CSE 120: Midterm Solution

August 23, 2007

Name: _____

Student ID:_____

Write your test number on *all* pages because the pages will be separated for grading.

No books, but calculators are allowed. Two double-sided 3x5 cards of handwritten notes is allowed.

If you think something about the question is open to interpretation, ask!.

Leaving a question blank will give you 25% partial credit. An answer that we can't read, can't understand, or is true, but not directly relevant to the question will receive a 0.

Be extremely concise in your answers.

Problem	Score
1	/4
2	/6
3	/4
4	/4
5	/4
6	/10
7	/8
8	/14
9	/10
Total	/64

1. *4 pts.* Suppose a thread in a user program is in a critical section of its code, having acquired all necessary locks. Can it get context-switched? Why or why not?

   Solution: Yes, it can get context-switched. The scheduler is unaware of locks.

   2 points for yes. 2 for explanation.

2. *6 pts.* What are some of the hardware differences between user mode and kernel mode. Name at least three.

   Solution:

   - Interrupt handlers run in kernel mode
   - On reset, hardware is in kernel mode
   - Kernel mode can read/write OS memory
   - Kernel mode can read/write certain registers (including kernel mode bit!)
   - Kernel mode can read/write Page Table registers
   - Kernel mode bit is 0 in kernel mode, 1 in user mode
   - Some instructions available only in kernel mode

   2 points for each of three differences.

3. *4 pts.* What is saved and restored (and where is it saved from and restored to) on a context switch between two threads in the same process? Between two threads in a different process? Be explicit.

   Solution: In the same process, must save register, sp, and pc into tcb of old thread (1 point) must restore registers, sp, pc from tcb of new thread. (1 point)

   In a different process, must save and restore as above (1 point), but must also restore the Page Table Base Register from the pcb of new thread (1 point).

4. *4 pts.* What is a monitor?

   Solution: Synchronization method (often built into a language) provided to support mutual exclusion and synchronization.

   It uses a lock (that provides mutual exclusion between all procedures in the monitor) and zero or more condition variables (that are used for synchronization).

5. *4 pts.* What is the difference between Hoare-style and Mesa-style scheduling for monitors? Be explicit, but brief.

   Solution: In Mesa-style monitors, when a thread does a signal on a condition variable, that causes one of the waiting threads to be put on the ready queue.

   In Hoare-style monitors, when a thread does a signal on a condition variable, it loses the lock, and one of the waiting threads is run immediately. When that thread is done, it gives the lock back and then execution continues with the original thread.

   2 points each

6. *10 pts.* Joe wrote the following code to solve the Dining Philosphers problem. Unfortunately, it has the potential for deadlock.

   Note that there are five philosophers each running the same `Philosopher` code, but with a different value of `i`.

```
shared Semaphore chopsticks[5]; // Each initialized to 1
Philospher(int i) {
  for (;;) {
    Think();
    chopsticks[left(i)].P();
    chopsticks[right(i)].P();
    Eat();
    chopsticks[left(i)].V();
    chopsticks[right(i)].V();
  }
}
```

   (a) *2 points* How can deadlock occur with this solution (be specific, but brief)?

   Solution: If every philosopher executes their first P before any executes the second P.

(b) *4 points*

For each of the four requirements of deadlock, show how it is evidenced in the situation:

Solution:

i. Mutual exclusion: All philosophers are blocked.

ii. Hold and wait. Each philosopher is holding a fork and waiting for another

iii. No preemption: No philosopher can be forced to give up the chopstick s/he has.

iv. Circular wait: Each is waiting on the philosopher to his/her right.

1 each.

(c) *4 points*

Fix the code by:

- picking one of the four requirements of deadlock,
- changing the code so that it no longer meets that requirement, and
- explaining how the change breaks the requirement.

Solution: Easiest is to break circular wait:

```
if (i == 3) {
chopsticks[right(i)].P();
chopsticks[left(i)].P();
} else {
chopsticks[left(i)].P();
chopsticks[right(i)].P();
}
```

Now we wont have philosopher 3 waiting on 4 and 2 waiting on 3.

Other solutions might be possible.

Points:

2 points for code, 2 points for explanation

7. *8 pts.* Assume a single-level page table with the following parameters: 2^{32} bytes of addressable physical memory; page size of 2^{11} bytes; 2^{28} bytes of logical address space, each page table entry includes a valid bit, a read-only bit, and an execute-allowed bit.

   (a) How many bytes in each Page Table Entry?
   Solution: 3 (21 + 1 + 1 + 1) / 8. 4 is OK *only* if state PTEs are word-aligned).

   (b) How many bytes in a frame?
   Solution: 2^{11}: same as the page size

   (c) How many bits in the physical address specify the frame?
   Solution: 21: 32 (entire address) - 11 (offset)

   (d) How many bytes in the page table?
   Solution: $3 \cdot 2^{17}$ bytes.

8. *14 pts.* The following table lists the arrival time and execution time of 5 threads.

Thread	Arrival time	Execution time
A	0	6
B	2	3
C	4	2
D	7	7

Fill in which thread is running at each time using the following scheduling algorithms: non-preemptive Shortest-Job-First (SJF), Round-robin (quantum=2) (RR), First-come, first-served (FCFS). Assume for the latter two that when a job arrives, it's put at the end of the queue

Fill in the following table showing what thread is running at any given time. Assume that context-switch time is negligible.

Time slot	SJF	RR	FCFS
0	A	A	A
1	A	A	A
2	A	B	A
3	A	B	A
4	A	A	A
5	A	A	A
6	C	C	B
7	C	C	B
8	B	B	B
9	B	A	C
10	B	A	C
11	D	D	D
12	D	D	D
13	D	D	D
14	D	D	D
15	D	D	D
16	D	D	D
17	D	D	D
18			
19			
20			

Solution: 2 points for each correct column (-1/2 for each error in each column).

What are the wait and turnaround times for each thread?

Time	SJF	RR	FCFS
Thread A wait time	0	5	0
Thread A turnaround time	6	11	6
Thread B wait time	6	0	4
Thread B turnaround time	9	3	7
Thread C wait time	2	2	5
Thread C turnaround time	4	4	7
Thread D wait time	4	4	4
Thread D turnaround time	11	11	11

Solution: 1 point for every correct row (based on your table in part 1).

9. *10 pts.* An operating system has 3 types of resources with the following quantities of each type:

Resource Type	R1	R2	R3
Quantity of that type	12	9	12

There are 5 threads. Each has declared the maximum number of resources of each type it will need, and each has allocated some quantity so far:

	R1	R2	R3
Thread 1 Max	4	3	4
Thread 1 Allocated	2	1	3
Thread 2 Max	5	3	3
Thread 2 Allocated	1	2	3
Thread 3 Max	6	4	3
Thread 3 Allocated	5	4	3
Thread 4 Max	4	1	2
Thread 4 Allocated	2	1	2

Show *all* of the calculations (including any intermediate matrices) to determine whether or not this system is in a *safe state*.

Solution:

`available` matrix:

R1	R2	R3
2	1	1

`needs` matrix:

	R1	R2	R3
Thread 1	2	2	1
Thread 2	4	1	0
Thread 3	1	0	0
Thread 4	2	0	0

Here's the order of threads we can first give their entire need, and then wait for them to return their entire allocation:

Thread	avail=avail-need	avail=avail+allocated
Thread 3	(2 1 1) - (1 0 0) = (1 1 1)	(1 1 1) + (6 4 3)= (7 5 4)
Thread 4	(7 5 4) - (2 0 0) = (5 5 4)	(5 5 4) + (4 1 2)= (9 6 6)
Thread 1	(9 6 6) - (2 2 1) = (7 4 5)	(7 4 5) + (4 3 4)= (11 7 9)
Thread 2	(11 7 9) - (4 1 0) = (7 6 9))	(7 6 9) + (5 3 3)= (12 9 12)

Since there's an order of satisfying all possible requests, we're in a safe state.

Points:
3 for calculating `needs`. 3 for calculating `available`

2 for saying "yes" 4 points for explanation