Homework Assignment 2

Respond to the questions as briefly as possible in your own words. Submit your answers via Canvas as a PDF document by the due date.

1. Consider the following program. What output is displayed on the console when this program runs? Why?

```
int myValue = 10;
int main(int argc, char **argv) {
        fork();
        myValue += 10;
        fork();
        myValue += 10;
        fork();
        printf("myValue = %d\n", myValue);
}
```

2. At a computer science department far, far away, everything is going well except for one problem. The department has a group of graduate students doing operating system research and a group of graduate students doing computability theory research. The two groups share a common graduate student lounge and fights are breaking out over the expresso machine when both the OS and theory groups want expresso. Being a Computer Science Department, the faculty decided to solve the problem using semaphores (the OS faculty suggested this). They decided to write a solution that would allow each group to access the expresso machine without interference from the other. That is, when the first OS student starts using the expresso machine, any OS student can use it but theory students are blocked. Only when the last OS student finishes with the expresso machine can waiting theory students use it. Once the first theory student begins using the expressor machine, any theory student can join in, but OS students are blocked.

   They began by setting up the framework below, but they were called away to a faculty meeting before they could finish. Complete solution to the vexing expresso dilemma using the framework below. (Hint: consider the readers/writers problem on pages 220-221 of your text and consider both groups as readers.)

```
int osCounter = 0;
int theoryCounter = 0;
Semaphore osMutex, theoryMutex, takeExpresso

OS student i:                                   Theory student j:
while (1) {                                     while (1) {

// code to enter the critical section           // code to enter the critical section

        getExpresso();                                  getExpresso();

// code to leave the critical section           // code to leave the critical section

}                                               }
```

3. Consider the following Pthreads program:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int passit = -1;
int AFlag = 1;
int BFlag = 0;
int save[10];
```

```
void *A(void *args) {
```

```
void *B(void *args) {
```

```
        int temp;

        int i;
        srand(12345);
        for (i = 0; i < 10; i++) {
                while (AFlag != 1)
                        sched_yield();

                temp = rand();
                save[i] = temp;
                passit = temp;
                AFlag = 0;
                BFlag = 1;
        }
}
```

```
        int temp;
        int i;
        for (i = 1; i <= 10; i++) {
                while (BFlag != 1)
                        sched_yield();

                temp = passit;
                passit = -1;
                printf("Thread B prints random
number %d\n", temp);

                BFlag = 0;
                AFlag = 1;
        }
}
```

```
int main (int argc, char **argv) {

        pthread_t a, b;

        pthread_create(&a, 0, A, 0);
        pthread_create(&b, 0, B, 0);
        pthread_join(a, 0);
        pthread_join(b, 0);

        int i;
        for (i = 0; i < 10; i++)
                printf("Main prints random number %d as %d\n", i+1, save[i]);
}
```

The main program starts two threads, one running function A and the other running function B. Thread A creates a random number and places it in a shared variable *passit* and also saves it in the current location of the array *saved*. Thread B displays each random number using the shared variable *passit* as it is created by thread A. After the two threads are finished, the main thread displays all numbers in the array *saved*. If you compile and run the program on the Linux virtual machine (you don't need to), you'd see that thread B printed all of the numbers passed to it from thread A.

The code in thread A and thread B that use the shared variable *passit* are critical sections for the two threads and must be performed under mutual exclusion. Furthermore, thread A cannot be allowed to "run ahead" of thread B; that is, thread A must not be allowed to overwrite the value in *passit* before thread B has a chance to display it.

Explain how the two threads synchonize with respect to their use of *passit* without using any of the techniques discussed in class: semaphores, mutex locks/condition variables, Peterson's algorithm, monitors, etc. (Hint: focus the threads using of the AFlag and BFlag variables and the three requirements for eliminating race conditions: 1) mutual exclusion on the critical section; 2) a thread can enter the critical section if no other thread is in it; and 3) bounded wait when attempting to enter the critical section. This program relaxes requirement 2).

4. Using the textbook's monitor syntax, implement a semaphore.
5. Suppose that you have a set of processes that are controlling a set of valves and switches. Valve 1 lets chemical X into the mixing vat. Valve 2 lets chemical Y into the mixing vat. Switch 1 starts the agitator in the mixing vat. Valve 3 releases the contents of the mixing vat into the holding tanks. Process A is responsible for delivering the indicated amount of chemical X into the mixing vat. Process B is responsible for delivering the indicated amount of chemical Y into the mixing vat. Process C is responsible for agitating the chemical for the requisite amount of time. Process D is

responsible for emptying the vat into the holding tanks. The activities of the processes must be controlled according to the following constraints:

1. Chemical X must be put into the mixing vat before chemical Y
2. both chemicals must be in the mixing vat before agitation begins,
3. agitation must be complete prior to the emptying of the mixing vat into the holding tanks
4. and, the mixing vat must be completely emptied before the next cycle of mixing can begin

Write pseudo-code using semaphores and P and V operations to implement the processes according to the above constraints. Assume that valves are controlled with functions open_valve(i), close_valve(i) and switches are controlled by functions on_switch(i) and off_switch(i).

6. Explain what is meant by *priority inversion.*
7. Spin locks (busy waiting locks) are more efficient than mutex sleep locks for very short critical sections. Suppose that the context switch time for a system (the time it takes to save the current process and load the next) is time *T*. How long can a critical section be before it is more efficient to use a mutex sleep lock rather than a spin lock?
8. Distinguish between a *preemptive* and *non-preemptive* CPU process scheduler.
9. What are *CPU bursts* and *IO bursts*?
10. Consider the set of processes shown in the table below:

Process Number	CPU Burst	Arrival Time
1	50	0
2	20	10
3	70	10
4	10	15
5	40	15

Assume the processes arrive into the system at the times shown and run till their CPU bursts are complete. Show the Gannt chart (use the textbook style Gannt chart) for each scheduling algorithm. What is the average turnaround time for each of the disciplines below?

1. FCFS
2. SJF
3. Preemptive SJF
4. *RR with quantum size 5.*