Northern Michigan University (Marquette Co, MI)

CS444-01-25F: Parallel & Distributed Processing (Andrew A. Poe) Name: ____________________________
Practice Endterm Examination (Exam 2) Page 1/2 Monday 1 December 2025 9:00 A.M. EST

Answer all questions. Show all work.

Time: 50 minutes.

1. Imagine a 4-way Yield sign at an intersection. This is like a 4-way stop sign but you don't actually have to stop if nothing is coming. Obviously, if the only cars at the intersection are moving north or south, nobody has to wait. Similarly, if the only cars are going east and west, nobody has to wait. The only time to have to wait is when east or west cars are at the intersection with north or south cars. (For this problem, we will assume that no one is turning right or left.)

Write semaphore code that correctly and fairly allows cars to pass safely over the intersection. Remember that any number of cars going in the same or opposite directions may go simultaneously, but cross traffic is disallowed.

```
void EnterIntersection () {

 P (admin);
 if (dir=='N' || dir=='S') {
  if (new > 0 || neww > 0) {
   cout << "WAITING" << endl;
   nnsw++;
   V (admin);
   P (NS);
  } else nns++;
  cout << "ENTERING" << endl;
  if (neww==0 && nnsw > 0) {
   nnsw--;
   nns++;
   V(NS);
  } else V(admin);
 } else {
  if (nns > 0 || nnsw > 0) {
   cout << "WAITING" << endl;
   neww++;
   V (admin);
   P (EW);
  } else new++;
  cout << "ENTERING" << endl;
  if (nnsw==0 && neww > 0) {
   neww--;
   new++;
   V(EW);
  } else V(admin);
 }
}

void ExitIntersection (){

 P(admin);
 if (dir=='N' || dir=='S') {
  nns--;
  if (nnsw > 0 && neww==0) {
  nnsw--;
  nns++;
  V(NS);
 } else if (neww > 0 && nns==0) {
  neww--;
  new++;
  V(EW);
 } else V(admin);
} else {
  new--;
  if (neww > 0 && nnsw==0) {
  neww--;
  new++;
  V(EW);
 } else if (nnsw > 0 && new==0) {
  nnsw--;
  nns++;
  V(NS);
 } else V(admin);
}
```

2. The same problem but use monitor code. Write a monitor to solve the problem and indicate how individual processes would use the monitor.

Notice that the monitor code is essentially the same, except without admin.

```
void EnterIntersection () {

if (dir=='N' || dir=='S') {
  if (new > 0 || neww > 0) {
   cout << "WAITING" << endl;
   nnsw++;
   P (NS);
  } else nns++;
  cout << "ENTERING" << endl;
  if (neww==0 && nnsw > 0) {
   nnsw--;
   nns++;
   V(NS);
  }
 } else {
  if (nns > 0 || nnsw > 0) {
   cout << "WAITING" << endl;
   neww++;
   P (EW);
  } else new++;
  cout << "ENTERING" << endl;
  if (nnsw==0 && neww > 0) {
   neww--;
   new++;
   V(EW);
  }
 }
}

void ExitIntersection (){

if (dir=='N' || dir=='S') {
  nns--;
  if (nnsw > 0 && neww==0) {
  nnsw--;
  nns++;
  V(NS);
 } else if (neww > 0 && nns==0) {
  neww--;
  new++;
  V(EW);
 }
} else {
  new--;
  if (neww > 0 && nnsw==0) {
  neww--;
  new++;
  V(EW);
 } else if (nnsw > 0 && new==0) {
  nnsw--;
  nns++;
  V(NS);
 }
}
```

This code would be in a monitor. The threads would call the monitor methods with parameters to indicate the specific car driving, perhaps a pointer to the car object.

3. The same problem but with message passing code. Have processes represent the cars and have them send messages to the master process (my_proc==0) when they want to use the intersection. Write code for the master process and indicate how the other processes would communicate with it (you only have to write the code, though, for the master process).

```
bool NS[tot]. EW[tot];
int nsct=0, ewct=0;
while (true) {
 MPI_Recv (&msg,&proc);
 if (msg=="Enter") {
  if (NSCar(proc)) {
   if (EWwait() > 0)
    NS[proc] = true;
   else {
    MPI_Isend ("Go",proc);
    nsct++
   };
  } else {
   if (NSwait() > 0)
```

```
    EW[proc]= true;
   else {
    MPI_Isend ("Go",proc);
    ewct++;
  }
 } else {
  if (NSCar(proc)) {
   nsct--;
   if (nsct==0)
    for (int i=1; i < tot; i++)
     if (EW[i]) {
      ewct++;
      MPI_Isend ("Go",i);
      EW[i] = false;
    }
  } else {
   ewct--;
   if (ewct==0)
    for (int i=1; i < tot; i++)
     if (NS[i]) {
      nsct++;
      MPI_Isend ("Go",i);
      NS[i] = false;
    }
  }
 }
}
```

NSCar is a boolean method that returns true if the process represents a north or south going car and false if it represents an east or west going car.

The EWwait and NSwait methods just check the EW or NS array to see if any entry is true, representing a waiting car.

The other processes will simply send a message to the master when they're ready to enter the intersection, wait for the "Go" message, and then send an exit message to the master when they have left the intersection.

4. Now, real intersections allow cars to turn left or right. You don't have to write code to handle this possibility, but indicate which operations (east-bound going straight, north-bound turning left, etc.) should be allowed to proceed simultaneously with each other, and which operations should not be handled simultaneously with each other. Assume that this intersection is in the USA and cars drive in the right lane.

If I am north going straight, I am OK with: north straight, north left, north right, south straight, south right, east right.

If I am north turning left, I am OK with: north straight, north left, north right, east right,

If I am north turning right, I am OK with: north straight, north left, north right, south straight, south right, east right, west straight, west left, west right

Similar for the other directions.