CS 475 Spring 2017

Assignment 2 - PART A

DUE 11:59 PM Thursday, March 9

Instructions:

- PART A of Assignment 2 is to be completed individually by each student. No collaboration among students is permitted.
- Please upload your solutions to PART A to Blackboard in PDF or MS Word format.
- Late submissions will not be graded. There are no slip days for Assignment 2 - PART A.

Problem 1. (15 points):

This problem tests your understanding of exceptional control flow in C programs. Assume we are running code on a Unix machine. The following problems all concern the value of the variable `counter`.

Part I

```
int counter = 0;

int  main()
{
    int i;

    for (i = 0; i < 2; i ++){
        fork();
        counter ++;
        printf("counter = %d\n", counter);
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

A. How many times would the value of `counter` be printed: _____

B. What is the value of `counter` printed in the first line? _____

C. What is the value of `counter` printed in the last line? _____

Part II

```
pid_t pid;
int counter = 0;

void handler1(int sig)
{
    counter ++;
    printf("counter = %d\n", counter);
    fflush(stdout);   /* Flushes the printed string to stdout */
    kill(pid, SIGUSR1);
}

void handler2(int sig)
{
    counter += 3;
    printf("counter = %d\n", counter);
    exit(0);
}

main() {
    signal(SIGUSR1, handler1);
    if ((pid = fork()) == 0) {
        signal(SIGUSR1, handler2);
        kill(getppid(), SIGUSR1);
        while(1) {};
    }
    else {
        pid_t p; int status;
        if ((p = wait(&status)) > 0) {
            counter += 2;
            printf("counter = %d\n", counter);
        }
    }
}
```

What is the output of this program?

Part III

```
int counter = 0;

void handler(int sig)
{
    counter ++;
}


int  main()
{
    int i;

    signal(SIGCHLD, handler);

    for (i = 0; i < 5; i ++){
        if (fork() == 0){
            exit(0);
        }
    }

    /* wait for all children to die */
    while (wait(NULL) != -1);

    printf("counter = %d\n", counter);
    return 0;
}
```

A. Does the program output the same value of `counter` every time we run it? `Yes` `No`

B. If the answer to A is `Yes`, indicate the value of the `counter` variable. Otherwise, list all possible values of the `counter` variable.

Answer: `counter =` _____

Problem 2. (25 points):

You have been hired by a company to work on a program that does climate modelling. The inner loop of the program matches atoms of different types as they form molecules. In an excessive reliance on threads, each atom is represented by a thread.

A Your task is to write code to form carbon dioxide out of one carbon thread and two oxygen threads (C02). You are to write the two procedures: `CArrives()` and `OArrives()`. A carbon dioxide molecule forms when two O threads are present and one C thread; otherwise the atoms must wait. Once all three are present, one of the threads calls `MakeCO2()`, and only then, all three depart.

B Extending the product line into beer production, your task is to write code to form alcohol (C2H60) out of two carbon atoms, six hydrogens, and one oxygen.

You must use locks and Mesa-style condition variables to implement your solutions. Obviously, an atom that arrives after the molecule is made must wait for a different group of atoms to be present. There should be no busy-waiting. There should also be no useless waiting: atoms should not wait if there is a sufficient number of each type to make a particular molecule.

Problem 3. (10 points):

Consider the code written by Harry Q. Bovik for the following problem. Please assume that all necessary header files are included in the code and all system calls and accessory functions always succeed. You may assume that the locks have been initialized correctly in `main()`, which we do not show.

1. Bovik comes to you and complains that `func1()` seems to misbehave. Is he lying or is there something wrong with his code? Defend your answer in 1-3 sentences.

2. Bovik is complaining about `func2()` as well. He insists to your boss that this program sometimes hangs and your boss would like your opinion. Is Bovik lying again or is there a problem? Defend your answer in 1-3 sentences.

```
pthread_mutex_t *count_l, *l_count;
int ref_count, tid_1, tid_2, tid_3, tid_4;

void *thread1(void *vargp) {
     tid_2 = pthread_self();
     pthread_mutex_lock(count_l);
     ref_count++;
     pthread_mutex_unlock(count_l);
     return(0);
}
void *thread2(void *vargp) {
     tid_1 = pthread_self();
     pthread_mutex_lock(count_l);
     pthread_kill(pthread_self(), SIGKILL);
     ref_count++; pthread_mutex_unlock(count_l);
     return(0);
}
void *thread3(void *vargp) {
     tid_3 = pthread_self();
     pthread_mutex_lock(count_l);
     pthread_mutex_lock(l_count);
     ref_count++;
     pthread_mutex_unlock(l_count);
     pthread_mutex_unlock(count_l);
     return(0);
}
void *thread4(void *vargp) {
     tid_4 = pthread_self();
     pthread_mutex_lock(l_count);
     pthread_mutex_lock(count_l);
     ref_count--;
     pthread_mutex_unlock(l_count);
     pthread_mutex_unlock(count_l);
     return(0);
}
```

```
void func1(void) {
     pthread_t t1,t2;
     pthread_create(&t1,NULL, thread1, NULL );
     pthread_create(&t2, NULL, thread2, NULL);
     pthread_join(t2, NULL); pthread_join(t1, NULL);
     exit(0);
}

void func2(void) {
     pthread_t t3, t4;
     pthread_create(&t4,NULL, thread4, NULL );
     pthread_create(&t3, NULL, thread3, NULL);
     pthread_join(t3, NULL);
     pthread_join(t4, NULL);
     exit(0);
}
```

Problem 4. (10 points):

Consider the following three threads and four semaphores:

```
/* Initialize x */
 x = 1;
/* Initialize semaphores */
s1 =
s2 =
s3 =
s4 =
```

```
void thread1()
{
  while (x != 360)
  {

     x = x * 2;


  }
  exit(0);
}
```

```
void thread2()
{
  while (x != 360)
  {

     x = x * 3;


  }
  exit(0);
}
```

```
void thread3()
{
  while (x != 360)
  {

     x = x * 5;


  }
  exit(0);
}
```

Provide initial values for the four semaphores and add P(), V() semaphore operations (using the four semaphores) in the code for thread 1, 2 and 3 such that the process is guaranteed to terminate.