CSC139 Operating System Principles

Sample Midterm Question on Assignment 1 (Producer/Consumer)

Consider the following function that implements one of the two processes (consumer or producer) in Assignment 1. Assuming that everything else is the same as given in the distributed solution, answer the following questions:

void FuncX(int bufSize, int itemCnt, int randSeed){

int i, in = 0, out = 0, val;

1 srand(randSeed);

2 for (i=0; i<itemCnt; i++) {

3 out = GetOut();

4 while((in+1)%bufSize == out){

5 out = GetOut();

6 }

7 val = GetRand(0, 100);

8 MemOp2(in, val);

9 in = (in+1) % bufSize;

10 SetIn(in);

}

}

int MemOp1(int indx){

1 int val;

2 void\* ptr = gShmPtr + 4\*sizeof(int) + indx\*sizeof(int);

3 memcpy(&val, ptr, sizeof(int));

4 return val;

}

void MemOp2(int indx, int val){

1 void\* ptr = gShmPtr + 4\*sizeof(int) + indx\*sizeof(int);

2 memcpy(ptr, &val, sizeof(int));

}

1. Does FuncX implement the consumer or the producer? Briefly justify your answer.

This is the producer, because it is setting the in. Also, the conditional in the while loop, and is writing to the buffer.

2. What does each of memory operations MemOp1 and MemOp2 do?

It copies data into the buffer.

3. What’s the purpose of calling GetOut on Line 3 of FuncX?

To get the updated value of out from the shared memory and update the local variable.

4. What’s the purpose of calling SetIn on Line 10 of FuncX?

Updates the shared variable <in> in shared memory

5. What’s the purpose of adding 4\*sizeof(int) on Line 2 of MemOp1?

Retains the header information, and starts at the beginning of the pointer.

6. Trace as thoroughly as possible the consequences of the following code change, assuming everything else remains the same. Your answer should precisely describe the change in the program’s behavior and specify the number of items produced/consumed. Assume that the buffer size is n.

Deleting Line 3 from FunctX

Items produced would only be bufSize – 1 and it will stop. Out is kept at 0. From the consumer point of view, the consumer has consumed everything, but the produce doesn’t have the proper out variable. This will create a deadlock. The consumer will consume bufSize – 1 items.

7. Modify FuncX above such that it prints a message as long as it is waiting for the other process. Your code should not print more than one message for each item the process waits for.

Set a Boolean outside the while loop and set up the print statement to only fire once.

8. Suppose that we chose to use message passing rather than shared memory to communicate the values of “in” and “out” between the consumer and user. Whenever a process updates “in” or “out”, it send a message to the other process. Replace the GetOut and SetIn in FuncX with either send or receive operations (don’t worry about syntax or parameters), and indicate whether it will be more efficient to use blocking or non-blocking send/receive.

The getOut outside the while loop should be nonblocking, but inside it should be blocking. The SetIn should be non blocking because setting in should not prevent me from producing.

9. If we modify the problem such that the buffer size is always guaranteed to be greater than or equal to the number of items produced (unbounded buffer), modify FuncX to account for this change as efficiently as possible. Also, describe the changes that will have to be made on the other side.

The while loop and getOut would be eliminated. Because there is no need to wait for the buffer to be empty. SetIn is still needed, because the consumer needs to know where to eat. The in will only need to be incremented, and not modulo with bufSize

10. If we modify the problem such that there are multiple producers but a single consumer, modify FuncX to account for this change as efficiently as possible. Also, describe the changes that will have to be made on the other side. Assume that there are x producers, each producing produces itemCnt, and that the consumer knows the number of producers and consumes all produced items.

We are going to need to lock the SetIn in the producer to code to prevent the producers from having a conflict because they are accessing a shared variable.