

Full Name: _____

Concurrent & Distributed Systems (CS 475)
Spring 2010
Midterm Exam

Instructions

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 80 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Good luck!

Problem 1 (10 pts)

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.

If multiple processes execute the program concurrently, the same sequence number can be assigned to multiple print jobs.

The program should be modified to use semaphores as a lock to ensure mutual exclusion. In other words, a (system-level) semaphore should be declared in the program initialized to 1. The code for the program should be bracketed in P() and V() operations on the semaphore to ensure that the reads and writes of the file storing the sequence number are executed as a critical section.

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 */*

```
semaphore machines = NMACHINES;  
semaphore mutex = 1;
```

```
int allocate()                                /* returns index of available machine */  
{  
    int i;  
    P(machines);  
    P(mutex);  
    for (i = 0; i < NMACHINES; i++) {  
        if (available[i] == 1) {  
            available[i] = 0;  
            break;  
        }  
    }  
    V(mutex);  
    return(i);  
}
```

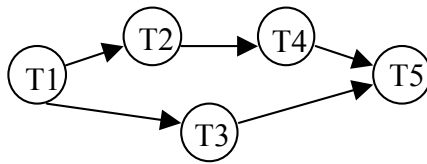
```
release(int machine)                        /* releases machine */  
{  
  
    P(mutex);  
    available[machine] = 1;  
    V(mutex);  
    V(machines);  
  
}
```

Problem 3 (20 pts)

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:

Semaphore S2, S3, S4, S5a, S5b = 0;

T1 { Body; V(S2); V(S3); }

T2 { P(S2); Body; V(S4); }

T3 { P(S3); Body; V(S5a); }

T4 { P(S4); Body; V(S5b); }

T5 { P(S5a); P(S5b); Body; }

Problem 4 (10 pts)

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 = 15
Explain your answer below.

val has an initial value of 10. When the child exits, a SIGCHLD is delivered to the parent. The signal handler increments val by 5 leading to the final value of 15 being printed out.

Note that the child has its own copy of val. Changes made to the child's copy do not affect the parent's copy.

Problem 5 (10 pts)

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 Y
- B. 34002 N
- C. 30402 Y
- D. 23040 Y
- E. 40302 N

Problem 6 (10 pts)

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.

No 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.

Yes, a deadlock will occur if lines 2, 11, and 20 are executed.