Concurrent & Distributed Systems (CS 475)
Fall 2017
Review Questions

Problem 1

In UNIX systems, processes that wish to print a file on a printer place a "job" in the print queue associated with the printer. Each print job is assigned a *unique sequence number* by the process that places it in the print queue. A file is used to store the next sequence number to be assigned to a print job. The file consists of single line containing the sequence number in ASCII. Each process that needs to assign a sequence number goes through the following steps:

1. It reads the sequence number from the file
2. It uses the number
3. It increments the number and writes it back to the file

A program that does this is shown below. My contention is that this program has a serious bug in it. Explain what the bug is, and how you would modify the program to remove this bug.

```
#define SEQFILE "seqno"                    /* filename */
#define MAXBUFF 100
main()
{
        int fd, n, seqno;
        char buff[MAXBUFF + 1];

        if ((fd = open(SEQFILE, 2)) < 0)
           err_sys("can't open %s", SEQFILE);
        lseek(fd,0L,0);                    /* rewind before read */
        if (( n = read(fd,buff,MAXBUFF)) <= 0)
           err_sys("read error");
        buff[n] = `\0`;                    /* null terminate for sscanf */
        if (( n = sscanf(buff, "%d", &seqno)) != 1)
           err_sys("sscanf error");
        seqno++;                           /* increment the sequence number */
        sprintf(buff,"%d\n",seqno);
        n = strlen(buff);
        lseek(fd,0L,0);                    /* rewind before write */
        if (write(fd,buff,n) != n)
           err_sys("write error");
}
```

A1.

Problem 2 (20 pts)

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 are there in the laundromat)

Using semaphores, write the code to allocate and release the machines. Assume that the available array is initialized to all ones. You are free to use as many semaphores and other variables as needed for your solution. Please specify the initial values of the semaphores and/or any global variables used in your solution.

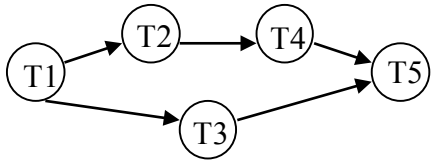
```
/* Specify initial values below  */


int allocate()                /* returns index of available machine */
{




}

release(int machine)                /* releases machine */
{




}
```

Problem 3

A precedence graph is a directed, acyclic graph. Nodes represent tasks, and arcs indicate the order in which tasks are to be accomplished. In particular, a task can execute as soon as its predecessors have been completed. Assume that the tasks are processes and that each process has the following outline:

```
Process T{
        Wait for predecessors, if any;
        Body of the task;
        Signal successors, if any;
}
```

Using semaphores, show how to synchronize five processes whose permissible execution order is specified by the following precedence graph:



Do not impose constraints that are not specified in the graph. For example, T2 and T3 can execute concurrently after T1 completes.

Answer:

Problem 4

Consider the following C program. (For space reasons, we are not checking error return codes. You can assume that all functions return normally.)

```
int val = 10;

void handler(sig)
{
     val += 5;
     return;
}


int main()
{
     int pid;

     signal(SIGCHLD, handler);
     if ((pid = fork()) == 0) {
          val -= 3;
          exit(0);
     }
     waitpid(pid, NULL, 0);
     printf("val = %d\n", val);
     exit(0);
}
```

What is the output of this program? `val` = _____

Explain your answer below.

Problem 5

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
int main () {
   if (fork() == 0) {
         if (fork() == 0) {
            printf("3");
         }
         else {
              pid_t pid; int status;
              if ((pid = wait(&status)) > 0) {
                printf("4");
              }
         }
   }
   else {
     printf("2");
     exit(0);
   }
   printf("0");
   return 0;
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program.

A. 32040

B. 34002

C. 30402

D. 23040

E. 40302

Problem 6

Part A
Consider the following C code. Assume that each function is executed by a unique thread on a uniprocessor system.

```
1 void thread1() {
2     P(lock1);
3     P(lock2);
4     P(lock3);
5
6     /* do some work */
7
8     V(lock2);
9     V(lock3);
10    V(lock1);
11 }


12 void thread2() {
13    P(lock1);
14    P(lock2);
15    P(lock3);
16
17    /* do some work */
18
19    V(lock1);
20    V(lock2);
21    V(lock3);
22 }
```

Does this code contain a deadlock? If so, write a sequence of line numbers that, when executed in that order, will cause the deadlock.

Part B

Consider the following C code:

```
1 void thread1() {
2     P(lock1);
3     P(lock2);
4
5     /* do some work */
6
7     V(lock2);
8     V(lock1);
9 }


10 void thread2() {
11    P(lock3);
12    P(lock1);
13
14    /* do some work */
15
16    V(lock1);
17    V(lock3);
18 }


19 void thread3() {
20    P(lock2);
21    P(lock3);
22
23    /* do some work */
24
25    V(lock3);
26    V(lock2);
27 }
```

Does this code contain a deadlock? If so, write a sequence of line numbers that, when executed in that order, will cause the deadlock.

Problem 7

Why is it **strongly recommended** that in multi-threaded programs, the *pthread_cond_wait()* call (in Pthreads) or the *wait()* call (in a Java program) be embedded in a while loop? Your explanation should be based on the actions taken by the Pthreads thread library on a *pthread_cond_wait()* call or by the Java run-time system on a *wait()* call. Give an example to illustrate any problems that can occur if the *pthread_cond_wait()* call or the *wait()* call is not inside a while loop

Problem 8

Are semaphores more powerful as a synchronization mechanism than monitors (the combination of mutex locks and condition variables)? Are monitors more powerful than semaphores?
How does one prove or disprove that one synchronization mechanism is more “powerful” than another one?