

Systems Qualifying Exam 2004

There are 8 questions on this exam from which you are to choose 5. If you attempt more than 5 questions, you must tell us which to consider in grading or else we will grade the first 5 (numerically) that you attempt. All questions count 20%.

1. Synchronization

You are to develop synchronization primitives for a threaded OS which has a time sliced scheduler. A thread has (at least) the following states:

`Running, Ready, Waiting`

The currently running thread is

`Thread *current`

and its state is

`current->state`

Preemption is implemented by an interrupt routine which calls `sched()` if and only if

`current->state == Running`

The procedure `sched()` yields the cpu by

- If `current->state == Running` move `current` to the ready list and set its state `Ready`.
- Move a thread from the ready list to `current` and set its state to `Running`.
- Perform a context switch to `current`

In addition, there is another scheduling primitive `ready(p)` which moves a thread (other than `current`) to the ready list and sets its state to `Ready`. Thus, a thread can block by setting its state to `Waiting` and calling `sched()`. A blocked thread `p` can be made ready by a call to `ready(p)`.

You are given a single synchronization primitive

`int tas(int *)`

which implements an atomic test and set operation.

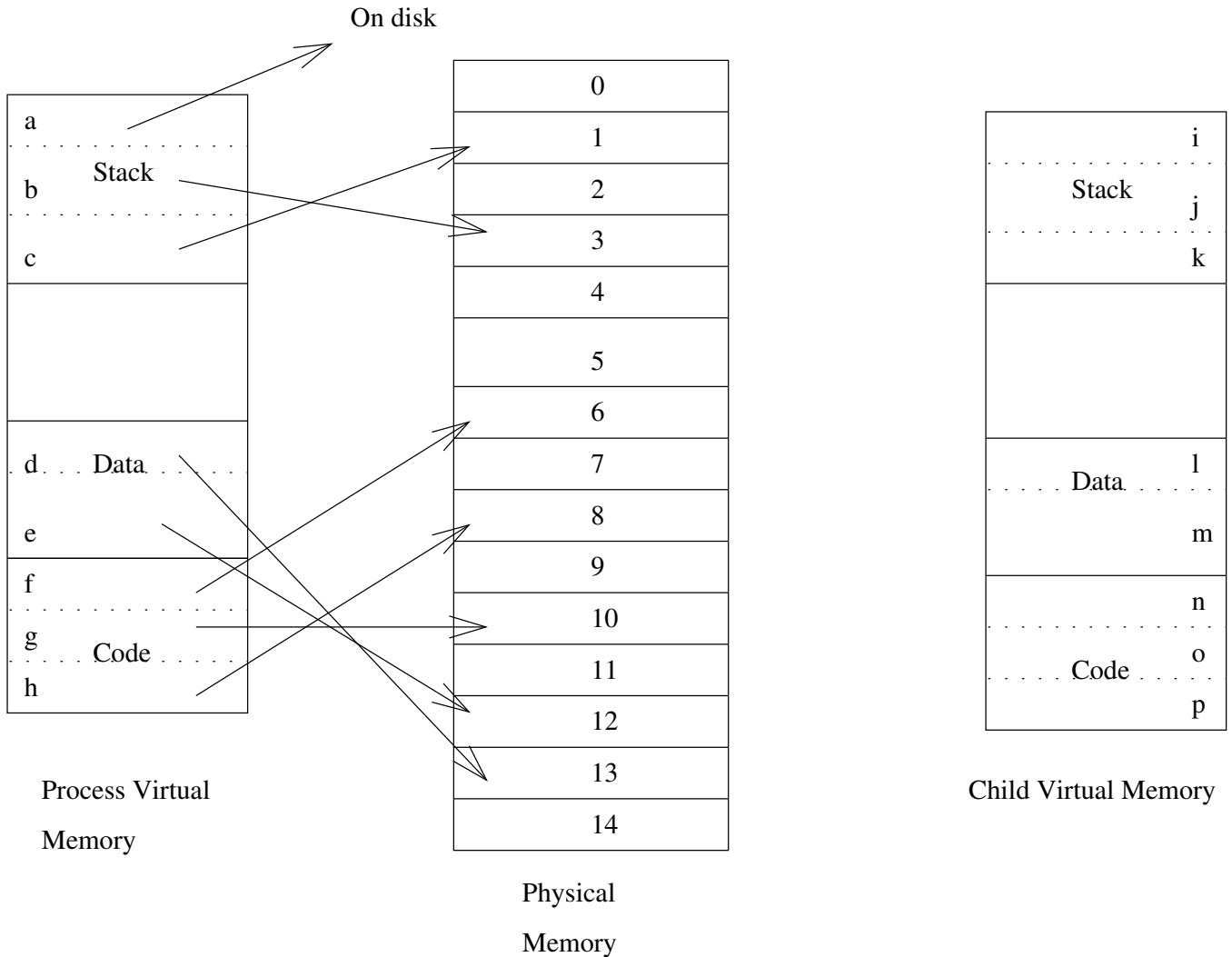
You are to implement a semaphore object. You may assume a thread list object with the following interface:

```
void TListInsert(ThreadList *, Thread *);
Thread *TListRemove(ThreadList *); // returns 0 for empty ThreadList
void TListInit(ThreadList *);
```

- (a) Define the Semaphore data structure and a suitable semaphore init function `SemInit`.
- (b) Implement the wait (P) function `SemWait`.
- (c) Implement the signal (V) function `SemSignal`.

2. Virtual Memory

The following memory map represents a typical executing UNIX process.



- What access permissions are associated with each page of the process' virtual memory – use the flags **r** (read), **w** (write), **x** (executable), and **p** (present in memory). You should write a table showing the page to permission mapping (use the page labels given in the chart).
- Suppose the process forks a child and copy-on-write is used, define the the child's virtual memory map with a table providing the mapping from virtual to physical pages and write the appropriate permissions for each virtual page. Similarly, define the parent's permissions.

3. Replacement Algorithms

Given reference string **5, 3, 4, 3, 1, 2, 1, 4, 3, 4, 5, 3, 2** which gives the sequence of page accesses by running program, you are to develop tables for several page replacement algorithms using the following table structure:

5	3	4	3	1	2	1	4	3	4	5	3	2

- (a) With three (initially empty) page frames, how many page faults will occur with the optimal page replacement algorithm ? Show the contents of the page frames at each “step” circling newly loaded pages.
- (b) With three (initially empty) page frames, how many page faults will occur with LRU page replacement ? Show the contents of the page frames at each “step” circling newly loaded pages.
- (c) With three (initially empty) page frames, how many page faults will occur with FIFO page replacement ? Show the contents of the page frames at each “step” circling newly loaded pages.

4. Concurrent Programming

Your goal is to implement a program with two coroutines using threads and semaphores. Recall that coroutines are a special case of threads where scheduling is handled explicitly in the coroutine – one coroutine explicitly relinquishes use of the cpu by transferring to another specific coroutine.

- (a) Define any shared variables and the initialization routine. Assume that you can create threads with `threadCreate(void (*t)())` where `t()` is the code for a given thread.
- (b) Write the skeleton for each coroutine – you do not have to be explicit about what a coroutine does when it has control, for example, it might just do `state = Update(state)` and then transfer control.

5. Scheduling

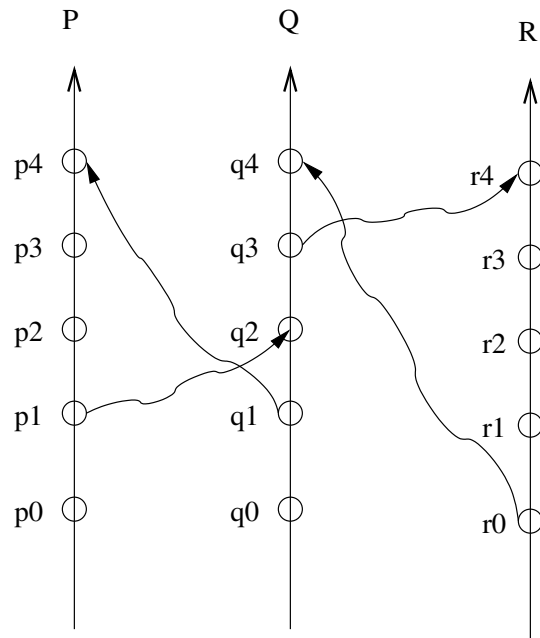
For each of the following scheduling algorithms, describe a major failing of the algorithm and if appropriate provide a pathological example with a short Gantt chart that illustrates this failing.

- (a) First Come First Served
- (b) Shortest Job First
- (c) Priority Scheduling

6. Time in distributed Systems.

In any distributed coordination problem, it is necessary to be able to determine the order in which two events occurred. In a distributed system it is sometimes impossible to say which of two events occurred first. The happened-before relation is used in obtaining a total ordering of events in a distributed system. The happened-before relation (denoted by \rightarrow) is defined as follows: first, if A and B are events in the same process, and A was executed before B , then $A \rightarrow B$. Second, if A is the event of sending a message by one process and B is the event of receiving that message by another process, then $A \rightarrow B$. Third, if $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$.

- (a) In a centralized system, it is always possible to determine the order in which two events occurred, but in a distributed system this is not always possible. Why?
- (b) From the diagram below of relative time for three concurrent processes, what can you say about the relationship between q_0 and p_2 ? What can you say about the relationship between p_0 and r_4 ?



- (c) For each of the three rules defining happened-before, substantiate why the rule must hold. (Hint: assume the property does not hold and discuss the consequence.)

7. Parallel Machines

Once you remove all the window dressings, a “parallel machine” such as the Origin 2000; a “cluster”, such as the Thor cluster in the machine room of 8 dual processor 2.8 GHz Intel machines; or a “distributed system” such as all workstations in the graduate lab, each consists of a set of processors communicating through some interconnection network. All three architectures support multi-process applications; the processes must communicate with each other during the course of execution. The fundamental differences are in process communication, synchronization, and sharing.

- (a) For each class of machine, identify a communication and a synchronization strategy.
- (b) What operating systems issues may be relevant in one class of machine but irrelevant in another?
- (c) What are the characteristics of applications that may be appropriate for each of these classes of machines?

8. File Systems

File systems in use today are optimized from studies of file system workloads conducted in the late 1980's and early 1990's. Ousterhout et al. in 1985 traced three servers running BSD UNIX for slightly over three days. This study, which we call the BSD study, introduced metrics such as run length, burstiness, lifetime of newly written bytes, and file access sequentiality. Baker et al. in 1991 conducted the same type of analysis on four two-day sets of traces of the Sprite file system. In this study, which we call the Sprite study, the authors collected traces at the file servers and augmented them with client information on local cache activity.

Recently Roselli et al. (in 2000) conducted a new study to understand how modern workloads affect the ability of file systems to provide high performance to users. As you may guess, applications like web servers, which serve web pages to users on the Internet, can have a read to write ratio of 300:1 in terms of total megabytes read and written. You have been tasked with proposing changes to existing UNIX based file systems needed to adapt to modern workloads.

- (a) Given a file system with disk blocks that are 4K in size, an i-node structure that has 12 direct data pointers, and two levels of indirect pointers, what is the maximum file size?
- (b) Suppose there are two environments on which a file system runs. The first has 8MB of local cache and a write delay of 30 seconds. The second has 64 MB of local cache and a write delay of 1 hour. Discuss the impact of each environment on disk read and write traffic.
- (c) For purposes of this question, assume that a disk block is 8KB and an i-node structure has 12 direct data pointers and two levels of indirect pointers. Assume also that "large" files are files over 96KB. The Sprite study found that while most files that were accessed were small, the size of the largest files had increased since the BSD study. The Roselli study shows that this trend has continued. While the large files still constitute about the same percentage of total files, today large files are accessed a significantly greater percentage of the time than in the past. Roselli further found that large files are more likely to be accessed randomly. Most files are still small. Suppose you are designing a new file system for modern workloads. What design decisions would you make to address these changes to workload?