Northern Michigan University (Marquette Co, MI)

CS444-01-25F: Parallel & Distributed Processing (Andrew A. Poe) Name: _____
Midterm Examination (Exam 1) Wednesday 8 October 2025 9:00 A.M. EDT

1. Consider the following program:

```
int x = ?????;
co {

      <await (x >0) x = -x;>
%%
      <await (x < 0) x += 2;>
%%
      <await (x==0) x++;>
}
```

What integer constants could I replace ????? with that would cause this program to DEFINITELY terminate? What values would cause the program to POSSIBLY terminate? What values would cause the program to NEVER terminate?

x < -2 2 fires...and then nothing since x is still < 0
x = -2 231 fires, x is -1
x = -1 21 and then nothing since 3 cannot fire
x = 0 312 fire and x is 1
x =1 12 and then nothing since 3 cannot fire
x = 2 123 x is 1
x > 2 12 fire and then nothing since 3 cannot fire

DEFINITELY: -2, 0, 2 NEVER: <-2, -1, 1, >2

2. You have four integers stored in the variables A, B, C, and D. Write parallel code to sort the variables so that A >= B >= C >= D. Don't worry about making explicit threads. Use the "co" command instead. You can use awaits and atomic statements and that sort of thing, but this code must be as parallel as possible.

Here's one way:

```
if (A<D) {int t=A; A=D; D=t;}
co {
 if (A<C) {int t=A; A=C; C=t;}
%%
 if (B<D) {int t=B; B=D; D=t;}
}
if (B<C) {int t=B;B=C; C=t;}
co {
 if (A<B) {int t=A; A=B; B=t;}
%%
 if (C<D) {int t=C; C=D; D=t;}
}
```

There are MANY other ways.

Northern Michigan University (Marquette Co, MI)

CS444-01-25F: Parallel & Distributed Processing (Andrew A. Poe) Name: _____
Midterm Examination (Exam 1) Wednesday 8 October 2025 9:00 A.M. EDT

3. MONGOLIAN BARBECUE. There are many sauces and many customers at the Mongolian Barbecue and each customer must put three sauces on their meat, but no two customers can use the same sauce at the same time. Each customer ranks the sauces from favorite to least favorite, and runs the following strategy.

LOCK (favoritesauce);
LOCK (secondfavoritesauce);
LOCK (thirdfavoritesauce);
Add sauces to meat.
UNLOCK (favoritesauce);
UNLOCK (secondfavoritesauce);
UNLOCK (thirdfavoritesauce);

Give an example that shows that this strategy might result in Mongolian Deadlock.

ME: Lock (barbecue); Lock (Mustard); Lock (soy)
YOU: Lock (Mustard); Lock (Barbecue); Lock (soy)
DEADLOCK: I'm waiting for mustard; you're waiting for barbecue!

Suggest a strategy that STILL requires the customers to lock three sauces before using any of them, but prevents deadlock.

Don't base it around favorites. Base it on "fixed order." The sauces are arranged in a row on the table. Lock the one closest to the near end of the table, then the second closest, then the third closest. Everyone gets the three sauces they want and there is no deadlock because no one's locking the same sauces in a different order!

4. The basic operations on a semaphore are P (wait) and V (signal). Imagine that we augmented a semaphore with the operation signalall, which would release all threads waiting, if any, on that semaphore.

Under what circumstances would {sem.signalall(); sem.wait()} cause a thread to wait on the semaphore, but {sem.signal(); sem.wait()} would not?

If there are no threads waiting on sem, sem.signalall() would not release anything and then wait.

However, sem.signal() would increment sem and so the sem.wait() would not actually result in the thread waiting.

If I had regular old semaphores without the signalall() method, how could I use these semaphores to emulate the operation of a signalall() method?

By passing the baton! Keep track of how many threads are waiting on the semaphore. When a thread is released, it signals the semaphore to release another one, and so forth until all are released. If there aren't any waiting in the first place, then don't signal anything.