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

Spring 2004 Anthony D. Joseph

Midterm Exam *Solutions*

March 18, 2004
CS162 Operating Systems

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

General Information:
This is a **closed book and notes** examination. You have 120 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	17	
2	16	
3	27	
4	22	
5	18	
Total	**100**	

1. (17 points total) Short answer questions:
   a. (12 points) True/False and Why?
      i) (4 points) Because of the overhead of context switching, programs that use threads will always take longer to execute than programs that do not use threads.

TRUE FALSE

Why?

FALSE. *Threading allows I/O to be overlapped with computation which can provide substantial performance improvements.*
4 points for each. 2 points for partial answers.

      ii) (4 points) Protection and isolation between applications and between applications and the operating system can be provided without hardware support.

TRUE FALSE

Why?

TRUE. *Using strong typing or software fault isolation can provide the same protection and isolation capabilities as hardware.*

      iii) (4 points) After a UNIX fork operation, the parent and new child process are identical in all respects.

TRUE FALSE

Why?

FALSE. *One register is different between the parent and child, and is used to determine which process is the parent and which is the child..*

   b. (5 points) Consider a system with a mixture of I/O bound processes and CPU bound processes
      i) (3 points) Explain how this mixture of processes maximizes system utilization:
      We want to keep all parts of the system busy by overlapping I/O operations with CPU operations. Overall, if we have some processes using the CPU while others are waiting on I/O, then the system will be more utilized.

ii) (2 points) Explain why this combination is more important in batch systems than it is on most computers sitting around in our department:
In general, our systems are underutilized (there are a lot of wasted cycles both on I/O devices as well as CPU), and user interaction is more important. Batch systems were expensive systems that were busy all the time.

2. (16 points total) CPU Scheduling.

a. (6 points) 5 identical jobs are run once on a non-preemptive scheduler, and then again on a preemptive scheduler using a round robin scheduling discipline. The jobs are purely computational - they do almost no I/O. On the non-preemptive scheduler it takes 24 hours for all 5 of them to complete; on the preemptive one it takes 24 hours and 2 minutes for all 5 of them. If the time quantum of the preemptive scheduler is 0.05 sec (50 milliseconds), how long does a context switch take? (Show **all** your work; it's OK to leave the answer in symbolic / long form).

The non-preemptive time is $5y + 4x = 24 * 60 * 60$, or $y = (-4/5\, x + 24 * 60 * 12)$
The preemptive time is $(y/0.05 * 5 - 1) * x + 5y = (24 * 60 + 2) * 60$

*Over the 24 hour period there are 24 hrs * 60 min/hr * 60 sec/min * 20 switches/sec = 1728000 context switches. These take 120 seconds. Each context switch takes 120 / 1728000 sec = 1/14400 sec = 69.4 microseconds*
We took off four points if you left out the 20 switches/sec. We deducted six points if you mixed up the 24 hours with 2 minutes, deducted two points for subtracting 5 from something incorrect, and took off two points for the inverse answer.
We subtracted two points if you had 0.05 and 20 in the wrong places.
If you skipped the context switches, we subtracted 5 points

b. (4 points) Name 2 things that need to be saved on a context switch.

CPU registers, Program Counter, Stack Pointer, Page Table Base Register, Segment Table

We subtracted one point for saving the ppn or vpn, one point for saving the page table, one point for saving the stack space, 2 points for saving the address space, and two points for saving thread states

c. (6 points) Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound processes and yet not permanently starve CPU-bound processes?
We awarded three points for saying that I/O bound processes generally sit idle for relatively long periods of time. When they finally get their I/O, they will not have run for a long time and will therefore be favored in scheduling relative to processes which have just been running. We awarded an additional three points for saying that if CPU-bound processes get blocked long enough by I/O-bound processes, they too will eventually not have run for a long time and they will be scheduled. Also, if there aren't too many I/O-bound processes, the CPU-bound ones will get to run when the I/O-bound ones are blocked.

We subtracted two points for not explicitly stating that I/O uses less CPU, subtracted two points for saying that the CPU-bound processes won't run until the end, and subtracted two points for saying that round robin scheduling was the reason.
We deducted three points for saying that you have to wait until all I/O-bound processes finish, subtracted three points for saying that the CPU-bound processes run because of interrupts, subtracted three points for saying that ***at first*** *CPU-bound process get to run.*

We took off two points for saying that the minimum priority process gets CPU time.

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

The following is an excerpt from The Washington Post's Style Invitational invitation to readers to: take any word from the dictionary, alter it by adding, subtracting, or changing one letter, and supply a new definition.

Here are this year's winners:

1. Reintarnation: Coming back to life as a hillbilly.

2. Bozone (*n.*): The substance surrounding stupid people that stops bright ideas from penetrating. The bozone layer, unfortunately, shows little sign of breaking down in the near future.

3. Giraffiti: Vandalism spray-painted very, very high.

4. Sarchasm: The gulf between the author of sarcastic wit and the person who doesn't get it.

5. Inoculatte: To take coffee intravenously when you are running late.

6. Hipatitis: Terminal coolness.

7. Osteopornosis: A degenerate disease. (This one got extra credit.)

8. Karmageddon: It's like, when everybody is sending off all these really bad vibes, right? And then, like, the Earth explodes and it's like, wow, a serious bummer.

9. Decafalon (*n.*): The grueling event of getting through the day consuming only things that are good for you.

10. Glibido: All talk and no action.

11. Dopeler effect: The tendency of stupid ideas to seem smarter when they come at you rapidly.

12. Arachnoleptic fit (*n.*): The frantic dance performed just after you've accidentally walked through a spider web.

13. Beelzebug (*n.*): Satan in the form of a mosquito that gets into your bedroom at three in the morning and cannot be cast out.

And the pick of the literature:
14. Caterpallor (*n.*): The color you turn after finding half a grub in the fruit you're eating.

3. (27 points total) Concurrency problem: Implementing Semaphores.

You are programming on a multiprocessor system using threads. The system includes monitors and condition variables, with the following classes and methods:

```
public class Monitor {
        public Monitor() {
                /* Creates a new monitor */
                …
        }

        public void Enter() {
                /* Enters the monitor */
                …
        }

        public void Exit() {
                /* Exits the monitor */
                …
                }
}
```

```
public class ConditionVariable {
        public ConditionVariable(Monitor mon) {
                /* Creates a condition variable
                   associated with monitor mon */
                …
        }
        public void Wait() {
                /* Blocks on the condition variable */
                …
        }
        public void Notify() {
                /* Wakes up one thread waiting on
                   cv, if there is such a thread */
                …
        }
        public void Broadcast() {
                /* Wakes up all threads waiting on cv,
                   if there are such threads */
                …
        }
}
```

Provide an implementation of general semaphores using this system. In other words, write the file Semaphore.java, implementing the following methods:

```
public class  Semaphore {
        public Semaphore(int initialValue) {
                /* Create and return a semaphore with initial value: initialValue*/
                …
                }
        public P() {
                /* Call P() on the semaphore */
                …
        }
        public V() {
                /* Call V() on the semaphore */
                …
        }
}
```

Write your solution on the following page.

3. (continued) Concurrency problem: Implementing Semaphores.

Write your solution here:

public class Semaphore {

Semaphores can be implemented with a monitor and a condition variable. The monitor protects access to the semaphore's value, and the condition variable lets you block/wakeup threads. Note too the < 0 in Semaphore.P() and =0 in Semaphore.V(). You can argue that the conditional in Semaphore.V() is unnecessary, for if the value is greater than zero, there will be no threads waiting on the conditional variable.

```
public class  Semaphore {
        private Monitor mon = null;
        private ConditionVariable  cv = null;
        int value;

        public Semaphore(int initialValue) throws InvalidInitValue{
                /* Create and return a semaphore with initial value: initialValue*/
                if (initialValue < 0) throw InvalidInitValue;
                mon = new Monitor();
                cv = new ConditionVariable();
                value = initialValue;
        }
        public P() {
                /* Call P() on the semaphore */
                mon.Enter();
                value--;
                if (value < 0) cv.Wait();
                /* OR "while (value <= 0) cv.Wait();   value--;" */
                mon.Exit();
        }
        public V() {
                /* Call V() on the semaphore */
                mon.Enter();
                value++;
                cv.Notify();  /* You could optionally include "if (value <= 0)" */
                mon.exit();
        }
}
```

- *We gave you one point for each correct class variable, but subtracted one point for each extraneous class variable Each of the three procedures in the class was worth 8 points, subject to deductions for errors.*
- *If you omitted monitors entirely, you received 0 points, or if you used interrupts or locks in a procedure, you received 0 points for the procedure (up to 16 points subtracted).*

- *We deducted three points if you did not sanity check the semaphore's initial value, deducted 6 points if you limited the initial value of the semaphore to a strict subset of ? , and deducted 8 points(from V) if you placed a max value on the semaphore.*
- *We deducted 16 points (8 each for P and V) if you used two monitors (one for condition variables and one for synchronization/mutual exclusion).*
- *If you accessed the queues of the condition variables, we deducted 12 points (6 each for P and V). If you accessed the private variables or methods of the monitor or condition variables, we deducted 2 points for each occurrence. If you violated the semantics of condition variables or monitors, we deducted 4 points for each violation.*
- *If you had an extra counter and incremented or decremented it out of order, we subtracted 2 points for each error. We deducted four points for not decrementing when value is 0.*
- *We deducted 1 point for each unnecessary check, 2 points for each extra or missing synchronization construct, 8 points for each non-atomic P() or V().*
- *If you used a while instead of an if in P(), we deducted three points. But, if the check is **before** the decrement and value is 0, then you must do a while **before** the else (otherwise, we deducted 3 points).*
- *If you waited in V(), we deducted 8 points). If you decremented after calling wait() in P() (and it's protected by value < 0), we deducted 6 points. If you mixed up your < and <='s, we deducted 4 points.*
- *If you used a broadcast without using a while(), we deducted 6 points, or if you used a broadcast with a while(), we deducted 3 points. We deducted 3 points for each unnecessary notify. If you inadvertently forgot a monitor enter or exit, we deducted 3 points. If you consistently forgot to include monitor enter and exits, we deducted four points.*
- *If you made minor Java mistakes, we deducted 3 points, for major Java mistakes, we deducted six points.*
- *If your answer didn't really make any sense, but you had the class set up right, we only gave you points for the class variables.*

4. (22 points) Deadlock:

A restaurant would like to serve four dinner parties, P1 through P4. The restaurant has a total of 8 plates and 12 bowls. Assume that each group of diners will stop eating and wait for the waiter to bring a requested item (plate or bowl) to the table when it is required. Assume that the diners don't mind waiting. The maximum request and current allocation tables are shown as follows:

Maximum Request	**Plates**	**Bowls**
P1	7	7
P2	6	10
P3	1	2
P4	2	4

Current Allocation	**Plates**	**Bowls**
P1	2	3
P2	3	5
P3	0	1
P4	1	2

a. (4 points) Determine the Need Matrix for plates and bowls.

Need	**Plates**	**Bowls**
P1	*5*	*4*
P2	*3*	*5*
P3	*1*	*1*
P4	*1*	*2*

b. (7 points) Will the restaurant be able to feed all four parties successfully? *Clearly explain your answer – specifically, why no or why/how there is a safe serving order*.

The vector of available resources is A = (2, 1).
First, serve P3, and A will be (2, 2).
Then, serve P4, and A will be (3, 4).
There are not enough resources available to serve P1 or P2. Therefore, the original resource allocation state is ***unsafe****. The restaurant cannot feed all four parties successfully.*

If you did not say that the allocation is unsafe, we deducted one point.
If you said "yes" or safe, we deducted 4 points.
If you said that there was deadlock or starvation, we deducted 3 points (saying you can't finish ever, cost2 points).
If your solution did not include the Banker's Algorithm or you were not specific in your explanation, we deducted 2 points

4. (continued) Deadlock

c. (11 points) Assume a new dinner party, P5, comes to the restaurant at this time. Their maximum needs are 5 plates and 3 bowls. Initially, the waiter brings 2 plates to them. In order to be able to feed all five parties successfully, the restaurant needs more plates.

i. (2 points) Determine the new Need Matrix for plates and bowls.

Need	Plates	Bowls
P1	*5*	*4*
P2	*3*	*5*
P3	*1*	*1*
P4	*1*	*2*
P5	*3*	*3*

ii. (6 points) At least how many plates would the restaurant need to add?

If add 1 plate, there are 9 plates and 12 bowls totally.
Initially, A = (1, 1).
Serve P3, A = (1, 2).
Serve P4, A = (2, 4).
So there are not enough resources for serving the 5 parties.

If add 2 plates, there are 10 plates and 12 bowls totally.
Initially, A = (2, 1).
Serve P3, A = (2, 2).
Serve P4, A = (3, 4).
Serve P5, A = (5, 4).
Serve P1, A = (7, 7).
Serve P2, A = (10, 12).

Therefore, at least 2 plates are needed.

We treated parts (ii) and (iii) as a combined 9 points.
If you had an incorrect number of plates, but the correct ordering, we deducted 5 points.
If you had an incorrect number of bowls, but the correct number of plates, and an incorrect ordering, we deducted 4 points.
If your had the correct number of plates, but an incorrect ordering, we deducted 6 points.

iii. (3 points) Show a safe serving sequence.

A safe serving sequence is P3-P4-P5-P1-P2.

5. (18 points) Paging:

Suppose you have a system with 32-bit pointers and 4 megabytes of physical memory that is partitioned into 8192-byte pages. The system uses an Inverted Page Table (IPT). Assume that there is no page sharing between processes.

a. (8 points) Describe what page table entries should look like. Specifically, how many bits should be in each page table entry, and what are they for? Also, how many page table entries should there be in the page table?

Virtual addresses are 32 bits, and split into two parts. The page number is the first 19 bits, and the offset within the page is the last 13 bits (2^{13} = 8,192).

Virtual Page Number	*Offset*
19 bits	*13 bits*

The inverted page table is a mapping of physical addresses to virtual addresses. Memory is 4 megabytes, partitioned into 512 pages. Therefore the inverted page table will consist of 512 entries. Each of these entries must have:

19 bits for the virtual page number of the physical page.
Some number of bits (16 in Unix) for the processs ID of the process that owns the page.
Protection bits (r/w/x)

We awarded two points each for: the correct number of IPT entries, storing the virtual page number in the IPT, storing the process ID in the IPT, and storing the protection bits(any reasonable set was accepted) in the IPT.
We subtracted one point for each extraneous item that you stored in the IPT, and subtracted one point for storing the physical page number instead of the virtual page number in the IPT.

b. (5 points) Describe how an IPT is used to translate a virtual address into a physical address.

A virtual address is translated to a physical address hashing virtual addresses (worth two points). Thus, in the normal case, the translation may be found in a few memory lookups rather than an entire table traversal. If it is found, and the owning process is equal to the current running process (owning PID check is worth 2 points), then its index in the table is the frame number of the physical page. If it is not found, then a memory fault must occur (not found fault is worth 1 point).
We also accepted a general search of the IPT, as described in the book and midterm review session.

c. (3 points) How can you make an IPT more efficient? *Explain your solution and how it works in detail.*
Add a Translation Lookaside Buffer (TLB), an associative memory that can perform fast lookup on virtual addresses and provide the translation in much less time than it takes to perform a memory access. TLB's are effective, especially when combined with caches, because programs tend to have locality in accessing pages. The use of a TLB was worth one point, the explanation was worth two points.

Alternatively, if you used the book solution for part (b), we accepted a hash-based enhancement for this part.

d. (2 points) What effect, if any, does your solution in part (c) have on what happens on a context switch?
On a context switch, the TLB must be flushed. That's it.

Alternatively, if you used the book solution for part (b) and a hash enhancement for part (c), then we accepted the answer "nothing" for this part.

This page intentionally left blank as scratch space.

Do not write answers on this page

Do not write answers on this page