COSC 350 System Software (Lab #10)

Task#1

Write complete two C programs "msgQsnd.c" and "msgQrcv.c" to communicate through message queue .

msgQsnd.c:

- Create a message queue with rw-r-r. To create a message queue, use existing file name "msgQsnd.c for creating a key value.
- Ask a message " two integers value" and send to the message queue.
- Keep asking a message until EOF (Ctr-D)
- Remove the message queue with termination.

msgQrcv.c:

- Receive a message (two integers) from the message queue created by msgQsnd.c.
- Calculate sum of two integers and display result on standard output.
- keep reading a message until EOF

What is producer Consumer Problem

- Two processes share a common, fixed-sized buffer size 10.
- Producer puts information into the buffer, and consumer takes it out.

```
#define N 10
int count = 0;
void producer()
{
    int item
    while (ture)
    {
          item = produce_item();
          if (count == N)
             sleep();
          insert_item(item)
          count = count + 1;
          if (count ==1)
             wakeup(consumer);
    }
}
```

```
void cunsumer()
{
    int item;
    while(true)
    {
           if (count == 0)
              sleep();
           item = remove_item();
           count = count – 1;
           if (count == N – 1)
             wakeup(producer);
           consume_item(item);
    }
}
```

Troubles arises

- When the producer wants to put a new item in the buffer, but it is already full.
- When the consumer tries to take an item from the buffer, but buffer is already empty.

Solutions for each case

- When the producer wants to put a new item in the buffer, but it is already full.
    - Solution – producer is going to sleep, awakened by customer when customer has removed on or more items.
- When the consumer tries to take an item from the buffer, but buffer is already empty.
    - Solution – customer is going to sleep, awakened by the producer when producer puts one or more information into the buffer.

Task#2. Write a complete C program to simulate producer consumer problem with shared memory without using semaphores or mutex for synchronization.

You need write three programs

- build.c:
    - create shared memory space for saving five integers. (array of integer)
    - must have extra information for insert new data or delete a data from the shared memory.
- producer.c: simulate producer
    - Producer runs forever
    - Producer create a random number between 1 to 10 and save proper slot in shared memory. If shared memory is full, producer need blocked somehow until space become available.
    - Then print contents of shared memory.
- consumer.c: simulate consumer
    - consumer runs forever.
    - If there is any data in the buffer, remove it from the shared memory. If there is no data in the buffer, consumer need blocked somehow until data become available.
    - Then, print contents of shared memory.

Task#3 Write a complete C program to simulate producer consumer problems with semaphores and shared memory. If you use semaphores properly, programmer does not need consider synchronization between two processes (mutual exclusion and race condition for shared memory).

You need write three programs

- build.c :
    - create shared memory space for saving five integers.
    - create a semaphore structure with three members for simulating producer consumer problems.
- producer.c: simulate producer
    - Producer runs forever
    - Producer create a random number between 1 to 10 and save proper slot in shared memory. Then print contents of shared memory.

- consumer.c: simulate consumer
    - consumer runs forever.
    - if there is any number in the buffer, remove it
    - Then, print contents of shared memory.

Algorithms for Solving Producer Customer Problem using semaphores

```
#define N 10
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer ()
{
    int item;

    while (true)
    {
           item = produce_item();
           down (&empty);
           down (&mutex);
           insert_item(item);
           up(&mutex);
           up(&full);
    }
}
```

```
void consumer()
{
    int item;

    while (true)
    {
           down(&full)
           down(&mutex)
           item = remove_item();
           up(&mutex);
           up(&empty);
           consume_item(item);
    }
}
```