

**CSC 369H1 F
OPERATING SYSTEMS**

**UNIVERSITY OF TORONTO
FALL 2005**

Midterm Test

NO AIDS ALLOWED

Please **PRINT** in answering the following requests for information:

Family Name: _____

Given Names: _____

Student Number: | _ _ _ | | _ _ _ | | _ _ _ |

Login (@cdf): _____

Notes to students:

1. This test lasts for 50 minutes and consists of 50 marks. Budget your time accordingly.
2. This test has 8 pages (including this one) and 6 questions; Check that you have all pages before starting.
3. **Write in pen. No pencils. Really, I mean it.**
4. Write your answers on this “question and answer” paper, in the spaces provided. Be concise. In general, the amount of space provided is an upper bound on the “size” of answer that is expected. If necessary, use space where available and provide explicit pointers.
5. In general, state your assumptions and show your intermediate work, where appropriate.
6. Do **not** go beyond here until instructed to do so. Write your student number at the top of each succeeding page once you get going.

Question	Marks
1	/ 6
2	/ 12
3	/ 8
4	/ 8
5	/ 10
6	/ 6
Bonus	/ 1
Total	/ 50

1. [6 marks, 1 each] True/False

Circle "T" if the statement is true. Otherwise circle "F".

- a) Context switch time on modern hardware is small enough to be ignored entirely when designing a CPU scheduler. **F**

Context switches are expensive operations, requiring many memory accesses to save/restore registers.

- b) The bakery algorithm ensures the Circular Wait condition cannot occur. **F**

That's the banker's algorithm. The bakery algorithm is a software soln to the critical section problem.

- c) Setting the "base" and "limit" registers are privileged operations. **T**

These registers define the memory a process can access, so an untrusted process cannot do this itself.

- d) Setting the program counter register is a privileged operation. **F**

Setting the PC is required for any control flow in a program (branches, function calls) and does not require special privileges. If the process tries to branch to a location it should not access, the memory access controls will prevent it from actually fetching instructions from that location.

- e) Deadlocks can always be detected by finding a cycle in a resource allocation graph. **F**

Only if there is a single instance of each type of resource. If there are multiple instances of a particular type of resource, then cycles may exist in the graph without indicating a deadlock exists.

- f) Shortest-job-first scheduling is not suitable for a general-purpose computer system. **T**

SJF requires the length of the job to be known in advance – this may be possible in specialized settings but for general purpose programs, knowing the job length in advance is unreasonable.

2. [12 marks; 4 each]

a) Suppose we want to solve the critical section problem in a single processor system. We have available either of the following two options:

- Option A: Disable interrupts.
- Option B: Use a built-in synchronization hardware instruction, such as test-and-set.

State one advantage of Option A over Option B, and identify the one used by OS/161.

Option A does not require busy-waiting if the critical section cannot be entered immediately, as does Option B. On a single processor system, busy-waiting is inappropriate since it wastes CPU cycles that would be better used by the process in the critical section. In contrast, once a process disables interrupts to enter the critical section, it is guaranteed to be the only process running until it leaves the critical section (either by yielding the CPU, or restoring interrupts), so no CPU time is wasted. OS/161 uses Option A for critical sections (such as the semaphore P() and V() functions).

b) **Explain** why an unsafe resource allocation state does not always lead to a deadlock state.

A safe state means that there is some order in which we can run the processes to completion, assuming that all processes continue to hold the resources they have already been granted until they complete, and that a process will need additional resources up to its maximum to complete. In an unsafe state, we cannot find such an order, so deadlock is possible, but may not actually occur depending on the actual behaviour of the processes. For example, some processes may release some of the resources they are currently holding, so that deadlock does not result, or they may no longer need their declared maximum to complete.

c) **State** the purpose of system calls (that is, why they are provided) and **briefly describe** their basic operation.

System calls allow unprivileged user-level processes to request services from the operating system, which may require privileged operations that the user-level process may not perform for itself. A user process makes a system call by setting designated registers with the type of system call and any arguments to that system call, and then traps to the OS (perhaps via a special "syscall" instruction as in the MIPS processors). The OS code then examines the type of system call and calls an appropriate handler function with the arguments. Results are returned to user level via registers as well.

3. [8 marks; 2 marks each]**a) Define internal fragmentation**

Wasted space within a unit of memory allocation. Internal fragmentation occurs when the units or chunks of allocated memory are larger than the size required or requested. The additional space is not needed by the process that it was allocated to, and cannot be used by any other process because it has been allocated.

b) Define external fragmentation.

Wasted space that is not allocated to any process. Free memory is broken into many small chunks which may be too small to satisfy allocation requests, even though there is enough total free memory to fill the request.

c) Circle

which one(s) can occur in paging systems:

internal external

which one(s) can occur in systems that use dynamic partitioning:

internal **external**

Internal fragmentation may also occur with dynamic partitioning, if we choose not to split very small chunks of unused memory from a chunk selected for allocation.

d) Briefly explain one reason dynamic partitioning might be used instead of paging.

Paging requires more sophisticated support for address translation of non-contiguous memory regions. Dynamic partitioning is thus used anywhere this hardware support is not available. The C library malloc()/free() routines for example, allocate memory within a virtual address space and the memory returned by malloc() must be contiguous within the virtual address space.

A second reason for using dynamic partitioning would be when the expected allocations were all much smaller than the size of a page, since the internal fragmentation caused by paged allocation would be very high in this case.

4. [8 marks; 4 each]

Most Unix systems provide a `setpriority()` function to set the priority of a user-level process. There is no guarantee about how the kernel CPU scheduler will actually use these priority values, however.

a) Describe an experiment using user-level processes that would allow you to determine if the scheduler is selecting processes according to their priority levels. Be sure to account for other factors (in addition to priority) that could affect scheduling.

Create a number of CPU-bound processes with different priorities and see in what order they complete. The test processes should all perform the same amount of work, by executing an identical computation, so that the completion order is the result of scheduling and not the amount of CPU time needed (Alternately, we could make the higher-priority processes need more CPU time). The computation should be long enough to run for several timeslices on the system under observation, to ensure that (multiple) scheduling decisions are actually being made. The order in which the processes are created should be different from the order in which they should complete under priority scheduling, to ensure we are not just seeing the effect of round-robin scheduling, or FCFS scheduling. To observe completion times, each process can just print a message with its priority level before exiting, but they should do no other I/O or any other system calls during the computation to make sure they are not blocked for any reason.

b) Describe a similar experiment that would allow you to determine if the scheduler is dynamically adjusting priorities to prevent starvation of low-priority processes.

Create a number of long-running CPU bound processes at high priority, together with a very short-running process at low priority, and observe the order in which they complete. With a strict priority scheduler, the low-priority process should always finish last, after all the high-priority processes. But if the low-priority process receives some CPU time, then it should finish ahead of (at least some of) the high-priority processes, because it needs very little CPU time.

5. [10 marks]

Consider a system that uses two-phase locking to support concurrent transactions. Each data item, X , has its own lock, with operations $\text{lock}(X)$ and $\text{unlock}(X)$.

a) [6 marks] Rewrite the following transaction, T_1 , inserting the necessary lock and unlock operations, so that the two-phase locking protocol is satisfied, and concurrent access to data items is maximized. Label the phases in your answer.

```
T1(A, B, C) :
  Read(A);
  Read(B);
  Write(C);
```

```
T1(A, B, C) :
  Lock(A)
  Read(A);
  Lock(B);
  Read(B);
  Lock(C);
  Unlock(A);
  Unlock(B);
  Write(C);
  Unlock(C);
```

Growing phase

Shrinking phase

b) [2 marks] What is the purpose of two-phase locking?

To guarantee conflict-serializable schedules of concurrent transactions. In other words, to guarantee that concurrent transactions always observe a consistent state, and that the result of several concurrent transactions is equivalent to some sequential execution of the same transactions.

c) [2 marks] What is the main problem with the two-phase locking protocol?

Deadlocks may occur. A transaction management system using two-phase locking must provide additional support to detect and recover from deadlocks.

6. [6 marks]

Consider the following solution to the critical section problem for 2 processes:

```

1.  boolean blocked[2];
2.  int  turn;
3.  void P (int id) {
4.      while (true) {
5.          /* lines 6-12 Entry to C.S. */
6.          blocked[id] = true;
7.          while (turn != id) {
8.              while (blocked[1-id]) {
9.                  /* do nothing */
10.             }
11.             turn = id;
12.         }
13.         /* line 14 Code for critical section */
14.         ...
15.         /* line 16 Exit from C.S. code */
16.         blocked[id] = false;
17.         /* remainder code */
18.         ...
19.     }
20. }
21.
22. void main() {
23.     blocked[0] = false;
24.     blocked[1] = false;
25.     turn = 0;
26.     /* Create 2 parallel threads,
27.      * one executes P(0),
28.      * the other executes P(1)
29.      */
30.     ...
31. }
```

Show an execution sequence that demonstrates the solution above is incorrect.

Process P(0)	Process P(1)	blocked[0]	blocked[1]	turn
	4. while (true)	False	False	0
	6. blocked[1] = true	False	True	0
	7. while(turn != 1) ← true	False	True	0
	8. while(blocked[0]) ← false Next line to execute is 11	False	True	0
	INTERRUPT!			
4. while (true)		False	True	0
6. blocked[0] = true		True	True	0
7. while (turn != 0) ← false Next line to execute is 14		True	True	0
14. Now in critical section		True	True	0
INTERRUPT!				
	11. turn = 1 (return to top of while)	True	True	1
	7. while (turn != 1) ← false Next line to execute is 14	True	True	1

../continued

	14. Now in critical section!	True	True	1
--	------------------------------	------	------	---

Extra space. Please indicate clearly which question(s) you are answering here, if any.

Jeopardy-style Bonus (1 mark, you supply the question): 42

Total marks = (50)

End of test