**Problem Set #2**

**1. (+15)**

Int east\_cars = 0, west\_cars = 0;

Lock mutex\_lock

Travel\_east(){

If(east\_cars > 0){

east\_cars++;

Drive\_onto\_bridge;

}

Else If(east\_cars == 0 && west\_cars == 0){

Aquire lock;

east\_cars++;

Drive\_onto\_bridge;

}

Else(){

Wait(Signal lock release)

}

}

Lock(){

While(east > 0 || west > 0)

Wait;

Signal(lock release)

}

And the same code for west with if(west\_cars > 0) and west\_cars++. This allows all east bound cars to follow while there is one east bound car still on the bridge. It is not starvation free. You could use semaphores but I thought you should only need a bridge lock because we don’t need to protect against starvation. With semaphores you can make it like the readers/writers problem but give a car waiting preference to acquire the bridge lock

**2. (+15)**  
  
Use a lock to signal between C1 and C2 that C1has finished and unlock C2. Cannot use a wait because T1 and T2 are in the same process.

Lock Mutex\_lock;

Condition condition1;

Int finished = 0;

T1

Aquire lock();

Run C1 code;

Finished = 1;

Release lock();

Flag condition();

T2

Aquire lock;

while(finished == 0)

Release lock();

Wait Condition();

Aquire lock();

Run C2 code;

Release lock();

**3. (+15)**

Because once a writer wants to write they signal readblock which blocks new readers and allows all current readers to finish thus signaling readcount = 0 and releasing writeblock. This prevents writer starvation.

**4. a. (+10)**

No the temp variable can changed by other functions if the process is not atomic, the local variable should be used to swap the values.

**b. (+10)**

Yes swap() is reentrant. There is no protection (lock mutex semaphore) so it will execute.

**5. (+10)**

Shared memory would be the best option because p2 won’t need to take or copy (unnessary and expensive time wise) the file from p1, just receive a pointer and compress it.

**6. (+15)**

A single monitor solution that allows both processes access to different functions concurrently.

Monitor 1(){

int variable1, variable2, variable3;

Conditions c1, c2, c3

Increment(){

Wait(c1);

Variable1++;

Signal(c1);

}

Decrement(){

Wait(c1);

Variable1--;

Signal(c1);

}

square(){

Wait(c2);

Square(Variable2);

Signal(c2);

}

Sqrt(){

Wait(c2);

Sqrt(Variable2);

Signal(c2);

}

sine(){

Wait(c3);

Sin(Variable3);

Signal(c3);

}

cosine(){

Wait(c3);

Cos(Variable3);

Signal(c3);

}