Northern Michigan University (Marquette Co, MI)

CS444-01-25F: Parallel & Distributed Processing (Andrew A. Poe) Name: ____________________________
Practice Final Examination Page 1/3 Monday 8 December 2025 7:00 P.M. EST

Answer all questions. Show all work.

Time: 110 minutes.

1. Consider the following parallel program;

```
x = 1; y = 2; z = 3;
co {
        x = x+y
%%      y = y*z
%%      x = x+z
}
```

What are all possible values for x, y, and z when this program halts?

ABC
x=6;y=6;z=3
ACB
x=6;y=6;z=3
BAC
x=10;y=6;z=3
BCA
x=10;y=6;z=3
CAB
x=6;y=6;z=3
CBA
x=10;y=6;z=3

2. Consider the following parallel program:

```
x=10, c = 1
co {
        <await x==0>; c = 0;
%%      while (c==1) <x = x – 1;>
%%      while (c==1) {if (x < 0) <x = 10>;}
}
```

Will this program terminate under a weakly fair system? Why or why not?

The only way this will ever terminate is if c becomes 0 at some point. Under a weakly fair system, this is not guaranteed, since whenever x is 0 there are two statements that might run, one that will terminate the program and one that will make x -1 and keep the program going. The await line might lose the race every time.

Will this program terminate under a strongly fair system? Why or why not?

Under a strongly fair system--which is very difficult to implement--x will be 0 infinitely often,so the await is guaranteed to fire sooner or later, thus terminating the program.

3. Bubble sorting is when array elements can only be compared to and swapped with their neighbors in the array. Describe, in English, how to implement a parallel bubble sort, as efficiently as possible. Regular bubble sorting runs in n^2 time. How fast can parallel bubble sorting be?

In parallel, you can compare (and swap if necessary) items <0,1>,<2,3>,<4,5.>, etc., simultaneously. Then <1,2><3,4><5,6>,etc., simultanously and repeat these two patterns a maximum of n times, resulting in a sorted array in n time.

4. A ballroom dance club has room for fifty dancers under the provision that there is never more men than women in the room (although there can be more women than men). (If you don't believe me, check out a ballroom dance club sometime.) The exception is that anyone can leave at any time; no one is a prisoner, so if a woman leaving results in more men than women being present, that is allowable, but no more men may enter until balance is achieved.

Write semaphore code that correctly and fairly allows people to enter and exit the dance club. Pseudo-code is perfectly acceptable for this problem.

```
EnterBallroom:

P(admin);
if (sex=='M') {
 if (m+f==50 || m>=f || mw > 0) {
  mw++;
  V (admin);
  P (male);
 } else m++;
} else {
 if (m+f==50 || fw > 0) {
  fw++;
  V (admin);
  P (female);
 } else f++;

if (mw > 0 && m+f < 50 && m < f) {
 mw--
 m++;
 V(male);
} else if (fw > 0 && m+f < 50) (
 fw--;
 f++;
 V(female);
} else V (admin);
```

```
ExitBallroom:

P(admin);
if (sex=='M')
 m--;
else
 f--;
if (mw > 0 && m+f < 50 && m < f) {
 mw--;
 m++;
 V (male);
} else if (fw > 0 && m+f < 50) {
 fw--;
 f++;
 V (female);
} else V (admin);
```

5. The same problem but use monitor code. Write a monitor to solve the problem and indicate how individual processes would use the monitor. Pseudo-code is acceptable.

```
EnterBallroom (Dancer *d):

if (d->sex=='M') {
 if (m+f==50 || m>=f || mw > 0) {
  mw++;
  wait (male);
 } else m++;
} else {
 if (m+f==50 || fw > 0) {
  fw++;
  wait (female);
 } else f++;

if (mw > 0 && m+f < 50 && m < f) {
 mw--
 m++;
 signal (male);
} else if (fw > 0 && m+f < 50) (
 fw--;
 f++;
 signal (female);
}

ExitBallroom (Dancer *d)

if (d->sex=='M')
 m--;
else
 f--;
if (mw > 0 && m+f < 50 && m < f) {
 mw--;
 m++;
 signal (male);
} else if (fw > 0 && m+f < 50) {
 fw--;
 f++;
 signal (female);
}
```

6. Let's say you have a piece of code designed to be run on a shared memory machine with semaphores. You don't have such a machine; you have a distributed memory machine with message passing. You would like to convert the shared memory code to message passing code. There are a lot of issues to worry about, but for the purposes of this problem you need only worry about getting semaphores to work. How would you emulate semaphores on a distributed memory machine in a message passing language which has no critical section structure (because there is no shared memory)?

Designate a process (perhaps 0) to handle semaphore requests. When another process wants to wait on a semaphore they could have code along these lines:

MPI_Isend ("WAIT SEMNAME",0);
MPI_Recv (&msg,0);

In this way, the process would wait on the semaphore until it received a message from process 0.

A signal would be

MPI_Isend ("SIGNAL SEMNAME",0);

And the process would not have to wait for a receive.

Now, the master process would handle the semaphores. If a WAIT request came in, and the semaphore was greater than 0, the master process would decrement the semaphore and send a response to the waiting process immediately. If the semaphore was 0, it would put the waiting process id in a queue and not send a response right away.

When a SIGNAL comes in, if the queue is non-empty, the master process would dequeue the first item, and send a response to that process, thus allowing it to continue. If the queue were empty, then the master process would just increment the semaphore.