PUBLISHED: 6/7/67
Minor Revision
Supersedes BG.6.00
6/6/66

Identification

Core Control
Peter G. Neumann and Mary R. Wagner

Purpose

Core control is the module which receives requests to assign and unassign core. It acts in accordance with the present occupancy of core and with the nature of the request. It maintains information about the use of core in the core map (Section BG.5).

Introduction

Core control maintains information on the availability of each (64-word) block of core and how this block is being used. This information is kept in the core map, described in Section BG.5. (It is assumed that the reader has read Section BG.5 - particularly the introduction - since definitions of the basic concepts of core management are given there.)

Core control is called to assign groups (contiguous core locations which are treated as single entities). An assigned group is of one of three types: it may contain a hyperpage (contiguous pages in contiguous core locations), a page table, or an entire unpaged segment. In general, all groups are requested by page control (Section BG.4). The only other modules to call core control are the initializer and the reconfigurer.

Section BG.6.00 contains the primitives of core control, an overview of the design and a sketch of the flow of control. The detailed design is described in Sections BG.6.01 through BG.6.06.

Pages and hyperpages

(The contents of this and the next paragraph should ultimately appear in Section BG.4, if at all. They are provided here only for background.) The hardware of the GE 645 provides pages of * two sizes, hereafter referred to as small pages and large pages, whose sizes are 64 words and 1024 words, respectively. There are arguments in favor of the use of each page size. The small pages provide greater variety as to what may be in core at any one time than do large pages. The large pages on the other hand are required in order to page segments of size larger than 16K (K=1024), since the hardware permits a paged segment to have at most 256 pages. In addition, large pages may tend to reduce the amount of hardware address computation by better utilizing the

available associative memory, although this is decidedly a second-order effect.

In Multics, the hyperpage (and not the page) is the unit of information which is transported (i.e., removed from core onto secondary or restored to core from secondary). The hyperpage size of a given segment may greatly influence the efficiency of the system with respect to transportation overhead, and its choice is far more critical than the page size. On one hand, it is clear that the transportation of small hyperpages (e.g., 64 words) may result in a large overhead because of the large number of requests for the transportation of hyperpages (e.g., missing-page faults and page removals). On the other hand, the transportation of large hyperpages may result in a large overhead because of the high volume of information transported, much of it unnecessarily). A compromise is obviously in order, although it must be able to vary with the circumstances. Thus the hyperpage size is chosen on a per-segment basis by the file system, and is essentially independent of the page size.

Primitives

The basic primitives of core control are assign, unassign, groupstat and poolsize. They are all privileged to segment control and page control. (Certain other primitives required only for initialization and reconfiguration are omitted here.) These primitives have the following basic purposes.

1. assign: Assign a group of specified size, type, status, etc., and return the absolute location of the assigned group to the caller.

2. unassign: Unassign the group with the specified location.

3. groupstat: Change either or both switches constituting the status of the group with specified location. The original status is returned to the caller.

4. poolsize: Change the size of the specified core pool.

The detailed structure of the arguments and a further description of each primitive is given below.

assign

```
loc=assign(atype,size,status,pagsiz,pageno,pool,
        descr,proc,sstptr,thresholdno,retopt,nsrtn,code);
```

The primitive to assign a group has the following arguments.

```
dcl loc bit (18),
    atype bit (2),
    size fixed bin (17),
```

Blank page on original

unassign

```
call unassign(loc,ifstat,nsrtn,code);
```

The primitive to unassign an assigned group with given location has the following arguments.

```
dcl loc bit (18),
    ifstat bit (2),
    nsrtn label,
    code fixed bin (17);
```

ifstat consists of the ifwired switch and the iflatched switch, from left to right. A group is unassigned if and only if each of these two switches is at least as large numerically as the corresponding wired or latchd switch in the core map. That is, unassign ifstat=11 means unassign even if the group is wired and latched; unassign ifstat=00 means unassign only if the group is unwired and unlatched; similarly, unassign ifstat=10, for example, means unassign even if the group is wired but then only if it is unlatched. ifstat=11 thus acts as an unconditional unassign. If the group cannot be unassigned because of a conflict in its status, a nonstandard return is given. The cases covered by code for the nonstandard return to nsrtn are as follows.

1. group wired
2. group latched
3. group not assigned
4. no group begins at this location
5. location not known

groupstat

```
oldstat=groupstat(loc,changestat,newstat,sstptr,proc,
            nsrtn, code)
```

The primitive to change the status of a group with given location has the following arguments.

```
dcl oldstat bit (2),
    loc bit (18),
    changestat bit (2),
    newstat bit (2),
    sstptr ptr,
    proc bit (1),
    nsrtn label,
    code fixed bin (17);
```

changestat consists of two bits, from left to right changew and changel. Similarly, newstat consists of two bits, from left to right neww and newl. For each '1' in changestat, the corresponding bit in the status is set tp the corresponding bit

in newstat. If changew=1, the wired switch in the group map entry for the group at location loc is set to the value given by neww. If changew=0, the wired switch remains unchanged. If changel=1, the latched switch in the group map entry for the group is set to the value given by newl. If changel=0, the latched switch remains unchanged. The original value of the status before the call is returned as oldstat, again in the order wired, latched. Note that the present status may be obtained by using the groupstat call with changestat=00. If sstptr is null, both it and proc are ignored. Otherwise, the values of sstptr and proc replace the values of the corresponding core map variables. The cases covered by code for the nonstandard return to nsrtn are as follows.

1. group not assigned
2. no group begins at this location
3. location not known
4. change of status not permitted

poolsize

```
call poolsize(poolno,psize,nsrtn,code);
```

This primitive has not yet been specified.

Lists and pools

Various lists are threaded through the group map by means of the pointer in each group map entry. There is a list of all free groups, the totality of these free groups forming the free pool. Each free group is available for immediate reassignment. There is also at least one list of reassignable groups containing hyperpages, i.e., groups which are eligible for removal of their contents. Each such eligible list contains groups forming an eligible pool. To which eligible pool a reassignable hyperpage group belongs is specified by the item pool in the group map entry. Initially there is only one eligible pool, with pool=0. (This is sufficient for normal operation. Other eligible pools will be available when system resource management (Section BT) and page control get around to figuring out how and when to use them. At that point, a primitive to alter the maximum pool size will be added to core control. The multiplicity of eligible pools may be useful in implementing certain classes of "guaranteed" service.) Each group in an eligible pool is available in the sense that its contents may be removed by calling page control, then unassigned by a back call from page control when removal is complete, and then reassigned. The hyperpages in an eligible list are kept in order of decreasing eligibility (within each list) for removal. Each eligible list is periodically reordered by the ranker on the basis of the use bits in the appropriate page tables. Newly assigned groups are added to the end of the eligible list indicated by the argument pool in the assign call. When it is deemed desirable to remove the contents of some group on a given eligible list, the group at

the beginning of the list is chosen.

The modules and the flow of control

The six basic functional modules of core control are the cartologer, the allocator, the replenisher, the contiguator, the ranker and the parametizers. Their functions are as follows. The cartologer contains the entry points for all core control primitives, and governs the flow of control within core control. It is named for its map managing characteristics. The allocator attempts to satisfy requests for core space by assigning free groups when desirable or possible. The replenisher and the contiguator (collectively the removal modules) determine which groups should have their contents removed from core, whenever the need arises. The removal of the contents of a group is initiated by calling page control, which actually accomplishes the removal. In particular, the replenisher attempts to compensate for the loss from the free pool resulting from allocation from that pool. The contiguator, on the other hand, attempts to satisfy a given request which either cannot or may not be satisfied out of the free pool. It does so by a combination of using free groups and removing the contents of eligible groups.

Associated with the cartologer, the allocator and the contiguator are several submodules. These are the splitter, the coalescer, the waitlister and the waker. The splitter (functionally a part of the allocator) breaks up a free group when required, and returns the surplus to the free pool. The coalescer combines adjacent unassigned groups as desired. The waitlister maintains the wait list, which indicates which partially satisfied requests are waiting for completion. It sets up an event whereby the process can be reawakened, and calls the file system wait module (see Section BG.15.01). The waker scans the wait list after unassign and status calls to see if a partially satisfied request is now complete. If so, it calls the file system notify module (Section BG.15.01), which in turn causes all process waiting for the given event to be scheduled.

The remaining two modules of core control are logically independent of the above four. The ranker keeps the order of the eligible list for each reassignable hyperpage pool (that is the order of eligibility for removal) reasonably related to actual use. Finally, the parametizers observe the various parameters of core control and the ways in which these parameters influence the behavior of the system. The parametizers are capable of adjusting these parameters dynamically.

Throughout the design of core control, algorithm-dependent modules have been identified and isolated as much as possible. In this way it is possible to modify the algorithms or to replace an algorithm-dependent module with another module. The allocator, the splitter, the coalescer, the waitlister, the waker, the ranker and the parametizers are all such entities.

Similarly, the contiguator and the replenisher are also. Any one of these modules may be modified or replaced, as desired.

For the assign primitive, the flow of control is somewhat as follows. The cartologer receives the call, strips off the arguments, and invokes the allocator. The allocator determines whether or not it is desirable to satisfy the request by reassigning a free group or groups, based on various allocation parameters and on the priority of the request implicit in the threshold argument. If it is desirable, an attempt is made to obtain the necessary free group. The allocator attempts to find the smallest group of adequate size, splitting it if necessary, and returning the surplus (if any) to the free pool. If this attempt is successful and the replenish option of retopt is 1, the replenisher is invoked. The replenisher determines whether or not it is desirable to remove hyperpages in order to compensate for the loss of the newly assigned group from the free pool. It does so on the basis of the replenisher parameters, in particular, the roof and the floor, which are the desired upper and lower limits for the amount of free core. If it is desirable, the replenisher decides which groups' contents to remove. For each hyperpage to be removed, page control is called to remove its contents from core. Page control returns immediately, with the return specifying one of three situations. First, the group may be unassigned directly (made free), as for example in the case of pure procedure which need not be removed before unassignment. Second, the group's contents are going to be removed by page control (whence the group may have its type changed to evacuated); a back call from page control (unassign) will signal removal (at which time the group may be made free). Third, this group is not available for removal at this time (in which case the replenisher most likely will try another group). Whether or not the replenisher causes any hyperpages to be removed, control is then returned to the caller, along with the location of the assigned group.

If on the other hand the attempt to obtain the necessary free group fails, there are still several possibilities. First, if the block option of retopt is 1, the waitlister is called to enter this process into the waitlist, preparatory to a future blocking by page control. Next, if the contiguate option of retopt is 1, the contiguator is called. The contiguator first decides whether or not it is desirable to attempt to satisfy the request at this time. If so, the contiguator determines which groups' contents should be removed. It then makes an appropriate entry in the coalesce list, indicating the size and location of the desired group. The contiguator then successively calls page control to remove each desired hyperpage from core. Page control returns, specifying one of the three situations noted above. First, the group may be unassigned directly (made reserved). Second, the group's contents are going to be removed (whence the group may have its type changed to evicted); a back call (unassign) will signal removal (at which time the group may be made reserved). Or third, try elsewhere. Finally, return is

made, via nsrtn, with the error code "group is not immediately available".

Subsequent to each call to page control to remove the contents of a group is a later back call (unassign) from page control to core control, announcing the availability of that group. This call occurs within whichever process is running at the time, i.e., not necessarily within the originating process. If the group to be unassigned is of type evacuated, its type becomes free. If it is of type evicted, its type becomes reserved. On the other hand, an unassign call may originate from segment control or from page control (e.g., the unassigning of an unpaged segmment after its removal, or the unassigning of a page table after the unloading of a segment). In these cases the group becomes free immediately. In any event, the coalescer (running under the process in control) coalesces the unassigned group as desired. If the group is reserved, the coalesce list (established previously by the contiguator) indicates how this unassigned group fits in to a larger group being established. The group retains the type reserved while being coalesced with immediately neighboring groups, as indicated by the coalese list, until the coalesced group of desired size has finally been established. At this point the coalesced group in the reserved pool is added to the free pool. The purpose of the reserved pool is of course to be able to guarantee when necessary that small parts of the would-be large group do not get gobbled up before they can be coalesced into the desired group. Following transferral of the coalesced group to the free pool, the waker is invoked. The waker examines the waiting list of unsatisfied requests. If at least one process is waiting for a free group of the given size, or smaller, the cartologer calls the notify module (see Section BG.15.01). As a result, all processes waiting for that particular event type reschedule themselves. At a later time, when such a process comes unblocked (c. u.), it invokes the c. u. later allocator. Thus the scheduler or schedulers in effect determine the order in which processes are actually assigned groups which have in this way been unassigned. Note that there is no explicit correspondence between the group or groups which were removed by the original process and the group which is ultimately assigned to that process. Note further that although the coalesce list guarantees the way in which a group may be established out of unassigned groups (when this approach is desired), the existence of this list in no way affects which process may ultimately obtain the established group. Incidentally, unless the roof is suitably large, the coalesced group may be split by the allocator satisfying other requests before a process requiring that group can get to it. For this reason, temporary raising of the roof is desirable. (If the coalesced group has been split prematurely, the entire allocator-contiguator cycle is repeated.)

Each algorithmic module is given a facility whereby its parameters may be controlled external to the algorithm, wherever this is meaningful. Each such module thus has an associated

module which observes and adjusts the parameters of that module on the basis of current and past activity. These modules are called parametizers, and are themselves replaceable. They provide a simple means for experimenting with system performance without having to alter the modules themselves. The parametizers may be dummied initially, with all parameters remaining constant as a result. The parametizers are thus capable of growing with the system, as are the algorithmic modules themselves. Note that each existing parametizer must be invoked occasionally, even if the parameters indicate that the associated module need not be called. Otherwise, a parametizer could effectively tune its module out of existence. The parametizers may dynamically control the roof, the thresholds, the amount of core which may be latched at any one time (to prevent total latching), the coalesce strategy, and how often to call the ranker. How often to call the parametizers themselves must of course not be left entirely to the parametizers.

As an example, consider the parametizer for the allocator. The initially implemented version may return immediately to the caller without doing anything. That is, the actual values of the thresholds and any other allocation parameters remain constant. Subsequent versions might use various algorithms for adjusting the thresholds. The ultimate version of this parametizer is speculative at this time, and may provide considerable amusement to Multics-watchers. Similarly, the parametizer for the replenisher may adjust the roof.

The detailed specifications of the core control modules are contained in the following sections. These are organized as follows.

BG.6.01 - The cartologer
BG.6.02 - The allocator
BG.6.03 - The replenisher
BG.6.04 - The contiguator
BG.6.05 - The ranker
BG.6.06 - The parametizers