Question#1:

1. Mutual exclusion is satisfied because Pi enters CS only if: flag[j] == False. Pi sees whether Pj wants to enter or not, if Pj wants to and Pj also get the turn then Pi will set it’s flag[i] to false and wait until after Pj execute critical section and sets turn = i;
2. Progress is satisfied because if the no processes are executing in critical section, the turn will let exactly one to enter the critical section and another process will lose it priority by changing its flag to false.
3. It satisfies the bounded waiting requirement. Turn choses who can enter to the critical section. If Pj in the critical section, Pi will in the ‘while(turn == j);’ to wait Pj comes out from critical section and sets turn to i. Pi would not wait the Pj loses its priority infinitely.

Question#2:

a)(1)Process C gets mutex and mutex-1 = 1-1 = 0, Process C enter critical section but Process C needs goC which is held by ProcessA and Process C will be blocked in the goC queue. Because ProcessC holds the mutex then no other Processes can access the critical section. All three processes block permanently.

(2)Process A, Process C, Process B. Process A release goB but CPU schedules Process C which needs goC held by ProcessB. Then Process C and Process B are blocked permanently.

(3)Process A, Process B, Process C. Process A release goB and Process B gets goB and release goC. In the end, Process C gets goC. No process blocks permanently.

b)(1) if Process B goes first, it will be block for no goB and sets mutex = 0 and then Process A can’t get into the critical section. Both processes block permanently. If Process A go first and at any time the amount of goB process B needs isn’t bigger than Process A can release then neither process blocks permanently.

(2) In this case, there doesn’t exist any execution scenario to fix this problem because in the end, the amount of goB process B needs will definitely larger than Process A can release.

Question#3:

1. if(empty == N)

printf(“Buffer is empty\n”)

and if(full == N)

printf(“Buffer is full\n”)

have problem. They are not atomic operation if Producer if(full==N) and this time consumer buys a item the buffer is not full but Producer will still printf(“Buffer is full\n”)

1. Code for Producer process: Code for Consumer process:

Do{ Do{

Produce(item); wait(full);

Wait(empty); wait(mutex);

Wait(mutex); item = Buffer[outp];

Buffer[inp] = item; outp = (outp + 1)%N;

Inp = (inp + 1)%N; if(empty == N-1)

If(full == N-1) printf(“Buffer is empty\n”);

Printf(“Buffer is full”); consume(item);

Signal(mutex); signal(mutex);

Signal(full); signal(empty);

} while(True) } while(True)

Question#4:

Class Monitor FileControl{

Boolean isWriting = false;

Int readerReadingNumber = 0;

Condition variable r;// reader waiting queue

Condition variable w;//writer waiting queue

Void WriterEntry(){

If(readerReadingNumber == 0 && isWriting == false){

Wait(r);

isWriting = true;

} else{

wait(w)

}

}

Void WriterExit(){

isWriting = false;

if(w != null) {

signal(w);

} else{

signal(r);

}

}

void ReaderEntry(){

if(isWriting == true || w != null){

wait(r);

} else{

readerReadingNumber++;

signal(r);

}

}

void ReaderExit(){

readerReadingNumber--;

if(readerReadingNumber == 0){

signal(w);

}

}

}

Question#5:

If signal is the last step then it should become the process which can get lock from others.