University of Waterloo
Midterm Examination Model Solution
Winter, 2005

1. (10 marks)

Given the following atomic swap operation for two boolean variables, write pseudocode that can be used to protect a critical section.

```
void Swap(boolean *a, boolean *b) {
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```

Your solution should include the following three parts, each of which should be clearly labeled:

Part 1: declarations of any variables used, for those variables

Part 2: code to be executed by a thread before it enters the critical section,

Part 3: code to be executed by a thread after it leaves the critical section.

For any variable declared in Part 1, you must indicate whether it is a shared variable or a variable that is private to each thread. In addition, you must specify the initial value of each variable.

```
// Part 1
boolean lock = false;  // shared variable
boolean key = true;    // private (per thread) variable

// Part 2
key = true;
while (key == true) {
  Swap(&lock,&key);
}

// Part 3
key = false;
Swap(&lock,&key);
```

2. (10 total marks)

Suppose that two processes, P_1 and P_2, are running in a system. Each process requires C units of execution time to complete its program. Neither program performs any system calls that might cause it to block, and there are no other processes in the system.

The system uses a preemptive, round-robin scheduler with a quantum of q. Each context switch (from one process to the other) involves some overhead. Let w ($w < q$) represent the amount of overhead per context switch.

a. (4 marks)

Suppose that both processes become runnable at time $t = 0$, and the scheduler chooses P_1 to run first. Let N_1 denote the number of times that P_1 will be preempted before it completes the execution of its program. Give an expression for N_1 in terms of q, w, and C.

$$\boldsymbol{N_1 = \lfloor \frac{C}{q - w} \rfloor}$$

b. (3 marks)

Let t_1 denote the time at which P_1 completes the execution of its program. Give an expression for t_1 in terms of N_1, q, w, and C. (Note: N_1 is the preemption count that you determined in part (a). If you did not get part (a), just answer this part of the question assuming that N_1 is known.)

$$\boldsymbol{t_1 = 2qN_1 + C \bmod (q - w)}$$

c. (3 marks)

Let t_2 represent the completion time of process P_2, and let $\Delta = t_2 - t_1$. Give an expression for Δ in terms of q, w, N_1, and C.

$$\boldsymbol{\Delta = C \bmod (q - w) + w}$$

3. (12 total marks)

a. (5 marks)

Consider the following list of actions. Put a check mark in the blank beside those actions that should be performed only by the kernel, and not by application programs.

- _**X**__ changing the contents of a page table entry
- ____ changing the value of the program counter (PC)
- ____ reading the value of the program counter (PC)
- ____ changing the value of the stack pointer (SP)
- _**X**__ changing the contents of a TLB entry
- ____ executing a system call instruction
- _**X**__ changing the contents of the page table base register
- _**X**__ changing the contents of the page table limit register
- _**X**__ halting or rebooting the computer

b. (2 marks)

What are the key differences between a user-level thread and a kernel thread?

User-level threads may be less expensive to use, as they do not require system calls to create, yield, or destroy. User-level threads are not known to or controlled by the kernel.

c. (2 marks)

What is the difference between an exception and an interrupt?

Exceptions are caused by instructions being executed by the currently running process. Interrupts are caused by hardware devices.

d. (3 marks)

Consider a system that implements simple paging, using a hardware controlled TLB. In such a system, the responsibility for *translating virtual addresses to physical addresses* is shared by hardware and the kernel. Briefly describe the responsibilities of each (with respect to address translation).

The hardware is responsible for searching the TLB to find translations. In case of a TLB miss, the hardware is also responsible for finding the appropriate entry from the process page table and loading it into the TLB, and for generating an exception if that entry is not valid.
The operating system is responsible for setting up and managing the contents of the page tables, and for handling any exceptions generated by the hardware.

4. (10 total marks)

Consider a NachOS system in which k ($k > 1$) processes are running. One process is running at low priority, and the remaining $k - 1$ processes are running at normal priority. Each process' program is described by the following pseudo-code:

```
REPEAT 5 TIMES {
    compute for C time units
    write a single character to the NachOS output console
}
```

Characters are written to the synchronous output console using the NachOS `Write` system call. The low priority process writes the character "L". The normal priority processes write the character "N". The synchronous output console outputs characters one at a time in the order in which the `Write` requests are made. Assume that that output console requires W time units to output a single character.

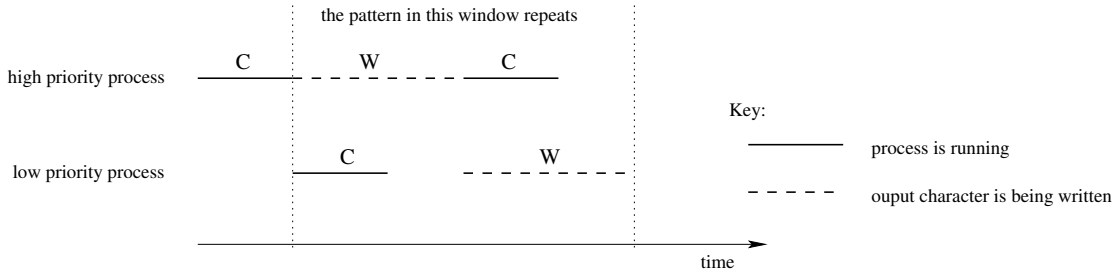
Assume that the NachOS scheduler is a round robin preemptive scheduler with quantum q that has been modified to understand priorities, as was required for Assignment 1. Specifically, a lower priority process will never run if there is a runnable process of higher priority.

For the purposes of this question, ignore context switching overhead.

a. (5 marks)

Suppose that $k = 2$ and $C < W$. Under this assumption, what is the sequence of N's and L's that will be produced on the output console by the processes? Explain your answer.

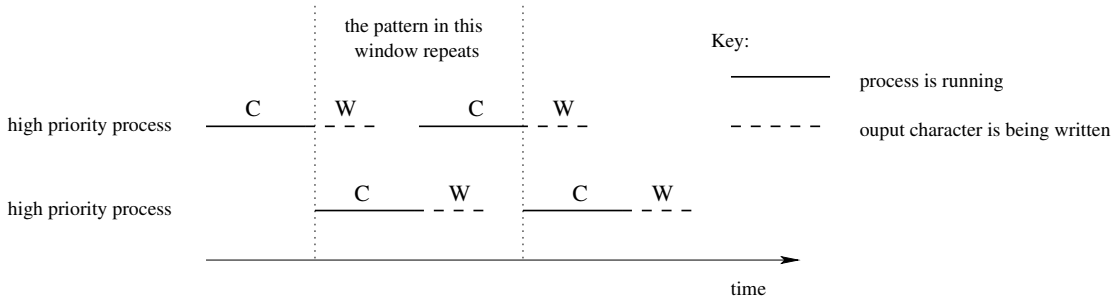
NLNLNLNLNL



b. (5 marks)

Suppose that $k = 3$ and $W < C < q$. Under these assumptions, what is the sequence of N's and L's that will be produced on the output console by the processes? Explain your answer.

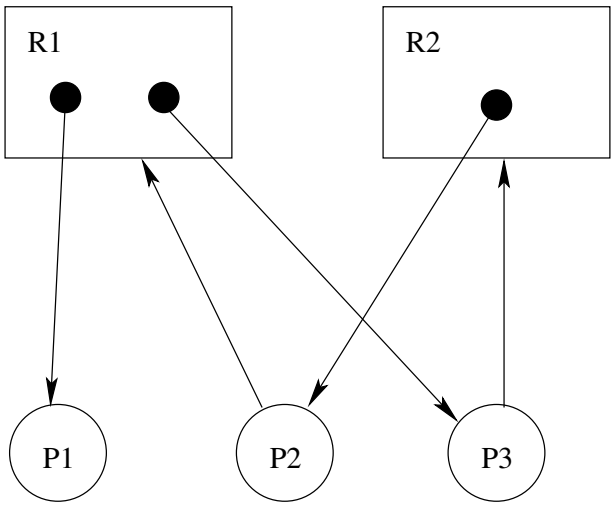
NNNNNNNNNNLLLLL



Note: since there is always a runnable high priority process, the low priority process will run once both high priority processes have finished

5. (8 total marks)

Give an example of a resource allocation graph that includes a cycle, yet does not have a deadlock. Briefly explain why your example does not have a deadlock. Keep your example simple – unnecessarily complex answers will be penalized.



There is no deadlock in this graph because the processes can finish as follows. P1 finishes, releasing an instance of R1. P2 acquires the released instance of R1 then finishes, releasing instances of R1 and R2. P3 acquires the instance of R2 and then finishes.

6. (10 total marks)

The local laundromat has just entered the computer age. As each customer enters, he or she puts coins into slots at one of two stations and types in the number of washing machines he/she will need. The stations are connected to a central computer that automatically assigns available machines and outputs tokens that identify the machines to be used. The customer puts laundry into the machines and inserts each token into the machine indicated on the token. When a machine finishes its cycle, it informs the computer that it is available again. The computer maintains an array *available[NMACHINES]* whose elements are non-zero if the corresponding machines are available (NMACHINES is a constant indicating how many machines there are in the laundromat), and a semaphore *nfree* that indicates how many machines are available. The *available* array is initialized to all ones, and *nfree* is initialized to NMACHINES. The code to allocate and release machines is as follows:

```
binary semaphore lock = 1;  // code added for part (b)

int allocate() {  /* Returns index of available machine.*/
  int i;
  P(nfree);     /* Wait until a machine is available */
  P(lock); //  code added for part (b)
  for (i=0; i < NMACHINES; i++) {
    if (available[i] != 0) {
      available[i] = 0;
      V(lock); // code added for part (b)
      return i;
    }
  }
}

release(int machine) { /* Releases machine */
  P(lock);  //  code added for part (b)
  available[machine] = 1;
  V(lock);  //  code added for part (b)
  V(nfree);
}
```

a. (4 marks)

It seems that, if two people make requests at the two stations at the same time, they will occasionally be assigned the same machine. This has resulted in several brawls in the laundromat, and you have been called in by the owner to fix the problem. Assume that one thread handles each customer station. Explain how the same washing machine can be assigned to two different customers.

A problem can arise if there is a context switch just before `available[i] = 0`. Each thread tests whether a washer is free and then claims it if it is free. However, the test-and-claim is not atomic. A context switch after the test may cause two threads to claim the same washer.

b. (6 marks)

Clearly describe how to modify the code above to eliminate the problem. Feel free to denote your changes directly on the code shown above.

Answer has been included in the code above.

7. (8 total marks)

A system implements paging using a page size of 4096 (2^{12}) bytes. Page tables for only two of several executing programs (Program A and Program B), are shown below. They represent part of the state of the system prior to the execution of the instructions that generate the address translations used in this question. Note that VPN refers to the virtual page number and PFN refers to the corresponding physical frame number. D, U, V, and RO are the dirty, use, valid and read-only bits, respectively. (The D and U bits are not used in this question.)

	VPN	PFN	D	U	V	RO
0	0	5	0	0	0	1
1	1	5	0	0	0	1
2	2	0	0	0	0	0
3	3	7	0	0	1	1
4	4	3	0	0	0	1

Page Table A

	VPN	PFN	D	U	V	RO
0	0	2	0	1	1	1
1	1	5	1	1	1	0
2	2	5	0	0	0	0
3	3	6	1	1	1	0
4	4	3	0	0	0	0
5	5	1	0	0	0	1
6	6	7	0	0	0	1
7	7	1	0	0	1	0

Page Table B

The following unordered subset of physical address translations (shown in hexadecimal) was produced without generating any page faults or other exceptions. For each shown **physical address**, give the original virtual address whose translation resulted in that physical address. Specify the virtual addresses in hexadecimal. Also indicate which program was running when the address was referenced, and indicate whether the specified address was being accessed with a read. If the information being requested can not be determined from the information provided write "unknown".

Physical Address	Virtual Address	Program (A or B)	Read?
0x7c8c	**0x3c8c**	**A**	**yes**
0x5231	**0x1231**	**B**	**unknown**
0x0000	**unknown**	**unknown**	**unknown**
0x5238	**0x1238**	**B**	**unknown**
0x2f38	**0x0f38**	**B**	**yes**
0x1008	**0x7008**	**B**	**unknown**
0x3cd1	**unknown**	**unknown**	**unknown**
0x6109	**0x3109**	**B**	**unknown**

8. (12 total marks)

Consider the following system call interface:

```
int ThreadFork(void (*func)(int), int arg);
```

The ThreadFork system call creates a new thread in the *same* address space as the calling thread. The new thread initially runs the function `func`, which is passed a single parameter `arg`.

Sketch how you would implement this system call in a system similar to NachOS. Ideally, your answer should consist of the major steps that the kernel's handler for the `ThreadFork` system call would need to implement the call. Details, such as the actual names of particular variables or data structures from the NachOS system, are not required.

- **Create a new (kernel) thread. Add it to the ready queue.**
- **Increase the size of the process's address space (or choose an unused part of the address space) to create a stack for the new thread.**
- **Update the process info (e.g., the process table) to indicate that the process has another thread.**
- **Initialize the new thread's stack and create an initial register context for the new thread.**

9. (10 total marks)

Consider a virtual memory system that uses pure segmentation (no paging). The system provides each process with up to 16 segments, each of which may be as large as 128 megabytes (2^{27} bytes). The system supports physical memories as large as 4 gigabytes (2^{32} bytes).

a. (2 marks)

How many bits does a virtual address have in this system?

31 bits, including 4 bits for the segment ID and 27 bits for the offset into the segment.

b. (4 marks)

What fields will be found in each entry of a segment table in this system? Describe the purpose of each field, and indicate how large (number of bits) each field must be.

segment length: 27 bits, indicates the size of the segment

segment base: 32 bits, indicates the physical address of the beginning of the segment.

flags: 1 bit each, e.g., a dirty flag, a use flag, protection flags, a valid flag.

c. (2 marks)

What is the maximum size (in bytes) of a segment table in this system? Briefly justify your answer.

27 bits (length) plus 32 bits (base) is 59 bits per segment. The use of 8 bytes (64 bits) per segment would leave 5 bits per segment for flags. There can be at most 16 segments, so the total size of the segment table would be $16*8 = 128$ **bytes.**

d. (2 marks)

How many memory references must this system's MMU make in order to translate a virtual address to a physical address?

At least one memory reference will be required, to read the segment table entry. If a single read retrieves 4 bytes (as would be true on a 32-bit machine), two reads would be required to read an 8 byte segement table entry. Note that this does not include the cost of reading (or writing) the contents of the physical address that results from the translation.