University of Waterloo
Midterm Examination

Spring, 2006

Student Name:	
Student ID Number:	
Section:	

Course Abbreviation and Number	CS350
Course Title	Operating Systems
Sections	01 (14:30), 02 (11:30)
Instructor	A. Aboulnaga

Date of Exam	June 20, 2006
Exam Returned	June 22, 2006
Appeal Deadline	July 4, 2006
Time Period	19:00-21:00
Duration of Exam	2 hours
Number of Exam Pages (including this cover sheet)	16 pages
Exam Type	Closed Book
Additional Materials Allowed	None

NOTE: KEEP YOUR ANSWERS AS CONCISE AS POSSIBLE.

Question 1: (14 marks)	Question 4: (12 marks)	Question 7: (11 marks)
Question 2: (11 marks)	Question 5: (12 marks)	Question 8: (17 marks)
Question 3: (10 marks)	Question 6: (13 marks)	
Total: (100 marks)		

Question 1. (14 marks)

a. (2 marks)
What is *context switching*?

Sample Answer:
Context switching is when the kernel switches the CPU from one process to another. This requires the kernel to save the state of the currently running process in the PCB.

---

b. (3 marks)
List three ways in which execution can switch from user space to kernel space (i.e., three ways in which an OS kernel can get back the CPU from a user proceess).

Sample Answer:
1- System calls, 2- Exceptions, 3- Interrupts

---

c. (2 marks)
What is the *timer interrupt*? And how does the OS use it for scheduling?

Sample Answer:
Most computers have hardwre clocks (or timers) that generate periodic interrupts. These interrupts are known as the timer interrupts. The OS uses the timer interrupt to measure the passage of time and determine when the quantum of a running process has expired, at which point the process is preempted. The timer interrupt also ensures that the kernel will get the CPU back from a running process so that it can make scheduling decisions.

---

d. (3 marks)
Why is Least Recently Used (LRU) considered to be impractical for use as a replacement policy in virtual memory systems?

Sample Answer:
Because it requires an LRU list or a per-page last access time stamp to be updated by the MMU with every memory access. This is prohibitively expensive.

---

e. (2 marks)
In virtual memory systems, why is replacing a clean page faster than replacing a dirty page?

Sample Answer:
When the OS replaces a dirty page, it has to write this page to the swap disk. A clean page can be replaced without writing it to the swap disk.

---

f. (2 marks)
List one factor that favours having larger virtual memory pages and one factor that favours having smaller pages.

Sample Answer:
Factors favouring large pages: smaller page tables, fewer entries in the TLB needed to cover the same amount of memory, more efficient I/O to swap pages in.
Factors favouring small pages: reduced internal fragmentation, less likely to page in unnecessary data.

Question 2. (11 marks)

a. (4 marks)
Processes (or threads) can be in one of three states: **Running**, **Ready**, or **Blocked**. In which state is the process (or thread) for each of the following four cases?

(i) Waiting for data to be read from a disk.
(ii) Spin-waiting for a lock to be released.
(iii) Having just called `wait()` on a condition variable in a monitor.
(iv) Having just completed an I/O and waiting to get scheduled again on the CPU.

Sample Answer:

(i) Waiting for data to be read from a disk. **Blocked**
(ii) Spin-waiting for a lock to be released. **Running**
(iii) Having just called `wait()` on a condition variable in a monitor. **Blocked**
(iv) Having just completed an I/O and waiting to get scheduled again on the CPU. **Ready**

---

b. (4 marks)

Consider the following list of actions. Put a check mark in the blank beside those actions that should be performed by the kernel, and not by user programs. (0.5 marks per action)

- ____ reading the value of the program counter (PC).
- ____ changing the value of the program counter (PC).
- ____ changing the value of the segment table base register.
- ____ changing the value of the stack pointer (SP).
- ____ increasing the size of an address space.
- ____ creating a memory segment that is shared between multiple processes.
- ____ writing to a memory segment that is shared between multiple processes.
- ____ disabling interrupts.

Sample Answer:

- ____ reading the value of the program counter (PC).
- ____ changing the value of the program counter (PC).
- _XX_ changing the value of the segment table base register.
- ____ changing the value of the stack pointer (SP).
- _XX_ increasing the size of an address space.
- _XX_ creating a memory segment that is shared between multiple processes.
- ____ writing to a memory segment that is shared between multiple processes.
- _XX_ disabling interrupts.

---

c. (3 marks)

Which of the following is shared between threads of the same process? Put a check mark in the blank beside the items that are shared. (0.5 marks per item)

- ____ integer and floating point registers.
- ____ program counter.
- ____ heap memory.
- ____ stack memory.
- ____ global variables.
- ____ open files.

Sample Answer:

- ____ integer and floating point registers.
- ____ program counter.
- _XX_ heap memory.
- ____ stack memory.
- _XX_ global variables.
- _XX_ open files.

Question 3. (10 marks)

Is each of the following statements True or False? Explain your answer.

a. (2 marks)

A multi-threaded program is, in general, non-deterministic. i.e., the program can give different outputs in different runs on the same input.

Sample Answer:

True. The threads can be scheduled differently by the OS in different runs, depending on factors such as when I/O completes or when interrupts happen. The threads can, in general, access shared data, and the different scheduling of thredas will result in different values of the shared data and hence different output. The different scheduling of the threads can also result in different ordering of program output.

---

b. (2 marks)

When *monitors* are used for synchronization in a multi-threaded program, they (the monitors) eliminate the possibility of threads waiting indefinitely for a synchronization event to occur.

Sample Answer:

False. Monitors simplify the task of writing concurrent programs, but indefinite waiting can still happen. For example, a thread in a monitor can call wait() on a condition variable, and there may be no thread that will call signal() on this variable.

---

c. (2 marks)

It is not possible for thrashing to occur in a system that uses a recency-based page replacement policy such as LRU.

Sample Answer:

False. Thrashing happens when the multiprogramming level is too high for the available physical memory and so processes cannot keep their working sets in memory. Thrashing has nothing to do with the replacement policy.

---

d. (2 marks)

One way to reduce the likelihood of thrashing is to increase the amount of main memory (RAM) in the system.

Sample Answer:

True. Increasing the amount of memory allows processes to keep more pages in memory, so the processes are more likely to have their working sets in memory, so the likelihood of thrashing is reduced.

---

e. (2 marks)

If every page in memory is accessed (used) between any two page faults, then the Clock replacement algorithm behaves exatcly like FIFO.

Sample Answer:

True. If every page in memory is used between any two page faults, the use bits of all frames on the "clock face" will be set by the MMU between replacements. Thus, the "clock hand" will not find a frame whose use bit is clear until it makes a full cycle around the face of the clock, and comes back to the frame on which it started. That frame will have a clear use bit because the clock hand will have cleared it when it started its sweep, so this is the frame that will be replaced. Thus, if the clock hand start at frame i, it will make a full sweep of the clock face and come back to replace frame i. Next time, it will replace frame $i + 1$, and so on. So the frames will be replaces in the order $0, 1, 2, \ldots$, which is FIFO.

Question 4. (12 marks)

a. (4 marks)
Briefly describe how the operating system gets loaded into memory and starts executing when the machine is powered up (also known as "booting" the operating system).

Sample Answer:

1. CPU starts executing instructions from a known location in memory when it is powered up.
2. That location is in ROM, and it has a simple program that loads a "bootstrap program" from a fixed location on disk and starts executing this bootstrap program.
3. The bootstrap program initializing the system (registers, memory, devices), and loads the full operating system into memory and start executing it.

---

b. (3 marks)
In class, we discussed three heuristics for memory placement when we are using variable sized memory allocation: *first fit*, *best fit*, and *worst fit*. Briefly describe the *worst fit* allocation strategy, and explain the motivation behind it.

Sample Answer:
The worst fit heuristic specifies that the OS should allocate the *largest* free memory area or "hole" to satisfy a memory request. This results in the largest possible leftover hole after satisfying the memory request. The motivation behind worst fit is that such a large leftover hole would be more useful for satisfying future memory requests that the smaller leftover holes that we get with first fit or best fit.

---

c. (5 marks)

Consider an OS that uses local page replacement (i.e., each process gets its own set of frames independent of any other process). Assume that this OS uses the Clock page replacement policy. Consider a process in this OS that has a working set of M pages, and is allocated exactly M frames.

(i) Do you expect the page fault frequency for this process to be high or low? Explain briefly.

(ii) Consider the case when this process incurrs a page fault. On average, how many steps will the "clock hand" advance to find a victim for replacement (i.e., how many times will the victim pointer be incremented)? Explain your answer.

Sample Answer:

(i) Low. A process mostly accesses pages from its working set. Since the working set is entirely in memory, page fault frequency will be low.

(ii) The process incurrs page faults only infrequently. Thus, when a page fault does happen, it is likely that all the M frames will have their use bits set. The victim pointer will therefore have to go through all M frames clearing the use bits before it can find a candaidte for replacement. Thus, the clock hand will advance M times on each page fault.

Question 5. (12 marks)

a. (4 marks)

Suppose that two long running processes, P_1 and P_2, are running in a system. Neither program performs any system calls that might cause it to block, and there are no other processes in the system. P_1 has 2 threads and P_2 has 1 thread.

(i) What percentage of CPU time will P_1 get if the threads are *kernel threads*? Explain your answer.

(ii) What percentage of CPU time will P_1 get if the threads are *user threads*? Explain your answer.

Sample Answer:

(i) If the threads are kernel threads, they are independently scheduled and each of the three threads will get a share of the CPU. Thus, the 2 threads of P_1 will get 2/3 of the CPU time. That is, P_1 will get 66% of the CPU.

(ii) If the threads are user threads, the threads of each process map to one kernel thread, so each *process* will get a share of the CPU. The kernel is unaware that P_1 has two threads. Thus, P_1 will get 50% of the CPU.

---

b. (3 marks)

When a thread is waiting for an event to occur (e.g., waiting for a lock to be released), describe a situation in which busy waiting (also known as spin-waiting) can result in better performance than blocking while waiting. Explain why busy waiting never performs better than blocking while waiting for single-CPU systems.

Sample Answer:

If the thread will wait for a short amount of time (compared to the time required for a context switch), then it is better to spin-wait. Blocking while waiting requires two context switches, one to block the thread and one to unblock it. If a thread t_1 is waiting for an event in a single-CPU system (e.g., waiting for a lock to be released), then the thread t_2 that will cause the event to happen (e.g., the thread that will release the lock) must be scheduled for t_1's waiting to end. By spin-waiting, t_1 is preventing t_2 from being scheduled, and is therefore prolonging its own wait as well as wasting system resources. In a multi-CPU system, t_2 would be scheduled on a different CPU.

---

c. (5 marks)

Some hardware provides a `TestAndSet` instruction that is executed atomically, and that can be used to implement mutual exclusion.

(i) Briefly describe what the `TestAndSet` instruction does.

(ii) Write a fragment of code or pseudo-code to show how `TestAndSet` can be used to implement critical sections. Be sure to indicate which variables in your code or pseudo-code are shared between the threads and which are local to each thread.

Sample Answer:

(i) The `TestAndSet` instruction sets the value of a variable and returns the old value. It is executed atomically.

(ii)

```
boolean lock; // Shared, initially false.
while (TestAndSet(&lock, true)){ }
Critical Section
lock = false;
```

Question 6. (13 marks)

Consider the following code for inserting into a hash table that will be concurrently used by multiple threads. Some details of the code are omitted for clarity. The hash table is implemented as an array of `N` linked lists, and each node in these linked lists contains an integer hash key, `key`, and its associated object, `obj`.

```
void hashInsert(int key, void *obj) {
    int listNum = key % N; // Determine which list the item belongs to.
    listInsert(hashArray[listNum], key, obj);
}

void listInsert(node *head, int key, void *obj) {
    // Create a new node, nn, that contains key and obj.
    nn->next = head;
    head = nn;
}
```

a. (4 marks)

Show a problem that can occur if two threads try to insert items into the hash table simultaneously.

Sample Answer:

A race condition can happen that would lead to one of the insertions being lost.

Thread 1	Thread 2
`nn-> next = head;`	
	`nn->next = head;`
	`head = nn;`
`head = nn;`	

Here, the insertion of Thread 2 was lost.

---

b. (4 marks)

Add a single lock to the above code to fix the problem. Feel free to annotate the code given above or to copy it in the space below.

Sample Answer:

```
Lock l; // Lock variable shared between threads.

void hashInsert(int key, void *obj) {
    int listNum = key % N; // Determine which list the item belongs to.
    l.Acquire();
    listInsert(hashArray[listNum], key, obj);
    l.Release();
}
```

---

c. (2 marks)

What is the biggest performance problem with this single-lock solution?

Sample Answer:

This solution serializes access to `hashArray`. If multiple threads want to update different lists in `hashArray`, they would still have to wait for each other.

---

d. (3 marks)

Show how you can use more locks to improve the performance of accessing the hash table by allowing more concurrency.

Sample Answer:

We can have an array of locks, one for each entry in `hashArray`.

```
Lock l[N]; // Array of lock variables shared between threads.

void hashInsert(int key, void *obj) {
    int listNum = key % N; // Determine which list the item belongs to.
    l[listNum].Acquire();
    listInsert(hashArray[listNum], key, obj);
    l[listNum].Release();
}
```

Question 7. (11 marks)

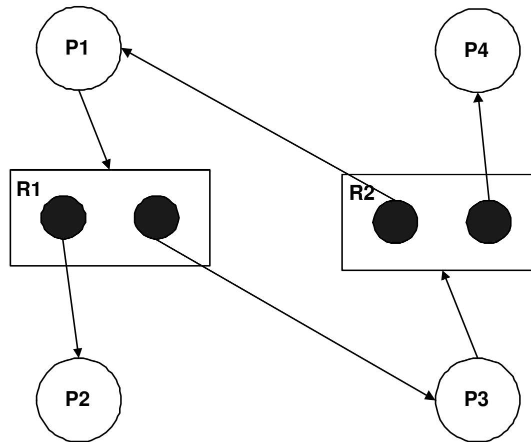
Consider the following information about resources in a system:

- There are two classes of allocatable resource labelled R1 and R2.
- There are two instances of each resource.
- There are four processes labelled P1 through P4.
- There are some resource instances already allocated to processes, as follows:
  - one instance of R1 held by P2, another held by P3
  - one instance of R2 held by P1, another held by P4
- Some processes have requested additional resources, as follows:
  - P1 wants one instance of R1
  - P3 wants one instance of R2

a. (5 marks)

Draw the resource allocation graph for this system. Use the style of diagram from the lecture notes.

Sample Answer:



---

b. (2 marks)

What is the state (runnable, waiting) of each process? For each process that is waiting, indicate what it is waiting for.

Sample Answer:
P1 waiting.
P2 runnable.
P3 waiting.
P4 runnable.

---

c. (4 marks)

Is this system deadlocked? If so, state which processes are involved. If not, give an execution sequence that eventually ends, showing resource acquisition and release at each step.

Sample Answer:
Not deadlocked (even though a cycle exists in the graph). One possible execution sequence: P2, P1, P4, P3. and

---

Question 8. (17 marks)

a. (4 marks)

Using the page references string shown below, fill in the frames and missing information to show how the **OPT** (optimal) page replacement algorithm would operate. Use a dash "–" to fill in blank locations. Note that when there is more than one page that is a possible victim, always choose the one with the **lowest** frame number.

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	B	E	C	G	D	A	G	D	B	E	C
Frame 1	A														
Frame 2	–														
Frame 3	–														
Frame 4	–														
Fault ?	X														

Sample Answer:

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	B	E	C	G	D	A	G	D	B	E	C
Frame 1	A	A	A	A	A	A	A	A	A	A	A	A	B	B	C
Frame 2	–	B	B	B	B	E	E	E	E	E	E	E	E	E	E
Frame 3	–	–	C	C	C	C	C	G	G	G	G	G	G	G	G
Frame 4	–	–	–	D	D	D	D	D	D	D	D	D	D	D	D
Fault ?	X	X	X	X	–	X	–	x	–	–	–	–	X	–	X

b. (4 marks)

Fill in the frames and missing information below to show how the **LRU** (least recently used) page replacement algorithm would operate. Use a dash "–" to fill in blank locations. Note that when there is more than one page that is a possible victim, always choose the one with the **lowest** frame number.

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	B	E	C	G	D	A	G	D	B	E	C
Frame 1	A														
Frame 2	–														
Frame 3	–														
Frame 4	–														
Fault ?	X														

Sample Answer:

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	B	E	C	G	D	A	G	D	B	E	C
Frame 1	A	A	A	A	A	E	E	E	E	A	A	A	A	E	E
Frame 2	–	B	B	B	B	B	B	B	D	D	D	D	D	D	D
Frame 3	–	–	C	C	C	C	C	C	C	C	C	C	B	B	B
Frame 4	–	–	–	D	D	D	D	G	G	G	G	G	G	G	C
Fault ?	X	X	X	X	–	X	–	X	X	X	–	–	X	X	X

c. (4 marks)

Assume that there are 5 pages, A, B, C, D, and E. Fill in the page reference string and complete the rest of the information in the table below so that LRU is the *worst* page replacement algorithm (i.e., it results in the maximum number of page faults). Use a dash "–" to fill in blank locations. Note that when there is more than one page that is a possible victim, always choose the one with the **lowest** frame number.

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	E										
Frame 1	A														
Frame 2	–														
Frame 3	–														
Frame 4	–														
Fault ?	X														

Sample Answer:

Num	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Refs	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
Frame 1	A	A	A	A	E	E	E	E	D	D	D	D	C	C	C
Frame 2	–	B	B	B	B	A	A	A	A	E	E	E	E	D	D
Frame 3	–	–	C	C	C	C	B	B	B	B	A	A	A	A	E
Frame 4	–	–	–	D	D	D	D	C	C	C	C	B	B	B	B
Fault ?	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

d. (2 marks)

Describe the pattern of references in the reference string that you came up with in part (c).

Sample Answer:

Repeated sequential scan of a set of pages.

---

e. (3 marks)

Suggest a replacement policy that would minimize the number of page faults for the pattern of references that you indentified in part (d). Your replacement policy cannot require advance knowledge of future page references.

Sample Answer:

Most Recently Used is a good policy in this case. MRU would behave just like OPT here.