DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Lab 09 – Process Synchronization

Objective(s):

- Learn about Producer Consumer Problem
- Learn about Preston's Algorithm

Producer-Consumer problem

The Producer-Consumer problem is a classical multi-process synchronization problem, that is we are trying to achieve synchronization between more than one process.

There is one Producer in the producer-consumer problem, Producer is producing some items, whereas there is one Consumer that is consuming the items produced by the Producer. The same memory buffer is shared by both producers and consumers which is of fixed-size.

The task of the Producer is to produce the item, put it into the memory buffer, and again start producing items. Whereas the task of the Consumer is to consume the item from the memory buffer.

Let's understand what is the problem?

Below are a few points that considered as the problems occur in Producer-Consumer:

- The producer should produce data only when the buffer is not full. In case it is found that the buffer is full, the producer is not allowed to store any data into the memory buffer.
- Data can only be consumed by the consumer if and only if the memory buffer is not empty. In case it is found that the buffer is empty, the consumer is not allowed to use any data from the memory buffer.
- Accessing the memory buffer should not be allowed to producer and consumer at the same time.

The code for the producer process can be modified as follows:

```
while (true) {

/* produce an item in next produced */

while (counter == BUFFER SIZE); /* do nothing */
buffer[in] = next produced;
in = (in + 1) % BUFFER SIZE;
counter++;

}
```

The code for the consumer process can be modified as follows:

```
while (true) {

while (counter == 0); /* do nothing */
next consumed = buffer[out];
out = (out + 1) % BUFFER SIZE;
counter--;

/* consume the item in next consumed */

}
```

Although the producer and consumer routines shown above are correct separately, they may not function correctly when executed concurrently. As an illustration, suppose that the value of the variable counter is currently 5 and that the producer and consumer processes concurrently execute the statements "counter++" and "counter--". Following the execution of these two statements, the value of the variable counter may be 4, 5, or 6! The only correct result, though, is counter == 5, which is generated correctly if the producer and consumer execute separately.

Peterson's algorithm

Peterson's algorithm (or **Peterson's solution**) is a concurrent programming algorithm for mutual exclusion that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication.

The producer consumer problem (or bounded buffer problem) describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue. Producer produces an item and puts it into a buffer. If the buffer is already full then the producer will have to wait for an empty block in the buffer. Consumer consume an item from a buffer. If the buffer is already empty then consumer will have to wait for an item in the buffer. Implement Peterson's Algorithm for the two processes using shared memory such that there is mutual exclusion between them. The solution should be free from synchronization problems.

Peterson's Algorithm is used to synchronize two processes. It uses two variables, a bool array **flag** of size 2 and an int variable **turn** to accomplish it.

```
do {

    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = false;

        remainder section

} while (true);
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

CODE:

```
#include <iostream>
#include <thread>

int flag[2];
int turn;
const int MAX = 100;
int ans = 0;

void lock_init()
{
    // Initialize lock by reseting the desire of
    // both the threads to acquire the locks.
    // And, giving turn to one of them.
    flag[0] = flag[1] = 0;
    turn = 0;
}

// Executed before entering critical section
void lock(int self)
{
    // Set flag[self] = 1 saying you want to acquire lock
    flag[self] = 1;

    // But, first give the other thread the chance to
    // acquire lock
    turn = 1-self;

    // Wait until the other thread looses the desire
    // to acquire lock or it is your turn to get the lock.
```

```
    while (flag[1-self]==1 && turn==1-self) ;
}

// Executed after leaving critical section
void unlock(int self)
{
    // You do not desire to acquire lock in future.
    // This will allow the other thread to acquire
    // the lock.
    flag[self] = 0;
}

// A Sample function run by two threads created
// in main()
void func(int s)
{
    int i = 0;
    int self =s;
    std::cout<<"Thread Entered: \n"<< self;

    lock(self);

    // Critical section (Only one thread
    // can enter here at a time)
    for (i=0; i<MAX; i++)
        ans++;

    unlock(self);
}
```

```
// Driver code
int main()
{
    // Initialized the lock then fork 2 threads
    lock_init();

    // Create two threads (both run func)
    std::thread threadObj1(func, 0);
    threadObj1.join();
    std::thread threadObj2(func, 1);
    threadObj2.join();

    std::cout<<"Actual Count:"<<ans<<" | Expected Count: \n"<< MAX*2;

    return 0;
}
```

Basically, Peterson's algorithm provides guaranteed mutual exclusion by using only the shared memory. It uses two ideas in the algorithm:

1. Willingness to acquire lock.
2. Turn to acquire lock.

A thread expresses its desire to acquire a lock and sets **flag[self] = 1** and then gives the other thread a chance to acquire the lock. If the thread desires to acquire the lock, then, it gets the lock and passes the chance to the 1st thread. If it does not desire to get the lock then the while loop breaks and the 1st thread gets the chance.

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Lab Task: Submit the Peterson algorithm code example explained in the lab manual.

ASSIGNMENT # 09

1. Write a program for producer consumer problem.

SUBMISSION GUIDELINES

- Take a screenshot of each task(code and output).
- Place all the screenshots in a single word file labeled with Roll No and Lab No. e.g. **'cs181xxx_Lab01'.**
- Convert the file into PDF.
- Submit the file at LMS