Assignment 09 | Operating System CE-092

Assignment submission for Operating System subject week 9.

nevilparmar24@gmail.com

Sem_init function :

#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);

sem_init() initializes the unnamed semaphore at the address pointed to by sem. The value argument specifies the initial value for the semaphore.
The pshared argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.

If pshared has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads (e.g., a global variable, or a variable allocated dynamically on the heap).

If pshared is nonzero, then the semaphore is shared between processes, and should be located in a region of shared memory.
sem_init() returns 0 on success; on error, -1 is returned, and errno is set to indicate the error.
E.g.
sem_t mutex;
sem_init(&mutex,0,1);

Sem_wait function :

#include <semaphore.h>
int sem_wait(sem_t *sem);

sem_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the

function returns immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement. This function returns 0 on success; on error, the value of the semaphore is left unchanged, -1 is returned, and errno is set to indicate the error.
E.g.
sem_t mutex;
sem_wait(&mutex);

Sem_post function :

#include <semaphore.h>
int sem_post(sem_t *sem);

sem_post() increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait call will be woken up and proceed to lock the semaphore.
sem_post() returns 0 on success; on error, the value of the semaphore is left unchanged, -1 is returned, and errno is set to indicate the error.
E.g.
sem_t mutex;
sem_post(&mutex);

Task 1:

Write a program to implement a solution of bounded buffer producer consumer problem using semaphores.

Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include<unistd.h>
#include <stdio.h>

#define MaxItems 5 // Maximum items a producer can
```

```
produce or a consumer can consume
#define BufferSize 5 // Size of the buffer

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno);
void *consumer(void *cno);

void *producer(void *pno)
{
    int item;
    while(1) {
        item = rand(); // Produce an random item
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        // Critical Section Begins

        buffer[in] = item;
        printf("Producer : Insert Item %d at %d\n",
buffer[in],in);
        in = (in+1)%BufferSize;
        /*
        * This sleep is optional
        * It is added just to observe the output
properly
        * becaue without the prog runs very quickly
        */
```

```
        sleep(1);

        // Critical section ends here
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}

void *consumer(void *cno)
{
    while(1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        // Critical section begins

        int item = buffer[out];
        printf("Consumer : Remove Item %d from %d\n",
item, out);
        out = (out+1)%BufferSize;
        /*
        * This sleep is optional
        * It is added just to observe the output
properly
        * becaue without the prog runs very quickly
        */
        sleep(1);

        // Critical section ends here
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
```

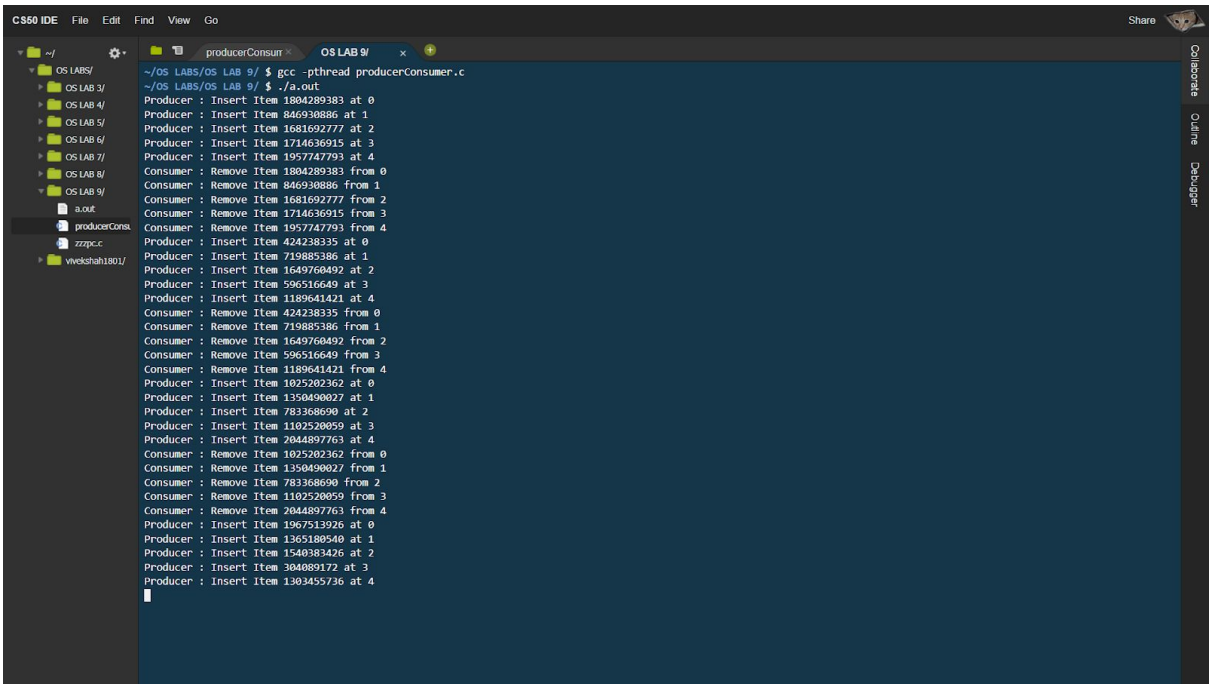
```c
int main()
{
    pthread_t pro,con;
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);

    pthread_create(&pro, NULL, (void *)producer, NULL);
    pthread_create(&con, NULL, (void *)consumer, NULL);
    pthread_join(pro, NULL);
    pthread_join(con, NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}
```

Output:



Nevil Parmar
CE-092
https://nevilparmar.me