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

Spring 2006 Anthony D. Joseph

Midterm Exam *Solutions*

March 8, 2006
CS162 Operating Systems

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

General Information:
This is a **closed book and notes** examination. You have 90 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	21	
2	54	
3	12	
4	13	
Total	**100**	

1. (21 points total) Short answer questions:

a. (4 points) True/False and Why?
Lottery scheduling can be used to implement any other scheduling algorithm.

TRUE FALSE

Why?
FALSE. *Lottery scheduling cannot implement strict priority scheduling, FIFO, or real-time scheduling. The correct answer was worth 2 points and the justification was worth an additional 2 points.*

b. (5 points) Inverted Page Tables:

i) (3 points) Give a two to three sentence description of an inverted page table.
An inverted page table uses a hash function to map virtual addresses to physical addresses..

ii) (2 points) Briefly (2-3 sentences) state the problem it is intended to solve.
An inverted page table supports a large virtual address space while only requiring a physical page table that is proportional in size to the available physical memory (an forward page table would require physical memory proportional to the size of virtual memory). We did not give credit for answers that stated the only problem it solves is reducing the number of lookups to memory.

c. (4 points) Why would two processes want to use shared memory for communication instead of using message passing? Your answer should be brief (no more than 5 sentences).
Shared memory has ***higher performance*** *because you* ***avoid the context switch overhea****d of using message passing through the kernel. We gave full credit only if your answer included both performance and context switch overhead.*

d. (4 points) We say that the operating system is an “illusionist”. Name two illusions that it provides:

i) (2 points) Illusion #1:

Any of the many illusions we’ve discussed were acceptable: exclusive access to CPU, memory, disk or I/O devices; “infinite” memory; etc..

ii) (2 points) Illusion #2:

See above.

e. (4 points) For system calls, we explained that arguments are sanity checked twice.

i) (2 points) When is each check performed?

Before entering the kernel and after entering the kernel.

ii) (2 points) Why is this redundancy important?

The first check is a sanity check of the arguments, the second check is for security purposes to detect race conditions between the current thread and other threads that attempt to maliciously modify the arguments (Time of Check to Time of Use vulnerability). We only gave full credit for answers that explicitly mentioned the race condition.

2. (54 points total) CPU Scheduling. Here is a table of processes and their associated arrival and running times.

Process ID	**Arrival Time**	**Expected CPU Running Time**
Process 1	0	5
Process 2	3	5
Process 3	5	3
Process 4	7	2

a. (15 points) Show the scheduling order for these processes under First-In-First-Out (FIFO), Shortest-Job First (SJF), and Round-Robin (RR) with a quantum = 1 time unit. *Assume that the context switch overhead is 0 and new processes are added to the* ***head*** *of the queue except for FIFO.*

Time	**FIFO**	**SJF**	**RR**
0	*1*	*1*	*1*
1	*1*	*1*	*1*
2	*1*	*1*	*1*
3	*1*	*1*	*2*
4	*1*	*1*	*1*
5	*2*	*3*	*3*
6	*2*	*3*	*2*
7	*2*	*3*	*4*
8	*2*	*4*	*1*
9	*2*	*4*	*3*
10	*3*	*2*	*2*
11	*3*	*2*	*4*
12	*3*	*2*	*3*
13	*4*	*2*	*2*
14	*4*	*2*	*2*
15			

b. (18 points) For each process in each schedule above, indicate the queue wait time and turnaround time (TRT).

Scheduler	Process 1	Process 2	Process 3	Process 4
FIFO queue wait	*0*	*2*	*5*	*6*
FIFO TRT	*5*	*7*	*8*	*8*
SJF queue wait	*0*	*2*	*7*	*3*
SJF TRT	*5*	*7*	*10*	*5*
RR queue wait	*4*	*7*	*5*	*3*
RR TRT	*9*	*12*	*8*	*5*

The queue wait time is the *total* time a thread spends in the wait queue. The turnaround time is defined as the time a process takes to complete after it arrives.

Part a) 5 points per column, part b) 3 points per row. We deducted one point per error up to the maximum score for the column/row. We deducted one point for a wrong overall arrival time.

c. (6 points) Approximate Shortest Remaining Time First.

i) (3 points) How can you approximate Shortest Remaining Time First scheduling without knowing how long a job takes ahead of time?

You can approximate it by looking at the past CPU burst behavior to predict the future CPU burst behavior. Exceeding a quantum indicates the job is long running, while yielding before the quantum indicates the job is short running.

ii) (3 points) How would you implement this approximation in a Lottery Scheduler?

The basic solution is to take/give tickets based upon CPU usage. We gave full credit for answers that explained how you give or take tickets in a quantum period based on CPU usage, and starting with one ticket.

d. (15 points) Scheduling Algorithm Pros and Cons. For each of the four algorithms discussed in class, list the pros and cons of the algorithm, and give one example situation where each algorithm would be a bad choice.

i) (3 points) First-Come, First-Served.

(1) Pros:

The answers were the ones we talked about in class, or in the book. We gave one point for each pro/con answer.

Easy to implement. Fair to all in order arrived.

(2) Cons:

Short jobs can be stuck behind long jobs – "convoy" effect.

(3) Bad choice example:

One long job followed by a short job..

ii) (3 points) Round Robin.

(1) Pros:
Easy to implement. Good for short jobs (they don't get stuck behind long jobs)..

(2) Cons:
Context switch times are a high overhead for long jobs. High TRT for equal length jobs..

(3) Bad choice example:
Multiple equal length jobs..

iii) (3 points) Shortest Job First/Shortest Remaining Time First

(1) Pros:
Optimal algorithm (i.e., best response time). Saying just "optimal" was not given points..

(2) Cons:
Long jobs are starved. Predicting the future is hard..

(3) Bad choice example:
An example with long jobs being starved or programs doing extra I/O in order to appear to have short CPU bursts.

iv) (3 points) Lottery

(1) Pros:
Lowest priority jobs not starved. Highest priority jobs prioritized over lower priority jobs. Flexible (adaptable to behave as other algorithms).

(2) Cons:
Hard to debug. Cannot exactly control or predict next job..

(3) Bad choice example:
Some example that shows necessity to tightly control what executes next, which may make lottery worse than priority (given that it tries to improve on priority). We didn't give credit for simply making an argument about overhead or approximation of RR when equal-sized jobs. The same applies to iv) 2.

No Credit – **Problem X** (000000000000 points)

The 2005 Ig Nobel Prize Winners

The 2005 Ig Nobel Prizes were awarded on Thursday October 6, 2005 at the 15th First Annual Ig Nobel Prize Ceremony, at Harvard's Sanders Theatre.

AGRICULTURAL HISTORY: James Watson of Massey University, New Zealand, for his scholarly study, "The Significance of Mr. Richard Buckley's Exploding Trousers."
REFERENCE: "The Significance of Mr. Richard Buckley's Exploding Trousers: Reflections on an Aspect of Technological Change in New Zealand Dairy-Farming between the World Wars," James Watson, Agricultural History, vol. 78, no. 3, Summer 2004, pp. 346-60.

PHYSICS: John Mainstone and the late Thomas Parnell of the University of Queensland, Australia, for patiently conducting an experiment that began in the year 1927 – in which a glob of congealed black tar has been slowly, slowly dripping through a funnel, at a rate of approximately one drop every nine years.
REFERENCE: "The Pitch Drop Experiment," R. Edgeworth, B.J. Dalton and T. Parnell, European Journal of Physics, 1984, pp. 198-200.

MEDICINE: Gregg A. Miller of Oak Grove, Missouri, for inventing Neuticles – artificial replacement testicles for dogs, which are available in three sizes, and three degrees of firmness.
REFERENCES: US Patent #5868140, and the book Going Going NUTS!, by Gregg A. Miller, PublishAmerica, 2004, ISBN 1413753167.

LITERATURE: The Internet entrepreneurs of Nigeria, for creating and then using e-mail to distribute a bold series of short stories, thus introducing millions of readers to a cast of rich characters – General Sani Abacha, Mrs. Mariam Sanni Abacha, Barrister Jon A Mbeki Esq., and others -- each of whom requires just a small amount of expense money so as to obtain access to the great wealth to which they are entitled and which they would like to share with the kind person who assists them.

PEACE: Claire Rind and Peter Simmons of Newcastle University, in the U.K., for electrically monitoring the activity of a brain cell in a locust while that locust was watching selected highlights from the movie "Star Wars."
REFERENCE: "Orthopteran DCMD Neuron: A Reevaluation of Responses to Moving Objects. I. Selective Responses to Approaching Objects," F.C. Rind and P.J. Simmons, Journal of Neurophysiology, vol. 68, no. 5, November 1992, pp. 1654-66.

ECONOMICS: Gauri Nanda of the Massachusetts Institute of Technology, for inventing an alarm clock that runs away and hides, repeatedly, thus ensuring that people DO get out of bed, and thus theoretically adding many productive hours to the workday.

CHEMISTRY: Edward Cussler of the University of Minnesota and Brian Gettelfinger of the University of Minnesota and the University of Wisconsin, for conducting a careful experiment to settle the longstanding scientific question: can people swim faster in syrup or in water?
REFERENCE: "Will Humans Swim Faster or Slower in Syrup?" American Institute of Chemical Engineers

NUTRITION: Dr. Yoshiro Nakamats of Tokyo, Japan, for photographing and retrospectively analyzing every meal he has consumed during a period of 34 years (and counting).

FLUID DYNAMICS: Victor Benno Meyer-Rochow of International University Bremen, Germany and the University of Oulu , Finland; and Jozsef Gal of Loránd Eötvös University, Hungary, for using basic principles of physics to calculate the pressure that builds up inside a penguin, as detailed in their report "Pressures Produced When Penguins Pooh -- Calculations on Avian Defaecation."
PUBLISHED IN: Polar Biology, vol. 27, 2003, pp. 56-8.

3. (12 points total) Concurrency problem: Dining Philosophers.
The goal of this exercise is to implement a solution to the Dining Philosophers problem using *only* semaphores (your solution may not use locks, monitors, or other synchronization primitives). Create a method `Dine()`, which waits until a diner has two chopsticks and can eat, then calls `Eat()`, and then releases the chopsticks before returning. Your solution should allow multiple philosophers to eat at the same time (as long as there are sufficient chopsticks in a pile in the middle of the table). Assume you are given the variable, `chopsticks`, which starts off initialized to the total number of chopsticks available. *Your solution should avoid deadlock.*

a. (4 points) Specify the correctness constraints. Be succinct and explicit in your answer.

A diner waits for two chopsticks (4 points).
We gave one point for some other type of related constraint (e.g., you can't steal chopsticks, there's one chopstick per diner, etc.) We subtracted one point for saying something that is implementation dependent or otherwise not a correctness constraint.

b. (8 points) Implement the `Dine()` method.
The key insight is to set the initial value of the semaphore to half the number of chopsticks. Then, each P operation effectively grabs two chopsticks and each V operation relinquishes two chopsticks.

```
Int chopsticks = N;         // Total number of chopsticks
Semaphore sticks = new Semaphore(floor(chopsticks/2));

    Dine() {
      sticks.P();           // This "acquires" two chopsticks
      Eat();
      sticks.V();           // This "releases" two chopsticks
    }
```

-8 pts: using other synchronization primitives (i.e., locks, CVs, interrupts)
-7 pts: didn't enforce correctness constraint (need two chopsticks to eat).
-7 pts: trying to use a semaphore in an illegal way (e.g., trying to check the value)
-6 pts: only one diner can eat at a time.
-5 pts: no call to Eat()
-5 pts: deadlock possible
-5 pts: busy waiting
-5 pts: solution fails if chopsticks is initially set to some value
-1 pt: extraneous/redundant semaphores
-1 pt: didn't show initialization of semaphore value

Another solution:

```
Int chopsticks = N;        // Total number of chopsticks
Semaphore Lock = new Semaphore (1);    // mutex
Semaphore Sem1 = new Semaphore (chopsticsk);  // scheduling
Dine {
  Lock.P();          // This “acquires” mutex
  Sem1.P();          // This “acquires” one chopstick
  Sem1.P();          // This “acquires” one chopstick
  Lock.V();          // This “releases” mutex
  Eat();
  Sem1.V();          // This “releases” one chopstick
  Sem1.V();          // This “releases” one chopstick

}
```

Yet another solution:

```
Int chopsticks = N;        // Total number of chopsticks
Semaphore reach = new Semaphore (1);    // mutex
Semaphore waiting = new Semaphore (0);  // scheduling

    Dine() {
      reach.P();           // This “acquires” mutex
      while (chopsticks < 2) {
        reach.V();         // This “releases” mutex
        wait.P();          // This “waits” on semaphore
        reach.P();         // This “acquires” mutex
      }
      chopsticks -= 2;
      reach.V();           // This “releases” mutex
      Eat();
      Reach.P();           // This “acquires” mutex
      chopsticks += 2;
      reach.V();           // This “releases” mutex
      wait.V();            // This “wakes” waiters
    }
```

4. (13 points) Virtual Memory.
The Lemon company has hired you to design the virtual memory system for their new line of desktop computers, the iniM caM. Each computer will have 32 bit virtual and physical addresses, and memory will be allocated in 2 KByte pages.

a. (4 points) For a single-level page table, how many bits will be used to index the page, and how many will be the offset within the page?
i) (2 points) Number of bits for page number?
Using the answer of 11 bits from ii), the remainder is 32-11 = 21 bits.

ii) (2 points) Number of bits for offset within the page?
Each page is 2 Kbytes in size which equals 2^{11} bits, so 11 bits are needed for the offset.

b. (5 points) Each page table entry will also include three bits for bookkeeping (Valid, Read, and Write bits).
i) (2 points) How many bytes are required for each page table entry?
The page table entry contains the physical page number (21 bits) and the 3 bookkeeping bits, so the total size is 24 bits or 3 bytes. We deducted one point for saying the PTE included virtual page numbers or for not converting the value into bytes. We allowed the answer of 4 bytes only if you stated that PTEs are word aligned and padded.

ii) (3 points) How much physical memory is required to store the table?
The table requires 3 x 2^{21} bytes (6 Mbytes). For consistency, if we accepted your answer of 4 bytes for part i), we expected an answer of 4 x 2^{21} bytes (8 Mbytes).

c. (4 points) If the iniM caM has 16 megabytes or less of physical memory, we can use 24 bit physical addresses (and still have 32 bit virtual addresses). How large would each page table entry and the entire table be now?
The PTE would now require 24-11 = 13 bits for the page portion, adding the 3 bits for bookkeeping, yields a PTE of 16 bits or 2 bytes. The table will now be 2 x 2^{21} bytes (4 Mbytes). We did not accept answers that assumed the PTE would have 2^{13} entries, since only an inverted table would have 2^{13} entries, but it would have a larger PTE.