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

Spring 1999 Anthony D. Joseph

Midterm Exam Solutions and Grading Guidelines

March 3, 1999
CS162 Operating Systems

Your Name:	
SID:	
TA:	
Discussion Section:	

General Information:
This is a **closed book** examination. You have two hours 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 (you needn't cover every available nano-acre with writing). If there is something in a question that you believe is open to interpretation, then please go ahead and interpret, **but** state your assumptions in your answer.

Good Luck!!

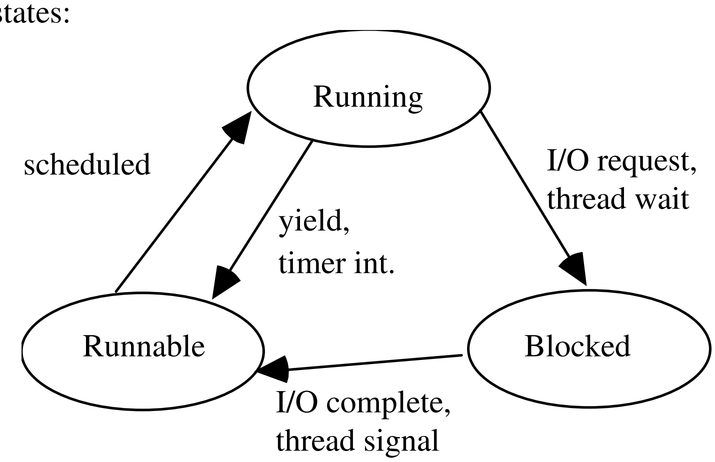
Problem	Possible	Score
1	13	
2	24	
3	18	
4	30	
5	15	
Total	**100**	

1. Threads (13 points total):
   a. (6 points) What state does a thread share with other threads in a process and what state is private/specific to a thread? Be explicit in your answer.

      Shared state:
      (1) Contents of memory (global variables, heap)
      (2) I/O state (file system)
      Having both answers is worth 3 points. Having one answer is worth 2 points.

      Private state:
      (1) CPU registers (including, program counter and stack pointer)
      (2) *Execution stack*
      Having both answers is worth 3 points. Having one answer is worth 2 points.

   b. (7 points) Draw a picture of the three states of a thread and label the transitions between the states:



      Each state and each arc is worth 1 point.

2. Context switching and CPU scheduling (24 points total):
   a. (3 points) What state is saved on a context switch between threads?

      CPU registers, program counter, and stack pointer.
      Each answer is worth 1 point.

   b. (6 points) List two reasons why Nachos disables interrupts when a thread/process sleeps, yields, or switches to a new thread/process?

      Interrupts are disabled for two reasons:
      (1) To prevent context switching from occurring while registers are being saved/restored.
      (2) To enable a thread/process to add itself to a queue (e.g., a wait queue) without allowing another thread to interrupt the action.

      Each answer is worth 3 points. If the answer was not specific about the potential conflict that could occur, we deducted 1 point.

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

Process Name	Arrival Time	Processing Time
1	0	3
2	1	5
3	3	2
4	9	2

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

Time	**FIFO**	**RR**	**SRTCF**
0	**1**	**1**	**1**
1	*1*	*2*	*1*
2	*1*	*1*	*1*
3	*2*	*3*	*3*
4	*2*	*2*	*3*
5	*2*	*1*	*2*
6	*2*	*3*	*2*
7	*2*	*2*	*2*
8	*3*	*2*	*2*
9	*3*	*4*	*2*
10	*4*	*2*	*4*
11	*4*	*4*	*4*
Average response time	*3+7+7+ 3/ 4 = 5*	*6+10+4 +3/ = 5.75*	*3+2+9+3/ 4= 4.25*

Each column is worth 5 points: 3 for correctness of the schedule (we deducted 1/2/3 points if you made minor/intermediate/major mistakes), and 2 for the average response time (1 point was deducted for minor errors).

3. Concurrency control (18 points total):

c. (6 points) Match the terms in column A with the most appropriate definition from column B.

Column A	Column B
1. Synchronization	a. Piece of code that only one thread can execute at once
2. Mutual exclusion	b. Ensuring that only one thread does a particular thing at a time
3. Critical section	c. Isolating program faults to an address space
	d. Using atomic operations to ensure cooperation between threads

1. *d*

2. *b*

3. *a*

Each answer is worth 2 points.

d. (12 points) For the following implementation of a banking application, say whether it 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. Note that `ThreadFork` does the obvious thing.

```
BankServer() {
       while (TRUE) {
              ReceiveRequest(&op, &acctId1, &acctId2, &amount);
              if (op == transfer) ThreadFork(Transfer(acctId1, acctId2, amount));
              else if …
       }
}

Transfer(acctId1, acctId2, amount) {
       account[acctId1]->Lock();
       acct1 = GetAccount(acctId1); /* May involve disk I/O */
       account[acctId2]->Lock();
       acct2 = GetAccount(acctId2); /* May involve disk I/O */
       if (acct1->balance < amount) return ERROR;
       acct1->balance -= amount; acct2->balance += amount;
       StoreAccount(acct1); /* Involves disk I/O */
       StoreAccount(acct2); /* Involves disk I/O */
       account[acctId1]->Unlock(); account[acctId2]->Unlock();
       return OK;
}
```

This implementation is (iii) dangerous, it will work sometimes (2 pts for this answer). Two problems: I/O may occur while the second account is being retrieved, leading to deadlock if there is a cycle of simultaneous transfers (A->B and B->A). Deadlock may also occur when an ERROR occurs, since the locks are not released (2 pts for describing each error). Solution: Acquire locks in the order of acctId values and release locks upon error. Each good solution is worth 3 pts. We deducted: 1 pt for solutions that used a global lock around the acquisition of both locks (it serializes the start of all transfers), 2 pts for solutions that used a global lock around the entire transfer (very inefficient). Adding a check to ensure that the acctId's are not identical was worth 1 pt of extra credit. No extra credit was given for performance improvements.

```
Transfer(acctId1, acctId2, amount) {
       If (acctId1 > acctId2) {
              account[acctId1]->Lock();
              account[acctId2]->Lock();
       } else {
              account[acctId2]->Lock();
              account[acctId1]->Lock();
       }
       acct1 = GetAccount(acctId1); /* May involve disk I/O */
       acct2 = GetAccount(acctId2); /* May involve disk I/O */
       if (acct1->balance < amount) {
              account[acctId1]->Unlock(); account[acctId2]->Unlock();
              return ERROR;
       }
       acct1->balance -= amount; acct2->balance += amount;
       StoreAccount(acct1); /* Involves disk I/O */
       StoreAccount(acct2); /* Involves disk I/O */
       account[acctId1]->Unlock(); account[acctId2]->Unlock();
       return OK;
}
```

4. Memory management (30 points total):

a. (6 points) Consider a memory system with a cache access time of 100ns and a memory access time of 1200ns. If the effective access time is 10% greater than

the cache access time, what is the hit ratio H? (fractional answers are OK)
Effective Access Time: $T_e = H \times T_c + (1 - H)(T_m + T_c)$,
where $T_c = 100ns$, $T_e = 1.1 \times T_c$, *and* $T_m = 1200ns$.

Thus, $(1.1)(100) = 100H + (1 - H)(1200 + 100)$
$110 = 100H + 1300 - 1300H$
$H = 119/120$

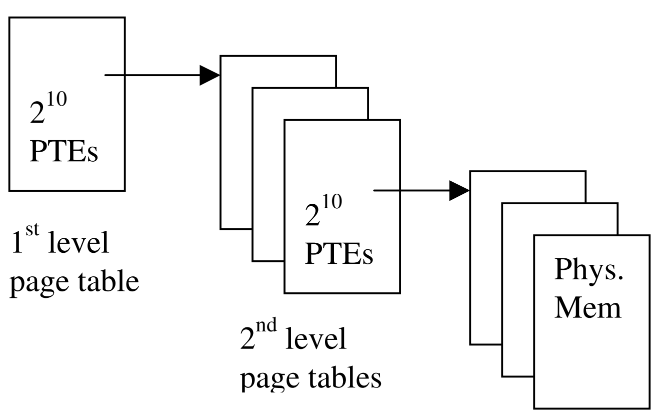
We awarded 3 pts for the correct formula and 3 pts for the correct answer. Many students missed the fact that the miss time includes ***both*** *the memory access time and the cache access time. If the formula was missing the cache access time, we deducted two points – if the answer based upon this incorrect formula was correct, we did not deduct any additional points.*

b. (6 points) Assuming a page size of 4 KB and that page table entry takes 4 bytes, how many levels of page tables would be required to map a 32-bit address space if every page table fits into a single page? Be explicit in your explanation.

A 1-page page table contains 1024 or 2^{10} *PTEs, pointing to* 2^{10} *pages (addressing a total of* $2^{10} \times 2^{12} = 2^{22}$ *bytes). Adding another level yields another* 2^{10} *pages of page tables, addressing* $2^{10} \times 2^{10} \times 2^{12} = 2^{32}$ *bytes. So, we need 2 levels.*

The correct answer is worth 2 pts. Correct reasoning is worth up to 4 pts (1 pt for identifying that there are 2^{10} *PTEs per page, 1 pt for describing how page tables are nested, and 2 pts based upon the quality of the argument).*

10 bit page	*10 bit page*	*12 bit offset*



c. (18 points) Consider a multi-level memory management using the following virtual addresses:

Virtual seg #	Virtual Page #	Offset

Each virtual address has 2 bits of virtual segment #, 8 bits of virtual page #, and 12 bits of offset. Page table entries are 8 bits. *All values are in hexadecimal.*

Translate the following virtual addresses into physical addresses:

Virtual Address	Physical Address
0x204ABC	***0x46ABC***
0x102041	***0x10041***
0x304F51	***Invalid segment!***

Virtual Address	Physical Address
0x23200D	***7400D***
0x1103DB	***Virt. page too big!***
0x010350	***0x16350***

Each correct translation is worth 3 pts. For answers that listed a translation when it is invalid, we deducted 1 pt.

Segment Table

Start	Size	Flags
0x2004	0x40	Valid, read only
0x0000	0x10	Valid, read/write
0x2040	0x40	Valid, read/write
0x1010	0x10	Invalid

Physical Memory

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x0000	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
0x0010	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D
….																
0x1010	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
….																
0x2000	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
0x2010	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21
0x2020	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31
0x2030	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40	41
0x2040	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51
0x2050	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61
0x2060	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71
0x2070	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81

5. Design tradeoffs (15 points total):

A hardware designer asks for your input on the design of a new processor and computer. You can spend $1500 dollars on the following components:

Item	Latency	Minimum Size	Cost
TLB	10ns	16 entries	$20/entry
main memory	200ns	16MB	$2/MB
disk	10ms (10M ns)	2GB	$0.20/MB

The page size is fixed at 16KB. Assume you want to run 3 – 4 applications simultaneously. Each application has an overall maximum size of 64 MB and a working set size of 256KB. TLB entries do not have Process Identifiers. Discuss how you would divide the available funds across the various items to optimize performance.

We start with the disk. Since the disk is the slowest component of the system, we take the minimum size, 2048MB or $410, leaving us with $1,090. Since the TLB does not contain process identifiers, we only need the minimum number of entries to map the working set – 256KB / 16KB = 16 entries or $320, leaving us with $770. With the remaining money, we could buy 385 MB. However, we will only ever need to have 4 ×64MB = 256 MB of memory or $512. So, we could spend the leftover $258 on another 12 TLB entries ($240). Increasing the TLB by this amount will improve performance somewhat, since the working set will change over time. It does not make sense to increase the TLB any more, since that would require that we decrease the memory size below the requirements for the applications; a situation that will cause the system to start paging.

We awarded 3 points per choice, based upon the reasonableness of the choices. We used the following table:

Item / Points	3 points	2 points	1 points	0 points
TLB	$29 \geq$ TLB ≥ 16	$32 \geq$ TLB $\succ 29$	TLB $\succ 32$	TLB $\prec 16$
Memory	≥ 256	$256 \succ$ Mem ≥ 128	$128 \succ$ Mem ≥ 64	Mem $\prec 64$
Disk	2.0 GB	$\succ$ 2.0GB, if using extra money for disk instead of memory or TLB	$\succ$ 2.0GB	$\prec$ 2.0GB

We awarded another 3 points if the TLB answer was based upon an analysis of the applications' working set size (i.e., only 16 entries are necessary, since they will map the working set and the TLB does not include process identifiers).

We awarded an additional three points based upon the overall reasonableness of the answer. For example, a system with a small amount of paging would lose a point, while a system with a significant amount of paging (e.g., only 128 MB of memory), would lose two points.