University of Waterloo
Midterm Examination Model Solution
Winter, 2003

Student Name: _____

Student ID Number: _____

Course Abbreviation and Number	CS354
Course Title	Operating Systems
Sections	01 and 02
Instructor	M. Karsten and K. Salem

Date of Exam	February 25, 2003
Time Period	19:00-21:00
Duration of Exam	2 hours
Number of Exam Pages (including this cover sheet)	11 pages
Exam Type	Closed Book
Additional Materials Allowed	None

Question 1: (20 marks)	Question 4: (8 marks)	Question 7: (8 marks)
Question 2: (10 marks)	Question 5: (12 marks)	Question 8: (12 marks)
Question 3: (8 marks)	Question 6: (10 marks)	Question 9: (12 marks)
Total: (100 marks)		

1. (20 total marks)

a. (2 marks)
In the Nachos simulated machine, the input and output consoles are *asynchronous* devices. Explain briefly what it means for a device to be asynchronous.

Requests to an asynchronous device return immediately, before the device has finished handling the request. The device generates an interrupt when it has finished handling the request.

b. (3 marks)
The Nachos `Exec` system call is used to create a new process. To implement `Exec`, the Nachos operating system must determine the size of the new process's virtual address space. Explain, briefly, how this is accomplished.

The sizes of the code, initialized and uninitialized global data, read-only data segments are determined from the NOFF file for the new process. Additional space is added for the stack.

c. (3 marks)
Briefly explain the difference between the user mode and the priviledged mode of a CPU. Why is that difference important for the goals of an operating system?

In user mode, certain instructions are unavailable, certain registers are inaccessible, and memory protection applies. These restrictions are used by the OS to maintain control over the hardware and to protect itself from accidental or deliberate interference from user programs.

d. (2 marks)
What is the difference between an Interrupt and a System Call?

An interrupt is asynchronously triggered by hardware. A system call is triggered by a special machine instruction executed by a user process.

e. (2 marks)
What is the difference between FIFO and round-robin scheduling?

Round-robin scheduling employs time-slicing and preemption.

f. (3 marks)
Briefly explain the usage and benefits of a translation lookaside buffer (TLB) for paging

The TLB is a cache of virtual-to-physical (page number to frame number) translations. The MMU tries to resolve address translations using the TLB, falling back to a page table if it fails. The benefit provide by the TLB is a reduction in the average time required to perform an address translation, since address translation through the TLB is much faster than address translation through a page table.

g. (3 marks)
The page replacement algorithm last-recently-used (LRU) could perform very well. Why is it hard to implement plain LRU in a real system? What techniques can be used to approximate LRU?

To implement LRU page replacement, the OS must be aware of the order in which page references occur. However, the OS is normally only made aware - via a page fault - of references to non-resident (out of memory) pages. The clock algorithm is an example of a technique that can be used to approximate LRU by taking advange of the "use" bit in each page table entry.

h. (2 marks)
An operating system uses a variable resident set policy with a global page replacement algorithm. Why may it be necessary to keep track of both a global and a per-process page fault frequency?

The per-process page fault frequency can be used to indicate whether the memory allocation for an individual process is too high or too low. The global page fault frequency can be used to indicate whether the system as a whole is overloaded.

2. (10 total marks)

Assume three processes with estimated CPU bursts:
p1: 7 time units
p2: 1 time units
p3: 16 time units

a. (5 marks)

Assume that all three processes are ready for execution. Explain and calculate the benefit of using short-process-next scheduling over just FIFO scheduling (first p1, then p2, then p3).

The benefit of shortest-process-next scheduling is lower turnaround times. Under shortest process next scheduling, the turnaround times of the process are 1 time unit (p2), 8 time units (p1) and 24 time units (p3), for an average turnaround time of 11 time units. Under FIFO scheduling, the turnaround times are 7 (p1), 8 (p2), and 24 (p3), for an average of 13 time units.

b. (5 marks)

Now, assume that p1 and p3 are ready for execution, but p2 becomes ready only after 2 time units. Explain what is necessary to still achieve an optimal execution order that minimizes the average turnaround time. What is the resulting execution schedule?

Preemption is needed. With preemption (SRT), p2 has a turnaround time of 1 (no waiting time), p1 has a turnaround time of 8 (1 time unit waiting), and p3 has a turnaround time of 24 (8 time units of waiting), as was the case which shortest-process-next scheduling in part (a).
The execution schedule is:

- *p1 runs for 2 time units*
- *p2 runs for 1 time unit*
- *p1 runs for 5 time units*
- *p3 runs for 16 time units*

3. (8 total marks)

a. (4 marks)

Suppose that an operating system supports concurrent threads. Suppose further that a process is running, and that it contains two threads, called ThreadX and ThreadY. These are system threads, scheduled by the operating system.

Thread X is executing the `ThreadXFunction`, shown below. Thread Y is executing the `ThreadYFunction`. In these functions, `ReadInput`, `WriteOuput` and `Exit` are system calls. The `ReadInput` call reads a character from the input device into the specified buffer. It is a *blocking* call - the caller will block for a short time while an input character is obtained. The `WriteOuput` call writes a character from the specified buffer to the output device. It is a *non-blocking* call. The `Exit` system call terminates the calling process.

For the purposes of this question, you should assume that the operating system uses a scheduling quantum large enough that a running thread will never use its entire quantum.

Code for Thread X	Code for Thread Y

void
ThreadXFunction() {
  char buf;
  for($i = 0$;$i < 5$;$i++$) {
  ReadInput(&buf);
  WriteOutput(&buf);
  WriteOutput("X");
  }
  Exit(0);
}
	void
	ThreadYFunction() {
	  char buf;
	  for($i = 0$;$i < 5$;$i++$) {
	  ReadInput(&buf);
	  WriteOutput(&buf);
	  WriteOutput("Y");
	  }
	  Exit(0);
	}

Assume that the following sequence of input characters is typed on the input console:

$$a\ b\ c\ d\ e\ f\ g\ h\ i\ j$$

Show a sequence of characters that could be generated on the output console by this process.

Since ReadInput blocks and the threads are system threads, they will switch at each ReadInput call. Output will look like:

$$aXbYcXdYeXfYgXhYiX$$

or like

$$aYbXcYdXeYfXgYhXiY$$

depending on which thread goes first. Note that the last character (j) will not appear on the output because the first thread to Exit will terminate the process (both threads).

b. (4 marks)

Repeat the problem from part (a). This time, however, assume that the operating system supports only sequential processes (one operating system thread per process.) That is, the operating system is not aware of the existence of multiple threads in a process. ThreadX and ThreadY are user-level threads, implemented by a non-preemptive user-level library.

In this case, the blocking system call will not result in a thread switch. The output will look like

$$aXbXcXdXeX$$

or like

$$aYbYcYdYeY$$

depending on which thread is initially running. The other thread will not run at all.

4. (8 total marks)

Nachos in its plain version has four scheduling states: JustCreated, Ready, Running, and Blocked. Ignoring the JustCreated state, which transistions exist between the other three states and when are they typically triggered?

- *Running → Ready: triggered when the running thread yields, or by a timer interrupt (quantum expiration).*
- *Ready → Running: as above.*
- *Running → Blocked: triggered, for example, when the running process suffers a page fault or performs console input/output*
- *Blocked → Ready: triggered when the blocked process gets what it was waiting for, e.g., on completion of an I/O request.*

5. (12 total marks)

Suppose that an operating system uses a preemptive multi-level feedback scheduler with (for simplicity) an infinite number of levels. New processes start at level 0. A process that uses its entire quantum at level i gets demoted to level $i+1$. A process that does not use its entire quantum moves back to level 0, regardless of the level that it had previously been at. The scheduling quantum for level i is 2^i time units.

Initially, there is one process in the system. It requires a total of 9 time units of running time before it exits. After every two units of running time, this initial process will create a new process. These new processes require only 1 time unit of running time each before they exit. (Note: process creation does not cause preemption of the creating process.)

None of these processes perform any blocking system calls - once started, they run until they exit or until they are preempted.

What will be the turnaround time of the initial process? That is, how long will it take for the initial process to finish? Justify your answer, preferably with a diagram.

From Time	*To Time*	*Running Process*	*Note*
0	*1*	*initial*	*level* 0 *quantum ends*
1	*2*	*initial*	*first child process created*
2	*3*	*initial*	*level* 1 *quantum ends*
3	*4*	*first child*	
4	*5*	*initial*	*second child process created*
5	*6*	*initial*	
6	*7*	*initial*	*third child process created*
7	*8*	*initial*	*level* 2 *quantum ends*
8	*9*	*second child*	
9	*10*	*third child*	
10	*11*	*initial*	*fourth child process created*
11	*12*	*initial*	*initial process finishes*
12	*13*	*fourth child*	

The turnaround time of the intial proces is 12 time units.

6. (10 total marks)

Two processes, Process A and Process B, are running in a system. Their page tables are shown below. Each page table entry includes a frame number and use (U), dirty (D), and valid (V) bits. An entry with $V = 1$ is a valid entry. For simplicity, assume the page size is 100 bytes.

Page Table for Process A

Page	Frame	U	D	V
0	6	0	1	1
1	4	1	1	1
2	5	1	0	1
3	0	0	0	0
4	1	1	0	1
5	9	1	0	1

Page Table for Process B

Page	Frame	U	D	V
0	0	0	0	1
1	7	1	0	1
2	8	0	0	1
3	2	0	1	1
4	0	0	0	0
5	3	1	1	1

a. (2 marks)

To which physical address does virtual address 410 of Process A map? If it does not map to a physical address, write "does not map".

110

b. (2 marks)

To which physical address does virtual address 64 of Process B map? If it does not map to a physical address, write "does not map".

64

c. (2 marks)

To which physical address does virtual address 410 of Process B map? If it does not map to a physical address, write "does not map".

does not map (page table entry is not valid)

d. (2 marks)

To which virtual address, in which process, does physical address 90 map? If it does not map to a virtual address, write "does not map".

Virtual address 90 *in process B.*

e. (2 marks)

To which virtual address, in which process, does physical address 890 map? If it does not map to a virtual address, write "does not map".

Virtual address 290 *in process B.*

7. (8 total marks)

a. (6 marks)

An operating system supports paged virtual address spaces using demand paging. Suppose that the system has allocated three physical frames to hold the pages of process P. Process P uses the following sequence of pages from its virtual address space.

0 1 2 4 5 1 2 4 0 3 1 4 2

Each number in the sequence is a virtual page number.

The three frames allocated to process P are initially empty. What is the *minimum* number of page faults process P will experience as a result of this sequence of page requests?

The optimal replacement policy will result in the minumum number of page faults. It will result in 9 page faults when applied to the given request sequence (see below).

b. (2 marks)

Assuming that optimal page replacement is used, which pages will be in process P's frames after this sequence of requests? (There is more than one correct answer - choose any *one* correct answer.)

Reference	*Frame 1*	*Frame 2*	*Frame 3*	*fault?*
0	*0*			*yes*
1	*0*	*1*		*yes*
2	*0*	*1*	*2*	*yes*
4	*4*	*1*	*2*	*yes*
5	*5*	*1*	*2*	*yes*
1	*5*	*1*	*2*	*no*
2	*5*	*1*	*2*	*no*
4	*4*	*1*	*2*	*yes*
0	*4*	*1*	*0*	*yes*
3	*4*	*1*	*3*	*yes*
1	*4*	*1*	*3*	*no*
4	*4*	*1*	*3*	*no*
2	*4*	*1*	*2*	*yes*

The last page replacement is arbitrary, so the contents of the frames may be any one of $2, 1, 3$ *or* $4, 2, 3$ *or* $4, 1, 2$

8. (12 total marks)

Assume that the following table shows the current occupancy of frames with pages, i.e. frame 0 currently holds page 17, frame 1 holds page 32, etc. Furthermore, assume that the clock algorithm is used for page replacement (with pointer advancement after replacement) and the usage bit is set to 1, if a page is brought into memory. The third row of the table below shows the current setting of the respective usage bits. Note that this table does not necessarily correspond to any real operating system data structure!

Frame Number	0	1	2	3	4	5	6	7
Page Number	17	32	41	5	7	13	2	20
Usage Bit	1	0	0	0	0	1	1	0

If the clock pointer currently points to frame 4, which pages are being replaced by the following page request schedule? Show the frame occupancy and usage bits at the end of the schedule in the table below.

Page request schedule: 32, 14, 15, 2, 18, 14

- *reference to 32 sets use bit to one, no replacement*
- *reference to 14 replaces 7 with 14, sets use bit to 1*
- *reference to 15 replaces 20 with 15, sets use bit to 1, clears use bits for 13 and 2*
- *reference to 2 sets use bit for 2*
- *reference to 18 replaces 41 with 18, sets use bit to 1, clears use bits for 17 and 32*
- *reference to 14 does not change anything, since use bit is already 1*

Pages Replaced? Pages 7, 20, and 41 are replaced

Frame Number	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Page Number	*17*	*32*	*18*	*5*	*14*	*13*	*2*	*15*
Usage Bit	*0*	*0*	*1*	*0*	*1*	*0*	*1*	*1*

New Clock Pointer? Clock pointer finishes at frame 3.

9. (12 total marks)

Suppose that you wish to design a virtual memory system with the following characteristics:

- The size of a page table entry is 4 bytes.
- Each page table must fit into a single physical frame.
- The system must be able to support virtual address spaces as large as 2^{38} bytes (256 GB).

a. (2 marks)

Suppose that you decide to use a multi-level paging scheme with no more than two levels of page tables. What is the minimum page size that your system must have?

Let 2^p be the page (and frame) size. The maximum number of page table entries that can fit in a single frame is 2^{p-2}, since page table entries are 4 bytes. For two level paging, each virtual address is composed of two page numbers (each one $p-2$ bits long) and a p-bit offset. For a 2^{38} byte address space we need:

$$(p-2)+(p-2)+p=38$$

so $p=14$. The minimum page size is 2^{14} bytes.

b. (4 marks)

Draw a diagram indicating how the bits of a virtual address will be interpreted by the address translation mechanism. Indicate which bits (and how many) are used to index the page tables at each level, and which bits form the page offset. Draw neatly.

The 12 most significant bits determine the page table entry number at the first level, the next most significant 12 bits determine the page table entry at the second level, and the remaining, 14 bits specify the page offset.

c. (2 marks)

Suppose instead that you are willing to use a three-level paging scheme. What is the minimum page size that your system must have in this case?

As in part (a), we have

$$(p-2)+(p-2)+(p-2)+p=38$$

so $p=11$. The minimum page size is 2^{11} bytes.

d. (4 marks)

Draw a diagram indicating how the bits of each virtual address will be interpreted by the address translation mechanism. Indicate which bits (and how many) are used to index the page tables at each level, and which bits form the page offset. Draw neatly.

The 9 most significant bits determine the page table entry number at the first level, the next most significant 9 bits determine the page table entry at the second level, the next most significant 9 bits determine the page table entry at the third level, and the remaining, 11 bits specify the page offset.