J. SALTZER

INTERDEPARTMENTAL

MASSACHUSETTS INSTITUTE OF TECHNOLOGY *CAMBRIDGE, MASS. 02139*

from the office of Information Processing Center

July 6, 1971

TO:	F. J. Corbato	J. R. Steinberg
	R. Roach	N.I. Morris
	W. Hayden	S. H. Webber
	J. W. Gintell	J. H. Saltzer
	V. Voydock	

FROM: T. H. VanVleck, K. A. Willis

The attached is a draft of our MCB on User control.

Attachment

How about a queue for System full ?

INTERDEPARTMENTAL

MASSACHUSETTS INSTITUTE OF TECHNOLOGY *CAMBRIDGE, MASS. 02139*

from the office of Information Processing Center

DRAFT

date 7/7

TO: Distribution

FROM: Thomas H. VanVleck, Keith A. Willis

SUBJECT: System Control Changes

DATE:

The attached documents describe some of our ideas about changes to the system control programs. All kinds of changes are proposed, from big to little and from obvious to not-so-obvious. We would like additional ideas.

Attachment

Proposed change:

Divide the information in the answer table among 3 tables.

Purpose:

There is currently a restriction that a logged in user be associated with exactly one process and no fewer than one terminal. (The dial command allows a process more than one terminal.) The change will free this restriction allowing the following conveniences.

1. Multiple processes - A user may wish to create additional processes to execute "at the same time" as his main process.

2. Absentee users - The proposed implementation of absentee users keeps a separate list of active processes, mainly because an answer table entry must currently be associated with a terminal. The change will allow all information about absentee users to be managed in the same manner as other users.

3. Disconnect - If a user wishes to leave his terminal but keep his process running, he will be able to do so with a "disconnect" command.

Design requirements:

Split the answer table information among 3 tables: login_user table, process table and terminal table, the login_user table will contain an entry for each user with a master process. The process table will contain an entry for each active process. The terminal table will contain an entry for each existing terminal line.

Each entry in the terminal table will be associated with an event channel as entries in the current answer table. Also, each entry in the process table will be associated with an event channel because a process need no longer be associated with a terminal and its event channel.

This will increase the events being listened to by the number of active processes. The number of processes will continue to increase, of course, as the load units increase. Currently these events exist:

1	for each entry in lines file (104)
1	for updating system tables (up_syctl_)
1	for dial channel
1	for absentee
1	for accounting update

temporary events:

2 for shutdowns
several others, too.

<u>Implementation:</u>

The entries in these tables will be related as follows.
An entry in the login_user table will contain:

1. a relative pointer to an entry in the process table if the user has at least one process,
2. a relative pointer to an entry in the terminal table if the user has at least one terminal,

If a user has more than one process, each entry in the process table will contain a relative pointer to the next entry. If a user has more than one terminal, each entry in the terminal table will contain a relative pointer to the next entry.

Each active entry in the terminal table and process table will contain a relative pointer to the associated login_user table entry

Examples:

A. A logged in user who uses none of the new features will have 1 entry in the login_user table which points to 1 entry in the process table and 1 entry in the terminal table.

B. An absentee user will have 1 entry in the login_user table which points to 1 entry in the process table.

C. A user who has multiple processes will have 1 entry in the login_user table which points to the entry for the master process which points to a slave process. (If more than 1 slave process, the 2nd will be pointed to by the first.)

D. A user who dials into another terminal will have no login_user entry. His terminal will be chained to the terminal entry associated with the login_user he dialed.

The new data bases will save approximately 6 records. The size of the answer table is currently 15 records in 1 segment. The login_user table will occupy no more than 4 records of a segment until more than approximately 62 users are logged in. The process table will occupy 1 record of a segment unless more than 63 processes are active. The terminal table will occupy about 4 records of 1 segment.

Answering service programs to be changed:

The major functional changes are that a new entry point will be added in dialup_ to receive event messages from processes and that after an absentee process has been created it will appear as any other process.

act_ctl

Simple changes that will not alter cpu time or size of program considerably.

$act_ctl_init - It will be passed pointers to all 3 tables.

$open_account - minor change - The pointer passed will be to a login_user table entry, so in order to get some information into the accounting card a reference will be made to the process entry. This assumes that there will be only 1 accounting card for each user (even though multiple processes).

$dp - It must reference the terminal entry to find login_time.

$update - Instead of searching through the answer table to charge cpu usage, it will search the process table. It will also search through the terminal table to charge connect time. A user will be inactive only if all his processes are inactive. A reference must be made to the login_user table.

admin

Minor changes which will keep efficiency and size about the same.

$admin_init - It will be passed pointers to all 3 tables.

$bump_user - It is passed a pointer to the login_user entry to be bumped. It must now check if the user has a terminal before informing him he is bumped. Then it must wake up the master process, so a reference will be made to the process table.

$bump - When a bump for all users is requested, a search should be made through the login_user table and then the master process for all active entries will be woke-up.

$remove - It will search through terminal table instead of answer table.

$warn - If a user is found in the login_user table, then all associated terminals will be given the message.

as_

Very minor changes.

as_$init - Initiate all 3 tables and pass pointers to all 3 tables to the following procs:

dialup_$dialup_init

lg_ctl_$lg_ctl_init

dial_ctl_$dial_init

asu_$asu_init

admin_$admin_init

act_ctl_$act_ctl_init

pass a ptr to login_user to up_sysctl_$up_sysctl_init

asu_

Minor changes which decrease size of program.

$asu_init - It is passed pointers to all 3 tables

$asu_remove - It should signal process event instead of destroying process and logging out itself.

$asu_attach - It will search through the terminal table instead of the answer table.

$asu_listen - It will use the terminal table instead of the answer table.

aswa_

Minor changes.

All entries will get pointers to a terminal table entry.

$tty_new_proc - It will also be passed the process id.

dial_ctl

Minor changes should increase efficiency.

$dial_init - It is passed pointers to 3 tables.

$dial_term - It is passed a pointer to the terminal table entry. To find the master process, a simple search through the processes corresponding to the login_user entry for terminal is needed instead of a search through the entire answer table.

$dial_req - It will search through the process table to find the process that signalled it, instead of the answer table. To find if dialok, it must reference the login_user entry.

$dial_ctl_ - It will be passed person, project, account, result as parameters instead of taking that info from the answer table. It searches through the login_user table instead of the answer table to find if user is ok. If dial is ok, a link will be made to the master process to get info(procid etc.) to put in terminal entry for slave.

$dial_broom - It is passed a pointer to the process entry and instead of searching answer table for all slaves, will simply link to all terminal entries from the login_user entry.

up_sysctl_

Minor changes-simply use login_user table instead of answer table.

up_pnt_

Minor changes - simply use login_user table instead of answer table.

up_pdt_

Minor changes - Use login_user table instead of answer table and when search is made through login_user to find if a user should be bumped, a reference must be made to the process table if any are found. The bump should be sent over the process channel.

dialup_

Extensive changes which will increase the size of code and may take longer to execute.

$dialup_init - It will be passed pointers to all 3 tables. The procedures lock and unlock will use login_user table instead of answer table. (They may be eliminated.)

$dialup_ - It will be signalled by an event channel from a terminal. (as explained later events from processes are sent to dialup_$dialup_ proc.) If the sender of a message is not known, then a search must be made through the terminal table.

to login - It will use login_user table in place of answer table header information and will call dial_ctl_ and lg_ctl_$login with extra arguments (person, project, account, and result).

If login is successful, an entry in the process table will be created and linked to the lu entry created by lg_ctl_. An event channel will be created and set to signal dialup_$dialup_proc. The process will then be created by cpg_ (which may have to be changed ***********). act_ctl_$open_account will then be called (it must wait until after process is created).

dialup8 - executed when terminal hangs up

If a dialed console, then the master will be found and notified and the terminal entry will be freed. Otherwise, the master process associated with the login_user entry is found and control is passed to dialup_proc which treats it as a logout (except for sending a message to the terminal).

$dialup_proc - will be signalled by the following events:

preempt
bump
system shutdown
process destroyed
inactive too long
logout
logout hold
new proc
(a hangup will be entered at a slightly different piece of code)

The process associated with the event can be an absentee, a master or a slave. Therefore, it may have any number of terminals associated with it.

The sequence of actions will be to destroy his process and associated event channel and then do the same for any slave processes. If the process is a slave, it will be the only one destroyed. The process table entries will then be freed. All associated terminals are informed of the action and lg_ctl_ $logout will be called with a pointer to the login_user table. If the event was a logout hold, the master terminal will be found and a transfer made to login. All related terminals entries are cleaned up.

absentee - A change will need to be made only to the code after lg_ctl_$login absentee is called. It will then build an entry in the process table and create an event channel which signals dialup_$dialup_proc when sending a message. (dialup_proc may need a special case to call absentee cleanup after an absentee process is destroyed.)

lg_ctl

Major change which will lengthen code and cpu time slightly.

$lg_ctl_init - It will be passed pointers to 3 tables.

$login_ - It will be passed a pointer to the terminal entry and arguments: person, project, account, result. When it checks if user is already logged in, it will disregard absentee users. When checking if a user can be bumped, a search will be made through the login_user table and admin$bump will be called with a ptr to the login_user entry.

If the person is validated and there is room, and entry will be built in the login_user table and linked to the terminal entry (all info that is currently stored in the answer table must be saved until the login_user entry is built).

$login_absentee - It can check if a user is already logged in to allow absentee the option of holding an absentee until its user has logged out. Basically it does the same thing as login except that it returns a pointer to the entry created in the login_user table.

$logout - It is passed a ptr to login_user table entry which it will free.

system_control_

Minor change to replace answer table with login_user table.

Proposed Change

Modify accounting system to keep usage limits "on line" and delete the acctbl.

Purpose

1. System usage information is currently recorded in the segment "acctbl", with an entry for each logged-in user. In order to bill this usage, these entries must be collected into the hist file which contains one entry for each user. This change will allow charges to be placed directly in the pdt (modified structure) thereby eliminating the acctbl. (The increase in size of each pdt will still conserve records.)

2. The usage limits for each person/ project will be stored in each pdt, allowing a cutoff for users who have overspent their account.

3. Billing may be done for any time period without requiring a system shutdown. The current accounting system is restricted to a monthly billing period and requires a system shutdown to avoid losing usage figures (following billings) for any users logged-in at the time of billing.

Design

The structure of the pdts will be modified to contain project and user limits and usage totals. The header will contain the project totals and the user entries will contain the user totals.

Charges will be made to the user's periodic total, grand total, and the project's grand total by acct_ctl_$update. Charges will be made for cpu usage and terminal connect time with the idea that in the future charges may also be made for paging, total process time, record use, tape setup, and miscellaneous charges.

Billing will gather the periodic totals from all pdts into the hist file. Once the bills have been written from the hist file, a billing-reset program will be executed to subtract the usage figures in the hist from the periodic totals in the pdts.

Implementation

The following programs will be altered:

cv_pmf

Several master and normal keywords will be added to allow a project administrator to enter usage limits. These include "termination_date", "dollar_limit", "cpu_limit" and "terminal_connect limit". The last two may be split among shifts if desired. Defaults for all these values will be extracted from the pdt header (a user's default limit will be identical to the projects).

See program number 1 to be written for description of how header values are generated.

up_pdt

Certain fields of a pdt will not be replaced by a pmf. These are all usage totals and login times. Furthermore, this program will set the state of all pdt entries to 0 meaning inactive, or > 0 meaning active. (1 meaning user may login to pdt, or 2 meaning the user may login or charge to the pdt in proposed change for uncoupling project and account). This will be done to preserve the usage figures of an inactive user. This means the check for logged-in users no longer on a project will reference the state. The order of all entries will remain stable to allow the accounting update to reference the user entry without searching.

lg_ctl_$login

When searching for a user entry in a pdt, only pdt entries with state "active" will be considered. If a match is found, the subscript in the pdt must be saved so it may be stored in the login-user entry (after successful login). A pointer to the pdt is also stored in the login_user entry and the login_time is stored in the pdt entry.

acct_ctl_$open_account

This will only log the users and accounts. The rest of the code manages the building of an acctbl "card" and may be deleted after parallel operation of old and new schemes for a billing period. It will also keep track of the number of crashes for a user by inspecting and setting a flag in the pdt indicating the session is not finished.

acct_ctl_$update

For each entry in the login_user table, usage figures will be totaled for all associated processes and terminals. These totals will then be added to the following three fields in the user's pdt: user's periodic total, user's cumulative total, project's cumulative total. The location of these fields is obtained with the pdt pointer and user index stored in the login_user entry by lg_ctl_$login. Users could be checked here for limit overrun: see below.

The existing code which accumulates totals in the acctbl "card" will be retained until a checkout of the new accounting system is completed.

acct_ctl_$close_account

The final usage figures for one login_user entry will be totaled as in acct_ctl_$update. It will also set a flag in the pdt indicating that the session is complete.

acct_ctl_$daemon_acct_init

The usage figures will be stored in one accounting file for all types of daemons.

New Programs to be Written

1. New_proj This program will be used to create a new pdt, and add, change, or inactivate its user entries. It will be the only program authorized to fill in requistion amounts in the pdt header. It will prompt the administrator for all fields to be completed in the pdt, thoroughly check all input to the pdt and install it.

2. A command will be written to modify the structure of all existing pdts and fill in the information needed.

3. A billing command will be written. It will gather the periodic totals from all pdts and store them in a hist file along with the time of billing. The contents of the pdts will not be altered until after the bills are acceptable (see number 4).

   This command may be activated any time after the reset_billing command (number 4) is completed. (The billing periods are arbitrary.) If the system crashes before completion then the hist file should be deleted and the billing command rerun. (The billing will later be improved to provide automatic restart with protection of data bases.) The bills may then be written with the existing program "wb". If the bills are not completely written (crash during the writing) or if they have mistakes, "wb" may be rerun if the date in the current hist file matches the date printed at the start of billing (must be manually checked.) If not, the hist was lost and the billing program must be rerun.

4. Once the bills have been written correctly, the reset_billing program must be run. This program subtracts the usage amounts just billed and fills in a reset date for the pdt. (The amounts just billed exist in the hist.) If a crash occurs before the completion of the reset-billing program, it must be rerun. (Any pdts with a reset date after the date of the hist are not altered.) If a crash occurs at any time between the start of billing and the completion of resetting which destroys the hist file, then billing must be rerun. This insures that all usage will be accounted for (but may require more cpu time for billing). If a crash occurs at any time which causes a pdt to be lost, then the best that can be done is to recover the most recent pdt. The result is the unavoidable loss of some usage figures.

New Programs which do not need to be completed at the same time

1. acct_ctl_$check

   This entry will compare a user's cpu and terminal connect usage with his limits and deny him permission to login if his account is overspent or if the projects total dollars exceeds his requisition amount. It will also see if the time of login is later than a user's termination date.

2. Another entry in acct_ctl_ may be written to check if any users have overspent their accounts since the time of login.

3. decode_pdt

   This program will allow a project administrator to obtain an ascii file of his online pdt (pmf). Master keywords will be omitted since they would be impossible to determine.

4. proj_billing

   This command will generate a proj_list from a project pdt. It will allow a project administrator to inspect his account's usage at any time.

5. Several statistical summaries will be written.

6. A program will be written which will cleanup the pdts. It will add all deleted user entries which have been inactive for more than one billing period to the free chain. This will allow new users on the project to occupy that entry.

Proposed Change

Uncouple the concept of an account from a project.

Purpose

This proposal eliminates the current restriction that a user must charge to the project on which he is registered. A user will now charge to a drawing-account. This may be his project's pdt, another project's pdt or a drawing-account pdt. The drawing-account may be specified in the user's login line or as a default in his pdt entry. Some or all of the entries in a project's pdt may specify a default drawing-account. This proposal is dependent on the implementation of the proposed change to the accounting system.

Design

The user's pdt entry will contain a default account name which may be a project pdt or a drawing-account pdt. A drawing-account pdt has the same structure as a pdt but does not correspond to one unique project. Therefore, a project administrator may divide the charges of his registered users among several existing pdts or a drawing-account pdt.

Each pdt entry also contains a state flag which indicates that is is active or inactive, that a user may or may not login to the pdt, and that it may or may not charge to the pdt.

Implementation

The following programs must be altered:

cv_pmf

The master keyword Account and normal keyword account are now meaningful. If specified in the ascii pmf, they must be followed by the name of another pdt or a drawing-account pdt. The field state will be set to 2 if account is unspecified or identical to the pdt (indicates the user may login and charge to this pdt), or 1 if account specified is different from the pdt (the user may only login to the pdt). up_pdt will be modified in accounting system change to retain an entry and set its state to inactive (set to 0) if a pdt entry does not exist in the pmf replacing it.

lg_ctl_$login

If the user specified an account in his login line or if his pdt default account is not identical to the pdt name, the sat will be searched for a match with the default account. If no match is found, the user will not be allowed to login. If a match was found then the pdt will be initiated (unless corresponding pdt pointer is not null) and the pdt pointer entry in the sat will be replaced. This pdt will then be searched for a matching user entry with the state indicating the privilege of charging to the pdt (state=2). If not found, the user will not be allowed to login. If found, the pointer to this pdt and the index of the matching entry will be stored in the login user entry when it is built (if login is successful). Attributes and other pdt information stored in the login_user entry will be taken from this pdt entry (not the project's pdt entry).

The sat will be modified to contain entries for the drawing-account pdts. This must be done before an attempt to install a drawing-account pdt.

Proposed change:

arguments for "login"

Purpose

It is sometimes desirable to override the default specifications for user-process initialization. This writeup proposes a syntax.

Description

The login "command" (shouldn't we do something about nomenclature here?) has the following syntax now:

login Person -Project- -Account-

It is proposed to change this to:

login Person -Person- -Project- -options-

where options may be any or all of the following:

Keyword	Argument	Comment
-account	Accountid	
-ac	Accountid	
-initproc	path	relative to homedir
-ip	path	
-homedir	path	relative to default
-hd	path	
-attach	device	usually "tw_"
-at	device	
-nobump		won't bump even if usually could.
-nb		
-bf		shuts off all messages if OK
-brief		

Proposed change:

Automatic "crank"

Purpose

Certain accounting functions must be performed periodically, but take too long to sneak them in between logins. This proposal describes a plan to create a special process to perform these functions.

Description

A procedure will be added to the initializer process to manage the crank. One entrypoint will be called at system startup, as follows:

```
    call crank_ctl_$crank_ctl_init;
```

This call establishes an event call channel and sets a timer which will cause an event call when the crank is to run next. The data which determine crank scheduling are stored in the header of the SAT (or answer table, or somewhere) and are the following:

```
    dcl time_next_crank fixed bin (71),
            /* standard system time */
        crank_running bit (1) aligned,
            /* 1 if running. helps crash recovery */
        time_between_cranks fixed bin (71),
            /* 24 hours, usually */
        crank_wakeup_channel fixed bin (71);
```

More data may be needed.

The timer will eventually go off (right away, if the system crashed during a crank or missed one) and the following event call will be made:

call crank_ctl_(msgptr);

The crank ctl_ procedure will create a process by a call to cpg_, similar to the way absentee does. A special entry point may be necessary in cpg_.

The crank process will then be started. Care must be taken to count the CPU time spent in the crank process as "initializer" time, or better, to keep a separate total, so that the "discrepancy" error message doesn't come up.

Termination of the crank process will be signalled to crank_ctl_ over the same event channel, and will cause the "crank_running" flag to be reset and the "time_next_crank" to be incremented and the next wakeup requested.

There should be a shared segment which both the initializer and the crank use to communicate. Some of the items in this database will be:

a) items sent by the initializer to the crank, such as notice that the crank is being reinvoked after a crash or is late.

b) responses by the crank such as error indications.

c) summary statistics calculated by the crank.

The crank process

The crank should execute in the user ring as an unprivileged user. No special response treatment should be given it, although the initializer should wait for it to finish if a normal shutdown is attempted. If multiple periods are required (e. g. daily, weekly, and monthly) the "exec_com" file should use "if" to do simple conditionals.

Here is a list of some things that the crank should do:

daily

1. Check dollar balances and cut off overspent users.
2. Print account status report for user accounts.
3. Print system status report for administrators.
4. Accumulate spending from individual session record decks into account files and produce detailed journal records for archives.
5. Process request messages from group supervisors for quota changes, etc.
6. Perform disk usage accounting.
7. Send usage reports to supervisors.

weekly

1. Produce various usage reports and backup listings.
2. Perhaps backup of administrative data bases on tape.

monthly

1. Produce monthly bills.

2. Produce mailing labels.

3. Produce archival data as necessary (tape for SC4020?).

4. Produce monthly summary reports for administrators.

5. Clean up old files.

Other functions will, of course, be added as necessary.

Programs should be written to be "re-runnable" if they are used within the crank. Furthermore, if a program runs into a fatal error, it should do something to keep the tracks from being covered, and then leave a message for the initializer and terminate. The initializer will squawk so that operations can nofity the accounting-system maintainers.

Proposed change: AS

eleminate "communications"

Purpose

This segment is a problem to maintain and install. It is the only ALM component of the current answering service.

Description

Some of the items in "communications" belong in the header of "whotab", others in the SAT, etc.

Proposed change:

"trap" command on initializer

Purpose

To enable operations to send an emergency message to a user before he begins working.

Description

The operator types:

trap Name Proj -options-

or

trap ttyxxx -options-

or

trap -id ~~-~~console_id -options-

to set a trap on user ID, GIOC channel, or terminal ID, respectively.

options may be any of the following:

-nolog	forbid login. Normally a trapped user is able to log in.
-msg "text"	send the message given by "text" to the user when trapped.
-temp	cause the trap to be unset after it is sprung once.
-off or -reset	reset a set trap

Implementation

1. Add entrypoint "trap" to segment admin.

2. This entrypoint uses a segment called "trap_table" or some such, declared like so:

```
dcl 1 trap_table based aligned,
      2 uchain  fixed bin,
      2 cchain  fixed bin,
      2 ichain  fixed bin,
      2 fchain  fixed bin,
      2 arg(100),
        3 next fixed bin,
        3 user char (32),
        3 chan char (8),
        3 idcode char (8),
        3 text char (64),
        3 temp bit (1),
        3 nolog bit (1);
```

3. Make utility routine

```
search_trap_tab_ (i, p, . . . .);
```

which searches the table starting at "i" and checks the terminal located by "p" against each entry. If a match is found, sends message to operator, to user if requested, etc.

4. Change dialup_ to call search_trap_tab_ for both "cchain" and "ichain".

5. Change lg_ctl_ to call search_trap_tab_ on "uchain".

Proposed change:

Add tape control to initializer.

Purpose

There is currently no accounting for tape usage, and no control over how many tapes are up. The operator has no way to reply negatively to a "mount" request.

Description

When the system comes up, the initializer will attach all tapes. When a process needs a tape, it will send a message to the initializer, using a message segment. The initializer will request the operator to locate the tape and mount it. When the operator has completed the mount, he will type a command to the initializer. The initializer will then force the attachment of the tape to the user and send him a wakeup.

Detachment, runaway tape, automatic logout, etc. are easily handled.

These changes can be made completely invisible to user-ring programs which call the tape DIM.

Implementation

1. Changes to initializer

   a) in "as_" at startup read a list of tapes

   b) new module, "tape_ctl_" establishes event channel and responds to signals; does accounting

   c) changes to "admin" to allow tape commands

2. Changes to the ring-4 tape dim in the attach and detach area. System-wide data base which has the event channel id.

3. Possible later changes to simplify the ring zero tape stuff and the ioam, which now think tapes must be assigned in ring 0.

4. Possible later extensions:

   a) a "tape_who"

   b) reserver

Proposed Change

Reassignment of machine-room consoles.

Purpose

Currently four consoles are required in order to run Multics:

- initializer
- IO Daemon
- Translator
- Backup

Retrieval, complete dump, or other operations activity require a fifth. All of these consoles must be in a secure area while running, since highly privileged operations are possible from any of these user identites.

Starting up the system also involves the typing of several "login" commands on the different consoles. Operations need to know the daemon passwords, which could lead to trouble if they were accidentally leaked to a malicious user who was also so speedy a typist that he could login before an operator.

Many of the messages typed in the machine room are not directed at the operator. Instead, they are intended for system programmers attempting recovery procedures or failure diagnosis. While this information is sometimes useful, perhaps we pay too much for it (or at least, maybe other installations won't want it).

This proposal suggests that the "daemon" processes be recoded to run without a console optionally, and that they be automatically logged in by the initializer. This will cut the number of terminals needed for Multics from 4 - 5 down to 1 - 2, provide quicker startup, and still allow the use of more terminals if desired.

Description

The Multics I/O switch provides most of the necessary flexibility to accomplish this proposal. Basically, what would be done for each daemon would be to attach the stream normally connected to the console to a special program, capable of using the message segment primitives to communicate with the initializer.

This special DIM would be able to use a console if one were allocated to it by the initializer, so that, for example, an operator could command the initializer to hook a console onto the backup daemon if an error message was received.

Messages written on the daemon's stream "error_output" would always be sent to the initializer, and the daemon would also be given a subroutine for non-error communication with the initializer.

We can handle input requests in one of two ways. One is to program the daemons so that all input requests are errors, and then always request input through a program which signals and gets input from the initializer. The other way is to keep a mode switch which says when an error has occurred, and take input from a file or from the message segment depending on mode. The second method allows conversion of almost any function to daemon operation, if we can replace "com_err_" etc. The first one leads to tighter code (do we care?) and maybe easier operation.

Initializer Changes

Adding the module "daemon_ctl_" to the initializer, and having it request the automatic login of the daemons, is pretty easy.

Giving the initializer some extra terminals is almost as easy. A new configuration-deck card or a modification to the current deck may be required. I suggest a new card, viz:

OPTTY TTY192 TTY194 TTY196 to tell the initializer about hard-wired terminals.

Hard-wired consoles can then be given automatically to daemons, etc. Other terminals can be surrendered to the initializer by some variant of the "dial" command. An operator command of the form--

```
give IO.SysDaemon TTY196
```

could then give daemon a console. Terminals not hard-wired to the GIOC should be handled by an adaptation of the "dial" command. Attaching a terminal to the initializer would require two steps:

1. A "dial" command executed on the terminal
2. A "give" command executed by the operator on the initializer console

Further Thoughts

Even fancier I/O attachment graphs could be constructed to handle the problem of too much output. Suppose, for example, that the initializer had a streamname "login_messages" which was used to write the login and logout messages on the initializer console. If a video terminal were available, it could be attached to the initializer and the "login_messages" stream put out over it, leaving the main initializer console for error messages and operator input.

A similar trick could be done for the daemons.

Proposed Change

"dial" command

Purpose

Some applications arise where a single process wants to use multiple terminals. CTSS and CP provide such a facility, which has been used for game-playing, for simulating multiple-access language systems, and for checkout of systems. A rudimentary facility of this type now exists in Multics, and this proposal describes the fancying-up necessary to make the facility releasable to users.

Description

The user who wishes to operate more than one terminal should be able to set this up with the answering service by issuing a subroutine call. Connecting an additional terminal to a process which has made this preparation should involve nothing more than the substitution of a "dial" command instead of login.

The current implementation supports both of these functions. The following deficiencies exist, though:

1. The only acceptable form for the dial command is:

   dial Name Project

   that is, no abbreviations or defaults apply. Ideally, the argument of "dial" should specify a conventional name not necessarily any person's; e.g., "dial housing_service". This suggestion leads to some sort of registry file for "dial names" and more software.

2. A user does not pay for dialed consoles, currently.

3. The process with multiple terminals has a problem with QUITs. Either it does not enable quit for a terminal - in which case it can't detect them - or it does - and then if

it gets the QUIT IPS, it can't tell which console hit it.

4. With the current pinch on phone-line access to Multics, allowing dial consoles would increase the probability of busy signals. This suggests that we may need some administrative control beyond the current all-or-nothing switch, to limit the number of consoles per process and/ or the total number.

Proposed Change

Dynamic load leveling

Purpose

The Multics system's load bears little relation to the number of logged-in users. Thirty system programmers may need more resources per minute than fifty students, and the same programmer can represent a very different load at different times of day. This proposal is for a set of changes to the system control facility to allow it to regulate Multics access depending on some function of the system's scheduling parameters.

Description

Whenever an "accounting update" is performed, the initializer determines some figures about the system load. Currently, it just notes the values, and doesn't take any action.

If a new module called "load_ctl_" were made up and called by accounting updates, it could operate like the CTSS load-leveler. Specifically, it could vary the "maxunits" parameter of the system within preset limits to attempt to forestall overload.

The CTSS load leveler, when properly adjusted, never had to "kill" any user. The proposed module will not even have this feature, so we will not have to introduce the bad P.R. aspects of bumping for overload. This makes extra sense when the number of users is large.

The general strategy is to check, every 15 minutes, and whenever anyone logs out, for overload due to any of several conditions. The first four suggested are:

a. Queue length
b. Response
c. Multiprogramming idle
d. Thrashing index

For each of these, overload will be defined to be "having a value above some threshold, and having been above the threshold last time we looked too." If an overload is detected, then the "maxunits" will be set to:

max(min(maxunits, cur_units), floor)

if the system is not currently full, to stop further logins. If the system is already full, "maxunits" will be allowed to decrease below "cur_units" to the value "floor", by steps of 1.0.

When the overload eases, "maxunits" will be set to 2.0 more than "cur_units" and allowed to increase to some maximum.

Reconfiguration can change system response so drastically that we might consider having a method of reconfiguring which had a delay (at least for deleting equipment). Thus, the operator might tell the system to drop a CPU; the system would be able to stop accepting logins for several minutes before the module was dropped.

The parameters we can vary are:

1. threshold value for each monitored quantity
2. floor on "maxunits"
3. ceiling on "maxunits"
4. "value" of configuration elements, in case we remove them

User weight in "load units" would be considered only at login time. Perhaps we could keep an exponentially smoothed estimate of a user's weight from one login to the next, but certainly the initial version of "load_ctl_" can live with the current method.

In order to start the system off at a stable point, experiments should be conducted in which the "load_ctl_" module just types out (and logs) messages describing what it would have done.

Proposed Change

Edit-only login responder

Description

When Multics is "full", users are currently turned away. This prevents the user from working at all, something like locking users out of the keypunch room when the batch queue is long. In some cases, we turn away system use which we could support (sale of terminal time) because we are short on some other resource (CPU). This proposal is for a "restricted" class of user who may only edit files. Users may be permitted to use this service even when the system is "full" of programmers.

A super-limited-service overseer which locked the user into "edm" is very easy to write. When Voydock's new editor is written we can use it instead. In any case, "load_ctl_" can be modified to have a number of "slots" for edit-only users. When a user would normally get the "sorry, system full" message, we can ask --

"System temporarily full. Do you wish edit-only service?"

instead, and allow the user to login with a special initial procedure if so.

The login responder would set-up some event channel such that when the initializer detected room for the user as a full user, the user could be allowed to do a new_proc. This will require some table giving the "deferred weight" of each edit-only user.

The responder would probably skip the message of the day and just give a "ready" message. The user could then type any of:

edm
program_interrupt
logout
addname

```
deletename
delete
rename
start
print
list
calc
help
who
```

(I am a bit worried about "list" consuming lots of resources. One could quibble with the command table endlessly, though. The point is that there would be one.)

Proposed Change

Cutoff users on projects with overspent accounts

Purpose

Users on projects which have exceeded their usage limits should not be allowed to login. This change will be in effect only until the proposed change to the accounting system has been implemented.

Design

The system administration table (sat) contains an entry for each project on the system. A previously unused character in each entry will be set to indicate the status of the project's account and the action to be taken if a user tries to login on the project. (see Appendix A). When a user tries to login this code will be inspected and the appropriate action will be taken.

In order to keep the status of the cutoff indicator up to date, system usage will be billed daily. The daily billing procedure will be identical to the monthly except that no individual bills will be written and the usage figures will not be reset in any of the data bases: hist, disk_stat, projfile, reqfile, and misc. The billing program will also generate each project's cutoff code in a copy of the sat. Therefore, the sat must be installed after each daily billing.

Implementation

Programs to be modified

atime

Usage information will only be collected from monthly accounting cards which were not previously billed and which have a completed session. A flag will be set in the accounting card to indicate the completion of the usage collection.

In case of crashes which leave hist and accounting deck out of step:

1. If a crash occurs during the accumulation of accounting cards, the hist will be deleted and a copy of it will be used for rerunning the billing procedure. In order to start accumulation at the same accounting card as during the crash any card which has the collected flag on with a logout time of the previous day will be collected again.

2. A crash which leaves the hist intact but uses the collected flag in an accounting card will be recognized if the date the hist was written is a day or more after the card's logout time. The procedure to be taken then will be to restart the billing with the previous day's hist.

wb

This program will be copied, renamed daily_billing and modified as follows:

1. The summary file only will be produced
2. The usage figures for each project will be compared to the requisition amount and the appropriate cutoff code inserted in the sat
3. The termination date of a project will be inspected and the appropriate cutoff code will be inserted in the sat

act_ctl_$check

This is currently a dummy entry called by lg_ctl_$login before allowing a user to login. It will be changed to locate the user's project in the sat and print an error message if its cutoff code is non-blank. It will return a non-zero code if user is not allowed to login.

lg_ctl_$login

The message printed when act_ctl_$check returns a non-zero error code will be modified.

installation_data

A new segment will be generated to contain installation dependent messages. act_ctl_$check will obtain its messages from this segment.

Appendix A

The cutoff code in the sat will be one of the following:

Code	Meaning
X	Project is out of funds and not allowed to login
Y.	Project is out of funds but allowed to login
T	Project has past termination date and not allowed to login
S	Project has past termination date but allowed to login
W	Project is within x dollars of limit
R	Project is within x days of termination date
- (blank)	Ok