Chapter 1

# Using DJM

*DJM* stands for Distributed Job Manager, a system initially developed at the Minnesota Supercomputer Center. DJM was designed to

- manage the flow of jobs through a Connection Machine CM-5 system
- avoid resource conflicts
- provide a load-balancing capability among the various partitions of a Connection Machine CM-5 system

## 1.1 Introduction

In order to balance system load, DJM handles all application processes running on the CM-5, interactive jobs as well as batch jobs. It allows jobs for which there are sufficient resources to execute immediately; it queues all other jobs for later execution.

- Interactive jobs execute *in the foreground*. They allow terminal input and output.
- Batch jobs execute *in the background.* They allow input and output only to specified files.

DJM ensures that it handles all jobs by trapping jobs that are invoked directly, rather than going through DJM's job-submission commands. It has the ability to penalize such trapped jobs (which it calls *foreign* jobs); whether it does so is up to the system administrator at each site.

DJM allows users and administrators alike a great deal of flexibility. Users can request either dedicated time or multi-user access. They can specify


*Copyright © 1993 Thinking Machines Corporation*

many parameters for their job, or only a few — or they can accept DJM's default parameters. They can ask that DJM send them mail when a job begins or ends execution. They can move a job from interactive execution (the *foreground*) to batch execution (the *background*), or vice versa.

CM system administrators are in charge of configuring DJM to meet the needs of their sites. They can choose how many queues to set up and what restrictions to put on various queues (which users can use the queue, what resource limits are placed on jobs using this queue, etc.). They can choose when to allow dedicated access to the system. They can determine how *foreign* jobs (jobs not submitted via DJM commands) are to be handled.

In general, DJM relies on a small number of queues to manage the job flow. At a minimum, there will be two queues at your site:

- **Default**: A general-purpose queue. Unless otherwise specified, all DJM jobs will be submitted to this queue.
- **Foreign:** A queue that handles jobs that are submitted from outside of DJM. While it is possible to bypass DJM and run jobs directly, they will still be managed by DJM, and end up in the foreign queue. The foreign queue usually has severe limitations imposed on it to force the use of DJM queues, so unless you are running a small, fast job, you should submit the job for DJM to handle.

### 1.1.1 Compatibility with NQS

Sites currently using the Network Queuing System (NQS) should note that DJM is compatible with NQS at the user level in its handling of requests for batch job submissions. Indeed, the system administrator can configure the system in such a way that user-level NQS commands (**`qsub`**, **`qlimit`**, **`qdel`**, and **`qstat`**) will continue to work as they did previously.

For a comparison of DJM and NQS, see Appendix G.


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

### 1.1.2 DJM Commands

Table 1 lists the DJM user commands and their NQS equivalents. These commands are described later in this chapter, as well as in Chapter 3, which contains UNIX-style man pages for all DJM commands.

**Table 1. DJM and NQS user commands.**

| DJM Command | NQS Command | Meaning |
|---|---|---|
| `jsub` | `qsub` | Submits a job for background processing. |
| `jrun` | `--` | Submits a job for interactive processing. |
| `jdel` | `qdel` | Deletes or signals a DJM job. |
| `jstat` | `qstat` | Displays status of DJM system. |
| `jconnect` | `--` | Connects a job to a terminal (tty). |
| `jmsg` | `--` | Sends a message to a job's output buffer. |
| `js` | `--` | Displays system-wide status of DJM system. |
| `jwait` | `--` | Waits for a DJM job to start or finish. |

## 1.2 Learning to Use DJM

A "job" is any user process that executes on the CM-5: an application program, a database query, and so on. To try out DJM, therefore, you can use any program or query that executes quickly and consumes only modest resources.

In order to create a consistent set of examples and offer a minimal program to users who typically run much larger programs, Section 1.2.1 offers two extremely simple sample programs (`simple.fcm` and `hello.c`) and provides instructions on how to compile and link them. You may use either of these programs as your own introductory program if you wish. We will use `simple.fcm` to generate examples throughout this chapter.


Copyright © 1993 Thinking Machines Corporation

### 1.2.1 A Simple Program

The program shown below is written in CM Fortran. It sets up three arrays of five elements each. It assigns values to each array element: array A receives the values 1,2,3,4, and 5, while all elements of array B receive the value 2. The program then squares each of these values, adds each element of A to the corresponding element of B, and puts the results in array C. It then prints the results. (Needless to say, this is not a typical data parallel program.)

```
program simple

integer a, b, c, n
parameter (n=5)
dimension a(n), b(n), c(n)

data a /1,2,3,4,5/
b=2

c= a**2 + b**2

print */ 'Array C contains:'
print *, c
end
```

Type this program in a file as you normally would; call the file **simple.fcm**. (Remember that in Fortran, each program statement must begin in column 7.)

To compile the program, issue the following command at your UNIX prompt (represented in this manual by the percent sign):

```
% cmf simple.fcm -o simple
```

You now have a CM Fortran program named **simple** that is ready for execution on the CM.

### 1.2.2 A "Hello World" Program

The following example shows a "hello world" program that is written in C and includes calls to the CM-5's CMMD communications library.

```
#include <stdio.h>
#include <cm/cmmd.h>
```


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

```
main(argc, argv)
   int argc;
   char *argv[];
 {
 CMMD_fset_io_mode(stdout, CMMD_independent);
 if (CMMD_self_address() < 16)
 printf("Hello world from node %d\n",CMMD_self_address());
 }
```

The program first sets the standard output stream to **CMMD_independent** mode, allowing each node to write to the output stream independently of all other nodes. It then asks nodes 0 − 15 to print out "Hello world from node *n*," where *n* is the node's logical address (**CMMD_self_address**) within the partition. Other nodes in the partition do nothing. Because output is independent, the nodes may respond in any order.

Type the program in a file; call the file **hello.c**. This program should be compiled and linked in two separate steps. Use the following command lines at the UNIX prompt:

```
% cc -c hello.c
% cmmd-ld -comp cc -vu -node hello.o -o hello_world
```

Your link line may vary in any of the following ways:

- **-comp cc** tells **cmmd-ld** to use the **cc** compiler. You may wish to specify a different compiler.
- **-vu** tells **cmmd-ld** that the program will run on a CM-5 equipped with vector units. Omit this argument if the CM-5 you use is not so equipped.
- **-o hello_world** tells **cmmd-ld** to name the executable file **hello_world**. You may want to choose another name.

You should now have a C program named **hello_world** that is ready for execution on the CM-5. Running the program will produce output like this (remember, the order in which nodes respond will vary):

```
% hello_world
Hello world from node 7
Hello world from node 8
Hello world from node 0
Hello world from node 15
Hello world from node 2
```


Copyright © 1993 Thinking Machines Corporation

```
Hello world from node 9
Hello world from node 4
Hello world from node 3
Hello world from node 12
Hello world from node 1
Hello world from node 13
Hello world from node 5
Hello world from node 10
Hello world from node 14
Hello world from node 6
Hello world from node 11
```

## 1.3 Submitting Jobs via DJM

The job submission commands for DJM are

- **jsub**, to submit a job for batch (background) execution
- **jrun**, to submit a job for interactive (foreground) execution

Processes submitted via **jsub** or **jrun** run under timesharing, just like any other processes.

### 1.3.1 Batch Jobs

To submit a batch job, you can either

- issue the command **jsub**, using as an argument the name of a script file that contains the name of the program to be run
- submit the program to **jsub** directly, by supplying the program name on the command line

For more details, see Section 1.4 below.

**NOTE:** Your system administrator may have configured DJM to accept the **qsub** command as an equivalent to **jsub**. If so, you can use **qsub** anywhere you would normally use **jsub**.


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

### 1.3.2 Interactive Jobs

You submit an interactive job by using the **jrun** command. The argument you give **jrun** can be

- the name of the script file that contains the name of the program or programs to be run
- the name of the program itself (this gives you an interactive "login" shell)

For more details, see Section 1.5 below.

### 1.3.3 Information Required

As mentioned above, DJM manages the flow of jobs on the CM system, balancing the load on each partition manager to avoid bottlenecks. When the resource requirements of a program exceed those available, DJM queues the job, then executes it when resources become available.

In order to manage the job mix effectively, DJM requires that each job submission include information about your expected usage of three resouces. You must specify

- number of processors required
- CPU time required
- total memory required

You may specify other requirements as well, by using options to the **jrun** and **jsub** commands. See Tables 2 and 3 for a list of these options. You specify requirements either in the shell script, on your command line, or as responses to DJM queries, as explained in the following sections.

### 1.3.4 Soft and Hard Resource Limits

DJM treats the essential information you give it when you submit a job as *soft limits* to help it allocate resources and schedule jobs. When you supply DJM with the amount of memory and CPU time you require, you are in essence contracting with DJM for those resources.


Copyright © 1993 Thinking Machines Corporation

If you overstep your estimates, DJM will not automatically kill your job. If there are no jobs waiting to run, your job may run to completion, even though you underestimated the time or memory your program required. However, once your job exceeds its requested limits, it does become vulnerable to being killed. The greater the overrun, and the greater the number of jobs currently waiting to run, the greater the chance your job will be killed.

DJM also provides two kinds of *hard limits.* The system administrator sets one type of hard limit on a per-queue basis: thus, a particular queue may allow jobs only a certain amount of CM time. The user sets the other limits, by specifying options to the **jsub** or **jrun** command, as shown later in this chapter.

### 1.3.5 Finding Information

To obtain information about the status of the DJM system (including items such as queue limits), issue the **jstat** command (or the **qstat** command, if your system is so configured).

The following output from a sample **jstat** command shows six jobs currently running, with three more queued and waiting to run. The jobs are running on a CM-5 that has four active partitions: a 128-PN partition in dedicated mode, and three smaller partitions in normal mode:

```
% jstat

USER     JID  MACH SERV  PROCS   TIME    MEM  STATUS   COMMAND
janet   4339  cm5          128  15:00     4G  Que nde  testcase6
joe     4338  cm5  cm5-2    64   0:00i     0  Running  /bin/csh
bert    4333  cm5  cm5-2    64   0:17i   58M  Running  td/job5
sally   4324  cm5  cm5-3    32   3:54i  162M  Running  hilbert3
daphne  4214  cm5          128 360:00D    4G  Que bsy  longjob5
brian   4329  cm5  cm5-2    64  14:48i  297M  Exiting  spec-6
lou     4308  cm5  cm5-4    32   2:46i     0  Running  /bin/csh
david   4270  cm5           64  30:34   120M  Que ded  testcase6
susan   4266  cm5  cm5-1   128   1:06D     0  Running  benchmark4
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

The fields in this **jstat** output are as follows:

| | |
|---|---|
| USER | The name of the user whose job this is. |
| JID | The job identifier, assigned by the system. |
| MACH | The machine on which the job is running or queued. |
| SERV | The server (partition manager) on which the job is running. This field is blank for queued jobs. |
| PROCS | For a running job, the number of processors in use. For a queued job, either the minimum or the exact number of processors requested. |
| TIME | For a running job, the amount of time used so far. For a queued job, the amount of time requested. The suffix **i** indicates an interactive job; **D** indicates a dedicated job; **f** would indicate a foreign job. |
| MEM | The amount of memory requested or in use. |
| STATUS | Currently executing jobs are marked as **Running**. Queued jobs are marked by **Que** plus a 3-letter code. In this example, **Que bsy** indicates a job that is queued because the processors it needs are busy running a dedicated job. **Que ded** shows a dedicated job queued on a non-dedicated partition. **Que nde** marks a non-dedicated job that is queued on a dedicated partition. |

For more information on the **jstat** command and its output, see the **jstat** man page in Chapter 3.

## 1.4 Submitting a Batch Request: The jsub Command

Use the **jsub** command to submit a program for execution when you want to run the job in the background. The program must not require interactive use.


Copyright © 1993 Thinking Machines Corporation

### 1.4.1 Examples

To execute the program **simple** from the default queue, use the program's name as the argument to **jsub**:

```
% jsub simple
```

Since you didn't specify the number of processors, CPU time, or memory, DJM prompts you for these:

```
Number of processors (64)? 32
Estimated CPU time (5min)? 1
Estimated memory (128M)? 2m
```

(**M** [or **m**; it is case-insensitive] specifies megabytes of memory. Note also that it represents total memory, not memory per processor.) Entering a carriage return for any value gets you the default value, which DJM has presented in parentheses.

DJM then submits the job, and tells you the job ID it has assigned it:

```
Job submitted successfully. Job id is 43.
```

When **simple** executes, DJM places its output in the current directory, in a file named **simple.o***jobid*: in this case, **simple.o43**.

Rather than waiting for prompts, you could specify your job's resources on the command line:

```
% jsub simple -nproc 32 -cputime 1min -mem 2m
```

The system would then respond:

```
Job submitted successfully. Job id is 44.
```

Finally, you could submit the job via a shell script.* A script that equates to the command line shown above would look like this:

```
#!/bin/sh
#JSUB-nproc 32
#JSUB-cpu 1min
#JSUB-mem 2m
#
```

* Shell scripts are command files; they provide job control information. Section 1.10 provides brief information about shell scripts, for users unfamiliar with them. Handbooks on UNIX and introductions to UNIX provide more information.



Copyright © 1993 Thinking Machines Corporation

```
# This is my program
simple
```

Having created a file containing your shell script, you would simply invoke the script to submit your job. For instance, if you had named your script **run_simple**, you would type

```
% jsub run_simple
```

### 1.4.2 Output of a Batch Request

When DJM executes your job, it places the job's output in a file named *jobname*.o*jobid*. The file contains a banner, the output from the program, and an account of the actual resources used by the program. This account can be used to estimate the resources needed for the next run of the same program. This allows DJM to manage your job more effectively.

The following example shows user Newton first submitting our sample C program, **hello_world**, on a 4-PN test machine named Mercury. It then shows Newton examining the resulting log file, using the UNIX **cat** command to do so:

---

```
% jsub hello_world
Number of processors (4)?          (Carriage returns accept
Estimated cpu time (5min)?         the default values.)
Estimated memory (4M)?
Job submitted successfully. Job id is 18.

                                   (After the job finishes,
% cat hello_world.o18              look at the log file.)
*** DJM V2.0.1, cm5 (mercury) TMC
*** Job hello_world (18) procs 4, mem 4M, cpu 5min, server mercury:0
*** newton hello_world started on Thu Sep 30 15:39:45 1993
Hello world from node 0
Hello world from node 3
Hello world from node 1
Hello world from node 2
*** Job terminated at Thu Sep 30 15:40:04 1993
*** exitcode = 0400
*** procs 4, mem 0, cpu 0
```

---


Copyright © 1993 Thinking Machines Corporation

### 1.4.3 jsub Options

The **jsub** command has many options associated with it. This chapter describes some of the more important ones:

- how to specify a particular queue (**-queue**)
- how to request exclusive access to a partition (**-dedicated**)
- how to set hard limits on resources
- how to request particular resources (**-machine**, **-need**, **-server**, **-exact_procs**)
- how to set priority (**-priority**)
- how to defer job execution (**-after**)
- how to specify use of a particular shell (**-shell**)
- how to request mail about a job (**-mail_begin**, **-mail_end**, **-mail_user**)

Table 2 summarizes the **jsub** options supported on the CM-5. See the man page for **jsub** in the Chapter 3 for a complete discussion of all options, including NQS compatiblity options.*

* Users accustomed to working on IBM systems will note that **jsub** options (and **jrun** options) serve much the same function as a Job Control Language, telling the system how to handle each separate job.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**Table 2. Options for the jsub command.**

| Option | Meaning |
|---|---|
| **-jobname** *name* | Defines a symbolic name for the job. |
| **-machine** *machine* | Specifies the CM-5 on which to run the job. |
| **-cputime** *n* | Requests *n* units of CPU time; you specify units as **sec**, **min**, or **hr** (for seconds, minutes, or hours). |
| **-memory** *n* | Requests *n* units of memory; you specify units as **G**, **M**, or **K** (for gigabytes, megabytes, or kilobyes). |
| **-nproc** *n* | Requests at least *n* processors. |
| **-dedicate** | Requests dedicated access to a partition. |
| **-after** *time* | Defers execution of the job until after the indicated time or date. |
| **-chdir** *path* | Changes the working directory to *path* before running the job. |
| **-exact_proc** | Runs the job on a partition with exactly the number of processors that were requested by **-nproc**. |
| **-execute** *file* [*file−args*] | Passes command line arguments to the submitted program. |
| **-export** | Exports all environment variables to the job. |
| **-hard_cputime** *n* | Specifies an absolute CPU time limit for the job. |
| **-hard_memory** *n* | Specifies an absolute memory limit for the job. |

*(continued)*


*Copyright © 1993 Thinking Machines Corporation*

**Table 2. Options for the jsub command** *(continued).*

| Option | Meaning |
|---|---|
| **-help** | Displays a summary of the options to **jsub**. |
| **-hold** | Prevents the job from running until it is released. |
| **-interruptable** | Tells the scheduler that the job can be interrupted. |
| **-linelimit** *n* | Limits the number of output lines the program can produce. |
| **-mail_begin** | Sends mail when the job begins execution. |
| **-mail_end** | Sends mail when the job completes execution. |
| **-mail_user** *user* | Sends mail to *user* when job starts and ends. |
| **-maxrerun** *n* | Sets the maximum number of times the job can be rerun. |
| **-need** *rsrcs* | Requires that the job run on a partition that offers resources *rsrcs*. |
| **-nodedicate** | Specifies that the job does not require exclusive access to the partition (default). |
| **-noexact_proc** | Specifies that the job does not require the exact number of processors (default). |
| **-nointerruptable** | Specifies that the job is not interruptible (default). |
| **-nopreamble** | Does not print the job startup output (from **.login/.***shell***rc**) (default). |
| **-noterminal** | Does not connect the job to a terminal (tty) (default). |
| **-notty** | Synonym of **-noterminal**. |

*(continued)*



*Copyright © 1993 Thinking Machines Corporation*

**Table 2. Options for the jsub command** *(continued).*

| Option | Meaning |
|---|---|
| **-preamble** | Prints startup output (from **.login/**.*shell***rc**). |
| **-priority** *n* | Sets user-defined priority to *n*. |
| **-processors** *n* | Synonym of **-nproc**. |
| **-procs** *n* | Synonym of **-nproc**. |
| **-project** *name* | Specifies the name of the project the job is being run for. |
| **-queue** *name* | Specifies the queue the job should be submitted to. |
| **-range** *m*−*n* | Specifies the processor range the job is to be run in. |
| **-safety** *n* | Sets the level of run-time safety checking to *n*. |
| **-server** *name* | Specifies that the job be run on a specific server. |
| **-shell** *name* | Specifies the shell for interpretation of the job submission script. |
| **-silent** | Submits the job without presentation of the job ID. |
| **-sp_disk** *n* | Requests *n* units of front-end disk space. |
| **-stderr** *name* | Sets the standard error file to *name*. |
| **-stdin** *name* | Sets the standard input file to *name*. |
| **-stdout** *name* | Sets the standard output file to *name*. |
| **-terminal** | Assigns the job a tty. Must be done if you want to **jconnect** to it while executing. |
| **-time** *n* | Synonym for **-cputime**. |
| **-tty** | Synonym for **-terminal**. |


Copyright © 1993 Thinking Machines Corporation

### 1.4.4 Specifying the Queue

CMs can be operated in two basic modes: *shared* mode (also called *normal* mode) and *dedicated* mode. Usually, a special period of time is set aside each day or week for the CM to operate in dedicated mode. In this mode, each job submitted has exclusive access to all the resouces of the partition. The rest of the time, the CM is shared by a number of users who are engaged in tasks such as debugging, testing, running small programs, etc.

When the CM is configured for *normal* access, there may be one or more queues that are active and can have jobs submitted to them. To specify the queue you want the job to be submitted to, use the **`-queue`** *queue* option to **`jsub`**. Your job will be submitted to the queue you specify. When the CM is in dedicated mode, only one queue (the dedicated queue) will be active. That queue is specified by the **`-dedicated`** option.

#### 1.4.4.1 Running Dedicated Jobs

If you require exclusive access to the CM partition, use the **`-dedicated`** option to **`jsub`** or **`jrun`**. This option declares that you require all the resources of the system. When you use this option, DJM adds your job to the dedicated queue. Your job then waits for DJM to be reconfigured to run the dedicated queue. At that time, the jobs in that queue will start to run, one at a time.

Two things are required for DJM to put any partition into dedicated mode:

- The system administrator must execute a reconfiguration command.
- There must be at least one job in the dedicated queue by a cut-off time prior to the scheduled reconfiguration. If no dedicated jobs are waiting by the cut-off time, the reconfiguration is cancelled. The amount of time between the cut-off time and the reconfiguration is site-specific; you should consult your system administrator to find out what it is at your site. The default is four hours.

If you need dedicated access to a CM, therefore, you must use the **`-dedicated`** option to **`jsub`**, so that DJM will recognize that there is at least one dedicated job that needs to run.

While the partition is in dedicated mode, DJM automatically kills any foreign jobs attempting to run. It sends such jobs an appropriate message.


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

The partition remains in dedicated mode either until the last dedicated job finishes running, or until a preset time (determined by the system administrator) is reached. When the first of these occurs, the system automatically reverts to normal (shared) mode. If dedicated jobs are not complete when the system reverts to normal mode, they will continue execution in shared mode, subject to system availablity. Dedicated jobs that have not yet been started will wait until the next stretch of dedicated time to begin execution.

### 1.4.5 Setting Limits on a Batch Request

As shown in Table 2, the **`jsub`** command has many options you can specify to set limits on the ammount of resources a batch request can use. The batch queue itself has its own set of limits. You can find out what they are by issuing the **`jstat`** command, with the **`limits`** subcommand.

You should set your limits as low as is reasonable to obtain more favorable scheduling for your request. If the limit you set exceeds the limit for the queue, the queue will not accept the request.

### 1.4.6 Choosing a Server

DJM performs implicit load balancing. Your job will run on whatever server has the most resources available to meet your job's needs. If you must run on a specific server, you should use either the **`-machine`** option or the **`-server`** option to **`jsub`** to specify the server you want the job to run on.

### 1.4.7 Choosing a Shell

When you use a shell script to submit a job, the system must decide what shell to use to interpret the script. It does so in the following manner:

1. DJM reads the first line in the script. If it contains the standard UNIX shell prefix **`'#!'`**, DJM uses the specified shell to interpret the script.

2. If the script's first line does not specify a shell, DJM looks for a **`-shell`** option on the command line, and uses the shell specified by that option.


Copyright © 1993 Thinking Machines Corporation

3. Failing that, DJM checks for your **SHELL** environment variable.
4. If there's no **SHELL** environment variable to use, DJM uses the shell specified in your login shell.
5. Failing all the above, DJM uses **/bin/sh**.

(Note for NQS users: These procedures are equivalent to NQS's concept of free and login shell strategy. DJM provides no counterpart to NQS's fixed shell strategy.)

### 1.4.8 Setting a Priority for a Batch Request

To set a priority for your batch request in its queue, use the **jsub** option **-priority** *n*. Minimum and maximum allowable priorities may vary from queue to queue at your site. To find out what limits have been placed on queues you may want to use, give the **jstat queue** *name* command and examine the **min_pri** (minimum priority) and **max_pri** (maximum priority) fields.

If you don't specify a priority, DJM assigns your request a default priority.

DJM uses priorities to determine how likely a job is to run when system resources are limited. (For full information on how it determines which job runs when, see Appendix C.)

### 1.4.9 Receiving Mail about a Batch Request

Use the **jsub** options **-mail_begin**, **-mail_end**, and **-mail_user** to specify that DJM is to send mail about your batch request. Specify **-mail_begin** to get mail when the request begins execution; specify **-mail_end** to get mail when the request ends execution. Use **-mail_user** to send mail to *user* rather than yourself.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

## 1.5 Submitting an Interactive Request: The jrun Command

You use the **jrun** command to run a job interactively from your terminal. As with any interactive program, the job will take input from your keyboard and direct output to your terminal. (You can also have the output captured in a file, if you wish, by using **jrun**'s **-capture** option.)

As with other jobs, DJM requires you to declare your program's needs for CPU time, etc. For additional job control, you can specify any of a number of options: all those available for **jsub** (listed in Table 2, page 13) as well as those available only with **jrun** (listed in Table 3, page 24).

After evaluating all information, DJM decides which partition (if any) is best able to run the job. If resources are available, it starts the job. Otherwise, it queues the request for later execution.

When **jrun** queues a job, you have three choices:

- You can wait for resources to become available.
- If you do not want to wait for resources to become available, and if your program can run in the background, you can disconnect your job from the terminal using **~**Ctrl-B (tilde Control-B). When resources become available, your job will run in the background, as if you had used **jsub** to submit it.
- If you cannot wait, and the job cannot run in the background, you can type **~**Ctrl-C to kill the pending job.

It is also possible to put a job into the background (as if you had used **jsub** to submit it) after the job has started running, by detaching the job from the terminal. To do this, you type the disconnect character (usually **~**Ctrl-B). You can reconnect the job to a terminal later on using the **jconnect** command (see Section 1.6 below).

While your job is disconnected from the terminal, job input and output are redirected to a disk file (as specified by the **-stdout** and **-stdin** flags); the default output filename is *jobname*.o*jobid*.


Copyright © 1993 Thinking Machines Corporation

### 1.5.1 Examples

In order to demonstrate running an interactive program, let's modify the program **simple** to prompt the user for the value of array B. Do this by adding the lines marked with **:

```
program simple

integer a, b, c, n, bval                    **
parameter (n=5)
dimension a(n), b(n), c(n)

data a /1,2,3,4,5/

print *, 'Enter value for array b:'         **
read *, bval                                **
b = bval                                    **

c= a**2 + b**2

print */ 'Array C contains:'
print *, c
end
```

Compile the program as you did previously:

```
% cmf simple.fcm -o simple
```

You now have a program that will prompt you for the value of the array B at run time.

#### 1.5.1.1 The Simplest Command Line

To execute an interactive program on the default queue, use the program's name as the argument to **jrun**:

```
% jrun simple
```

Since you didn't specify the number of processors, CPU time, or memory, DJM prompts you for these:

```
Number of processors (64)? <CR to take default>
Estimated cpu time (5min)? 1min
Estimated memory (64M)? 32M
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

DJM then responds with:

```
Job id is 195.
[Type ~^B to disconnect, ~^C to kill.]
[Connected to mars.think.com /dev/ttyp2.]
*** Job simple (195) procs 64, mem 64M, cpu 5min,
*** server mars.think.com:0
*** newton simple started on Sat Jan 23 15:51:46 1993
```

The program prompts for the value of array B:

```
Enter value for array b:
20
```

The program then runs to completion:

```
array c contains:
         401       404       409       416       425
*** Job terminated at Sat Jan 23 15:52:05 1993
*** exitcode = 0000
*** procs 64, mem 0, cpu 0
```

#### 1.5.1.2 Command Line with Options

You can specify your resource requirements on the command line (along with any other options you might want to specify):

```
% jrun simple -nproc 64 -cputime 1min -mem 2m
```

You will get the same responses as in the case above: first a job ID and some job information; then your program's prompt, to which you reply; and finally your program's output and some closing information.

#### 1.5.1.3 Shell Script

You can submit your job to **jrun** via a shell script. As with **jsub** shell scripts, the options are prefaced by either **jsub** or (at sites configured for NQS compatibility) **qsub**. In this example, the script looks exactly like a **jsub** script:

```
#!/bin/csh
#JSUB-nproc 64
#JSUB-cpu 1min
#JSUB-mem 2m
```


Copyright © 1993 Thinking Machines Corporation

```
#
# This is my program
simple
```

You then use **jrun** to submit the shell script:

```
% jrun submit.script
```

This results in:

```
Job id is 197.
[Type ~^B to disconnect, ~^C to kill.]
[Connected to mars.think.com /dev/ttyp5.]
*** Job submit_197 (197) procs 64, mem 2M, cpu 1min, server mars.think.com:0
*** newton /bin/csh submit started on Sat Jan 23 16:02:00 1993
simple
Enter value for array b:
20
array c contains:
         401         404         409         416         425
*** Job terminated at Sat Jan 23 16:02:18 1993
*** exitcode = 0000
*** procs 64, mem 0, cpu 0
```

#### 1.5.1.4 A Shell Script Invoking Prism

You can easily run Prism under DJM control. Simply issue the **jrun** command with the appropriate arguments and execute Prism in the DJM shell. The command line syntax is

```
% jrun scriptname
```

In this syntax, *scriptname* identifies a DJM shell script. A sample script might be:

```
#! /bin/csh
#JSUB -nproc 64
#JSUB -cpu 8hrs
#JSUB -mem 64M
#JSUB -export
```



Copyright © 1993 Thinking Machines Corporation

```
#
/bin/prism
```

If you name the above script **run_prism**, and give it execute permissions, you can use it as an argument to **jrun**. Note the use of the **-export** argument in the script. This argument ensures that all your environment variables are in effect when your job is run. Thus, the command line for the script shown above:

```
% jrun run_prism
```

runs Prism on the display referenced by your **DISPLAY** environment variable.

#### 1.5.1.5 Further Examples

Further examples of job submissions, including submissions of CM/AVS jobs and *Lisp jobs, are supplied in Section 1.10 at the end of this chapter.


Copyright © 1993 Thinking Machines Corporation

### 1.5.2 Options for jrun

The **jrun** command options are a superset of the **jsub** command options. Table 3 lists the options germane only to **jrun** on the CM-5. See the man page for **jsub** in Chapter 3 for a complete discussion of all options.

**Table 3. Options (in addition to those for jsub) for the jrun command.**

| Option | Meaning |
|---|---|
| **-capture** | Send output to both the terminal and a file specified by **-stdout**. |
| **-disconn** *char* | Specify the disconnect character. Default is **~**Ctrl-B. |
| **-echo** | Echo characters typed at the terminal (default). |
| **-eightbit** | Input to the job is interpreted as full 8-bit characters. Default is to strip the high bit and pass thru the lower seven bits only. |
| **-escape** *char* | Specify the escape character. Default is '**~**'. |
| **-nobell** | Do not ring bell when terminal connection is established. |
| **-nocapture** | Send output to only the terminal (default). |
| **-noecho** | Do not echo characters typed at the terminal. |
| **-noescape** | Disable the ability to escape from a job. |
| **-nostop** | Disable flow control (Ctrl-S and Ctrl-Q). |

### 1.5.3 Escape Commands

You can control the processing of an interactive job running under DJM by using escape commands. Each of these commands begins with an escape character. By default, this character is the tilde (**~**). However, you can use


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

the **-escape** option to **jrun** to define whatever character you like as the escape character.

Table 4 lists the escape commands.

**Table 4. jrun Escape commands.**

| Command | Meaning |
|---|---|
| ~Ctrl-B | Disconnect the terminal from the job. Synonyms: Ctrl-B (no escape character), ~B, ~d, and ~Ctrl-D. |
| ~Q | Terminate the job. Synonyms: ~. and ~Ctrl-C. |
| ~Ctrl-Z | Suspend the terminal session. (To bring it back to the foreground later, simply use the **fg** command as you would for any suspended UNIX job.) |
| ~Ctrl-Y | Suspend terminal input. Output continues to be sent to the terminal. |
| ~c | Toggle capture mode. |

## 1.6 Connecting a Job to a Terminal: The jconnect Command

You can use the **jconnect** command for jobs submitted with **jsub** or **jrun**:

- If you submit a job using the **jsub** command with the **-tty** or **-terminal** option, you can then use **jconnect** to attach a terminal to the job.
- If you submit a job using **jrun** and then disconnect the job from the terminal, you can reconnect the job to the terminal using **jconnect**.

Once the job is connected, you can interact with the job via the terminal.


Copyright © 1993 Thinking Machines Corporation

You use the job ID (given you by **jrun** or **jsub**) as an argument to the **jconnect** command. For example:

```
% jconnect -jid 145
```

connects job ID 145 to a terminal.

Remember: The job must have a psuedo-tty associated with it in order for the **jconnect** command to succeed. By default, jobs submitted with **jsub** do not have pseudo-tty's associated with them. You must specify the **-tty** (or **-terminal**) option to a batch job to be able to **jconnect** to that job.

## 1.7 Adding to the Job's Output Buffer: The jmsg Command

If you need to add text to a job's **stdout** buffer while the job is executing, use the **jmsg** command. If the job is connected to a terminal, then the message is sent to the terminal. Otherwise, it is sent to the file specified by the job's **-stdout** submission option.

In the example below, the user has submitted a job and uses the **jmsg** command to include text in the output file. As no **-file** argument is provided, text is read from standard input. A Ctrl-D terminates the input. The output file includes the message as expected.

```
% jsub -nproc 64 -mem 125m -cpu 10min -- fill_mem abc 50
Job submitted successfully.  Job id is 24.
% jmsg 24
This is a message for job 24
  Message sent.
%

*** DJM V2.0.0, cm5 (exp-vu)
*** Job fill_mem (24) procs 64, mem 125M, cpu 10min, server mars:0
*** newton /usr/diag/fill_mem started on Tue Aug 24 09:10:06 1993
This is a message for job 24
*** Job terminated at Tue Aug 24 09:11:35 1993
*** exitcode = 0000
*** procs 64, mem 123M, cpu 1min
```



Copyright © 1993 Thinking Machines Corporation

## 1.8 Waiting for a Job to Start: The jwait Command

Use the **jwait** command to perform a conditional command when a job starts or ends. When **jwait** command is issued with a job ID, the command waits until the job begins or ends (as specified by the **-begin** and **-end** options) and then returns. If the job fails to begin execution, **jwait** returns a nonzero return value.

In this example, job 38 is submitted and has to wait until resources are free to execute. When it completes, **jwait** returns.

```
% jsub -nproc 64 -mem 1536m -cpu 10min -- fill_mem abc 50
Job submitted successfully.  Job id is 38.
% jwait 38
Waiting..
Job 38 has terminated.
%
```

### 1.8.1 Chaining Jobs

You may wish the system to take action depending on the success or failure of a job. For example, if a short job runs successfully you may then want to run a longer version; if the short job exits with an error status, you may want to have an operator informed via a mail message.

**jwait** can be used to chain jobs. For example, you can submit a job and then wait for its completion to resubmit it, using **jwait** to perform the waiting.

```
% jsub script
Job submitted successfully.  Job id is 24.
% jwait -end && jsub script
```

You can also use **jwait** for cases where you need to synchronize several jobs on different machines and have them begin simultaneously. For example, you may have to perform steps that do not involve the CM but require the job that was running to have finished. Post-processing or spooling of output are two such examples. A script that copies the results when the CM program is done might look like this:

```
#!/bin/sh
jsub -stdout /sda/results myjob    # do some computing on the CM
jwait -end                         # wait for the job to finish
rcp /sda/results timbuktu@far.away # copy the results back to home
```


Copyright © 1993 Thinking Machines Corporation

## 1.9 Deleting a Job Request: The jdel Command

To delete a request (either batch or interactive) from a queue, issue the **jdel** command. As an argument, specify the job ID that was returned when you submitted the request. If you do not know the job ID of the job, you can use **jstat** to find out (see Chapter 2).

The syntax is

```
jdel job-id [-kill | -signo signal ]
```

The **jdel** command with no options sends a SIGTERM to the job. Five minutes later, a SIGKILL is sent. This allows the job five minutes to clean up before termination.

You can specify the initial signal to be sent with the **-signo** option. Also, you can use the **-kill** option to have a SIGKILL signal sent immediately to the job.

## 1.10 Further Examples

This section provides further introductory material on shell scripts. It also shows three examples of scripts used for DJM submissions:

- first, a heavily commented script used for running a benchmark
- second, a script for running CM/AVS under DJM
- third, a script for running *Lisp under DJM

### 1.10.1 UNIX Shells and Shell Scripts

A UNIX shell is a command interpreter. It takes a command line, interprets the command, checks syntax, invokes any UNIX calls, and returns the result to the user. Interpretation can include making use of features such as pattern-matching, if you type a wildcard; using the values of preset variables to determine the execution environment for a command; and so on. The general idea is to provide ease of use and flexibility for users.


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

Several types of shells exist: for example, the C shell, the Bourne shell, and the Korn shell. Each has slightly different features, but all share the features listed above.

All shells take input from "standard input." By default, this means commands that you type at the terminal; however, you can instruct the shell to take its input from other sources, such as shell scripts.

A shell script is simply a file that contains a list of commands, one per line. The commands may be UNIX commands, subsystem commands (such as DJM commands), or shell commands. When the shell script is invoked, the commands are executed in the order in which they appear in the file.

If you plan to execute a given program repeatedly, using a shell script may save you a good deal of typing time; you create the script once, then invoke it whenever you need it.

For further information about UNIX shells and shell scripts, you should consult one of the many introductions to UNIX that are available. Two such books are *UNIX for the Impatient*, by Paul W. Abrahams and Bruce R. Larson (Reading, Massachusetts: Addison-Wesley, 1992) and *UNIX in a Nutshell: A Desktop Quick Reference for System V and Solaris 2.0,* by O'Reilly and Associates (O'Reilly and Associates, 1992).

### 1.10.2 Running a Benchmark

This first example shows a shell script that contains UNIX, CMOST, and DJM commands. Comments precede lines in the script; they are set in italics.

Following the script, a sample logfile entry appears. To make the logfile simpler to read, output from the **jstat** and **cmps** commands has been edited. Normally these commands would return more information than this example shows.

---

```
    Indicate the choice of shell used to interpret non-DJM com-
    mands wtihin the script (here, the C shell).

#!/bin/csh
```


Copyright © 1993 Thinking Machines Corporation

*#JSUB <argument> is the notation for specifying resource requirements within a shell script. These lines are interpreted by DJM when it evaluates the submitted job.*

*This is a benchmark run, so ideally one would like it to run in single-user (dedicated = non-timeshared) mode.*

```
#JSUB -dedicated
```

*Provide information for project accounting.*

```
#JSUB -project bench-d
```

*Tell DJM that you want the exact number of processors specified as the argument to -nproc (see below).*

*-exact_proc is useful when performing benchmark timings that require a particular number of processors.*

```
#JSUB -exact_proc
```

*Specify the number of processors desired*

```
#JSUB -nproc 512
```

*Specify the estimated CPU time required by the job.*

```
#JSUB -cpu 15m
```

*In this script, the memory argument is not strictly necessary, since the user has already specified -exact_proc and a 512 PN partition, a combination which effectively requests 16G of memory. When estimating the amount of memory required by a job, keep in mind that the OS occupies a certain amount of memory per node (approximately 10%).*

```
#JSUB -mem 4G
```

*Specify that the logfile for this job be sent to the named file in the directory from which the script was submitted to DJM.*

```
#JSUB -stdout v1-512pn.log
```



*Copyright © 1993 Thinking Machines Corporation*

*The remaining commands in the script are processed sequentially once DJM has scheduled the user's job and it has begun to execute on a CM partition. The CP for that partition processes the commands through the shell interpreter specified on the first line of the script.*

*jstat lists all users currently running on the machine, the CP on which they are executing, the code they are running, etc. (See "man jstat".) It is invoked within the shell script in order to record the load on the machine at the time that the user's job begins to execute.*

```
jstat
```

*Define a short environment variable to stand for a long pathname. (To see all currently set environment variables within the C shell, type "env" at the shell prompt.)*

```
setenv R_DIR ~kepler/bench/ver4/cm5/512pn/benchmark3
```

*cmps provides information on the partition on which the code is executing (# PNs, available memory, user processes, etc.). It is the CMOST version of the regular UNIX "ps" command.*

```
cmps
```

*The next two commands prepare the user's directory on the CP for a new series of runs. In the particular environment for which this script was prepared, the system administrators have chosen not to permit direct user logins to the CPs. Therefore, all file management must be done within the DJM script.*

```
cd /scratch/kepler/bench/ver4/benchmark3-2
rm *
```

*Copy the executable from the scratch disk of the compile-server ("pluto") to the scratch disk of the CP (".", the current directory).*

```
cp /n/pluto/scratch/kepler/bench/ver4/1024grid/v1-512-p .
```

*Copy any input files from the user's file space to the current directory.*

```
cp ~kepler/bench/ver4/code/input.dat ./input.dat
```


Copyright © 1993 Thinking Machines Corporation

*Modify the permissions on the program to be universally executable. (These permissions are not preserved in the cross-machine copy.)*

```
chmod 777 v1-512-p
```

*Date/time stamp.*

```
date
```

*Execute the program.*

```
v1-512-p > out.512.1
date
```

*Copy the output from the CP scratch disk to permanent storage on the user's file system.*

```
cp out.512.1 $R_DIR
date
```

*Repeat the run another two times to test timing consistency and dependence on CP load.*

```
v1-512-p > out.512.2
date
cp out.512.2 $R_DIR
date
v1-512-p > out.512.3
date
cp out.512.3 $R_DIR
date
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

Here is a sample logfile from the script above:

```
*** Distributed Job Manager 0.9.14, cm5
*** Job v1-512-p (43266) procs 512, mem 16G, cpu 15min, server cm5-1:0
***
***                              DEDICATED MODE
***
*** kepler /bin/csh v1-512pn.sh started on Wed Oct  6 12:07:45 1993
jstat
USER       JID MACH  SERV    PROCS   TIME   MEM  STATUS   COMMAND
hazlitt  43214 cm5             512  360:00D   4G  Que bsy  myjob2
newton   43107 cm5             256  180:00D   2G  Que bsy  devt/cm5/longjob5
kepler   43066 cm5   cm5-1     512    0:00D    0  Running  bench/cm5
/v1-512pn.sh setenv R_DIR ~kepler/bench/ver4/cm5/512pn/benchmark3


cmps
Partition    Host  Size Start Stop  State   Description
-----------------------------------------------------------------
cm5-1+512+1  cm5-1 512  1024  1535  ACTIVE  CMOST-7.3.BETA.1
cm5-1:  512 PN System, 30224K mem. free, 7556K VU mem. free, 0 procs,
TS-9/25/
cd /scratch/kepler/bench/ver4/benchmark3-2
No match.
cp -i /n/pluto/scratch/kepler/bench/ver4/1024grid/v1-512-p .
cp -i ~kepler/bench/ver4/code/input.dat ./input.dat
chmod 777 v1-512-p
date
Wed Oct  6 12:08:07 MDT 1993
v1-512-p
date
Wed Oct  6 12:08:36 MDT 1993
cp -i out.512.1 /users/kepler/bench/ver4/cm5/512pn/benchmark3
date
Wed Oct  6 12:08:37 MDT 1993
v1-512-p
date
Wed Oct  6 12:09:05 MDT 1993
cp -i out.512.2 /users/kepler/bench/ver4/cm5/512pn/benchmark3
date
Wed Oct  6 12:09:05 MDT 1993
v1-512-p
date
```


*Copyright © 1993 Thinking Machines Corporation*

```
Wed Oct  6 12:09:33 MDT 1993
cp -i out.512.3 /users/kepler/bench/ver4/cm5/512pn/benchmark3
date
Wed Oct  6 12:09:34 MDT 1993

*** Job terminated at Wed Oct  6 12:09:34 1993
*** exitcode = 0000
*** procs 512, mem 2.49G, cpu 1.1min
*** Account bench-d will be billed approximately 297.733 megabyte hours
```

### 1.10.3 Running CM/AVS under DJM

In order to run CM/AVS under DJM, you must have AVS perform a DJM job submission for you. You can accomplish this by creating a copy of the following script in your local directory and having your **`.avs-hosts`** file reference it. Make sure that you give the file execute permissions.

```
#! /bin/csh -f

# Script that is called by AVS to run CM/AVS modules under DJM.
# list_dir doesn't need to run on the CM, but since AVS has a tight timeout
# limit on list_dir, using DJM slows things up so that it does indeed
# timeout. So, just run that outside of DJM.

# All other programs use jsub to force it to run under DJM.

# The values of nprocs, cpu, mem and queue should be modified to meet the
# needs of your site.

if ($argv[3] == list_dir) then
   $argv[1] $argv[2-$#argv]
else
  /usr/etc/jsub -nprocs 0 -cpu 8hr -mem 64M \
                -silent -queue default \
                -execute $argv[1] $argv[2-$#argv]
endif
```



Copyright © 1993 Thinking Machines Corporation

You would save this file in your local directory, then modify your **.avs-hosts** file to reference the script as a third argument to the **/usr/ucb/rsh** call (the first two arguments being the *hostname* and *−n* respectively). For example, if you named the file **.avs-djm-startup**, your **.avs-hosts** file might look like this:

```
mars-djm "/usr/ucb/rsh mars.think.com -n /users/joe/.avs-djm-startup" \
         /usr/lib/cmavs_library /usr/avs/data

venus-djm "/usr/ucb/rsh venus.think.com -n   /users/joe/.avs-djm-startup" \
        /usr/lib/cmavs_library /usr/avs/data
```

When you edit remote modules, you are prompted from the hosts named in your **savehosts**. It may take a few seconds longer to return the remote modules, as DJM is starting a job. This first job reads the remote modules and adds them to the network editor palette. It then exits. When you instantiate a CM/AVS module in the editor, a second DJM job starts. The job remains active as long as a CM/AVS module is active in the network editor, or until your time and memory allocation is consumed.

### 1.10.4 Running *Lisp under DJM

Like running Prism, running an interactive *Lisp band under DJM is simply a matter of creating a shell script to be used as an argument to **jrun**. The following can be used as an example:

```
#! /bin/csh
#JSUB -nprocs 32
#JSUB -cpu 1hr
#JSUB -mem 64M
#JSUB -queue interactive
#
/usr/bin/starlisp-vu-dev-f7600
```


Copyright © 1993 Thinking Machines Corporation

An example of running *Lisp from the command line (via DJM) is shown below.

```
% jrun -nproc 64 -mem 125m -cpu 60min -- /usr/bin/starlisp-vu-dev-f7600
Job id is 25.
[Type ~^C~^B to disconnect, ~~^C to kill.]
[Connected to mars.think.com /dev/ttyq5.]
*** Job starlisp (25) procs 64, mem 125M, cpu 1hr, server mars:0
*** newton /bin/csh /usr/bin/starlisp-vu-dev-f7600 started on Tue Aug 24 09:16:38 1993
%
exec /usr/etc/starlisp-dhod-f7600
/usr/etc/starlisp-dhod-f7600
;;; Lucid Common Lisp/SPARC
;;; Development Environment Version 4.1, 12 October 1992
;;; Copyright (C) 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992 by Lucid, Inc.
;;; All Rights Reserved
;;;
;;; This software product contains confidential and trade secret information
;;; belonging to Lucid, Inc.  It may not be copied for any reason other than
;;; for archival and backup purposes.
;;;
;;; Lucid and Lucid Common Lisp are trademarks of Lucid, Inc.  Other brand
;;; or product names are trademarks or registered trademarks of their
;;; respective holders.

;;; Loading source file "/usr/lib/starlisp/initializations/initializations.lisp"

;;; Looking for file-set starlisp-patch-f7600 directory definition in /usr/dfs/
site/starlisp-patch-f7600.dfs
;;; Loading source file "/usr/dfs/site/starlisp-patch-f7600.dfs"
;;; File-set directory for starlisp-patch-f7600 defined.
;;; Looking for def-file-set starlisp-patch-f7600 in /usr/lib/starlisp/patch/def-
file-set.lisp
;;; Loading source file "/usr/lib/starlisp/patch/def-file-set.lisp"
;;; File-set starlisp-patch-f7600 defined.
;;;         Lisp World: lcmw-4-1
;;;            Created: Tuesday, 17 November 1992 at 14:17:45 EST
;;;    Produced within: SunOS Release 4.1.2 (CPU) #8: Thu Sep 17 14:48:45 EDT 1992

;;;
;;; User World: starlisp-dhod-f7600
;;;    Created: Wednesday, 14 July 1993 at 11:29:07 EDT

%
```


*Copyright © 1993 Thinking Machines Corporation*

At this point *Lisp is active. A call to **cm:finger** will return output similar to **cmps**:

```
% (cm:finger)
64 PN System, 29468K mem. free, 7172K VU mem. free, 2 procs,
TS-7/21/93-1:56 (CMOST 7.3 Beta 2) Daemon up: 15:06

USER      PID   CMPID  TIME   TEXT    ILH     ILS     IGS    IGH    VUS    VUH
COMMAND
galileo   16246 1      0:00   496K   112K     48K      0K     0K     4K     0K  `
testsuite1
newton   *16247 2      0:00    64K   212K     48K      0K     0K     4K     0K
router_msgs_mimd

If you would rather see output similar to jstat, you can load a small
lisp file that redefines cm:finger. The lisp code for this is given at
the end of this section.

% (dfs:load-n "/u/newton/cm_finger")
(dfs:load-n "/u/newton/cm_finger")
;;; Compiling /u/newton/cm_finger.lisp
;;; You are using the compiler in DEVELOPMENT mode (compilation-speed = 3)
;;; If you want faster code at the expense of longer compile time,
;;; you should use the production mode of the compiler, which can be obtained
;;; by evaluating (lisp:proclaim '(optimize (compilation-speed 0)))
;;; Generation of full safety checking code is enabled (safety = 3)
;;; Optimization of tail calls is disabled (speed = 2)
;;; Reading source file "/u/newton/cm_finger.lisp"
;;; Writing binary file "/u/newton/cm_finger.sbin41x"
;;; Loading binary file "/u/newton/cm_finger.sbin41x"
;;; Warning: Redefining Function FINGER which used to be defined in "//proj/starlisp/
crtl/f7600/boot-code.lisp"
#P"/u/newton/cm_finger.sbin41x"

Calling cm:finger now produces output similar to jstat:

% (cm:finger)
USER       JID MACH  SERV PROCS    TIME    MEM STATUS  COMMAND
newton      28 exp-v phant   64    0:02f 21.2M Running /usr/diag/tsd/router_msgs_mimd
newton      25 exp-v phant   64    0:00i     0 Running /usr/bin/starlisp-vu-dev-f7600
```


*Copyright © 1993 Thinking Machines Corporation*

The lisp code to produce output like this is as follows:

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: cmi5; Base: 10; muser: yes-*-

(in-package 'cmi5)

(setq cm::*finger-program* "jstat")
(defvar cm::*jstat-args* '(:configuration :global :jobs :limits :machine
                           :processes :up :partitions :queues :servers :scores))

(defun cm:finger (&optional argument)
  (cond ((null argument)     ; no arg means use jobs
     (setq argument :jobs))
    ((not (member argument cm::*jstat-args*))    ; invalid arg
     (setq argument nil)))
  (if argument
    (progn
      (lcl:run-program cm::*finger-program* :arguments argument)
      (values))
    ;; argument was nil
    (error "Unknown argument")))
```



*Copyright © 1993 Thinking Machines Corporation*

## Chapter 2

# DJM Administration

This chapter describes how to configure and administer the DJM job management system on the Connection Machine CM-5 system.

DJM is designed so that you can configure it to meet the needs of your site. For example, you can:

- configure different types of queues for different classes of jobs
- restrict a queue to specified users or groups of users, thereby limiting access to the CM to high-priority jobs
- set limits on the resources a job in a queue can request

How each site configures DJM and the CM depends on the computing needs of the site. This chapter provides information that will permit the system administrator to configure DJM and the CM for maximum resource use.

This chapter provides, first, an overview of DJM, covering

- the daemons and commands that control DJM
- DJM partitions
- DJM modes
- DJM queues
- DJM's job scheduling philosophy
- DJM configuration files

Second, the chapter explains how to administer DJM. It discusses

- initial configuration tasks


*Copyright © 1993 Thinking Machines Corporation*

- daily operations
- troubleshooting

## 2.1 DJM Overview

### 2.1.1 DJM Daemons and Commands

DJM is composed of four daemons and an user interface program. (See Figure 1.) The daemons are:

**djm_master** This is the primary daemon. There must be one and only one for each Connection Machine at your site. **djm_master** provides the brains of the system. It schedules jobs and interfaces with the other daemons to obtain system status and job information. It can run on any system, and need not be run on a CM partition manager.

**djm_fingerd** This daemon runs on each partition manager. It is responsible for querying the timesharing daemon for information on what CM processes are running and the resources they are using. **djm_fingerd** communicates this information to **djm_master** for use in scheduling decisions.

**djm_execd** Like **djm_fingerd**, this daemon must run on each partition manager. It receives commands from **djm_master** to run a job. To do so, it spawns the fourth DJM daemon to execute the user process.

**djm_shepherd** One **djm_shepherd** daemon, spawned by **djm_execd**, exists for each user process. **djm_shepherd** reports job status to the **djm_master** and manages the pseudo-terminal interaction for interactive jobs. (UNIX afficionados will be interested to know that it manages them in a manner similar to the BSD **rlogin** daemon.)



Copyright © 1993 Thinking Machines Corporation

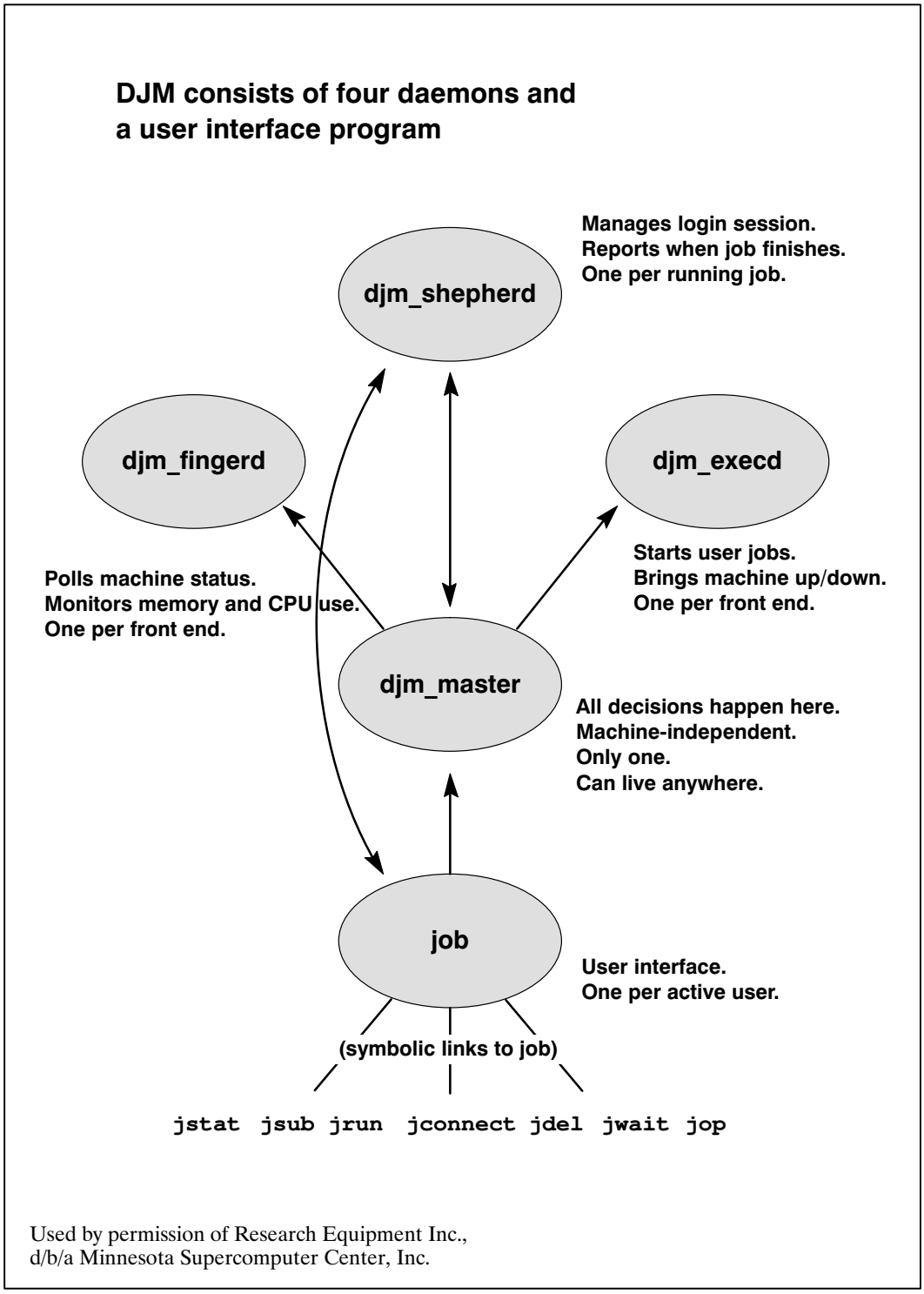


**Figure 1. Functional layout of DJM.**


Copyright © 1993 Thinking Machines Corporation

The user interface program is named **job**. The **job** program takes a number of subcommands, each of which is equivalent to a **jXXX** command:

| **job Subcommands** | **DJM Command** |
|---|---|
| **op** | **jop** |
| **run** | **jrun** |
| **submit** | **jsub** |
| **wait** | **jwait** |
| **msg** | **jmsg** |
| **connect** | **jconnect** |
| **status** | **jstat** |
| **delete** | **jdel** |

Thus, for example, the two commands "**job status**" and "**jstat**" produce identical results; you can use whichever you prefer. In fact, **jstat** and other DJM commands are symbolic links to **job**. When a **jXXX** command is executed, the **job** command looks at the name under which it was invoked and runs the appropriate **job** subcommand.

The DJM commands **jrun**, **jsub**, **jwait**, **jmsg**, **jconnect**, and **jdel** (and their **job** subcommand equivalents) are primarily user commands. **jstat** is used by both system administrators and users. These commands are discussed in Chapter 1 and in the appropriate man pages.

The **jop** command is used by system administrators. It has a number of subcommands, as shown in Table 5. Some of these subcommands are discussed later in this chapter; all of them are discussed in the **jop** man page in Chapter 3. Options to **jop** commands are discussed in the man page.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**Table 5. jop Commands.**

| **jop Commands** | **Meaning** |
|---|---|
| **General commands:** | |
| <null> | When executed with no arguments, **jop** enters a command shell; it prints a banner and a prompt and waits for you to type commands from the keyboard. |
| **help** | Lists and describes all **jop** commands. |
| **reconfig** | Configures DJM partitions. A partition must be configured before DJM will run jobs. There are many options, explained in the **jop** man page.<br><br>The DJM scheduler must be frozen before partitions can be reconfigured. |
| **wipe_reconfig** | Cancels one or all pending reconfigurations. |
| **freeze** | Halts all scheduling activity. Running jobs continue, but no new jobs are started. |
| **unfreeze** | Resumes normal scheduling activity after a **freeze**, or as part of system startup. |
| **goall** | Same as **unfreeze**. |
| **sync** | Writes all state information to the files in **/usr/spool/djm**. This occurs automatically every 5 minutes, and when DJM is frozen. You should use this command as a precaution before doing anything risky. |
| **reload** | Reloads all state information from disk. Use this command after tuning DJM parameters, before unfreezing DJM. |

*(continued)*


*Copyright © 1993 Thinking Machines Corporation*

**Table 6. jop Commands** *(continued).*

| **jop Commands** | **Meaning** |
|---|---|
| **Commands that control queues:** | |
| **enable** | Allows a queue to accept job submissions. |
| **disable** | Queue will not allow new submissions, but will run queued jobs. |
| **start** | Allows jobs to execute from a given queue. |
| **stop** | Stops a given queue. Currently running jobs will continue to execute, and additional jobs may be queued; but no new jobs will run. |
| **Commands that control user jobs:** | |
| **run** | Runs a specified job as soon as possible, giving it priority over all other queued jobs (but not over running jobs). |
| **rerun** | Reruns a previously run job; kills and re-queues a running job. |
| **hold** | Holds a queued job. The job will not be executed until it is released by the **job release** command. (Yes, that is "**job release**" not "**jop release**." Sorry about that.) |
| **suspend** | Suspends a running job. (Sends a SIGSTOP signal to the job. See **resume**.) |
| **resume** | Resumes a suspended job. (Sends a SIGCONT signal to the job.) |
| **delete** | Deletes a queued job, or kills a running one (after a 5-minute grace period). Works like **jdel**. |



*Copyright © 1993 Thinking Machines Corporation*

### 2.1.2 DJM Partitions

DJM views the Connection Machine much as the system administrator does: as a set of one or more partitions, each containing a certain number of contiguous processing nodes, each under the control of its own timesharing daemon. DJM, in fact, has its own logical "DJM partitions."

The system administrator creates DJM partitions by using the **`jop reconfig`** command. Once operational, the partitions are controlled by the **`djm_master`**, **`djm_fingerd`**, and **`djm_execd`** daemons, as explained in Section 2.1.1.

The **`jstat partition`** command shows the state of *running, current,* and *future* partitions. A *running* partition reflects the actual physical partitioning of the CM-5, as set by the **`cmpartition`** command. A *future* partition is a declaration of the types of jobs that DJM will be allowed to run on a range of nodes on the CM-5 at some time in the future. A *current* partition is a future partition that has been activated.

The remainder of this chapter, when it refers to *partitions*, will refer to DJM logical partitions.

### 2.1.3 DJM Modes

DJM partitions can operate in one of three modes: *normal*, *dedicated*, or *halted*. The mode is specified when the partition is configured.

In *normal mode*, the partition accepts jobs that meet the limits of the partition. It runs, under timesharing, as many jobs as will fit on the system. It queues other submitted jobs for execution when resources become available.

In *dedicated mode*, the partition runs only one job at a time; the job has access to all partition resources without competition. This allows large production jobs to be run without the impact of smaller, less important jobs competing for resources.

In *halted mode*, the partition does not schedule new jobs. Jobs that are running when the partition enters halted mode continue to run unless **`-kill`** is specified at reconfiguration time.


Copyright © 1993 Thinking Machines Corporation

### 2.1.4 DJM Queues

As initially installed, DJM uses two queues; the **default** queue and the **foreign** queue. You will probably want to add more queues to meet specific needs for your site. For example:

- Experience shows that most CMs are used both in normal and in dedicated mode. If your site has jobs that need exclusive access to a partition, you need to establish a **dedicated** queue.
- You may want to create an interactive queue that allows users to run Prism or CM/AVS jobs. Such a queue might have no memory usage penalties and might allow hours of CPU time.
- You may want special queues to run during nights and/or weekends.
- You may want to create queues that are tailored for small or large jobs, based on time and memory requirements.

A note on the **foreign** queue: While DJM is used to manage the resources of the CM, it is possible for users to invoke jobs on the CM outside of DJM (by not using **jsub** or **jrun**). DJM places these processes in the **foreign** queue. However, since DJM has no information about these jobs other than that supplied by the timesharing daemon, and hence doesn't know what these jobs require, excessive foreign jobs can cause DJM to mismanage the DJM jobs. To control this, the administrator can set limits on the **foreign** queue, controlling the amount of memory and/or time a foreign job may consume.

#### 2.1.4.1 How DJM Chooses Queues

If the user does not specify a queue when submitting a job via **jsub** or **jrun**, DJM chooses an appropriate queue from which to run the job. To do this, DJM looks at the **search_order** field in the queue configuration file. DJM searches the queues by search order, lowest to highest, trying to find a queue that meets all the user-specified criteria (memory, CPU time, resources, etc.) as well as DJM criteria (user and group access restrictions, maximum number of jobs a single user may submit, etc.). For more information on queue configuration files, see Appendix B.

When it finds a suitable queue, DJM adds the job to that queue. If DJM finds no queue that meets the needs of the job, it rejects the job.


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

For best DJM performance, therefore, the most restrictive queues should be given the lowest search orders. The **`default`** queue, on the other hand, should have the highest-valued **`search_order`** field and the least restrictive memory and CPU limits. Most DJM jobs will find a suitable queue before being placed in the **`default`** queue, and DJM will be able to manage the job mix accordingly.

### 2.1.5 Queues, Scores, and Scheduling

DJM does not use the term *queue* in the strict sense; that is, a DJM queue is not restricted to a FIFO ordering of jobs. Rather, DJM queues provide a mechanism for job classification, with different classes of jobs (dedicated or shared, interactive or batch-style, requiring few or many resources) being run from different queues.

When a user submits a job, DJM gives the job a numeric *score*, based on a number of factors. Jobs with high scores take priority over jobs with lower scores. A job runs when resources are available and its score is high enough. (See Appendix C for an explanation of DJM scores.)

#### 2.1.5.1 Soft Scheduling

Jobs request certain resources when submitted. DJM makes the resources available; in return, the user agrees not to exceed the request. If the user's job does exeeced the request, the system has the right to kill the job in favor of another job that is waiting to run.

DJM, however, uses soft scheduling for request limits. After all, there is no reason to kill a job if no other job is waiting. If a job exceeds its requested time and memory, it continues to run until another job requires the resources.

At first glance, it may seem that users could escape having their jobs killed by simply requesting the maximum resources that their system could supply. Doing this, however, could lower the job's score, and could well cause the request to spend a long time in the queue, waiting for the large amount of resources to become available. Hence, it behooves the user to make a realistic appraisal of resources required and to request no more than is likely to be needed.


Copyright © 1993 Thinking Machines Corporation

#### 2.1.5.2 Recalculations

DJM scores are calculated using two linear equations, one for queued jobs and one for running jobs. The scores are recalculated each time the DJM scheduler executes, so that they are constantly updated and reflect the state of the system accurately.

Note that both running jobs and queued jobs have scores. If the score for a queued job is higher than the score for a running job, then the running job will be killed to make room for the queued job. In this case the job is sent a SIGTERM and allowed 5 minutes to clean up (checkpoint itself) before it is sent a SIGKILL. The waiting job then starts execution.

Normally, a job that has not yet used the resources it requested should have a higher score than any queued job; otherwise, a job could be killed before it reaches its soft limits. The system administrator should ensure that the queue configuration files enforce this scoring strategy, unless there are particular reasons to do otherwise.

### 2.1.6 DJM Configuration Files

There are a number of files that DJM uses to store system status and configuration information.

**Filename** **Contains**

In **`/usr/spool/djm`**:

**`global`** Global parameters for DJM. (This is the global configuration file.)

**`queue_`***name* Parameters for the named queue. (This is the queue configuration file.)

**`partitions`** Information regarding the DJM partitions available to run jobs.

**`log`**.*x* The log file DJM keeps regarding its operations.

**`djm_`***daemon-name***`_pid`**.*hostname*
The process ID of the named daemon (**`master`**, **`execd`**, or **`fingerd`**), running on *hostname*.



Copyright © 1993 Thinking Machines Corporation

| | |
|---|---|
| *jid* | Information on the active job that has the identification number *jid*. (This is the job state file.) |
| **old/***jid* | Job file for a completed job, saved in the **old** directory so that the job can be rerun, if needed. |

Two other files are usually located in **/etc/cm/configuration**:

| | |
|---|---|
| **djm.conf** | The DJM system configuration file. |
| **djm.cookie** | A per-site DJM message authentication key. |

#### 2.1.6.1 DJM Global Configuration File

The global configuration file specifies the global parameters of the DJM system. Some of the fields are updated automatically by **djm_master**, and some are set by the system administrator. Appendix A provides a detailed discussion of the global configuration file. Section 2.2.2 discusses the procedure for changing the file.

#### 2.1.6.2 DJM Queue Configuration Files

For each queue available under DJM, there is a **queue_***queuename* file in **/usr/spool/djm**. The file specifies the parameters of the queue, and is updated by **djm_master** to reflect the current state of the system. Some of the fields are modifiable by the operator, some are not. The fields are described in detail in Appendix B.

Note, in particular, that the factors for the linear equations on which DJM scores are calculated are set in the **queue_***queuename* file. The equations that are used to calculate the scores are presented in Appendix C.

#### 2.1.6.3 The DJM Partition Configuration File

DJM stores information about the machine partitions under its control in a file named **partitions**, in the spool directory. Reconfiguration commands automatically update this file. There is no need for the system administrator to edit this file by hand. Appendix D presents an example of a **partitions** file.


Copyright © 1993 Thinking Machines Corporation

#### 2.1.6.4 DJM Log Files

DJM records its transactions in a log file, stored in the spool directory. This log file is rolled over at regular intervals (usually daily), using the **newdjm-log** script that is distributed with the DJM package. By default, 8 days of DJM logs are kept in the spool directory. To modify this, edit the **newdjm-log** script to the value appropriate to your site.

The log contains text entries for each job that is run, including foreign jobs. An example of a log entry is shown below. (Linebreaks have been edited to keep the example within margins.)

```
Mar 11 13:29:33 SUBMIT: user newton, cdir /u/newton/cmf/bgk, file djm-job
Mar 11 13:29:33 SUBMIT: requested_nproc 64, requested_cputime 300,
                        requested_memory 524288
Mar 11 13:29:33 SUBMIT: jid 1565, state Que, pri 50,
                        dedicated no, exact_proc no
Mar 11 13:29:33 SUBMIT: queue default
Mar 11 13:29:35 RUN: job 1565, user newton, server venus.think.com,
                        interface 0, nproc 64
Mar 11 13:29:35 RUN: score 100000, cputime 300, mem 524288, project unknown
Mar 11 13:31:05 FINISHED: job 1565 finished, exitcode 0000,
                        cputime 29, max_memory 403456.
Mar 11 13:31:05 FINISHED: user newton, file djm-job, project unknown.
Mar 11 13:31:05 DELETE: job 1565 deleted.
```

In this example, Newton submitted the job requesting 64 processors, 5 minutes of CPU time, and 512 Mbytes of memory. DJM assigned it job ID 1565 and put it in the default queue for scheduling with priority 50. Neither dedicated access nor an exact number of processors was required.

The job was run on the server Venus with a base score of 100000. Finally, the job completed successfully (exitcode 0000), taking 29 seconds of CPU time and approximately 404 Mbytes of memory.

Note that, due to the asynchronous nature of job submissions, a realistic log file will have job log messages interleaved with other job log messages. However, the overall order for each job remains the same.



Copyright © 1993 Thinking Machines Corporation

#### 2.1.6.5 PID files

In order for the DJM daemons and user programs (the **job** command and its associated subcommands) to exchange information, the process ID of each daemon is stored in a file in the spool directory. Each file is named by the daemon name and the hostname the daemon is running on. For example,

```
djm_execd_pid.mars.think.com
djm_fingerd_pid.mars.think.com
djm_master_pid.mars.think.com
```

Each of these files contains the **pid** of the named process on the named host. Don't edit these files or the DJM daemons will not be able to communicate.

#### 2.1.6.6 Job Files

DJM keeps a detailed snapshot of each job that is running, in a file named by the job ID of the job. When the job is completed, DJM moves the file to the **old** directory in the spool directory. Appendix E contains a sample of a job file. The job state file in the **old** directory can be used to rerun the job.

#### 2.1.6.7 System Configuration File: /etc/cm/configuration/djm.conf

The DJM system configuration file contains a number of fields and associated values. It defines the physical machines that comprise the system, establishes pathnames, etc. You must edit the file to make it conform to your site. Lines beginning with a '**#**', tab, space. or newline are ignored. Appendix F presents a sample configuration file and describes each field.

#### 2.1.6.8 Authentication and the djm.cookie File

In order to provide for secure message passing between the various DJM daemons and clients, each DJM TCP message has an authentication key included with it. The receiver of the message uses the key to determine whether to accept or reject the message.

The authentication key is made up of two parts. The first is the value contained in the **djm.cookie** file. This file, which should have root-only


Copyright © 1993 Thinking Machines Corporation

read/write permissions, contains a single 8- to 10-digit decimal number. This value is read by the daemons and combined with an internal (preset) second key to form the actual authentication key.

Having the internal key allows for the possibility of the **djm.cookie** file being compromised. If that happened, combining a bogus **djm.cookie** value with the internal key would produce a bogus authentication key, and the message would be rejected.

Both these keys (internal and external) must be changed for your site. Section 2.2.1 provides instructions for changing them.

### 2.1.7 Privileges

DJM recognizes three levels of access: *users*, *operators*, and *managers*. Operators and managers are defined in the **operators** and **managers** fields, respectively, of the global configuration file. DJM operators can use the **jop** commands explained in this chapter to control DJM partitions, queues, and jobs. DJM managers can do everything operators can do; they can also define managers and operators, and control the DJM daemons, using the **job mgr** (or **jop mgr**) command with one of its options, as shown below.

| | |
|---|---|
| **add_mgr** | Adds user to manager list. |
| **add_op** | Adds user to operator list. |
| **del_mgr** | Deletes user from manager list. |
| **del_op** | Deletes user from operator list. |
| **dump_trace** | Tells all daemons to dump their trace buffers. Equivalent to sending each daemon a SIGQUIT. The daemons do not exit. |
| **exit** | Tells all daemons to exit without dumping trace buffers. |
| **help** | Shows this output. |
| **panic** | Tells all daemons to panic (dump trace buffers and exit). |


Version 2.0, November 1993
Copyright © 1993 Thinking Machines Corporation

## 2.2 Running DJM

### 2.2.1 Initial Startup

A number of tasks are required to bring up DJM for the first time. These are enumerated below.

1. Create the DJM spool directories:
   **/usr/spool/djm**
   **/usr/spool/djm/old**

2. Edit **/etc/cm/configuration/djm.cookie**

   This file contains the external half of the daemon authentication key. The file should contain a 10-digit decimal number. For security, the file should be read/writable by root only.

   In the following example, root sets and verifies access permissions on this file:

   ```
   % chmod 600 /etc/cm/configuration djm.cookie
   % ls -l /etc/cm/configuration djm.cookie
   -rw------- 1 root 20 Sept 1993 /etc/cm/configuration/djm.cookie
   ```

   Alternatively, you can set an environment variable **DJM_COOKIE** to hold the external key. However, all users will need this variable set so that messages could be sent from the **job** command to the **djm_master** daemon.

3. Use the **assign_scrambler** script to set the other half of the authentication key in the daemons and user binaries. Supply an 8-digit hexadecimal number as an argument to the script. For example:

   ```
   % assign_scrambler fedcba98
   Changing scramble key in djm_master
   Changing scramble key in djm_fingerd
   Changing scramble key in djm_execd
   Changing scramble key in djm_shepherd
   Changing scramble key in job
   %
   ```

4. Edit the **newdjmlog** script to meet the needs of your site. Make a **cron** entry to have the script run on a regular basis. This keeps the log files from becoming so large that they fill the file system, and it facilitates removal of out-of-date job state files from the **old** directory.


Copyright © 1993 Thinking Machines Corporation

5. Install the **djm_startup** script on the systems that will serve as the DJM master and DJM server systems.

6. After completing the five steps listed above, you can start DJM by running the **djm_startup** script. This creates the initial queue configuration files, as well as the global and partition configuration files in **/usr/spool/djm**.

   You must supply the name of the CM (as specified in the **djm.conf** file) as an argument to the script. A sample of the output is presented below. In this example, a single system serves as both master and server system for the CM Sol:

   ```
   % djm_startup sol
   Starting /usr/etc/djm_master sol
   Starting /usr/etc/djm_fingerd
   Starting /usr/etc/djm_execd
   ```

   You should run this script on each master and server after the timesharing daemons on all partitions are running. This allows the **djm_fingerd** daemon to contact the timesharing daemon and obtain information regarding the processes that are running and the resources they are consuming.

7. When DJM is first installed, it has only two queues. Moreover, the **queue_default** configuration file implements a simple FIFO queue policy. You may want to change the queueing policy to reflect the needs of your site. In addition, you will probably want to define other queues. For instance, you probably want a **queue_dedicated**, so that users can run dedicated jobs on the CM. You will also want to ensure that the queues you create reflect the job management policy that best suits your site. Section 2.2.2 explains how to configure queues, and provides notes on special issues concerning default and foreign queues. Appendix B describes the queue configuration (**queue_** *queuename*) files in detail.

8. You will also probably want to edit the global configuration file (**/usr/spool/djm/global**) to restrict foreign access and otherwise complete the implementation of your job management policy. Section 2.2.2 discusses some issues connected with this file, notably those regarding foreign jobs (Section 2.2.2.5). Appendix A provides a complete description of the file.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

9. At this point DJM should be operational with the scheduler frozen (so that it will not run jobs). You must now configure the DJM system via the **jop reconfig** command to define DJM partitions and their capabilities. Section 2.2.2 explains how to configure DJM partitions.

   As discussed in Section 2.2.2, finding optimal values for the global and queue configuration files usually requires a gradual fine-tuning process.

## 2.2.2 Normal Operations

### 2.2.2.1 Configuring Partitions

When you bring up a CM-5 that has DJM on it, DJM starts up with the scheduler in a frozen state. This ensures that no jobs are scheduled before all parts of the system are ready. To allow jobs to run, you must first execute a **jop reconfig** command with the appropriate arguments, thus creating one or more DJM partitions, and then unfreeze the scheduler.

The following example shows the creation of a CMOST partition followed immediately by starting DJM and creating a DJM partition to match the CMOST partition.

1. First create and start the CMOST partition:

```
% cmpartition create -pn_range 0-63
% cmpartition list -l

CM System "SOL"
    64 Processors

Partition Managers
    0      mars.think.com

Available PN Ranges:
    All PNs in use

IOP Addresses:
    No IOPs in this machine

Name    Partition Manager   Size State     Nodes Description
mars    mars.think.com      64   ALLOCATED 0-63
```


Copyright © 1993 Thinking Machines Corporation

```
% cmreset
% cmpartition start -cmd ts-daemon
```

2. When timesharing is running (you can examine **`/var/adm/ts-daemon.log`** to determine this), start DJM on the server:

```
% djm_startup sol
Starting /usr/etc/djm_fingerd
Starting /usr/etc/djm_execd
```

   In this case, the system on which **`djm_startup`** is being run is not the **`djm_master`** system; hence, only the **`djm_fingerd`** and **`djm_execd`** daemons are started.

3. Now, execute the DJM reconfiguration command.

```
% jop reconfig -mode normal -range 0-63 -server mars
Future partition #1 will be reconfigured at Aug 12 08:31
```

   The message tells you that DJM has created a partitition for use in the future. This is standard DJM practice. Unfreezing the scheduler makes the future partition become the current partition.

4. Unfreeze the scheduler.

```
% jop unfreeze

Scheduler unfrozen.
```

5. At this point DJM is ready to schedule jobs submitted to it. For example:

```
% jsub -- router_test 1000
Number of processors (64)?
Estimated cpu time (5min)?
Estimated memory (4M)?
Job submitted successfully.  Job id is 6.

% jstat
USER     JID  MACH   SERV    PROCS TIME MEM    STATUS   COMMAND
newton   6    sol    mars    64    0:11 1.27M  running  router_test
```

**Checking Configuration Status.** The **`jstat partition`** command is used to examine the state of all partitions, both those that are running and those that are planned for the future.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

```
% jstat partition
Running partitions:
BPROC  NPROC   SERVER INT  FREE_MEM  TOT_MEM  TS LOAD
     0    64   mars    0     10M     10M       y    0

Current partitions:
BPROC  NPROC  SERVER  MODE  TS   TIME     MEM     MEM*CPU
     0    64   mars  normal  y   unlim    unlim   unlim
```

**Cancelling Planned Reconfigurations.** If for some reason you need to cancel a reconfiguration to a future partition, use the **jop wipe_reconfig** command before unfreezing the scheduler. This command takes the future partition ID number as an argument.

```
% jop wipe_reconfig 1
Future partition #1 deleted.
```

**Automating Reconfigurations**. As the **jop** command is simply a UNIX command, it is possible to write shell scripts to automate the reconfiguration of the CM. Such a shell script can then be installed in the system's **crontab** to make reconfiguration occur automatically. You can also use the **jop** command's **-at** and **-delay** options to cause reconfigurations to occur at specified times.

#### 2.2.2.2 Configuring Queues

If you look in **/usr/spool/djm** for a newly installed DJM system, you'll see two queue configuration files, one for the default and one for the foreign queue (named **queue_default** and **queue_foreign**, respectively). Once DJM is configured and running, you may want to add new queues.

As an example, we'll create a dedicated queue to allow for exclusive access to the CM during specified times.

1. First, halt the scheduler.

```
% jop freeze
Scheduler frozen.
```


Copyright © 1993 Thinking Machines Corporation

2. Then create the state file for the dedicated queue. It is usually best to simply copy an existing configuration file to a new name and edit it.

```
% cp queue_default queue_dedicated
% vi queue_dedicated
```

   At a minimum, you need to change the name field to "dedicated" and the dedicated field to "yes". Other fields can be changed to implement whatever queuing strategy your site requires. (See Appendix B for details.)

3. Finally, reload the queue state file information into the daemons and unfreeze the scheduler.

```
% jop reload
All state reloaded from disk.

% jop unfreeze
Scheduler unfrozen.
```

4. DJM is now ready to accept dedicated jobs for later execution.

```
% jsub -dedicated -- router_test 100000
Number of processors (64)?
Estimated cpu time (5hr)?
Job submitted successfully to the "dedicated" queue.
Job id is 7.

% jstat
USER JID MACH SERV PROCS TIME  MEM  STATUS  COMMAND
mars 7  poule       4  300:00D 128M Que ded router_test
```

The status field of the **jstat** output shows that the job has been queued and is awaiting dedicated access to the CM. This dedicated access time must be configured explicitly by the system administrator, as explained in the next section.

**NOTE:** Remember, when configuring queues, that you want to give the most restrictive queue the lowest **search_order**, and that you want the default queue to be your least restrictive queue and have the highest value in its **search_order** field. (The values you set for {**min**|**max**}**_memory** and {**min**|**max**}**_cputime** are probably the key values when deciding how restrictive a queue is.) If you set your queues up in this fashion, DJM will be able to place each job that does not specify a queue into a queue that


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

meets, but does not exceed its requirements. This will help DJM optimize system performance.

Note also issues raised in Section 2.2.2.4 (“Allowing Dedicated Access”); these also apply to queue configuration.

#### 2.2.2.3 Stopping DJM

To stop DJM, execute the following sequence of commands:

```
% jop freeze
```

```
% jop reconfig -mode halt -kill -range m-n -lock
```

This freezes the scheduler to prevent further jobs from being scheduled, and then puts the partition in halted mode to prevent jobs from being queued. The **-kill** option kills any remaining jobs in the partition and the **-lock** option assures that the partition will not be restarted if the system is rebooted.

```
% jop mgr exit
```

This forces all the DJM daemons to die. Any running **djm_shepherd** processes continue to babysit for the user processes until they are done.

#### 2.2.2.4 Allowing Dedicated Access to the System

To reconfigure the system to dedicated mode, you must freeze the scheduler, issue the appropriate reconfiguration command, and then unfreeze the scheduler.

For example:

---

```
% jop freeze
Scheduler frozen.

% jop reconfig -mode dedicated -delay 1hr -kill -range 0-64
  -server mars
Future partition #1 will be reconfigured at Aug 12 10:26.
```


Copyright © 1993 Thinking Machines Corporation

```
% jop unfreeze
Scheduler unfrozen.

% jstat partition
Running partitions:
BPROC  NPROC  SERVER INT FREE_MEM TOT_MEM  TS LOAD
     0   64   mars    0        4G      4G  y    0

Current partitions:
BPROC  NPROC SERVER MODE  TS   TIME      MEM     MEM*CPU
     0    64 mars   normal y  unlim    unlim       unlim

Future partitions:
 # BPROC NPROC SERVER MODE  TS   TIME     MEM   MEM*CPU   RECONFIG_TIME
 1     0    64 mars   dedica y  unlim   unlim     unlim   Aug 12 10:26 kill

% jstat queue
QUEUE       MACH SRCH DED FOR  ENA STA  QUE HLD RUN TOT
dedicated   sol   50  yes no   yes yes    1   0   0   1
foreign     sol   50  no  yes  yes yes    0   0   0   0
default     sol   50  no  no   yes yes    0   0   0   0
```

In this case, the reconfiguration request specifies that DJM be configured for dedicated access in one hour and that any jobs that are running in the partition be killed when dedicated mode begins. The users of these jobs will receive a warning to this effect 30, 5, and 1 minutes prior to the impending reconfiguration so that they can take appropriate action.

**Notes on Dedicated Access**. Establishing dedicated access to the CM requires consideration of a number of issues:

- **`-kill`** vs. **`-nokill`**

  When reconfiguring DJM, the system administrator declares whether jobs that are running at the time of reconfiguration should be allowed to run to completion or not. This is done by specifying the **`-kill`** or **`-nokill`** option to the **`reconfig`** subcommand. The default is **`-nokill`**, which allows running jobs to complete.

  Dedicated access forces an exception to **`-nokill`**. Nondedicated jobs that are submitted prior to reconfiguration are never allowed to run if their requested CPU time would put them into the dedicated access time.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

If the reconfiguration specified **-kill**, then running jobs are sent a warning at T minus 30, 5, and 1 minutes to warn the user of the impending kill. It is the user's responsibility to take action on the warning.

- Foreign jobs

  Dedicated access prohibits foreign jobs from running. Any foreign jobs that are running at the start of dedicated mode are killed; any foreign job found running during dedicated mode is killed as soon as DJM notices it (average of 1 minute, maximum of 2 minutes).

- How far in advance must dedicated jobs be queued?

  By default, there must be at least one dedicated job queued by a certain time prior to a scheduled reconfiguration to dedicated mode for the reconfiguration to take place. If no such dedicated job is queued, the reconfiguration is aborted.

  The default time interval is four hours. You can change the interval by editing the **dedicated_submit_by_limit** field in the global configuration file. As described above, freeze the scheduler, edit the value, execute the **reload** command, and unfreeze the scheduler.

- Volatility

  By default, dedicated mode is volatile. When there are no more dedicated jobs to run, the system automatically reverts to normal mode. This may be what you want, or it may not. If it is not, use the **-lock** option to the **reconfig** command.

  The **-lock** option makes dedicated mode nonvolatile in two ways:

  - It informs DJM that it must stay in dedicated mode until an explicit reconfiguration command is executed to switch the system to normal mode. Such explicit reconfigs can run from **cron** to provide a guaranteed dedicated time slot on a regular basis.

  - It forces reconfiguration to dedicated mode to happen even if no dedicated jobs have been queued.

- Killing dedicated jobs

  If a dedicated job is running at the end of dedicated mode, it is not automatically killed unless the reconfiguration to normal mode specifies **-kill**. It, like any other DJM job, has a score. It will continue to run, and may get to run to completion, even if it overruns the dedi-


Copyright © 1993 Thinking Machines Corporation

cated time window. However, its score will be adjusted by the **running_dedicated_late_factor**, making it vulnerable. If a queued job with a higher score appears, the dedicated job will be killed and the queued job will start to run.

#### 2.2.2.5 Limiting Foreign Access to the System

DJM cannot prevent users from invoking jobs from outside of DJM. It can, however, restrict the resources that such jobs can consume, in order to prevent them from interfering overmuch with its ability to schedule jobs submitted within DJM, via **jsub** and **jrun**.

DJM notices all jobs submitted from outside DJM. It considers them to be running in the foreign queue. If the system is running in dedicated mode, it kills them.

The global configuration file contains a number of parameters that can be modified to restrict the consumption of resources by foreign jobs. For example, the file contins a field for **foreign_requested_cputime** and one for **foreign_requested_mem_per_proc**. A foreign job has these values assigned to it as if the user had run the job under DJM with the **-cpu** and **-mem** switches. This allows DJM to calculate a score for the foreign job and, when the assumed values are exceeded, to apply the appropriate penalty to the score. This makes the foreign job subject to being killed by a DJM job with a higher score.

In addition, the fields **foreign_hard_cputime** and **foreign_hard_mem_per_proc** specify the absolute limits on a foreign job. Any foreign job that exceeds either of these limits is killed immediately.

Suggested values are:

| | | |
|---|---|---|
| **foreign_requested_cputime** | 86400 | (1 day) |
| **foreign_requested_mem_per_proc** | 0 | (unlimited) |
| **foreign_requested_total_mem** | *xxxx* | (total max memory) |
| **foreign_hard_cputime** | 900 | (15 minutes) |
| **foreign_hard_mem_per_proc** | 0 | (unlimited) |
| **foreign_hard_total_mem** | *xxxx* | (total max memory) |

Total max memory is site-dependent.

The key field is **foreign_hard_cputime**. This allows users to bypass DJM to run short benchmarks or to do short debuggings without having to use



Copyright © 1993 Thinking Machines Corporation

DJM. In no case should the value for this field be less that 120 (2 minutes). The maximum value is 0, which represents unlimited CPU time.

The 2-minute minimum is necessary because user processes may be swapped to disk. DJM assumes such swapped processes to be foreign. Hence, if the limit is too low, the swapped process would be killed, even if it were a valid DJM job. A minimum of 2 minutes ensures that this will not occur.

## 2.3 When Things Go Wrong

DJM has a number of features that assist in analyzing problems.

### 2.3.1 Log File

The main log file contains information regarding all jobs that have been run. This file can contain useful information if problems occur.

### 2.3.2 Trace Files

DJM daemons log a record of their actions internally. In the event of abnormal termination, the log is dumped to disk. The **`djm.conf`** file (see Appendix F) contains a line to specify the directory in which the trace files will be created. Each daemon creates a trace file called **`djm_`***daemon-name* **`.trace`**. Existing trace files are overwritten by new ones.

To force a DJM daemon to dump a trace log, simply send it a SIGQUIT signal (**`kill -3`** *pid of daemon*). This will dump the trace log without causing an interruption in the operation of the daemon.

A SIGTERM (**`kill -15`**) or SIGABRT (**`kill -6`**) causes the daemon to dump the trace log and quit. In the latter case, a core file is dumped as well.


Copyright © 1993 Thinking Machines Corporation

An example of a trace file follows :

```
12:06:25 MSG select(timeout=60)
12:07:25 MSG select returned 0
12:07:25 SCHED poll_status
12:07:25 PROC query_fingerd host=mars.think.com
12:07:25 MSG send_tcp_msg(sock=5,host=mars.think.com)
12:07:25 MSG data=f7fff70c, dlen=4
12:07:25 MSG recv_tcp_msg(sock=5,host=mars.think.com)
12:07:25 MSG data=f7fff500, dlen=516
12:07:25 MSG recv msg (prog=39001802,len=260)
12:07:25 MSG fingerd@mars.think.com: unable to send GET_CONFIG to ts daemon
12:07:25 SCHED update_resource_use
12:07:25 SCHED calc_job_scores
12:07:25 SCHED check_hard_limits
12:07:25 SCHED sched
12:07:25 MSG select(timeout=60)
12:07:50 SIG fatal_sig(15)
12:07:50 SCHED panic: terminate signal received
12:07:50 LOG errlog: djm_master terminated due to the following reason:
12:07:50 LOG log: djm_master terminated due to the following reason:
12:07:50 LOG errlog: terminate signal received
12:07:50 LOG log: terminate signal received
```

### 2.3.3 Checkpoints

The **djm_master** daemon checkpoints itself every 5 minutes. It also checkpoints itself upon receiving a **jop freeze** command. A checkpoint updates the state files in **/usr/spool/djm**.

In the event of a crash, therefore, the configuration files may be edited by hand. DJM can then be restarted using the **jop reload** command.

### 2.3.4 If jstat Hangs

Finally, it is possible that the **jstat** command may appear to hang. Either network problems or a slave daemon (**djm_fingerd** or **djm_execd**) waiting on a hung timesharing daemon can cause this.



Copyright © 1993 Thinking Machines Corporation

In this case, **djm_master** will wait up to 2 minutes for a response. If no response is forthcoming, the server is delared down. Thereafter, the **djm_master** daemon will wait only 5 seconds per poll before returning. When the server is back on-line, the 2-minute limit will be in force once again.


Copyright © 1993 Thinking Machines Corporation

## Chapter 3

# DJM Reference

This chapter contains the man pages for the DJM commands:

| | |
|---|---|
| **jconnect** | Connects terminal to a job. |
| **jdel** | Deletes or kills a job. |
| **jmsg** | Sends a message to a job. |
| **job** | User interface to the Distributed Job Manager. |
| **jop** | Operator interface to the Distributed Job Manager. |
| **jrun** | Runs a job interactively. |
| **js** | Shows job information for all machines. |
| **jstat** | Shows job information. |
| **jsub** | Submits a job for background execution. |
| **jwait** | Waits for a job to start. |

These man pages originated at the Minnesota Supercomputing Center and have been edited at Thinking Machines Corporation.


*Copyright © 1993 Thinking Machines Corporation*

# jconnect

Connects a terminal to a job.

---

### SYNOPSIS

**jconnect** [ *jobid* ] [ *options* ]

#### Options:

[[**-jid**] *jobid*] [**-jobname** *name*] [**-escape** *char*] [**-noescape**]
[**-machine** *mach*] [**-nobell**] [**-nostop**] [**-eightbit**]

---

### DESCRIPTION

**jconnect** establishes a connection between your terminal and a running job. If the job is not yet running, **jconnect** waits for the job to start.

After connection is established, your keystrokes are forwarded to the standard input of the job. After you disconnect and all your keystrokes have been read by the job, the standard input will be redirected back to its original source, usually a disk file or **/dev/null**, as indicated by the job's **-stdin** option.

To abort waiting for the job, type Ctrl-b. To cancel the job, type Ctrl-c.

All option names may be abbreviated to uniqueness.

### OPTIONS

[**-jid**] *jobid* — Connects to the job with the given *jobid* as returned by **jstat**(1).

**-jobname** *name* — Connects to the job with the name *name*.

**-escape** *char* — Sets the escape character to *char*. The default is a tilde (**~**). The escape character is used to prefix commands that affect the terminal session in some way. For example, **~**b disconnects the job from your terminal and **~**c toggles capture mode. See **jrun**(1) for a complete list of such commands. A control character can


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

be denoted with a caret (^A), given in octal notation ('\ 01') or typed directly. **-noescape** disables the escape character.

**-machine** *mach* Establishes a connection to machine *mach*. This is typically the name of a Connection Machine system.

**-nobell** Turns off ringing of bell when the connection is established.

**-nostop** Permanently disables Ctrl-s and Ctrl-q processing (flow control). By default these are enabled on the local terminal unless the job explicitly disables them via ioctl calls. This option is usually not needed, as programs such as emacs will disable flow control whenever necessary.

**-eightbit** Passes 8-bit characters through unchanged. By default the uppermost bit of each character is stripped off to create 7-bit logical characters.

## NOTES

You cannot connect to a job that does not have a pseudo terminal associated with it. This is indicated by the job's **-notty** option.

It is not possible to change the terminal type of a job (the **TERM** environment variable) once the job starts running.

## SEE ALSO

**job**(1), **jrun**(1), **jstat**(1), **jwait**(1)


*Copyright © 1993 Thinking Machines Corporation*

# jdel

Deletes or kills a job.

---

### SYNOPSIS

**jdel** [ *options* ]

#### Options:

[ [**-jid**] *jobid*] [**-jobname** *name*] [**-signo** *n*] [**-kill**] [**-machine** *mach*]

#### NQS Compatibility Options:

[**-h** *mach*] [**-k**] [**-***signo*]

---

### DESCRIPTION

**jdel** deletes or kills a job. If the job is queued, it is deleted. If it is running, then a signal is sent to it. The default signal is SIGTERM; the choice of signal can be changed with the **-signo** option.

**jdel** is upward-compatible with the command **qdel** in the Network Queuing System (NQS).

All option names may be abbreviated to uniqueness.

### OPTIONS

| | |
|---|---|
| [**-jid**] *jobid* | Deletes or kills the job with the given *jobid* as returned by **jstat**(1). |
| **-jobname** *name* | Deletes or kills the job with the name *name*. |
| **-signo** *n* | Sends signal *n* to the job. The default signal is 15 (SIGTERM). |
| **-***n* | Sends signal *n*. For NQS compatibility. |
| **-kill** | Sends signal 9 (SIGKILL) to the job. This signal cannot be caught or ignored. |


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**-machine** *mach* Selects machine *mach*. This is typically the name of a Connection Machine system.

## NOTES

Upon receipt of a signal, the job is given 5 minutes to clean itself up. After this period expires the job will be sent a SIGKILL.

## SEE ALSO

**job**(1), **jsub**(1)

---


*Copyright © 1993 Thinking Machines Corporation*

# jmsg

Sends a message to a job.

---

### SYNOPSIS

**jmsg** [ *options* ] [ *message...* ]

#### Options:

[ [**-jid**] *jobid*] [**-jobname** *name*] [**-file** *name*] [**-stderr**] [**-stdout**] [**-machine** *mach*]

#### NQS Compatibility Options:

[**-h** *mach*] [**-e**] [**-E**] [**-o**] [**-O**] [**-f** *name*]

---

### DESCRIPTION

**jmsg** sends a message to the output buffer of a job. If no message is given on the command line, then the message is read from the standard input. The output appears on the user's terminal if the job is connected. Otherwise, it goes to a disk file.

**jmsg** is upward-compatible with the command **qmsg** in the Network Queuing System (NQS).

All option names may be abbreviated to uniqueness.

### OPTIONS

| | |
|---|---|
| [**-jid**] *jobid* | Sends the message to the job with the given *jobid* as returned by **jstat**(1). |
| **-jobname** *name* | Sends the message to the job called *name.* |
| **-stderr** | Sends the message to the job's standard error file. This is the default. |
| **-e** | Same as above. For NQS compatibility. |


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

| | |
|---|---|
| **-stdout** | Sends the message to the job's standard output file. If the job has a combined stdout and stderr file, only one message is sent. |
| **-o** | Same as above. For NQS compatibility. |
| **-file** *name* | Reads the message from the file *name*. |
| **-machine** *mach* | Selects machine *mach*. This is typically the name of a Connection Machine system. |

## NOTES

**jmsg** always returns immediately, even if the output of the job is blocked by the user's terminal.

The Posix Batch Queuing standard (1003.15) reverses the job ID and message arguments, placing the message first. This implementation leaves the message last, in keeping with existing NQS practice.

## SEE ALSO

**job**(1)


*Copyright © 1993 Thinking Machines Corporation*

# job

User interface to the Distributed Job Manager.

---

### SYNOPSIS

**job** [ *command* ] [ *options* ]

#### Commands:

**connect delete exit help hold kill mgr msg oper ping query quit release rerun rls run show signal status submit wait**

#### Common options:

[ [**-jid**] *jobid*] [**-jobname** *name*] [**-machine** *mach*] [**-noprompt**] [**-tdisplay**] [**-tracemask** *n* ]

---

### DESCRIPTION

**job** is the primary user interface for the Distributed Job Manager. It allows you to issue commands to DJM to create and run jobs for execution.

There are three ways in which you can invoke **job**:

- When executed with no arguments, **job** prints a banner and a prompt and waits for you to type commands from the keyboard.
- A command can be invoked directly from the shell by placing the command and its associated options after the word **job** on the shell command line. Example: **job status**.
- A symbolic link consisting of the letter **j** followed by the command will invoke that command. Example: **jstat**.

All **job** commands and options may be abbreviated to uniqueness.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

## GETTING STARTED

To get started there are two basic commands that you need to know: **submit** and **status**. These allow you to run a job and to show information about that job.

You submit a job for execution by typing **submit myjob** where **myjob** is the name of the program or shell script you wish to have executed. An example is shown below.

```
% jsub myjob
Number of processors (32)? 256
Estimated CPU time (5min)? 60
Estimated memory (32M)? 400
Job submitted successfully.  Job id is 43.
```

You need to specify the resources that will be required by the job. These include the following:

- The number of processors required.
- The estimated amount of CPU time required.
- The estimated amount of memory required.

  It is in your best interests to make your resource estimates as accurate as possible with respect to the job's actual needs. If you understate your job's resource requirements it may be terminated if other jobs are waiting to run. On the other hand, you do not want to overestimate your resource requirements as it may cause your job's execution to be delayed.

The actual resource usage is written to the end of the job's output when it finishes. You can use this information to supply better resource estimates the next time you run your job.

Resource information can be provided in one of three ways:

- interactively, as shown in the example above
- on the job command line, for example:
  **jsub myjob -nproc 256 -cpu 60m -mem 32M**
- with embedded shell script options, as shown below:

```
#!/bin/sh                       |the name of the shell to be used
#JSUB -nproc 256 -cpu 60m -mem 32m     |options go here
#JSUB -stdout myfile                   |more options
a.out                           |your program goes here
```


*Copyright © 1993 Thinking Machines Corporation*

By default, the output from your job will be sent to a file *myjob.onnn* in the current directory, where *nnn* is the ID number assigned to the job by the system.

You can check up on the progress of your job with the command **status**. See **jstat**(1).

You can also run a job interactively from your terminal. See **jrun**(1).

## OPTIONS

The following options are common to all commands.

[**-jid**] *jobid* Searches for the job with the identifier *jobid* as shown by **jstat**(1). The leading **-jid** is optional. The identifier is assigned by the system when you submit the job.

**-jobname** *name* Searches for the job with the given *name*. **-jobname** is also used to set the job name in **jsub**(1) and **jrun**(1).

**-machine** *mach* Selects machine *mach*. This is typically the name of a Connection Machine system. By default the machine name is obtained from the environment variable **DJM_MACHINE**.

**-noprompt** Does not prompt for input. This can be used to suppress prompts when feeding commands into **job** from a shell script.

**-tdisplay** Turns on the trace display. This is used to print internal information for debugging purposes.

**-tracemask** *n* Sets the trace mask to *n*.

## COMMANDS

All commands may be abbreviated to uniqueness.

### connect

**connect** [[**-jid**] *jobid*] [**-jobname** *name*] [**-emacs**] [**-escape** *char*] [**-noescape**] [**-nobell**] [**-nostop**] [**-eightbit**]
Establishes a connection between your terminal and a running job. For details, see **jconnect**(1).


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

#### delete

**delete** [[**-jid**] *jobid*] [**-jobname** *name*] [**-signo** *n*] [**-***n*] [**-kill**]

Deletes or kills a job. If the job is queued, it is deleted. If it is running, a signal is sent to it. The default signal is SIGTERM; the choice of signal can be changed with the **-signo** option.

Upon receipt of a signal, the job is given 5 minutes to clean itself up. After this period expires the job will be sent a SIGKILL. For details, see **jdel**(1).

#### exit

Quits the job program.

#### help

Shows a list of all commands and a one-line description of each command.

#### hold

**hold** [[**-jid**] *jobid*] [**-jobname** *name*]

Holds a queued job. The job will not be executed until it is released. This command has no effect if the job is already running.

#### kill

**kill** [[**-jid**] *jobid*] [**-jobname** *name*] [**-signo** *n*] [**-***n*] [**-kill**]

Sends a signal to a running job. It is a synonym for **delete**.

#### mgr

**mgr** [*command*] [*options*]

Runs manager commands. For details see Chapter 2.


*Copyright © 1993 Thinking Machines Corporation*

#### msg

**msg** [[**-jid**] *jobid*] [**-jobname** *name*] [**-file** *fname*] [**-stderr**] [**-stdout**] [*message* ]

Sends a message to the output buffer of a job. If the job is connected (**jconnect**) then the output appears on the user's terminal, otherwise it is sent to a file. For more details, see **jmsg**(1).

#### oper

**oper** [*command*] [*options*]

Commands: **delete disable enable freeze goall help hold reconfig reload rerun resume start stop suspend sync unfreeze wipe_reconfig**

Runs operator commands. These commands are used by the operator to control job and queue behavior and to temporarily change system operation. For details see **jop**(8) and Chapter 2 of this manual.

#### ping

Sends a null message to the master daemon. This can be used to verify that DJM is still alive.

#### query

A synonym for **status**.

#### quit

Quits the job program.

#### release

**release** [[**-jid**] *jobid*] [**-jobname** *name*]

Releases a held job. The job is then scheduled for execution as normal. This command has no effect if the job is not held or is already running.

#### rls

A synonym for **release**.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

#### rerun

**rerun** [ [**-jid**] *jobid*] [**-jobname** *name*]

Reruns a job. If the job is dead, it is recovered and requeued as if the user had typed **jsub** or **jrun**. The job's shell script or program must still exist in its original location. If the job is running, it is killed before it is requeued.

#### run

**run** [ *file* ] [ *options* ]
**run** [ *options* ] −− *file* [ *arguments* ]

Options:

[**-jobname** *name*] [**-machine** *mach*] [**-cputime** *n* ] [**-memory** *n* ] [**-nprocs** *n* ] [**-capture**] [**-after** *time*] [**-chdir** *path*] [**-dedicated**] [**-eightbit**] [**-emacs**] [**-escape** *char*] [**-exact_mem**] [**-exact_proc**] [**-execute**] [**-export**] [**-hard_cputime** *n* ] [**-hard_memory** *n* ] [**-help**] [**-hold**] [**-interruptable**] [**-linelimit** *n*] [**-mail_begin**] [**-mail_end**] [**-mail_user** *user* ] [**-maxrerun** *n* ] [**-need** *rsrcs* ] [**-nobell**] [**-nocapture**] [**-nodedicated**] [**-noescape**] [**-noexact_proc**] [**-nointerruptable**] [**-nopreamble**] [**-nostop**] [**-noterminal**] [**-notimeshared**] [**-preamble**] [**-priority** *n*] [**-processors** *n* ] [**-procs** *n* ] [**-project** *name* ] [**-queue** *name* ] [**-range** *m-n* ] [**-safety** *n* ] [**-server** *name* ] [**-seq** *m-n* ] [**-shell** *name* ] [**-silent**] [**-sp_disk** *n* ] [**-sp_memory** *n* ] [**-stderr** *name* ] [**-stdin** *name* ] [**-stdout** *name* ] [**-terminal**] [**-time** *n* ] [**-timeshared**] [**-ucode** *n* ] [**-want** *rsrcs* ]

This command runs a job interactively from the terminal. For details see **jrun**(1) and **jsub**(1).

#### show

A synonym for **status**.

#### signal

A synonym for **delete**.


*Copyright © 1993 Thinking Machines Corporation*

#### status

**status** [*command*] [*NQS options*] [*jobid*] [*queuename*]

Commands:

**global jobs limits machine partitions queues servers scores up**

Displays the status of the Distributed Job Manager and its jobs. In addition to jobs, information on queues, partitions, servers, and other entities can also be shown. For details, see **jstat**(1).

#### submit

**submit** [ *file* ] [ *options* ]
**submit** [ *options* ] −− *file* [ *arguments* ]

Options:

[**-jobname** *name*] [**-machine** *mach*] [**-cputime** *n* ] [**-memory** *n* ] [**-nprocs** *n* ] [**-after** *time* ] [**-chdir** *path* ] [**-dedicated**] [**-exact_mem**] [**-exact_proc**] [**-execute**] [**-export**] [**-hard_cputime** *n* ] [**-hard_memory** *n* ] [**-help**] [**-hold**] [**-interruptable**] [**-linelimit** *n*] [**-mail_begin**] [**-mail_end**] [**-mail_user** *user* ] [**-maxrerun** *n* ] [**-need** *rsrcs* ] [**-nodedicated**] [**-noexact_proc**] [**-nointerruptable**] [**-nopreamble**] [**-noterminal**] [**-notimeshared**] [**-notty**] [**-preamble**] [**-priority** *n* ] [**-processors** *n* ] [**-procs** *n* ] [**-project** *name* ] [**-queue** *name* ] [**-range** *m-n* ] [**-safety** *n* ] [**-server** *name* ] [**-seq** *m-n* ] [**-shell** *name* ] [**-silent**] [**-sp_disk** *n* ] [**-sp_memory** *n* ] [**-stderr** *name* ] [**-stdin** *name* ] [**-stdout** *name* ] [**-terminal**] [**-time** *n* ] [**-timeshared**] [**-tty**] [**-ucode** *n* ] [**-want** *rsrcs* ]

This command submits a job to run in the background. The job can be connected to a terminal if the **-tty** option is set. For details see **jsub**(1).

#### wait

**wait** [*jobid*] [**-jobname** *name*] [**-begin**] [**-end**] [**-nobell**]

Waits for a job to begin or end execution. See **jwait**(1).


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

## SEE ALSO

**jconnect**(1), **jdel**(1), **jmsg**(1), **jop**(8), **jrun**(1), **js**(1), **jstat**(1), **jsub**(1), **jwait**(1)

---


*Copyright © 1993 Thinking Machines Corporation*

# jop

Operator interface to the Distributed Job Manager.

---

### SYNOPSIS

**jop** [ *command* ] [ *options* ]

#### Commands:

**delete disable enable freeze goall help hold reconfig reload rerun resume run start stop suspend sync unfreeze wipe_reconfig**

---

### DESCRIPTION

**jop** is the operator interface to the Distributed Job Manager. It allows you to control job and queue behavior and to change the partition configuration.

You must be authorized before you can use this command.

There are two ways in which you can invoke **jop**:

- When executed with no arguments, **jop** prints a banner and a prompt and waits for you to type commands from the keyboard.
- A **jop** command can be invoked directly from the shell by placing the command and its associated options after the word **jop** on the shell command line. Example: **jop unfreeze**.

All **jop** commands and options may be abbreviated to uniqueness.

### COMMANDS

#### freeze

Freezes the Distributed Job Manager. All scheduling activity will cease. Jobs that are running continue to run, but no new jobs are started. You must freeze DJM before hand editing the state files in **/usr/spool/djm**. This command causes all state to be implicitly sync'ed to disk. Once editing is complete, you can execute **reload** to load the state back in from disk and then **unfreeze** to resume the scheduler.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

#### unfreeze

Unfreezes the Distributed Job Manager. Normal scheduling activity will resume. When the system is first booted **djm_master** starts off in a frozen state. This ensures that no jobs are scheduled before the all the machinery is ready (time-sharing daemons, IOBAs, etc.). This command is executed manually as part of the system startup procedure. It is also used to resume scheduling after the system was frozen in order to hand-edit the state files.

A synonym for **unfreeze** is **goall**.

#### delete

**delete** [[**-jid**] *jobid*][**-signo** *n*] [**-***n*] [**-kill**]

Deletes or kills a job. If the job is queued, it is deleted. If it is running, a signal is sent to the job. Upon receipt of a signal the job is given 5 minutes to clean itself up. After this period expires the job is sent a SIGKILL. For details, see **jdel**(1).

#### disable

**disable** *queue*

Disables the queue. The queue will no longer accept new job submissions. Jobs may still be executed from the queue.

#### enable

**enable** *queue*

Enables the queue. The queue will once again accept new job submissions.

#### help

Shows a list of all commands and a one-line description of each command.

#### hold

**hold** [**-jid** *jobid*]

Holds a queued job. The job does not execute until it is released. This command has no effect if the job is already running.


*Copyright © 1993 Thinking Machines Corporation*

### treconfig

**reconfig** [**-at** *time* ] [**-delay** *n* ] [**-hard_cputime** *n* ] [**-hard_memory** *n* ] [**-hard_mem_per_proc** *n* ] [**-hard_cpu_times_mem_per_proc** *m*c* ] [**-hard_memcpu** *m*c* ] [**-kill**] [**-lock**] [**-mode** *name* ] [**-need** *rsrcs*] [**-nokill**] [**-notimeshared**] [**-offer** *rsrcs*] [**-queue** *queues*] [**-range** *m-n* ] [**-runlimit** *n* ] [**-seq** *m-n* ] [**-server** *name* ] [**-soft_cputime** *n* ] [**-soft_memory** *n* ] [**-soft_mem_per_proc** *n* ] [**-soft_cpu_times_mem_per_proc** *m*c* ] [**-soft_memcpu** *m*c* ] [**-silent**] [**-timeshared**] [**-unlock**] [**-wait** *n*]

Sets administrative controls on contiguous segments of processors (partitions) on the CM. A partition must be configured before DJM will run jobs. Configuring a partition may cause side effects, such as starting or stopping the timesharing daemon on the CM-2.

The system must be frozen (see **freeze**) before this command will be accepted.

This command creates a "future partition." A future partition is one that has not yet been realized on the machine. When DJM is unfrozen, it checks for the existance of future partitions. The future partitions whose reconfiguration time has arrived are then realized as current partitions.

**-at** *time* Peforms the reconfiguration at the indicated time or date. Example: **-at** "Thu 10pm". See the **-after** option of **jsub** for a list of of valid time and date formats. A warning is issued to all users who are to be affected by the change at T minus 30, T minus 5, and T minus 1 minutes. New jobs are not scheduled if the estimated completion time is beyond the time when reconfiguration is to occur. The default time is now.

**-delay** *n* Delays the reconfiguration for the indicated number of minutes or hours. Example: **-delay 2hr**. This allows you to give a relative instead of an absolute time. You cannot specify both **-at** and **-delay** in the same reconfiguration.

**-hard_cputime** *n*
Sets the hard CPU time limit to *n*. Example: **-hard_cputime 6hr**. Queued jobs requesting more than *n* units of CPU time are not allowed to run in the partition. Running jobs that exceed this amount are killed.

**-hard_memory** *n* Sets the hard memory to *n*. Example: **-hard_memory 300mb**. Queued jobs requesting more than *n* units of memory are not


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

allowed to run in the partition. Running jobs that exceed this amount are killed. Note that this is total memory.

**-hard_mem_per_proc** *n*
: Sets the hard memory limit to *n* units per processor. Queued jobs requesting more than this amount of memory are not allowed to run in the partition. Running jobs that exceed this amount are killed. You cannot specify both **-hard_memory** and **-hard_mem_per_proc** in the same reconfiguration.

**-hard_cpu_times_mem_per_proc** *'m*c'*
: Sets the limit on the product of CPU times memory to *m*c*. Example: **-hard_memcpu '16mb*2hr'**. Queued jobs requesting more than this product are not allowed to run in the partition. Running jobs that exceed this amount are killed. This option allows short jobs to use a lot of memory, or long-running jobs to use a little memory, while preventing long-running jobs from using a lot of memory. Quotes are needed to prevent the shell from expanding the asterisk. Note that this is memory per processor, not total memory.

**-hard_memcpu** *n* Synonym for **-hard_cpu_times_mem_per_proc**.

**-kill** Kills all jobs in the partition when the reconfiguration takes place. The default is **-nokill**. Users receive a warning at T minus 30 minutes, 5 minutes, and 1 minute.

**-lock** Prevents subsequent reconfigurations from taking effect. This allows you to override a predetermined schedule in order to take special actions, such as repairing the machine. Further reconfigurations are ignored until one is found with the the **-unlock** attribute. This option makes a dedicated partition non-volatile.

**-mode [***name***]** Sets the partition mode. The valid modes are **normal**, **dedicated**, and **halted**. **normal** allows regular non-dedicated jobs to run. **dedicated** allows only dedicated jobs to run. **halted** prevents all jobs from running. Note that dedicated jobs are never started in a non-dedicated partition and vice versa. Dedicated mode has special scheduling rules; for more information see Section 2.2.2.4.


*Copyright © 1993 Thinking Machines Corporation*

**-need** *rsrcs* Indicates a list of resources that jobs must specify in order to run on the partition. *rsrcs* is a comma-separated list of keywords. Example: **-need devel,new_os**. A job will not be run on the partition unless it specifies all the resources indicated. In the example above, the user must specify either **-need devel**, **new_os**, or **-want devel,new_os** in order to run on the partition.

**-offer** *rsrs* Indicates a list of offered resources. *rsrcs* is a comma-separated list of keywords. Example: **-offer 'hippi,sda'**. A job will not be run on the partition unless the partition offers at least the resources indicated by the user. You can think of **-need** as the "necessary" and **-offer** as the "sufficient" resources to allow a job to run.

**-nokill** Does not kill jobs when reconfiguration takes place. This is the default. Note that some changes, in particular the stopping of the timesharing daemon, will cause jobs to be killed anyway, even if **-nokill** is set. In such cases you should use **-kill** so that users receive the 30-minute warning.

**-notimeshared** (CM-2 only.) Indicates a non-timeshared partition. Jobs run on this partition will be directly attached to the CM-2 sequencer. The non-timeshared partition is not assigned to a particular server. Instead it "floats" between servers, being assigned to whichever front end has an available interface. The partition is automatically split or coalesced with adjoining non-timeshared partitions as necessary. You can assign a preferred front end with the **-server** option.

This option is not supported on the CM-5.

**-queue** *queues* Indicates a list of queues from which jobs will be accepted. *queues* is a comma-separated list of queue names. Example: **-queue 'small,large,fast'**. A job will not run on the partition unless it is located in one of the indicated queues.

**-range** $m-n$ Sets the range of affected processors to $m-n$. Example: **-range 0-127**. On the CM-2, you can alternatively use **-seq**. There is no default. All reconfigs must include a **-range** or **-seq** option.

**-runlimit** $n$ Sets the total job run limit to $n$. This controls the maximum number of jobs that may concurrently run on the partition. 0 indicates unlimited (default).


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**-seq** *m−n* (CM-2 only.) On the CM-2, sets the range of affected sequencers to *m−n*. Example: **2-3**. All reconfigs must include a **-range** or **-seq** option.

**-server** *name* Indicates the server that will control the partition. This option is currently ignored on the CM-5. It is required on the CM-2 if **-timesharing** is set. When **-notimesharing** is set, this option indicates a preferred server for a floating partition (CM-2 only). The server must be listed in the **djm.conf** file.

**-silent** Performs the reconfiguration silently unless an error occurs. This option is useful in cron scripts.

**-soft_cputime** *n*
Sets the soft CPU time limit to *n*. Queued jobs requesting more than *n* units of CPU time are not allowed to run unless they are **-interruptable**. Running jobs that exceed this amount suffer a reduction in their job score (see **running_cputime_exceeded_factor**), making them vulnerable to being killed if other jobs are waiting to run.

**-soft_memory** *n* Sets the soft memory limit to *n*. Queued jobs requesting more than *n* units of memory are not allowed to run unless they are **-interruptable**. Running jobs that exceed this amount suffer a reduction in their job score (see **running_memory_exceeded_factor**), making them vulnerable to being killed if other jobs are waiting to run. Note that this is total memory.

**-soft_mem_per_proc** *n*
Sets the soft memory limit to *n* units per processor. Queued jobs requesting more than this amount of memory are not allowed to run unless they are **-interruptable**. Running jobs that exceed this amount suffer a reduction in their job score (see **running_memory_exceeded_factor**), making them vulnerable to being killed if other jobs are waiting to run. You cannot specify both **-soft_memory** and **-soft_mem_per_proc** in the same reconfiguration.


*Copyright © 1993 Thinking Machines Corporation*

**-soft_cpu_times_mem_per_proc** *'m*c'*
: Sets the soft limit on the product of CPU times memory to *m*c*. Example: **-soft_memory '16mb*2hr'**. Queued jobs requesting more than this amount are not allowed to run unless they are **-interruptable**. Running jobs that exceed this amount suffer a reduction in their job score (see **running_mem_cpu_product_exceeded_factor**), making them vulnerable to being killed if other jobs are waiting to run. Quotes are needed to prevent the shell from expanding the asterisk.

**-soft_memcpu** *n* Synonym for **-soft_cpu_times_mem_per_proc**.

**-timeshared** Indicates a timeshared partition. This is the default. On the CM-2, the **djm_exec** daemon starts the timesharing daemon by invoking the shell script **start-ts.sh** with the appropriate interface and processor options. This option is ignored on the CM-5. The current version of DJM does not start or stop timesharing on the CM-5.

**-unlock** Reverses the effect of the **-lock** attribute. This reenables the processing of reconfigurations, allowing resumption of a predetermined partitioning schedule.

**-wait** *n* This is a synonym for **-delay**.

### wipe_reconfig

[ *reconfig_id* ]

Erases (wipes out) a pending reconfiguration. The *reconfig_id* is the unique identifier assigned to the reconfiguration by the system. If no ID is specified, all pending reconfigurations are erased except those with the **-lock** or **-unlock** attribute.

Here are some examples of partition reconfiguration:

Reserve all of a CM-5 for dedicated time at 10pm:

```
jop freeze
jop reconfig -mode ded -range 0-127 -kill -at 10pm
jop unfreeze
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

Restore normal use on the CM-5 in 4 hours:

```
jop freeze
jop reconfig -range 0-63   -mode normal -nokill -wait 240min
jop reconfig -range 64-127 -mode normal -kill -wait 240min
jop unfreeze
```

Reserve half of a CM-200 for dedicated time on Thursday at 2pm:

```
jop freeze
jop reconfig -seq 2-3 -mode dedicate -kill -notimeshare -at 'Thu 2pm'
jop unfreeze
```

Total shutdown of the CM-200 in 4 hours:

```
jop freeze
jop reconfig -mode halt -kill -wait 240min -seq 0-3 -lock
jop unfreeze
```

Dedicated mode is volatile. If no dedicated jobs are waiting to run, then the reconfig to dedicated mode is spontaneously converted into a reconfig to normal mode. Note that a grace period can be applied to delay the spontaneous conversion. The grace period is set by the parameter **`dedicated_grace_period`** in the global file.

Dedicated jobs must be queued at least 4 hours before the start of the dedicated time period. When the last dedicated job finishes, the partition is automatically reconfigured back to normal mode (after the grace period). This is explained more fully in Chapter 2.

None of these rules apply if the dedicated partition is locked (**`-lock`**).

#### reload

Causes all state information to be reloaded from disk. It is used after hand-editing the state files in **`/usr/spool/djm`** in order to load the modified information back into DJM. The system must be frozen (which syncs the data to disk) before this command will be accepted.


*Copyright © 1993 Thinking Machines Corporation*

#### rerun

**rerun** [**-jid**] *jobid*

Reruns a job. If the job is dead, it is recovered and requeued as if the user had typed **jsub** or **jrun**. The job's shell script or program must still exist in its original location. If the job is running, it is killed before it is requeued.

#### resume

**resume** [**-jid**] *jobid*

Resumes a suspended job. This sends a SIGCONT signal to all the processes in the job. See **suspend**.

#### run

**run** [**-jid**] *jobid*

Causes the scheduler to attempt to run the job as soon as possible, giving it preemptive priority over all other queued jobs. This command is much safer than simply boosting the job's score because the latter causes the scheduler to kill other running jobs to make room.

#### stop

**stop** *queue*

Stops the named queue. Jobs are no longer run from that queue. Currently running jobs are allowed to finish.

#### start

**start** *queue*

Starts the named queue. Jobs are once again run from that queue.

#### suspend

**suspend** [**-jid**] *jobid*

Suspends a running job. This sends a SIGSTOP signal to all the processes in the job. See **resume**.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**sync**

Writes all the state information to the files in **/usr/spool/djm**. This occurs automatically once every five minutes. You should run this command as a precaution before doing anything dangerous or risky. A **sync** is implicitly performed when DJM is frozen.

## SEE ALSO

**job**(1), **jstat**(1), **jdel**(1)


*Copyright © 1993 Thinking Machines Corporation*

# jrun

Runs a job interactively.

---

### SYNOPSIS

**jrun** [ *file* ] [ *options* ]
**jrun** [ *options* ] −− *file* [ *arguments* ]

#### Options:

[**-jobname** *name*] [**-machine** *mach*] [**-cputime** *n* ] [**-memory** *n* ] [**-nprocs** *n* ] [**-capture**] [**-after** *time*] [**-chdir** *path*] [**-dedicated**] [**-echo**] [**-eightbit**] [**-emacs**] [**-escape** *char*] [**-exact_mem**] [**-exact_proc**] [**-execute**] [**-export**] [**-hard_cputime** *n* ] [**-hard_memory** *n* ] [**-help**] [**-hold**] [**-interruptable**] [**-linelimit** *n*][**-mail_begin**][**-mail_end**][**-mail_user** *user* ] [**-maxrerun** *n* ] [**-need** *rsrcs* ] [**-nobell**] [**-nocapture**] [**-nodedicated**] [**-noecho**] [**-noemacs**] [**-noescape**] [**-noexact_proc**] [**-nointerruptable**] [**-nopreamble**] [**-nostop**] [**-noterminal**] [**-notimeshared**] [**-preamble**] [**-priority** *n* ] [**-processors** *n* ] [**-procs** *n* ] [**-project** *name* ] [**-queue** *name* ] [**-range** *m-n* ] [**-safety** *n* ] [**-server** *name* ] [**-seq** *m-n* ] [**-shell** *name* ] [**-silent**] [**-sp_disk** *n* ] [**-sp_memory** *n* ] [**-stderr** *name* ] [**-stdin** *name* ] [**-stdout** *name* ] [**-terminal**] [**-time** *n* ] [**-timeshared**] [**-ucode** *n* ] [**-want** *rsrcs* ]

---

### DESCRIPTION

The **jrun** command runs a job interactively. The output is directed to your terminal and the input is obtained from your keyboard.

To disconnect your terminal from the job, type a tilde (**~**) followed by Ctrl-b. After you disconnect, the input and output will be redirected to disk files or **/dev/null**, as indicated by the **-stdin** and **-stdout** options. The job can later be reconnected to the terminal using **jconnect**(1).

If no program is specified on the command line, your login shell is invoked and an interactive login session is created.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

The options to **jrun** are a superset of those provided by **jsub**. Options may be specified on the command line or they can appear as embedded comment directives within the job's shell script. See **jsub**(1).

At minimum you must specify the following options: **-nproc**, **-cputime**, and **-memory**. These are required by DJM in order to schedule jobs effectively.

When disconnected, output will be sent to a file with the same name as the job file with the suffix *.onnn* appended, where *nnn* is the job ID.

If the **EMACS** environment variable is set, **jrun** assumes it is running inside an emacs shell-mode buffer.

#### Escape Commands

A tilde (**~**) appearing as the first character on a line is an escape for commands that affect the local terminal. The following commands are recognized:

| | |
|---|---|
| **~**Ctrl-b. | Disconnects the terminal from the job. |
| **~**Ctrl-d | Disconnects the terminal from the job. Other synonyms are **~**b and **~**d. |
| **~**q | Terminates the job. Other synonyms are **~**. and **~**Ctrl-C. |
| **~**Ctrl-z | Suspends the terminal session. |
| **~**Ctrl-y | Suspends only the input. Output from the job will continue to be sent to your terminal. |
| **~**c | Toggles capture mode. In this mode the output is sent both to your terminal and to the job's **-stdout** file. The initial default can be set with **-capture** or **-nocapture**. |

### OPTIONS

The options unique to **jrun** are defined below. See **jsub**(1) for a definition of the remaining options.

| | |
|---|---|
| **-capture** | Always captures output to the **-stdout** file, even when the job is connected to the terminal. Normally output is captured only when the job is disconnected. |


*Copyright © 1993 Thinking Machines Corporation*

**-echo** Turns on echo. The session will echo characters typed at the terminal. This is the default.

**-emacs** Turns on emacs mode. This option is useful when using **jrun** inside an emacs shell-mode buffer. It disables character echo and disables Ctrl-s and Ctrl-q processing (flow control). This option overrides the **EMACS** environment variable.

**-noecho** The session should not echo characters typed at the terminal.

**-escape** *char* Sets the escape character to *char*. The default is a tilde (**~**). The escape character is used to prefix commands that affect the terminal session. A control character can be denoted with a caret (^A), given in octal notation ('\ 01'), or typed directly. **-noescape** disables the escape character.

**-nobell** Turns off ringing the bell when the connection is established.

**-noemacs** Turns off emacs mode. This option overrides the **EMACS** environment variable.

**-nostop** Disables Ctrl-s and Ctrl-q processing (flow control). Normally flow control is enabled on the local terminal unless a remote program explicitly disables them via ioctl calls. This option is usually not needed, as programs such as emacs will disable flow control whenever necessary.

**-eightbit** Passes 8-bit characters through unchanged. By default the uppermost bit of each character is stripped off to create 7-bit logical characters.

## EXAMPLE

```
% cat myjob
#!/bin/sh
#
# This script shows an example of embedded DJM comment directives
#JSUB -procs 8K
#JSUB -cpu 60min
#JSUB -mem 80M
#
a.out
% jrun myjob
Job id is 57.
```


*Copyright © 1993 Thinking Machines Corporation*

```
[Type ~^B to disconnect, ~^C to kill.]
[Connected to c3.msc.edu /dev/ttypa.]
*** job57 procs 8192, mem 80M, cpu 60min, seq 2, server c3:0
*** alan /bin/sh myjob started on Fri May 22 14:18:01 1992
Attached to {CM}* timesharing on CM-MSC, a CM200 on interface 0
Attached to 8192 CM200 processors on sequencer 2, microcode version 6104
1024K of memory with 64 bit floating point. Paris safety is on.
+ a.out
Working...
I'm done.
Detached from {CM}* Timesharing.
*** Job terminated at Fri May 22 15:20:15 1992
*** exitcode = 0000000
*** procs 8192, mem 43M, cpu 52min
***
*** You requested 80M of memory but only used 43M.
*** The next time you run this job, you should request
*** less memory. This will let your job run earlier
*** and you will get your results sooner.
```

## SEE ALSO

**jsub**(1), **jconnect**(1), **job**(1)


*Copyright © 1993 Thinking Machines Corporation*

# js

Shows job information for all machines.

---

### SYNOPSIS

**`js`** [ *command* ]

#### Commands:

**`global jobs limits machines partitions queues servers scores up`**

---

### DESCRIPTION

**`js`** displays the job status of all of the machines in the local site configuration. The command line options are the same as for **`jstat`**(1).

### NOTES

**`js`** is implemented as a shell script that executes **`jstat`**(1) on every machine in order to obtain the job information.

### SEE ALSO

**`jstat`**(1)

---


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

# jstat

Shows job information.

---

### SYNOPSIS

**jstat** [ *command* ] [ *options* ]

#### Commands:

**global jobs limits machine partitions queues servers scores up**

#### Options:

[ [**-jid**] *jobid*] [**-jobname** *name*] [**-noheader**] [**-machine** *mach*] [*jobid*] [*queue_name*]

#### NQS Compatibility Options:

[**-h** *mach*] [**-a**] [**-b**] [**-f** *queue*] [**-l**] [**-q**] [**-r**] [*jobid*] [*queue_name*]

---

### DESCRIPTION

**jstat** displays the status of the Distributed Job Manager and its jobs and queues. The default output is a summary of all jobs.

The arguments to **jstat** are upward-compatible with those for the command **qstat** in the Network Queuing System (NQS).

All options and commands may be abbreviated to uniqueness.

### OPTIONS

| | |
|---|---|
| [**-jid**] *jobid* | Displays information on the job with the given *jobid*. |
| **-jobname** *name* | Displays information on the job with the given name. |
| **-noheader** | Leaves header out of the output. |


*Copyright © 1993 Thinking Machines Corporation*

**-machine** *mach* Selects machine *mach*. This is typically the name of a Connection Machine system.

**-a -l -q -r** These are NQS-compatible options for display of jobs, limits, queues, and processes, respectively.

## COMMANDS

### jstat jobs

This command is the default. It shows a list of all jobs in the system. Running as well as non-running (queued) jobs are displayed.

```
USER     JID  MACH   SERV  PROCS TIME     MEM    STATUS    COMMAND
mxo1201  17   cm200  cb:1  8K    0:01     8M     Running   planenws1ptqd
mxm2206  10   cm200  ca:0  8K    24:00i   1G     Running   airtime
root     23   cm5    c3    32    11:27f   22.8M  Running   dvrcp
mxm1901  22   cm5    c1    512   4:02f    3.17G  Suspend   q43/hn3dx
mxb1201  24   cm5          256   240:00   600M   Que npr   SH/SH_APP/shayd_rdt
mxm2208  19   cm5          512   120:00D  4GX    Que nde   bubrun/mubble
mxm1901  22   cm5          512   45:00    3.17G  Hold      q43/hn3dx
```

**USER** The name of the user.

**JID** The job identifier. This is assigned to the job by the system.

**MACH** The name of the machine on which the job is running or queued.

**SERV** This is the server on which the job is running. For a queued job this field is blank. This is followed by the interface number on systems with more than one interface.

**PROCS** The number of processors. The suffix K indicates a multiple of 1,024 processors. For a running job this value is the number of actual processors in use. For a queued job this is the minimum number of requested processors (or the exact number if **-exact_proc** is specified).


*Copyright © 1993 Thinking Machines Corporation*

**TIME** The amount of CPU time. The format is *mmmm:ss* where *mmmm* is the number of minutes and *ss* is the number of seconds. For a running job this value is the amount of CPU time actually consumed. For a queued job this is the amount of CPU time requested. The suffix **f** indicates a foreign job — one that was run directly on the machine without going through the Distributed Job Manager. The suffix **i** indicates an interactive job. The suffix **D** indicates a job that requires dedicated use of machine resources. The suffix **?** indicates that DJM was unable to obtain information about the job within the past minute. (This usually indicates a crashed machine or a malfunctioning timesharing daemon.)

**MEM** The amount of memory. For a running job this is the amount actually in use. For a queued job this is the amount requested. The suffix **K** indicates kilobytes (1,024), **M** indicates megabytes (1,048,576), and **G** indicates gigabytes (1,073,741,824). The letter **X** means that the user has asked for an exact amount of memory (see **jsub -exact_memory**).

**STATUS** The job's current status. **Running** means the job is currently executing on the machine. **Hold** means the job is being held in the queue by the user or by the operator. **Suspend** means the job is not receiving any CPU time. SIGNAL means the job has received a signal and is waiting to terminate or checkpoint itself. DYING means the job was sent a SIGKILL but did not terminate (this usually indicates a problem with the server). QUE means the job is queued. This is followed by a three-letter explanation of why the job is not running, as described below:

- **dwn** The machine is down or halted.
- **stp** The queue is stopped.
- **npr** The requested number of processors is not available.
- **pqu** The job is not from an acceptable queue (**-queue**) for the partition.
- **pne** The job does not have the necessary resources (**-need**) for the partition.


*Copyright © 1993 Thinking Machines Corporation*

| | |
|---|---|
| **psu** | The partition does not have sufficient resources for the job. |
| **pme** | The requested memory exceeds the maximum allowed for the partition. |
| **pcp** | The requested CPU time exceeds the maximum allowed for the partition. |
| **pcm** | The requested product of CPU time and memory exceeds the maximum allowed for the partition. |
| **pnj** | Too many jobs are running on the partition. |
| **srv** | The requested server is not available. |
| **spr** | The specific number of processors is not available (**-exact_proc**) |
| **bsy** | The processors are busy running a dedicated or non-timeshared job. |
| **usr** | The user has held the job. |
| **opr** | The operator has held the job. |
| **mem** | Not enough memory is available. |
| **nde** | The partition is configured for dedicated mode but the job is not a dedicated job. |
| **ded** | The partition is configured for non-dedicated (normal) mode but the job is a dedicated job. |
| **dra** | Processors are being held idle in order to permit a higher priority job to run. This is called “draining” a partition. |
| **cnf** | A reconfiguration event is approaching where jobs will be killed, and the estimated completion time of this job plus the other running jobs would be too long. |


*Copyright © 1993 Thinking Machines Corporation*

**qnj** Too many other jobs are running from this queue.

**unj** Too many other jobs are running by this user.

**qnm** Too much memory is already in use by other jobs in this queue.

**tmj** Too many jobs are running. The global per-user or per-partition run limit has been reached.

**ts** System is waiting for timesharing to start or stop.

**COMMAND** For a running job, this field indicates the name of the process currently being executed. For a queued job, this indicates the name of the shell script or program that was submitted by the user. The user's home directory or work directory is stripped from the beginning of the pathname. Other messages include the following.

***** The scheduler is frozen. *****
The operator has frozen the DJM scheduler. No jobs will be run until the scheduler is unfrozen. Jobs can still be queued.

***** Simulate only. *****
The scheduler will not actually run or kill jobs. Instead it will report its intended actions and scheduler parameters.

***** Jobs will not be killed. *****
The DJM kill mechanism is disabled.


*Copyright © 1993 Thinking Machines Corporation*

**jstat job** *name*

The optional argument *name* will display detailed information about a specific job. The argument can be either the job's name or it's **`jid`** number. The information consists of attribute and value pairs separated by a tab. An example is shown below.

```
jid 165
uid 6257
sid 19990
state    Running
jobname  job165
queue_name    foreign
uname    mxm7001
file     rrec2
cdir     /net/ec/wrk2/mxm7001
stdout_name   /dev/tty
pri 50
when_queued   706771371
when_run 706772343
foreign_pid   0
active_pid    19990
active_cputime 5
active_memory 370688
active_when_started706772343
max_memory    370688
is_timeshared yes
requested_cputime  3600
requested_memory   4194304
requested_nproc    512
hard_cputime  0
hard_memory   0
actual_nproc  128
actual_baseproc    512
dedicated     no
rerun    0
sp_disk  0
sp_memory     0
server_name   c1.msc.edu
server_interface   0
foreign  yes
hastty   yes
echo     yes
onlcr    yes
linelimit     9999
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

```
exact_proc    no
interruptable no
```

The meaning of some of these fields is described below.

**jobname** The name of the job set by **-jobname**.

**queue_name** The name of the queue to which the job was submitted.

**file_name** The name of the shell script or program to be executed.

**cdir** The current working directory.

**stdout_name** The name of the standard output file or device.

**pri** The job's user-specified priority (**-pri**).

**when_queued** The time when the queue was enqueued (the timestamp).

**when_run** The time when the job began execution.

**active_pid** The process ID of the process that has consumed the most CPU time while attached to the machine.

**active_cputime** The CPU time used by the active process, measured in seconds.

**active_memory** The memory currently in use by the active process, measured in kilobytes.

**active_when_started**
The time when the most recent CM process began execution.

**max_memory** The maximum memory used by this job.

**is_timeshared** Whether the job is executing under a timeshare daemon.

**requested_cputime**
The CPU time requested by the user.

**requested_memory**
The memory requested by the user.

**requested_nproc**
The number of processors requested by the user.


*Copyright © 1993 Thinking Machines Corporation*

**hard_cputime** The hard CPU time limit.

**hard_memory** The hard memory limit.

**actual_nproc** The actual number of processors in use.

**dedicated** Whether this job requires dedicated access to a partition.

**rerun** The maximum number of times to rerun this job after an abnormal termination.

**sp_disk** The amount of scalar processor disk space requested by the user.

**sp_memory** The amount of scalar processor memory requested by the user.

**server** The name of the server on which the job is running.

**server_interface**
The number of the interface on which the job is running.

**foreign** Whether the job is foreign (not submitted through DJM).

**hastty** Whether the job has a pseudo tty.

**echo** Whether keystrokes are echoed back to the user.

**onlcr** Whether a newline is mapped to a carriage return-newline pair on output.

**linelimit** The job's output line limit.

**exact_proc** Whether the job must run in a partition with exactly the requested number of processors, and no more.

**interruptable** Whether it is safe to kill the job early. Jobs that are interruptible are allowed to began execution immediately before a scheduled shutdown.


*Copyright © 1993 Thinking Machines Corporation*

#### jstat proc

Shows a list of running processes. Unlike the jobs display, it is possible for multiple lines to be displayed for a single job. Jobs that are queued are not shown.

```
USER      SERV   PROCS  TIME   MEM     PID    COMMAND
mxm2206   ca 0     8K   24:00  1G      3939   airtime
mxo1201   cb 1     8K   0:01   8M      23687  planenws1ptqd
mxm2208   c1       512  28:11  400M    8831   /net/ec/wrk2/mxm2208/bubrun/mubble
mxm1901   c1       512  4:02   3.17G   8958   /net/ec/wrk1/mxm1901/q43/hn3dx
root      c3       32   11:27  22.8M   24154  /usr/local/unsupported/dvrcp
mxb1201   c3       32   0:34   355M    24170  /net/ec/u2/mxb1201/SH/SH_APP/shayd_rdt
```

**USER** The name of the user.

**SERV** The server on which the job is running. This is followed by the interface number on servers with more than one interface.

**PROCS** The number of processors. The sufix **K** indicates a multiple of 1,024 processors.

**TIME** The amount of CPU time consumed. The format is *mmmm:ss* where *mmmm* is the number of minutes and *ss* is the number of seconds.

**MEM** The amount of memory currently in use. The suffix **K** indicates kilobytes (1,024), **M** indicates megabytes (1,048,576), and **G** indicates gigabytes (1,073,741,824).

**PID** The process ID.

**COMMAND** The pathname of the program being executed.

#### jstat scores

Shows a list of jobs and their associated *scores*. A score is a value assigned to each job by the DJM master scheduler. The score indicates the relative ranking of the job in terms of access to system resources. For queued jobs, the job with the highest score will run next if the requested resources are available. The scores of the running jobs are examined to determine which job(s) should be killed or checkpointed to make room for the designated job. This occurs only if the score of the queued job is higher than the sum of the scores of all the jobs holding the required resources.


*Copyright © 1993 Thinking Machines Corporation*

DJM recalculates the scores of all jobs once per minute. The score is determined by a set of equations whose terms are linear combinations of factors based on CPU time, memory use, number of processors requested, job wait time, and other parameters. For details on these parameters, see **`jstat queues`** below.

The jobs are shown in order of descending scores.

```
USER     JID  JOBNAME  MACH   NPROC QUEUE     STATUS    SCORE
mxm1901  22   h3ndx    cm5    256   default   Running   1054233
root     23   dvrcp    cm5    32    foreign   Running   1006723
mxb1201  24   airfoil  cm5    256   foreign   Running   1004525
mxm2208  19   dyna3d   cm5    512   default   Que npr   45322
```

**`JOBNAME`** The name of the job as set by **`-jobname`**. By default this is the name of the submitted file.

**`QUEUE`** The name of the queue to which the job was submitted. Note that foreign jobs are automatically assigned to the **`foreign`** queue.

**`STATUS`** The status of the job. See the description under **`jstat jobs`**.

**`SCORE`** The score for this job.

#### jstat limits

Shows the resource limits associated with each queue. These represent the maximums on attributes such as CPU time and memory usage that can be used by the jobs submitted to that queue. For MEM limits, the suffix is printed in uppercase for total limits, and in lowercase for per-processor limits.

```
QUEUE        TIME   MEM    MEM/PROC*CPU  NPROC   UQ   UR   UPR TR   TPR TM     TPM
small_quick  15min  220M   unlim         unlim   999  2    1   12   6   unlim  unlim
small_slow   unlim  220M   unlim         unlim   999  2    1   8    4   unlim  220M
medium       2hr    440M   14.8M*hr      unlim   999  2    1   2d   1   unlim  440M
big_quick    15min  unlim  unlim         unlim   999  2    1   2    1   unlim  unlim
night        unlim  unlim  unlim         unlim   999  2    1   2    1   unlim  unlim
foreign      20min  128M   unlim         unlim   999  999  999 999  999 unlim  unlim
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**QUEUE** The name of the queue.

**TIME** The maximum amount of CPU time allowed per job. The suffix **min** means minutes, the suffix **hr** means hours, and the suffix **d** means days.

**MEM** The maximum amount of memory allowed per job. The suffix **K** means kilobytes, the suffix **M** means megabytes, and the suffix **G** means gigabytes.

**MEM/PROC*CPU** The maximum amount of (memory/processor) * CPU time allowed per job. The memory and time suffixes for MEM and TIME are used. For example, **30M*min** would mean that a job could request 10 megabytes/processor and 3 minutes, or 30 megabytes/processor and 1 minute.

**NPROC** The maximum number of processors allowed per job. The suffix **K** means 1,024 processors.

**UQ** The maximum number of jobs that may be queued by each user.

**UR** The maximum number of jobs that may be run concurrently by each user from this queue.

**UPR** The maximum number of jobs that may be run concurrently by each user on any one partition, from this queue.

**TR** The maximum (total) number of jobs that may be run concurrently from this queue.

**TPR** The total number of jobs that may be run concurrently from this queue, on any one partition.

**TM** The maximum amount of memory that may be consumed by all jobs from this queue.

**TPM** The maximum amount of memory that may be consumed by all jobs from the queue, on any one partition.


*Copyright © 1993 Thinking Machines Corporation*

### jstat queues

Shows a summary of the activity of all the queues. Each line represents one queue.

```
QUEUE         MACH   SRCH DED   FOR   ENA   STA   QUEd HLD   RUN   TOT
small_quick   cm5    40   no    no    yes   yes   2    0     1     3
small_slow    cm5    50   no    no    yes   yes   0    0     0     0
medium        cm5    60   no    no    yes   yes   0    0     1     1
big_quick     cm5    70   no    no    yes   yes   0    0     0     0
night         cm5    80   no    no    yes   yes   3    0     0     3
foreign       cm5    90   no    yes   yes   yes   0    0     0     0
```

**MACH** The name of the machine.

**SRCH** The search order. When a job is submitted, this parameter determines the order in which the queues are examined to determine which one will receive the job. At submit time, DJM searches the queues beginning with the queue with the lowest search order value. The queues with progressively higher values are examined until one is found that meets all the user-specified criteria.

**DED** Whether the queue is for dedicated jobs (yes) or normal jobs (no).

**FOR** Whether the queue is for foreign jobs — those that are run directly on the machine without going through the Distributed Job Manager.

**ENA** Enabled: whether the queue will accept (yes) or reject (no) submitted jobs.

**STA** Started: whether the will run (yes) or hold (no) queued jobs.

**QUE** The number of jobs queued.

**HLD** The number of jobs held by the user or by the operator.

**RUN** The number of running jobs.

**TOT** The total of the **QUE**, **HLD**, and **RUN** columns.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**jstat queue** *name*

The optional argument *name* will display detailed information about a specific queue. The information consists of attribute and value pairs separated by a tab. An example is shown below.

```
name     default
search_order  100
enabled  yes
started  yes
users 0 :
groups 3 : 345 546 3266
projects 2 : alpha1 research
max_memory    0
max_cputime   0
max_cpu_times_mem_per_proc   0
max_nproc     0
min_memory    0
min_cputime   0
min_cpu_times_mem_per_proc   0
min_nproc     0
min_pri  0
max_pri  100
foreign  no
range_baseproc     -1
range_nproc   0
dedicated     no
user_nsub_limit    999
user_nrun_limit    999
user_part_nrun_limit    999
queue_nrun_limit   999
queue_part_nrun_limit   999
queue_mem_run_limit     0
queue_part_mem_run_limit     0
queued_base_factor 100000
queued_wait_factor 1
queued_nproc_factor     -0.25
queued_memory_factor    0
queued_cputime_factor   0
queued_mem_cpu_product_factor   0
queued_already_queued_factor   -3600
queued_pri_factor  0
running_base_factor     1e+06
running_cputime_exceeded_factor   -2e+06
running_nproc_factor    0
```


*Copyright © 1993 Thinking Machines Corporation*

```
running_memory_factor   0
running_memory_exceeded_factor    -2e+06
running_mem_cpu_product_exceeded_factor  -2e+06
running_interactive_factor   1000
running_dedicated_late_factor     -2e+06
```

The meaning of some of these fields is described below. Times are in seconds, memory sizes are in kilobytes.

**search_order** The order in which the queue is searched relative to other queues.

**enabled** Whether the queue is enabled to receive jobs.

**started** Whether the queue is started to run jobs.

**users** A list of user IDs of users who are validated to submit jobs to the queue.

**groups** A list of group IDs of groups that are validated to submit jobs to the queue.

**projects** A list of project names that are validated to submit jobs to the queue.

**exact_mem_users**
A list of user IDs of users who are validated to use the **-exact_memory** option.

**groups** A list of group IDs of groups that are validated to use the **-exact_memory** option.

**projects** A list of project names that are validated to use the **-exact_memory** option.

**max_memory** The maximum amount of memory a job can request.

**max_cputime** The maximum amount of CPU time a job can request.

**max_cpu_times_mem_per_proc**
The maximum product of CPU time and memory processor that a job can request.

**max_nproc** The maximum number of processors a job can request.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**min_pri** The minimum priority a job can request. Priorities lower than this will be raised.

**max_pri** The maximum priority a job can request. Priorities higher than this will be lowered.

**foreign** Set to "yes" if the queue accepts foreign jobs.

**dedicated** Set to "yes" if the queue accepts dedicated jobs.

**user_nsub_limit**
The maximum number of jobs that can be submitted per user.

**user_nrun_limit**
The maximum number of jobs that can be run per user.

**user_part_nrun_limit**
The maximum number of jobs that can be run per user, in a single partition.

**queue_nrun_limit**
The maximum number of jobs that can be run from this queue.

**queue_part_nrun_limit**
The maximum number of jobs that can be run from this queue, in a single partition.

**queue_mem_run_limit**
The maximum amount of memory that can be used by this queue.

**queue_part_mem_run_limit**
The maximum amount of memory that can be used by this queue, in a single partition.


*Copyright © 1993 Thinking Machines Corporation*

The following parameters are constants used by the scheduler when calculating the score of each job:

**`queued_wait_factor`**
: This is multipled by the job's wait time in seconds.

**`queued_nproc_factor`**
: This is multipled by the job's number of requested processors.

**`queued_pri_factor`**
: This is multipled by the job's requested priority.

**`queued_cputime_factor`**
: This is multipled by the job's requested CPU time in seconds.

**`queued_memory_factor`**
: This is multipled by the job's requested memory size in kilobytes.

**`queued_mem_cpu_product_factor`**
: This is multipled by the job's requested CPU and memory, measured in kilobyte-seconds.

**`queued_already_queued_factor`**
: This is multipled by the the number of jobs already queued by this user.

**`running_base_factor`**
: This is added to the score of all running jobs.

**`running_nproc_factor`**
: This is multipled by the job's actual number of processors.

**`running_memory_factor`**
: This is multipled by the job's maximum actual memory use.

**`running_interactive_factor`**
: This is added to the score of foreign jobs.

**`running_cputime_exceeded_factor`**
: This is added to the score of jobs that have exceeded their requested CPU time or the soft limit of a partition (should be a negative number).


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**running_memory_exceeded_factor**
: This is added to the score of jobs that have exceeded their requested memory limit or the soft limit of a partition (should be a negative number).

**running_mem_cpu_product_exceeded_factor**
: This is added to the score of jobs that have exceeded the product of CPU and memory for a given partition (should be a negative number).

**running_dedicated_late_factor**
: This is added to the score of jobs that have overrun their dedicated time period and are now in a reconfigured non-dedicated partition (should be a negative number).

#### jstat partitions

Shows information about the partitions of the machine. A partition represents a portion of the machine that is connected to a particular front-end server. The partition information is divided into three types. A *running* partition reflects the actual state of the machine. A *current* partition is what the system administrator wants the partition configuration to look like. A *future* partition is what the system administrator wants the partition configuration to look like sometime in the future. This allows DJM to start and stop timesharing daemons and to repartition the machine automatically (not yet supported on the CM-5).

```
Running partitions:
BPROC    NPROC     SERVER    INT  FREE_MEM  TOT_MEM   TS   LOAD
0        256       c1        0    3.92G     7.48G     y    2
256      256       c2        0    0         7.48G     y    1
768      32        c3        0    101M      479M      y    1
```

**BPROC** The ordinal number of the first processor in the partition (base processor).

**NPROC** The total number of processors in the partition.

**SERVER** The server assigned to this partition.

**INT** The interface number on the server.

**FREE_MEM** The amount of free memory available.


*Copyright © 1993 Thinking Machines Corporation*

**TOT_MEM** The total memory available to run jobs. This equals the total physical memory minus the overhead for the timesharing system.

**TS** Whether timesharing is running on this partition.

**LOAD** The number of processes running on this partition.

```
Current partitions
BPROC   NPROC SERVER MODE     TS  TIME       MEM      MEM/PROC*CPU
0       256   c1     normal   y   120:00 H   1G   H     unlim
256     256   c2     dedicate y   unlim      unlim     3G*hrS S
768     32    c3     normal   y   120:00 S   128M S     unlim
```

**BPROC** The desired ordinal number of the first processor in the partition (base processor).

**NPROC** The desired number of processors.

**SERVER** The desired server.

**MODE** **Normal** means that the partition can be accessed by normal (non-dedicated) jobs. **Dedicate** means that only jobs with the **-dedicated** option may run. This is a special mode that is typically used for overnight jobs that require access to the entire machine. **Shutdown** means that the partition is off-line and no jobs will be scheduled.

**TS** Whether timesharing should be running or stopped.

**TIME** The CPU time limit. **S** indicates a soft limit, and **H** indicates a hard limit. If a partition has both soft and hard limits, only the most restrictive limit is shown. For a soft limit, a job that requests more than this amount will not run unless it is **-interruptable**.

**MEMORY** The memory limit.

**MEM/PROC*CPU** The limit for the product of memory/processor and CPU time.

Jobs that exceed any hard limits will be killed. Jobs that exceed any soft limits are vulnerable to being killed if other jobs are waiting to run.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

```
Future partitions:
# BPROC  NPROc  SERVER MODE     TS TIME   MEM    MEM/PROC*CPU  RECONFIG_TIME
01   0   512    c1     shutdo   y  unlim  unlim  unlim         Sep 03 02:00 kill
lock
02   0   512    c1     dedica   y  unlim  unlim  unlim         Sep 03 07:00 unlock
03   0   512    c1     normal   y  6hr H  unlim  unlim         Sep 03 08:00
```

**#** The partition reconfiguration ID. This ID can be used to erase (wipe out) a future partition. See **jop wipe_reconfig**.

**TS** Whether timesharing should be running or stopped.

**RECONFIG_TIME** The time at which reconfiguration will occur. The annotation **kill** means that jobs running at that time will be automatically killed. The annotation **lock** means that once the partition becomes current, no subsequent reconfigurations will be honored. **unlock** turns off the effect of **lock**.

#### jstat global

Shows a global summary of the state of the machine.

```
MACH      NAME  QUE   HLD   RUN   TOT   UQUE  URUN  PRUN  UPRUN
cm200     pa    1     1     5     7     4     4     4     1
cm5       pb    0     0     6     6     4     4     4     1
```

**MACH** The primary name of the machine.

**NAME** The secondary name of the machine. Also used for accounting purposes.

**QUE** The total number of jobs queued on the machine.

**HLD** The total number of jobs held on the machine.

**RUN** The total number of jobs running on the machine.

**TOT** The grand total of the **QUE**, **HLD**, and **RUN** columns.

**UQUE** The global maximum number of jobs that may be queued by each user.

**URUN** The global maximum number of jobs that may be run concurrently by each user.


*Copyright © 1993 Thinking Machines Corporation*

**PRUN** The global maximum number of jobs that may be run concurrently on each partition.

**UPRUN** The global maximum number of jobs that may be run concurrently by each user, on each partition.

### jstat machine

This command shows a summary of the configuration of the machine.

```
MACH      NAME       NPROC      MEMORY     SERVERS
cm200     pa         32K        4G         ca cb
cm5       pb         544        8.5G       c1 c2(DOWN) c3
```

**MACH** The primary name of the machine.

**NAME** The secondary name of the machine. Also used for accounting purposes.

**NPROC** The total number of processors on the machine.

**MEMORY** The total amount of physical memory on the machine.

**SERVERS** A list of servers for this machine. The annotation **(DOWN)** indicates that the server is not running.

### jstat servers

This command shows which servers are running. An alias for this command is **jstat up**.

```
STATUS    SERVER
UP        ca.msc.edu
DOWN      cb.msc.edu
UP        c1.msc.edu
DOWN      c2.msc.edu
DOWN      c3.msc.edu
UP        c4.msc.edu
UP        c5.msc.edu
UP        c6.msc.edu
```


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

| | |
|---|---|
| **STATUS** | **UP** indicates the server is alive and capable of running jobs. **DOWN** indicates that something is wrong (server is crashed, timesharing is hung, etc.) |
| **SERVER** | The fully qualified domain name of the server. |

## NOTES

To prevent thrashing on the servers, the information is updated no more than once every 15 seconds.

## SEE ALSO

**job**(1)


*Copyright © 1993 Thinking Machines Corporation*

# jsub

Submits a job for background execution.

---

### SYNOPSIS

**jsub** [ *file* ] [ *options* ]
**jsub** [ *options* ] −− *file* [ *arguments* ]

#### Options:

[**-jobname** *name*] [**-machine** *mach*] [**-cputime** *n* ] [**-memory** *n* ] [**-nprocs** *n* ]
[**-dedicate**] [**-after** *time* ] [**-chdir** *path* ] [**-exact_mem**] [**-exact_proc**]
[**-execute**] [**-export**] [**-hard_cputime** *n*] [**-hard_memory** *n*] [**-help**] [**-hold**]
[**-interruptable**] [**-linelimit** *n*] [**-mail_begin**] [**-mail_end**]
[**-mail_user** *user* ] [**-maxrerun** *n* ] [**-need** *rsrcs* ] [**-nodedicate**]
[**-noexact_proc**] [**-nointerruptable**] [**-nopreamble**] [**-noterminal**]
[**-notimeshared**] [**-notty**] [**-preamble**] [**-priority** *n* ] [**-processors** *n* ]
[**-procs** *n* ] [**-project** *name* ] [**-queue** *name* ] [**-range** *m-n* ] [**-safety** *n* ]
[**-server** *name* ] [**-seq** *m-n* ] [**-shell** *name* ] [**-silent**] [**-sp_disk** *n* ]
[**-sp_memory** *n* ] [**-stderr** *name* ] [**-stdin** *name* ] [**-stdout** *name* ]
[**-terminal**] [**-time** *n* ] [**-timeshared**] [**-tty**] [**-ucode** *n* ] [**-want** *rsrcs* ]

#### NQS Compatibility Options:

[**-a** *time*] [**-e** *file_name*] [**-o** *file_name*] [**-eo**] [**-h**] [**-jn**] [**-jy**] [**-ke**] [**-ko**]
[**-re**] [**-ro**] [**-lf** *n*] [**-lm** *n*] [**-lt** *n*] [**-ma**] [**-mb**] [**-me**] [**-mu** *mail_name*]
[**-p** *priority*] [**-q** *queue_name*] [**-r** *job_name*] [**-s** *shell_name*] [**-x**] [**-z**]
[**-A** *project_name*] [**-M** *mail_name*] [**-N** *job_name*] [**-S** *shell_name*] [**-V**]

---

### DESCRIPTION

The **jsub** command submits a file containing a program or a shell script for execution on a machine. The job will be executed in the same directory in which the job was submitted. The job will be run in the background and all output will be directed to a file. The job can be connected to a terminal if the **-tty** option is set. See **jconnect**(1).


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**jsub** is upward-compatible with the command **qsub** in the Network Queuing System (NQS).

Options may be specified on the command line or they can appear as embedded comment directives within the job's shell script. Such directives consist of a shell comment character (**#**) followed by a special prefix: either **J**, **JSUB**, **QSUB**, **PBS**, or **@$**. The first line of the script can begin with **#!** to specify a shell to interpret the script. Example: **#!/bin/sh**. If omitted, the login shell is used.

An example is shown below of using embedded comment directives in a job script.

```
#!/bin/sh                      # the name of the shell
#JSUB -nproc 8K
#JSUB -cpu 60min -mem 128mb
#JSUB
# a.out                        # run my program
```

Directives are evaluated up to the first non-blank, non-comment line or up to the first directive with no **-** present.

The output is sent to a file with the same name as the job with the suffix *.onnn* appended, where *nnn* is the job ID. By default, the standard output and the standard error are combined into a single stream and sent to this file. You can redirect the standard error to a different file with the **-stderr** option. This option does not work if **-tty** is set because there is no way to disambiguate output sent to a terminal device.

At minimum you must specify the following options: **-nproc**, **-cputime**, and **-memory**. These are required by DJM in order to schedule jobs effectively. It is in your best interest to estimate these resource values as accurately as possible with respect to your job's actual needs. If you understate your job's resource requirements, your job may be terminated by DJM if other jobs are waiting to run. On the other hand, you should not overestimate your resource requirements as that may cause your job's execution to be delayed.

These estimates are treated as "soft" limits when your job runs. Your job is not automatically killed if they are exceeded. The chance of getting killed depends on how egregious the overrun is and how heavily loaded the machine is.

The actual resource usage is written to the end of the output when your job finishes. You can use this information to supply improved usage estimates the next time you run your job.


*Copyright © 1993 Thinking Machines Corporation*

### OPTIONS

All option names may be abbreviated to uniqueness.

**-jobname** *name* Sets the job's name to *name*. You can use this name as a handle to refer to this job. Example: **jconnect -job myjob**. The name is limited to 8 characters in length and must be unique across the system. By default the name of the file is used, and if necessary the suffix *_jobid* is appended to make the name unique.

**-machine** *mach* Executes the job on machine *mach*. This is typically the name of a Connection Machine system. By default the machine name is obtained from the environment variable **DJM_MACHINE**.

**-cputime** *n* Requests *n* units of CPU time. The suffix **secs** indicates seconds, **min** indicates minutes, **hr** indicates hours, and **d** indicates days. The default is **min**. Suffixes may be abbreviated. Example: **-cpu 10h**. The amount of time is limited based on per-queue and global maximums. See **jstat limits**.

**-memory** *n* Requests *n* units of memory. The suffix **kb** indicates kilobytes, **kw** indicates kilowords (8192 bytes), **mb** indicates megabytes, **mw** indicates megawords, **gb** indicates gigabytes, and **gw** indicates gigawords. The default is megabytes. A single letter abbreviation indicates a byte (not word) unit. Example: **-mem 2g**. The amount of memory is limited based on per-queue and global maximums. Note that the amount represents *total* memory, not memory per processor.

**-nproc** *n* Requests at least *n* processors. The suffix **k** indicates 1024 processors. Note that this represents a minimum requirement: your job may actually be run on a partition with more processors than you requested if the partition is available. If you require exactly *n* processors, add the **-exact_proc** option.

**-dedicate** Requests dedicated access to the partition. Your job will have the right to access all resources (memory, CPU, nproc) of the partition during its execution. The **-memory** option is ignored, as you own all the memory by definition. The job will only be run when there exists a partition that is enabled for dedicated access. Typically the operator will create one such partition, encompassing the entire machine, overnight. Dedicated partitions have special scheduling rules. See Chapter 2 for more details. (**NOTE:** Do not confuse **-dedicated** with **-notime-**


*Copyright © 1993 Thinking Machines Corporation*

**shared**. The latter merely runs a job on a CM-2 sequencer that does not have a timesharing daemon.)

**-after** *time* Defers execution of the job until after the indicated time (or date). If the parameter contains embedded spaces, it must be enclosed in quotes, for example: **-after "July 4 12:30pm"**. The time argument can be in any reasonable format. Some examples are shown below.

```
12:30pm EDT
Fri July 4, 1992 12:30 EDT
4 jul 1992 12:30 E.D.T.
7/4 12:30
7/4/92 12:30:00
07041230   (``touch format'', required by IEEE Std 1003.15)
Friday 12:30
12am   (midnight)
12 P.M.  (noon)
tomorrow  (this time tomorrow)
tomorrow 3am
2:00 today
+1:00  (one hour from now)
+2 days  (two days from now)
@number  (absolute time in seconds since 1 Jan 70 00:00:00 GMT)
```

Dates that are potentially ambiguous are illegal:

```
92/4/7
92/7/4
7 July 4  (year 2007)
```

Missing components are interpolated with the current time and date. For example, "4 July" means run at the current time of day on the 4th of July in the current year.

**-chdir** *path* Changes the current working directory to *path* before running the job. By default, the job is run in the directory in which it was submitted. All files, including those indicated with **-stdin** and **-stdout**, will be accessed relative to the new path. This option can be used to avoid a design bug in the NFS automounter, where the the current working directory is not the actual path, but rather a temporary link created by the automounter.


*Copyright © 1993 Thinking Machines Corporation*

**-exact_mem** By default DJM releases unused memory back to the system 2 to 5 minutes after the job starts. The **-exact_mem** option tells DJM not to release the unused memory. Use of this option requires validation by the system administrator.

**-exact_proc** Runs the job on a partition with exactly the requested number processors. By default DJM runs a job on a partition with more than the requested number of processors if a partition with the exact number is not available.

**-execute** *file* [*file_arguments*]
This option allows you to pass command line arguments to your program. It must be the last thing on the command line. The parameters consist of the file to be submitted for execution followed by its arguments. The option **--** (two dashes) is a synonym for **-execute**. An example is shown below:

```
jsub -cpu 2hr -mem 1g -nproc 256 -- myjob arg1 arg2 arg3
```

**-export** Exports all environment variables to the job.

**-hard_cputime** *n*
Specifies a hard limit for CPU time. If your job exceeds this limit, it is killed immediately. This is useful for preventing a runaway job from consuming all of a user's account allocation.

**-hard_memory** *n* Specifies a hard limit for memory. If your job exceeds this limit, it is killed immediately. This is useful for preventing a runaway job from consuming all the machine's memory and swap space.

**-help** Displays a summary of all options along with a one-line description of each option.

**-hold** Holds the job. It will not be executed until it is released.

**-interruptable** Informs the scheduler that it is safe to interrupt the job. This option tells DJM that the job should be run right away, even if the job is not expected to finish before the scheduled shutdown time.

**-nointerruptable**
Does not run the job if a scheduled shutdown will occur sooner than when the job is expected to finish. This is the default.


*Copyright © 1993 Thinking Machines Corporation*

**r-linelimit** *n* Sets the limit on the amount of output to *n* lines of text. The default is 9999 lines. This limit applies only to **-tty** jobs when I/O is directed to a disk file.

**-mail_begin** Sends mail when the job begins execution.

**-mail_end** Sends mail when the job ends execution. The resource usage and exit code are displayed. If the job was killed by DJM, an explanation is also sent (e.g., memory limit exceeded).

**-mail_user** *name*
The name of the user to receive the mail described above.

**-maxrerun** *n* Sets the maximum number of times to rerun the job. If this value is greater than 0 and the job terminates with a nonzero exit code (i.e., the last command in the shell script exited abnormally) then the job is run again. The letter **r** is appended to the output file name to indicate that the output is from a second run of the same job.

**-need** *rsrcs* Requests that the job run on a partition that offers all the listed resources. *rsrcs* is a comma-separated list of site-defined keywords. For example, **-need 'new_os,sda,hippi'** might request a partition that has a new operating system, a Scalable Disk Array, and a High Performance Parallel Interface.

**-notimeshared** Requires that the job run on a partition that does not have a timesharing daemon. This option applies only to the CM-2.

**-notty** Does not connect the job to a pseudo tty. This is the default for **jsub** jobs, as opposed to **-tty** for **jrun** jobs. The job cannot be connected to a terminal. The primary advantage of **-notty** is that such jobs can be checkpointed. If checkpointing becomes feasible for open terminals, then this default may change. A synonym is **-noterminal**. For compatibility with NQS, the environment variable ENVIRONMENT is set to BATCH.

**-priority** *n* Sets the user-defined job priority to *n*. This is used as one of the factors in determining the job's score for scheduling purposes. The interpretation is site-dependent.


*Copyright © 1993 Thinking Machines Corporation*

**-preamble** Prints the output from the execution of the preamble. The preamble is a sequence of in-line shell commands that execute your **.cshrc** and **.login** files. By default this output is suppressed.

**-processors** *n* This is a synonym for **-nproc**.

**-procs** *n* This is another synonym for **-nproc**.

**-project** *name* Sets the project name. This is used for accounting purposes. By default the project name is obtained from environment variable **DJM_PROJECT**.

**-queue** *name* Submits the job to the specified queue. The job must meet the queue's maximum limits for memory, CPU time, and number of processors as well as controls on user validation and other attributes. By default the job is submitted to the first queue that meets all the requirements based on the queue search order (see **jstat limits**). The environment variable **DJM_QUEUE** is searched if this option is not specified.

**-range** *m-n* Requires that the job run in an exact set of physical processors. For example, **-range 768-799** means the job must run in 32 processors starting with physical processor number 768.

**-seq** *m-n* Requires that the job run in an exact set of CM-2 sequencers. For example, **-seq 2-3** means that the job must run on sequencers 2 and 3.

**-safety** *n* Sets the level of run-time safety checking to *n*. The environment variable **DJM_SAFETY** is consulted if this option is not specified. 0 indicates no checking and is the default. 10 indicates maximum checking. On the CM-2, this option sets the **CM_DEFAULT_SAFETY** environment variable. On the CM-5 this sets the **CMOS_SAFETY_LEVEL** environment variable.

**-server** *name* Requires that the job run on a specific server. The environment variable **DJM_SERVER** is consulted if this option is not specified.

**-shell** *name* Sets the shell for interpretation of the job script to *name*. This overrides the **#!/bin/sh** comment directive.

**-silent** *name* Submits the job silently. This supresses the normal display of the job's assigned *jobid*.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

**r-sp_disk** *n* Requests *n* units of scalar (front-end) disk space. The suffixes are the same as for **-memory**. The default is site-dependent.

**-sp_memory** *n* Requests *n* units of scalar (front-end) memory space. The suffixes are the same as for **-memory**. The default is site-dependent.

**-stdout** *name* Sets the standard output file to *name*. The default is the name of the job file followed by **.o***nnn* where *nnn* is the job ID. The letter **r** is appended to the name if the output is from a rerun, in order to avoid overwriting the output from the first run. By default the standard error stream is also sent to this file unless **-stderr** is specified.

**-stderr** *name* Sets the standard error file to *name*. This causes the standard output and standard error streams to be split and sent to different files. This option is not supported if **-tty** is set because there is no way to disambiguate **stdout** and **stderr** when sent to a terminal device.

**-stdin** *name* Sets the standard input file to *name*. The default is **/dev/null**. When the job is connected to a terminal, the input is redirected to your keyboard. After you disconnect and all your keystrokes have been read by the job, the standard input will be redirected back to this source.

**-time** *n* This is a synonym for **-cputime**.

**-timeshared** Requires that the job run on a partition that has a timesharing daemon. This option applies only to the CM-2.

**-tty** Assigns the job to a pseudo tty. This is the default for **jrun** jobs, as opposed to **-notty** for **jsub** jobs. A job cannot be connected to a terminal unless this option is set. This option is necessary because some machines do not support checkpointing of a job with an open terminal. A synonym is **-terminal**.

**-ucode** *n* Sets the microcode level to *n*. Example: **-ucode 6002**. This option applies only to the CM-2.

**-want** This is a synonym for **-need**.


*Copyright © 1993 Thinking Machines Corporation*

#### NQS Compatitibility Options

| | |
|---|---|
| **-e** *name* | Sets the standard error file to *name*. |
| **-o** *name* | Sets the standard output file to *name*. |
| **-eo** | Combines standard output and standard error. Unlike NQS, DJM uses this as the default setting. |
| **-ke -ko -re -ro** | These options are ignored. DJM does not make a distinction between local and remote output. |
| **-lM** *n* **-lm** *n* | Sets the per-job memory limit to *n*. Per-process limits are not supported. |
| **-lT** *n* **-lt** *n* | Sets the per-job CPU time limit to *n*. |
| **-lF** *n* **-lf** *n* | Sets the per-job scalar (front-end) disk space limit to *n*. |
| **-mb -me -mu** *user* | These are the same as **-mail_begin**, **-mail_end**, and **-mail_user** respectively. |
| **-q** *name* | Submits the job to named queue. |
| **-r** *name* | Sets the job's name to *name*. |
| **-s** *name* | Specifies the shell to interpret the job's shell script. |
| **-x** | Exports all environment variables to the job. |
| **-z** | Submits the request silently. |

### EXAMPLE

```
% cat myjob
#!/bin/sh
#
# This script shows an example of embedded DJM comment
directives
#JSUB -procs 8K
#JSUB -cpu 60min
#JSUB -mem 80M
#
a.out
% jsub myjob -mach cm200
Job submitted successfully.  Job id is 57.
```


*Copyright © 1993 Thinking Machines Corporation*

(one hour later)

```
% cat myjob.o57
*** Distributed Job Manager 1.0, CM-200 (pa) AHPCRC / Minn.
Supercomputer Ctr
*** job57 procs 8192, mem 80M, cpu 60min, seq 2, server cb:0
*** alan /bin/sh myjob started on Fri May 22 14:18:01 1992
Attached to {CM}* timesharing on CM-MSC, a CM200 on interface 0
Attached to 8192 CM200 processors on sequencer 2, microcode
version 6104
1024K of memory with 64 bit floating point. Paris safety is on.
+ a.out
Working...
I'm done.
Detached from {CM}* Timesharing.
*** Job terminated at Fri May 22 15:20:15 1992
*** exitcode = 0000
*** procs 8192, mem 43M, cpu 52min
***
*** You requested 80M of memory but only used 43M.
*** The next time you run this job, you should request
*** less memory.  This will let your job run earlier
*** and you will get your results sooner.
```

## NOTES

There are per-queue and per-user limits on the number of jobs that can be submitted at the same time.

A **jsub** job with the **-tty** option is indistinguishable from a **jrun** job.

The Posix Batch System (PBS, IEEE 1003.15) changed the way options are parsed, requiring all options to consist of a single letter. DJM, on the other hand, has many more arguments than available letters in the alphabet. This means that options that can be grouped together in PBS (for example, **-zV**) must be broken apart in DJM (**-z -V**).

## SEE ALSO

**job**(1), **jrun**(1), **jstat**(1), **jwait**(1)


*Copyright © 1993 Thinking Machines Corporation*

# jwait

Waits for a job to start.

---

### SYNOPSIS

**jwait** [ *jobid* ] [ *options* ]

#### Options:

[[**-jid**] *jobid*] [**-begin**] [**-end**] [**-machine** *mach*] [**-nobell**]

---

### DESCRIPTION

**jwait** waits for a job to begin or end.

### OPTIONS

| | |
|---|---|
| [**-jid**] *jobid* | Waits for the job with the given *jobid* as returned by **jstat**(1). |
| **-jobname** *name* | Waits for the job with the given *name*. |
| **-begin** | Waits for the job to begin execution. |
| **-end** | Waits for the job to end execution. This is the default. |
| **-machine** *mach* | Selects machine *mach*. This is typically the name of a Connection Machine system. |
| **-nobell** | Does not ring the bell until wait is complete. |

### NOTES

If the job fails to begin execution (e.g., it was deleted or it had a file permission problem), a nonzero exit code is returned.


*Version 2.0, November 1993*
*Copyright © 1993 Thinking Machines Corporation*

### SEE ALSO

**job**(1), **jconnect**(1), **jsub**(1)

---


*Copyright © 1993 Thinking Machines Corporation*