University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Spring 2001 Anthony D. Joseph

Midterm Exam *Solutions*

March 7, 2001
CS162 Operating Systems

Your Name:	
SID AND 162 Login:	
TA:	
Discussion Section:	

General Information:
This is a **closed book and notes** examination. You have ninety minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points given to the question; there are 100 points in all. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

Problem	Possible	Score
1	12	
2	24	
3	23	
4	18	
5	23	
Total	**100**	

1. (12 points total) "Lightweight" versus "Heavyweight" processes – Pros and Cons:
   a. (4 points) List one advantage of running several single-threaded applications as "heavyweight" processes over running the applications as multiple "lightweight" processes, all in one address space. Be explicit, but *concise*, in your answers.
      i)

      4 points for saying fault tolerance because of protection, 3 points for saying simple to implement and debug without saying why, 2 points for saying no synchronization needed without shared memory (with some justification), and 0 points for saying no synchronization needed without justification.

   b. (8 points) List two reasons why lightweight processes are better than heavyweight ones.
      i)

      4 points for each of the following reasons:

      - *Easier to share data between threads in the same address space than between processes*
      - *Faster to switch between threads than between processes*
      - *Sharing of code portion (or other portions) of applications possible versus heavyweight processes (not covered yet)*

      2 points for modularity, 1 point for less system resources (w/o significant specific advantages) or other minor advantages, and 0 points for concurrency or other statements not specific to lightweight processes.

      ii)

2. (24 points total) Suppose that we have a multiprogrammed computer in which each job has identical characteristics. In one computation period, *T*, for a job, half the time is spent in I/O and the other half in processor activity. Each job runs for a total of *N* periods. Assume that a simple round-robin scheduling scheme is used and that I/O operations can overlap with processor operation. We define the following quantities:
   - Turnaround time = actual time to complete a job.
   - Processor utilization = percentage of time that the processor is active (not waiting).

   For large N, compute approximate values for these quantities for one, two, and four simultaneous jobs, assuming that the period *T* is distributed in each of the following ways:

a. (15 points) I/O first half, processor second half.
i) 1 job, Turnaround time and Processor utilization:

5 points, 3 for correct answer for TAT and 2 for correct utilization.
TAT = NT, CPU utilization = 50%.

For the turnaround time, we accepted anything that was the correct answer plus or minus constant (dependent on T) factors. If you had any fraction of NT other than the correct answer (2/3 NT) this is ***wrong****.*

Where we could follow your logic, we gave credit for answers that were left unreduced. Many people left very complex expressions like: (7/4NT + 5/4NT + 3/4NT + 1/4NT) / 4. If we could see that it evaluated to the correct answer, they received credit. Most of these did not evaluate to the correct answer. Please actually read the question in the future. We were looking for APPROXIMATE values for large N.

We subtracted one point if the correct answers was given except that you wrote T or N instead of NT. "N periods" received full credit.

For utilization, we accepted any ratio that evaluated to the correct % as N goes to infinity. We accepted 99% for 100%, even though this is not truly correct. We saw a lot of answers like "utilization = NT". Yes, the CPU spends NT in computation (for 2 threads), but the question asks for percentage very clearly. We accepted (1 - t/n)% because we could see what they meant, even those this too is incorrect.

We rejected any answers that had two different values for the same quantity, even if one of them is right. Several people wrote multiple answers, such as "NT or 5/4 NT". If you guess, you have a 50% chance of getting it right. If you write them both, it's wrong.

Common errors:

The most common error was NT, 2NT, 4NT, with 50% utilization for all. If you did this, you ended up with a total of 8 points. The next most common error was just getting the wrong numbers (e.g., 7/4NT for 4 threads). Many people wrote completely impossible answers: NT for one thread, 2/3NT for two threads.

ii) 2 jobs, Turnaround time and Processor utilization:

TAT = NT, CPU utilization = 100%.

iii) 4 jobs, Turnaround time and Processor utilization:

TAT = (2N – 1)T, CPU utilization = 100%. We also accepted 2NT for TAT.

b. (9 points) I/O first and fourth quarters, processor second and third quarters.
i) 1 job, Turnaround time and Processor utilization:

2 points for TAT and 1 for utilization. Same answer as part a for 1 job.

ii) 2 jobs, Turnaround time and Processor utilization:

Same answer as above for part a for 2 jobs.

iii) 4 jobs, Turnaround time and Processor utilization:

Same answer as above for part a for 4 jobs.

3. (23 points total) CPU scheduling.
a. (8 points) Given CPU-bound tasks and a choice between FIFO and Round-Robin scheduling algorithms, choose the best algorithm for each of the following systems and specify why you chose the algorithm.
i) Multiprogrammed batch system:

2 points for correct answer, 2 points for good reason, 1 point for mediocre reason. FIFO is the optimal choice, since in a batch system, the primary concern is with throughput, and the less context switching, the more processing time is available for the process. Policies that minimize context switching are best.
We awarded 1 point for swapped incorrect answers for each part, IF there was a reasonable justification that indicated the student understood the tradeoffs involved.

ii) Interactive, time-sharing system:

Round-robin is the optimal choice, since the primary concern is with turnaround time. Time-slicing is preferred because it gives all the processes access to the processor over a short period of time.

b. (15 points) Consider the following processes, arrival times, and CPU processing requirements:

Process Name	Arrival Time	Processing Time
1	0	4
2	1	4
3	4	3
4	8	2

For each of the following scheduling algorithms, fill in the table with the process that is running on the CPU. Assume a 1 unit timeslice for timeslice-based algorithms. For RR, assume that an arriving thread runs at the beginning of its arrival time.

Time	**FIFO**	**RR**	**SRTF**
0	**1**	**1**	**1**
1	*1*	*2*	*1*
2	*1*	*1*	*1*
3	*1*	*2*	*1*
4	*2*	*3*	*3*
5	*2*	*1*	*3*
6	*2*	*2*	*3*
7	*2*	*3*	*2*
8	*3*	*4*	*4*
9	*3*	*1*	*4*
10	*3*	*2*	*2*
11	*4*	*3*	*2*
12	*4*	*4*	*2*
Average response time	*(4+7+7+5)/4 = 23/4 = 5.75*	*(10+10+8+5)/4 = 33/4 = 8.25*	*(4+1+3+2) = 21/4 = 5.25*

We awarded 5 points per column (scheduling policy). We subtracted 1 point for an error in each entry and 1 point for incorrectly choosing the run next job to run. Note that we only subtracted 1 point if you had consecutive errors in cells because of an error made in a previous cell.

A lot of people did not follow the statement: "For RR, assume that an arriving thread runs at the beginning of its arrival time". Also, the definition of SRTF states that if the arriving thread has less remaining time, it runs first.

Many people also made addition mistakes, but we did not take off points if you had the correct response times. Note that the definition of Response Time is Finish time – Arrival time.

No Credit – Problem X: (000000000000 points)

Late Night Fun in the Labs (*don't try this tonight*)

From a student in CS ###:
For a computer programming class, I sat directly across from someone, and our computers were facing away from each other. A few minutes into the class, he got up to leave the room. I reached between our computers and switched the inputs for the keyboards. He came back and started typing and immediately got a distressed look on his face.

He called the Lab TA over and explained that no matter what he typed, nothing would happen. The TA tried everything. By this time I was hiding behind my monitor and quaking red-faced.

I started to type, "Leave me alone!"

They both jumped back, silenced. "What the #$?!#$" the TA said.

I typed, "I said leave me alone!"

The student got real upset. "I didn't do anything to it, I swear!" It was all I could do to keep from laughing out loud. The conversation between them and HAL 2000 went on for an amazing five minutes.

Me: "Don't touch me!"

Student: "I'm sorry, I didn't mean to hit your keys that hard."

Me: "Who do you think you are anyway?!" Etc. Finally, I couldn't contain myself any longer and fell out of my chair laughing.

After they had realized what I had done, they both turned beet red.

Funny, I never got more than a C- in that class.

4. (18 points total) Concurrency problem: Building H_2O.

The goal of this exercise is for you to create a monitor with methods `Hydrogen()` and `Oxygen()`, which wait until a water molecule can be formed and then return. Don't worry about actually creating the water molecule; instead only need to wait until two hydrogen threads and one oxygen thread can be grouped together. For example, if two threads call Hydrogen, and then a third thread calls Oxygen, the third thread should wake up the first two threads and they should then all return.

a. (6 points) Specify the correctness constraints. Be succinct and explicit in your answer.

1) Each hydrogen thread waits to be grouped with one other hydrogen and oxygen before returning.
2) Each oxygen thread waits for two other hydrogens before returning.
3) Only one thread access shared state at a time

3 good answers received 2 points each. One point for partially correct. Full four points if the two correctness constraints for hydrogen and oxygen were CORRECTLY combined into one constraint.

b. (12 points) Observe that there is only one condition any thread will wait for (i.e., a water molecule being formed). However, it will be necessary to signal hydrogen and oxygen threads independently, so we choose to use two condition variables, `waitingH` and `waitingO`.

State variable description	Variable name	Initial value
Number of waiting hydrogen threads	`wH`	`0`
Number of waiting oxygen threads	`wO`	`0`
Number of active hydrogen threads	`aH`	`0`
Number of active oxygen threads	`aO`	`0`

You start with the following code:

```
Hydrogen() {
  wH++;
  lock.acquire();
  while (aH == 0) {
    if (wH >= 2 && wO >= 1) {
      wH-=2; aH+=2;
      wO-=1; aO+=1;
      waitingH.broadcast();
      waitingO.signal();
    } else {
      lock.release();
      waitingH.wait();
      lock.acquire();
    }
  }
  lock.release();
  aH--;
}
```

```
Oxygen() {
  wO++;
  while (aO == 0) {
    if (wH >= 2 && wO >= 1) {
      wH-=2; aH+=2;
      wO-=1; aO+=1;
      waitingH.signal();
      waitingH.signal();
    } else {
      waitingO.broadcast();
    }
  }
  aO--;
}
```

For each method, say whether the implementation either (i) works, (ii) doesn't work, or (iii) is dangerous – that is, sometimes works and sometimes doesn't. If the implementation does not work or is dangerous, explain why (there maybe several errors) and show how to fix it so it does work. Also, list and fix any inefficiencies.

i. `Hydrogen()`

Seven points total: 1 for say the routine is (iii) dangerous or (ii) doesn't work (the latter is acceptable only if one of the bugs in the student's answer is that the OS would crash if we called wait() after releasing the lock, since that's what Nachos does). Two points for each of three bugs and associated fixes (if the bugs mentioned implied a fix, then full credit was given):

1. *Starvation may occur because the method releases the lock before waiting.*
2. *The state variables wH and aO are modified outside of critical sections*
3. *Inefficient: The broadcast should be a signal.*

We awarded full credit to students that said you cannot wait on a condition variable without originally holding the lock, in place of saying that the lock release before the wait would cause starvation.

We took off 1 point for any additional bugs, since there were no others (except for a complete rewrite of the implementation – which wasn't what we were looking for).

ii. `Oxygen()`

Five points total: One point for (ii) doesn't work.
Two points for each reason:

1. *The broadcast should be a wait.*
2. *No locks around state variables.*

5. (23 points) Deadlock:

Consider the following snapshot of a system with five processes (p1, ... p5) and four resources (r1, ... r4). There are no current outstanding queued unsatisfied requests.

currently available resources

r1	r2	r3	r4
2	1	0	0

	current allocation				max demand				still needs			
Process	**r1**	**r2**	**r3**	**r4**	**r1**	**r2**	**r3**	**r4**	**r1**	**r2**	**r3**	**r4**
p1	0	0	1	2	0	0	1	2	*0*	*0*	*0*	*0*
p2	2	0	0	0	2	7	5	0	*0*	*7*	*5*	*0*
p3	0	0	3	4	6	6	5	6	*6*	*6*	*2*	*2*
p4	2	3	5	4	4	3	5	6	*2*	*0*	*0*	*2*
p5	0	3	3	2	0	6	5	2	*0*	*3*	*2*	*0*

a. (5 points) Compute what each process still might request and fill in the "still needs" columns.

One point for each correct row.

b. (8 points) Is this system currently deadlocked, or will any process become deadlocked? Why or why not? If not, give an execution order.

Four points for correct answer: not deadlocked and will not become deadlocked. Four points for using the Banker's algorithm, to determine the process finishing order: p1, p4, p5, p2, p3.

c. (10 points) If a request from p3 arrives for (0, 1, 0, 0), can that request be safely granted immediately? In what state (deadlocked, safe, unsafe) would immediately granting the whole request leave the system? Which processes, if any, are or may become deadlocked if this whole request is granted immediately?

Change available to (2, 0, 0, 0) and p3's row of "still needs" to (6, 5, 2, 2). Now p1, p4, and p5 can finish, but with available now (4, 6, 9, 8) neither p2 nor p3's "still needs" can be satisfied. So, it is not safe to grant p3's request. Correct answer NO (2 points). State is unsafe (4 points) as the system may or may not deadlock. Processes p2 and p3 may deadlock (4 points, each incorrect one was –2 points).