Lab 6 – More Semaphores, Deadlock

Due: Never, but look at these before the exam

This week's lab consists only of questions to look at on your own (not to be turned in). Make sure you understand these questions before the midterm exam.

Practice Questions

- Show that semaphores and monitors are equivalent (show how to implement each using the other).
- Describe how you would implement a "nice" semaphore, like the ones we assumed we had for the sleeping barber and the dining savages, using System V semaphores.
- SG&G 7.1
- SG&G 7.11
- SG&G 7.14
- A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Why or why not?

Old Exam Questions

These questions are from the midterm given in the last version of this course. There were actually more questions, but our exam will cover only up to deadlock. Last time, the exam was not held until after the first few lectures on memory management. That exam also had a more strict time limit of 120 minutes, so I will be able to ask longer questions this time around.

1. Processes and Threads (7 points)

a. Threads share more context than processes. Name two advantages that this affords threads over processes. (2 points)

b. Consider an environment where there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run. Explain why this model can make multithreaded programs run faster than their single-threaded counterparts on a uniprocessor system. (2 points)

c. What might the output be when the following program is run? Assume the `fork()` call is successful. Explain what is happening in a few sentences. (3 points)

```
#include <stdio.h>
#include <unistd.h>
```

```
int main(int argc, char *argv[]) {
  int x;
  pid_t pid;
  int status;

  x=13;
  printf("x is now %d\n",x);

  pid=fork();
  if (pid == 0) {
    x=x+4;
    printf("Add 4, and x=%d\n",x);
    exit(x);
  } else {
    x=x*4;
    printf("Multiply by 4, and x=%d\n",x);
    wait(&status);
  }
  printf("In the end, x=%d\n",x);
  return 0;
}
```

2. CPU Scheduling (12 points). The following table contains arrival times and CPU burst times for four processes.

process	arrival time	CPU burst time
P1	0	2
P2	3	8
P3	4	9
P4	7	3

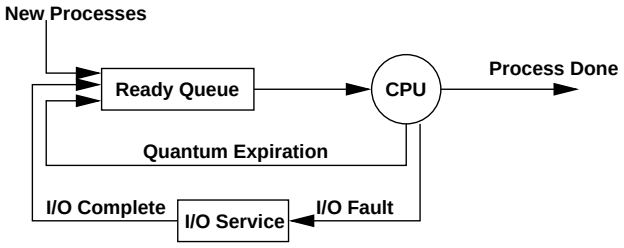
We consider two scheduling disciplines: *Shortest Job First* (SJF) and *Round Robin* (RR).

a. Draw two Gantt charts, showing the process occupying the CPU at each time, first using SJF and second using RR. For RR, use a quantum of 3. Label each context switch with its time (assume context switch time is zero). (2 points)

b. What are the turnaround times for each process with each scheduling discipline? (2 points)

c. What is the average waiting time with each discipline? (2 points)

d. Recall this diagram from the first project handout:



Why is there no transition shown from the ready queue directly to I/O service? (1 point)

For parts e and f, consider a variation of RR in which a process that has used its full time quantum is returned to the end of the ready queue, while one that has used $\frac{1}{2}$ of its time quantum is returned to the middle of the queue and one that has used $\frac{1}{4}$ of its time quantum is placed $\frac{1}{4}$ of the distance away from the beginning of the queue.

e. What is the objective of this scheduling policy? (2 points)

f. Discuss the advantages and disadvantages of its implementation. (3 points)

3. Resource Allocation. (7 points)

For parts a and b, let R1, R2, R3, and R4 be four different non-preemptible resource types. Assume maximum claims and current resource assignments for four processes P1, P2, P3, and P4, are given in the following table, along with the total number of each resource (including allocated resources).

	Maximum Claims				**Current Use**				
Resource	**P1**	**P2**	**P3**	**P4**	**P1**	**P2**	**P3**	**P4**	**Total**
R1	4	2	5	1	1	1	3	0	6
R2	2	2	1	3	0	1	0	2	5
R3	3	0	5	2	2	0	4	1	7
R4	1	5	0	5	1	2	0	1	9

a. Is this system currently in a safe state? If it is, give a safe sequence. If it isn't, what processes may end up in deadlock? (2 points)

b. If P3 relinquishes one instance of R3, and P2 is granted three instances of R4, will the system be safe? If it is, give a safe sequence. If it isn't, what processes may end up in deadlock? (2 points)

c. Consider a system consisting of six resources of the same type shared by n processes, each of which needs at most two instances of the resource at a time. For what values of n is the system is deadlock-free? (3 points)

5. Process Synchronization (15 points)

a. Consider this variation on the producer-consumer problem we studied in class. Assume one producer process and `n` consumer processes share a buffer, large enough to hold one item at a time. The producer deposits items into the buffer, consumers fetch them. Every

item deposited by the producer has to be fetched by all `n` consumers before the producer can deposit its next item.

Develop pseudocode for the actions of the producer and the consumers. Use semaphores for synchronization. Your solution should be deadlock-free and should avoid unnecessary waiting. (9 points)

b. The semaphores we have used are **counting semaphores**. Some operating systems provide a more restricted type of semaphore called a **binary semaphore**. Here, the semaphore can only have values of 0 or 1. A `signal()` operation on a binary semaphore with a value 1 is ignored, whereas a counting semaphore would increment its value to 2.

A counting semaphore `S` can be implemented using only binary semaphores and regular machine instructions, as follows:

- Data structure includes:

```
binary_semaphore S1=1;
binary_semaphore S2=0;
binary_semaphore S3=1;
int C=initial value of S;
```

- *wait* operation

```
wait(S3);
wait(S1);
C--;
if (C < 0) {
  signal(S1);
  wait(S2);
}
else {
  signal(S1);
}
signal(S3);
```

- *signal* operation

```
wait(S1);
C++;
if (C <= 0) {
  signal(S2);
}
signal(S1);
```

Describe the purpose of each of the three binary semaphores `S1`, `S2`, and `S3`. (6 points)