Resource Management and Task Scheduling

DRAFT 1

by
Walter A. Wallach

Introduction and Summary

This document presents the concepts of resource menagement and test scheduling as implemented in the R1000. Resource management is defined to be the control and apportionment of certain system elements, particularly CPU time, main memory and paging server bandwidth, to tasks. Scheduling is the activity by which tasks are given access to those managed resources.

1. Definitions

This section defines some terms and concepts upon which R1000 resource management is based.

1.1. Main Memory and Thrashing

Main memory in the R1000 is a very large set associative cache. Backing store for the cache is moving head disk. Cache faults are resolved either by creating a new logical page in an available cache frame, or by invoking software to locate a missing logical page on backing store and read it in. Management of the cache is implemented in microcode.

1.1.1. Available Cache Frames

Page frames are reclaimed by two mechanisms: tasks terminating and releasing their memory resources and aged pages being swapped out of the cache to backing store. The latter mechanism may be applied at page fault time by any faulting task, and explicitly by a page cleaner task. The page cleaner runs periodically to monitor the demand for cache frames and maintain a pool of clean frames available for allocation to new logical pages and receiving pages from backing store. When the page cleaner determines that the pool of available pages in the cache is not adequete, it causes the balance set to be reduced.

We require at least 25 percent of the frames on each cache line to be available (either clean or unallocated) at any time. If the page cleaner finds itself unable to maintain this level, the medium term scheduler is asked to reduce the balance set.

1.1.2. Backing Store Bandwidth

Page fault microcode monitors traffic to backing store, keeping track of the available backing store bandwidth in order to prevent oversubscribing backing store.

1.2. Tasks States

A tast may be either blocked or unblocked, and, at the same time, it may be in the balance set ot out of the balance set.

1.2.1. Blocked vs Unblocked

A task is blocked whenever it is unable to proceed until some external event occurs. A task is unblocked whenever it is able to proceed whenever it is given the resources it needs.

A task may be blocked awaiting disk I/O (DISK_WAIT), awaiting memory (PAGE_CLEAN), executing a delay (_DELAY), or awaiting an entry call (in an accept) or executing an entry call (_RENDEZVOUS_WAIT).

1.2.2. In or Out of the Balance Set

A task is in the balance set whenever it holds resources. In particular, we say it is in the balance set when its TCB is in memory. In all liklihood, such a task has other pages of its stacks in memory as well. The key idea here is that the tasks cache pages cannot be reclaimed without first cleaning them by copying them to backing store.

A task is out of the balance set when its TCB is not in memory, or, its TCB is in memory, clean and not threaded on any memory-resident queue (a page is clean if there is a copy of it on backing store, so the memory can be reclaimed without first copying the page to backing store).

A task is said to be *ready* if it is unblocked and in the balance set. All tasks on a ready queue are ready.

Normally, blocked tasks will remain in the balance set until their memory resources are required by other tasks. Then, their pages will migrate out to backing store as they age. When a task becomes unblocked, it will be made ready only if the system has the resources to support that task's entering the balance set.

Modules which are in the balance set are under the care of the memory resident resource managers: the short term scheduler and the memory manager. Tasks out of the balance set are in the care of the medium term scheduler and the delay manager.

The schedulers are discussed in the next section. The delay manager is not discussed explicitly, but may be subsumed by the MTS.

2. Short and Medium Term Scheduling

Scheduling occurs at two levels: *short term* dispatching of the processor and *medium term* scheduling. The short term scheduler is implemented in microcode and may run within a task, or on behalf of the system when no task is running on the processor. Periodically, the short term scheduler chooses the highest priority task among those ready to run, and gives that task the processor.

The medium term scheduler is a permanent system task implemented in software. It manages all unblocked tasks that are not in the balance set, and manages multiplexing tasks between its medium term scheduling queue and ready queues.

2.1. Task State Transitions

From the resource management point of view, a task's state is reflected in which queue, if any, he is on, and who can name him. A task that is blocked must be namable by an unblocked task before the task can again run. For example, if a task is blocked in an accept, it will remain blocked until another task executes an entry call to it. If no task can execute such an entry call, the task remains blocked forever.

Tasks which are blocked naturally are namable by the events which will unblock them. Tasks which have executed delays or are blocked on an accept or in an entry are examples of such tasks. Tasks which are become blocked because of failure to aquire a resource must be blocked until that resource becomes available. Such tasks are blocked unnaturally, and they must be remembered by the resource manager until the resource becomes available. These tasks are remembered on queues by the short term scheduler and the memory manager (tasks in the balance set) and by the medium term scheduler (tasks out of the balance set). Unnaturally blocked tasks include those awaiting disk I/O, awaiting memory and awaiting room in the balance set (swapped out).

A naturally blocked task can be removed from the balance at any time. When the task becomes unblocked, it will be reintroduced into the balance set, if there is room, or else it will be unnaturally blocked and placed in the medium term scheduler's care until it can be put back into the balance set. An unnaturally blocked task cannot be removed from the balance set without first removing it from its queue. In certain cases, it must be put on a similar queue for tasks out of the balance set (tasks executing delays).

2.2. Short Term Scheduling

Short term scheduling policy is last in first out preemptive resume (LIFO-PR), meaning that, when a task of the same or higher priority than the one currently running becomes ready, the current task is preempted and placed at the front of his ready queue, while the new task is given the processor. When the currently running task becomes unable to continue (e.g., because it has page faulted), it is removed from the processor and the last task to be put in the highest priority ready queue is given to processor.

On top of the LIFO-PR policy is a FIFO time slicing policy, which insures fairness. When a task has accumulated a certain amount of CPU time, it is removed from the front of the ready queue and placed at the back of the queue. Such a task will not gain access to the

Module State Transitions

```
       SHORT_TERM_SCHEDULER        ||  MEDIUM_TERM_SCHEDULER
          (balance set)            ||       (out of balance set)
                                   ||
         +------------------------ || <---------------------------+
         |                         ||                             |
        \|/                        ||                             |
+-->  RUNNABLE  -----------------> || -----------> SWAPPED  ----->|
|        |                   /|\   ||                             |
|       \|/                   |    ||                             |
|<--  RUNNING  -------+       |    ||                             |
|       ***           |       |    ||                             |
|        |            |       |    ||                             |
|        +----------- | ----->|    ||                             |
|                     |       |    ||                             |
|<--  DISK_WAIT  <----+       |    ||                             |
|         |           |       |    ||                             |
|         +---------- | ----->|    ||                             |
|                     |       |    ||                             |
|<--  PAGE_CLEAN <----+       |    ||                             |
|       _WAIT         |       |    ||                             |
|         |           |       |    ||                             |
|         +---------- | ------+    ||                             |
|                     |            ||                             |
|<--  SHORT_DELAY  <--+            ||        +---> LONG_DELAY  -->|
|          |          |            ||        |                    |
|          +--------- | ---------> || -------+                    |
|                     |            ||                             |
+---  SHORT        <--+            ||              LONG           |
        _RENDEZVOUS                ||                _RENDEZVOUS  |
 _WAIT    ----------------> || ------------>  _WAIT    ----+
         ***                       ||                 ***
```

Notes: (1) *** means that we do not explicitly maintain a set of tasks in this state. (e.g., no queue of such tasks).

processor until it has migrated to the front of his ready queue, either because other tasks become blocked, or are demoted according to the time slicing policy.

2.3. Medium Term Scheduling

Often, the aggregate resource requirements of all unblocked tasks exceed the resources available. In such circumstances, a subset of the unblocked tasks are readied, and the remainder are maintained in a medium term scheduling queue. The medium term scheduler periodically removes some of the ready tasks from the balance set to make room for tasks in the medium term scheduling queue.

2.4. Reducing the Balance Set

When system resource managers determine that a system resouce is oversubscribed, the medium term scheduler is invoked and asked to reduce the balance set.

The MTS then scans the ready queues and disk wait queues, choosing tasks to be removed from the balance set. These tasks are removed from their queues, entered into the medium term scheduling queue, and their TCBs are unwired. Their memory resources are available for reclaimation (we may choose to clean all dirty task pages at this time).

If the short term scheduler determines that there are insufficient resouces to run an unblocked task, the task is given to the medium term scheduler, and entered into the medium term scheduling queue. The MTS will run the task when sufficient resources become available.

2.4.1. System Shutdown

At system shutdown time, the MTS removes all permanent tasks from the balance set and places them in the medium term scheduling queue (tasks are removed from both the ready queues and the disk wait queue). All termorary tasks are purged from the medium term scheduling queue. Since the MTS is a permanent task, its queue persists until the next R1000 session.

2.4.2. Non-Paging I/O

Permanent tasks cannot perform I/O themselves. Permanent tasks cannot execute entry calls on termorary tasks. Therefore, at crash or shutdown time, we can ignore all outstanding I/O activity (tasks in the disk wait queue will fault their required pages into the cache when the system comes back up).

3. Short Term Scheduler

The short term scheduler manages the ready queues. When a task makes an entry call, the short term scheduler must quickly decide if the target task's TCB is still in the the cache and, if so, put is on a ready queue. If the target task is not still in the cache, the target task's TCB must be faulted in from backing store, and the short term scheduler must decide is there are sufficient resources available for entering the task into the balance set. If there are not, either the medium term scheduler must reduce the balance set, or the target task must be given to the medium term scheduler and run at a later time.

Normally, we expect tasks to remain in the cache for quite some time, unless explicitly swapped out to backing store. When executing an entry call, the calling task is suspended in favor of the accepting task. Therefore, the only real decision the short term scheduler must make is whether the target task's TCB can be wired into the cache.

When the target task exits the accept, both the caller and the target task are able to run. If the caller has been removed from the balance set, the short term scheduler must decide if there are sufficient resources to put him back into the balance set. Normally, we expect the caller to remain in the cache, and be moved back to the ready queue.

The most important decision made by the short term scheduler is whether to schedule a newly created module. When a module declares a package or task, a new TCB is allocated such that the TCB can be introduced into the balance set. If there are insufficient resources to support the new task (which will be in a phase transition and will generate a high demand for memory), the declaring task is suspended until resources are available for creating its child. Note that, in this situation, the declaring module is still unblocked. It can be suspended most easily by simulating a delay within the task, thereby blocking it.

The page replacement policy gives memory priority to TCBs of blocked tasks (they will not be swapped out unless no other page frame can be reclaimed without hurting the system). When a TCB is swapped out, all of that modules stack pages may be cleaned (this may be done more efficiently as a single operation rather than faulting each page out. It makes a larger number of cache frames available for reuse, and makes swapping the task back into the cache more efficient).

We may choose not to wire TCBs of non-system tasks, letting the page replacement policy (implemented in microcode) decide if a TCB must be swapped out. The page replacement policy would not reclaim a TCB unless no other page on that cache line could be used. If a TCB was reclaimed, it would be removed from its ready queue and all of its stack pages cleaner (as described above).

System tasks are always wired.

4. Medium Term Scheduler

The medium term scheduler is implemented as a permanent task. Its name is stored in the pack label on the system disk. At system initialization time, the MTS is recovered from the file system, and it chooses the tasks to be placed in the ready queues.

4.1. Balance Set Management

The MTS provides two primary entries: one for placing specific tasks into its queue, and the second to reduce the balance set.

When the microcode discovers that an unblocked task cannot be run (e.g., because there is not sufficient memory), or that a TCB must be cleaned and its page frame reclaimed,it calls this MTS entry, passing the task's name. After the task's name has been entered onto the MTS' queue space, the task may be removed from his resident queue.

When any resource manager discovers that the balance set is too large, the second MTS entry is invoked. This causes the MTS to scan the ready queues and disk wait queues, choosing tasks to be removed from the balance set and entering them into the MTS queue. The MTS may or may not call the file system to clean the chosen tasks' stack pages.

The MTS is organized as a pair of tasks: the MTS proper is a permanent task maintaining the MTS queue, and a temporary task, which interfaces the MTS with the resident queues. When the balance set is to be reduced, the temporary task is invoked to scan the resident queues and choose tasks to remove. Each chosen task is passed to the MTS' queue space, then its TCB is removed from its queue and its stack pages are cleaned.

4.2. Replacing the MTS

The medium term scheduler provides an interface for reading all tasks in its medium term scheduling queue. If a new MTS is replacing the current one, the microcode is informed not to call the MTS. The new MTS calls the old, requesting the contents of its queue. When the new MTS' queue has been built, it places itself in the pack label of the system disk, aborts the old MTS, and gives its name to microcode. The system then proceeds.

4.3. The Medium Term Scheduling Queue

The medium term scheduling queue is the finite sized, wired data stack of the MTS. If this queue is not large enough to hold all unblocked tasks, they must be (e.g.) delayed until there is room in the MT queue. Alternatively, the MTS may be an unwired task; this requires the microcode to wait until the MTS resolves page faults before completing balance set adjustments.

5. Page Replacement Policy

The page allocation and page cleaner microcode implement the page replacement policy. The page allocator is called within the requesting task whenever a cache frame is required for a logical page. This will occur when a stack is being extended and when a frame is required for loading a page from backing store.

5.1. Page Allocator

If the page allocator can find an available frame within the least recently used quarter of the cache line, it allocates this frame to the page. If there is no such frame, the page cleaner microcode is invoked within the requesting task. Page cleaner microcode is also invoked from the page cleaner task. [An available frame is a page frame containing an invalid page, a page thats not within anyone's stack, or a clean page. Any clean page must also be cataloged, and may be replaced without losing any data.]

5.2. Page Cleaner

Page cleaning is performed by the page cleaner microcode. This microcide is called from within a task when the page allocator fails to find an available page frame, and is called periodically from the page cleaner task to maintain the proper pool of available pages.

The page cleaner microcode locates a likely victim page to clean (copy to disk), and asks the file system to store the page. The store request may require several backing store operations before the missing page has been located. Each page that is read from backing store (for file system data bases) requires a frame to be allocated (which could result in more page cleaning). The page cleaner is implemented such that, if one cleaning operation requires another, the first is abandoned and the other is performed. Normally, the file system will have the required data base elements in the cache to complete the store operation.

The page cleaner task maintains information about the current demand for memory. If it cannot maintain the required number of free cache frames, it may run more often (providing that backing store bandwidth is available). It may also ask the MTS to reduce the balance set.

5.3. Backing Store Bandwidth

The page fault microcode and the disk controller tasks maintain metrics reflecting the portion of backing store bandwidth being used (and therefore, the amount of bandwidth in reserve). This metric is available to the short and medium term schedulers for making decisions about entering new tasks into the balance set. The algorithm is still to be determined. However, it must be able to predict the paging traffic caused by each task in the system, since this is one metric used in balance set management.

6. Load Control

It is difficult to identify any notion of working set, given the amorphous groups of cooperating tasks we envision to be running on the R1000. It is also critical that memory and backing store bandwidth not be oversubscribed.

6.1. Oversubscribed Memory

Memory may become over subscribed in two ways, because too many requests are being made to a single cache line (called *thrashing on a cache line*), or an insufficient pool of available pages exists throughout the cache. The former requires that selected tasks be removed from the balance set, probably be choosing a page on the line and finding its TCB, or choosing a TCB within the oversubscribed line. The latter requires that the MTS scan the ready queues. Care must be taken that, when a task faults a page into the cache, it is not stolen before the task can run and reference the page.

6.2. Oversubscribed Backing Store

The paging behavior of the task mix may oversubscribe backing store bandwidth without oversubscribing memory. This may occur is too many tasks are in phase transitions (i.e., are changing their resident set of pages). A newly created task is an example of this. Tasks in phase transitions consume page frames faster than tasks between transitions. This high consumption increases backing store bandwidth used for page cleaning and reduces the available page pool.

The short term scheduler will not introduce a new task into the system unless the system can support another task in a transition. A rule of thumb is that one phase transitioning task per backing store device can be supported. Transitioning tasks are not associated with any particular device, rather, a system with four backing store devices and support the aggregate backing store traffic associated with four tasks in phase transitions. We need not be this strict, as long as a proper available page pool can be maintained (we want the system optimized for task creation as well as rendezvous). Choice of the proper load control parameters requires experimentation with a running R1000.

6.3. Deadlock Control

Cleaning of cache frames requires that portions of the file system data bases be paged in as needed. In the limiting case, oversubscription of memory may prevent paging in such a portions, leading to a deadlock. The system attempts to prevent oversubscribing memory by monitoring availability and demand for memory. This section describes policies and mechanisms which deal with potential deadlocks.

6.3.1. Emergency Page Buffers

The Address Space Manager maintains enough wired page buffers to hold the maximum number of file system data base pages required to clean any page (the maximum depth of any logical to disk address translation is five levels, requiring, at most, five pages of emergency buffering). Normally, the ASM will read pages of its data base into memory, to be accessed associatively (like any other page in the system). When the page cleaner task makes a STORE request which fails because of no memory, it can call an entry to the ASM which will use the emergency buffers. When this entry is called, the ASM becomes single threaded; no other ASM requests are honored until the one using the emergency buffers completes, including waiting for disk requests to complete.

Single threading requests obviates the need to lock the emergency buffers. The page cleaner task is the only task able to call the emergency STORE entry to the ASM, and he calls it only in situations which appraoch deadlock.

6.3.2. The Page Cleaner Task Queue

The page cleaner queue is a list of all tasks in the system blocked awaiting available memory. We expect this queue to be empty as long as memory is not oversubscribed (since the page cleaner is maintaining the minimum number of available frames on each cache line).

The page cleaner task runs periodically, scanning all lines of the cache to make sure the required number of available pages is maintained. If he finds that he can clean no pages without receiving a page create barrier from the ASM, and either the number of available pages is not adequete or there are tasks on the page cleaner queue, he calls the emergency STORE entry to the ASM. This call effectively single threads the virtual memory system until the necessary number of available pages of memory has been restored.

Single threading the virtual memory system serves to throttle demand for memory, as well as simplifying the mechanisms required for breaking the insufficient memory deadlock. While the ASM is single threaded, requests may accumulate. Therefore, if memory management microcode cannot rendezvous with the ASM immediately, it executes a delay on behalf of the requesting task.

6.3.3. Page Allocate-time Page Cleaning

The page allocator will attempt to clean a page, if no page is available on the required cache line. If the ASM returns a create barrier, the page cleaning microcode puts the requesting task onto the page cleaner task's queue, blocking that task until the page cleaner can increase the number of available pages in the system. The requesting task is unnaturally blocked, and may be removed from the balance set as if it were a ready task.

6.3.4. Barriers to the Medium Term Scheduler

The medium term scheduler is an unwired task. Should he encounter a create barrier while cleaning a page, he is placed on the page cleaner queue. By using the emergency STORE entry and associated buffers, we should never enter a deadlock as a result of oversubscribed memory.

6.4. Task Unblocking

When a task decides to unblock another task (e.g., the task exits an accept), microcode locates the blocked task's TCB, and changes its state. The unblocking task requests the short term scheduler to ready the blocked task.

The short term scheduler decides if resources permit running the blocked task. [This probably hinges of whether the blocked task was already in the balance set. If so, then we assume resources exist to run that task.]

If the blocked task's TCB is not in memory, the unblocking task's attempt to change the TCB's state will fault it into the cache. This happens transparent to the unblocking task. Once faulted into memory, the blocked task can be unblocked and readied.

When cleaning a TCB, we may decide to clean all resident stack pages for that task as well. When someone tries to restore that task to the balance set by referencing its TCB and adding it to the ready queue, we may swap the entire task back into memory by reading all stack pages that were cleaned along with the TCB. This reduces the loading transient associated with running the task and uses backing store bandwidth more efficiently. To facilitate this, we want a short term scheduler call to unblock a task, rather than embedding that operation in instruction microcode.

Table of Contents