

CS 474
Midterm
March 6, 2000

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no calculators** are allowed.

Please note the following instructions. There will be a ten point deduction for failure to comply with them.

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in.

You have until 1:20 to finish the exam. The questions are equally weighted.

1. All computer systems (at least, all that I'm aware of!) have an instruction used to return from an interrupt. This instruction restores the old program counter and processor status word from stack, returning to the program that was running at the time the interrupt occurred, at the point where it was interrupted.

This is normally a privileged instruction (*i.e.* it can only be executed when in kernel mode). Describe how this instruction could be exploited to execute arbitrary code in kernel mode if it could be executed from user mode.

2. The C programming language's post-increment `++` operator (as in `i++`, not `++i`) increments the value of a variable, and returns its old value. What assumptions would you have to make about the `++` operator to use it in implementing mutual exclusion? Give the entry and exit code for a critical section using `++` for mutual exclusion.
3. Attached to this exam is a very simple linked list manipulation package (in order to simplify the question, the case of an empty list is not considered). Use semaphores to modify this code so the list can be in shared memory, accessible by several processes. A full-credit solution will only let one process at a time take a node from the head of the list, and will only let one process at a time put a node at the tail of the list, but will let a process insert and a process delete at the same time so long as there is more than one node in the list. You may assume that `malloc()` and `free()` are already protected from mutual exclusion problems (*i.e.* you don't have to worry about multiple processes calling them simultaneously). *Hint: use lots of semaphores.*
4. A nice feature of a program that uses a graphical user interface is for it to "background" itself - that is, to give the shell prompt back immediately, while the program continues to run.

This is actually a very easy feature to add to a program: we can do it by calling `fork()`, and terminating immediately if we are the parent but continuing if we are the child. Write the code that accomplishes this.

5. Suppose we have the following resource, allocation, and unsatisfied request tables:

Resources:		R0	R1	R2	R3	R4
	3	5	7	3	2	

Allocation:		R0	R1	R2	R3	R4
P0	1	2	3	2	1	
P1	1	2	2	0	0	
P2	0	0	1	1	1	

Request:		R0	R1	R2	R3	R4
P0	0	0	0	0	0	
P1	0	0	0	1	0	
P2	0	0	0	0	1	

A. Is the system currently deadlocked? If not, give a sequence by which all processes could continue to termination.

B. Suppose P0 were to request 2 units of R0. What would happen?

SSN: _____

Code for Question 3

```
struct node {
    void *contents;
    struct node* next;
};

struct list {
    struct node *head;
    struct node *tail;
}

void enqueue(struct list *this, void *data) {
    struct node *newnode;

    newnode = malloc(sizeof(struct node));

    newnode->contents = data;
    newnode->next = NULL;

    this->tail->next = newnode;
    this->tail = newnode;
}

void *dequeue(struct list *this) {
    void *retval;
    struct node *oldnode;

    retval = this->head->contents;
    oldnode = this->head;
    this->head = oldnode->next;

    free(oldnode);

    return(retval);
}
```