OPERATING SYSTEMS

Lab-9: Peterson solution, turn , flag , order of execution, producer consumer

Name – Vinay Santosh Menon
Registration Number - 20BAI1103

TASK:

Implement Peterson's solution for two processes.

To carry out the producer consumer problem

To carry out the order of execution problem

Peterson's Solution

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.

CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdbool.h>
#define _BSD_SOURCE
#include <sys/time.h>
#include <stdio.h>
#define BSIZE 8
#define PWT 2
#define CWT 10
#define RT 10
int shmid1, shmid2, shmid3, shmid4;
key_t k1 = 5491, k2 = 5812, k3 = 4327, k4 = 3213;
bool* SHM1;
int* SHM2;
int* SHM3;
```

```
int myrand(int n)
{
   time_t t;
   srand((unsigned)time(&t));
   return (rand() % n + 1);
}
int main()
{
   shmid1 = shmget(k1, sizeof(bool) * 2, IPC_CREAT | 0660);
   shmid2 = shmget(k2, sizeof(int) * 1, IPC_CREAT | 0660);
   shmid3 = shmget(k3, sizeof(int) * BSIZE, IPC_CREAT | 0660);
   shmid4 = shmget(k4, sizeof(int) * 1, IPC_CREAT | 0660);
   if (shmid1 < 0 || shmid2 < 0 || shmid3 < 0 || shmid4 < 0) {
       perror("Main shmget error: ");
       exit(1);
   }
   SHM3 = (int*)shmat(shmid3, NULL, 0);
   int ix = 0;
   while (ix < BSIZE)
       SHM3[ix++] = 0;
   struct timeval t;
   time_t t1, t2;
   gettimeofday(&t, NULL);
   t1 = t.tv_sec;
   int* state = (int*)shmat(shmid4, NULL, 0);
   *state = 1;
```

```
    int wait_time;
    int i = 0;
    int j = 1;
    if (fork() == 0)
    {
        SHM1 = (bool*)shmat(shmid1, NULL, 0);
        SHM2 = (int*)shmat(shmid2, NULL, 0);
        SHM3 = (int*)shmat(shmid3, NULL, 0);
        if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 ==
(int*)-1) {
            perror("Producer shmat error: ");
            exit(1);
        }
        bool* flag = SHM1;
        int* turn = SHM2;
        int* buf = SHM3;
        int index = 0;
        while (*state == 1) {
            flag[j] = true;
            printf("Producer is ready now.\n\n");
            *turn = i;
            while (flag[i] == true && *turn == i)
                ;

            index = 0;
            while (index < BSIZE) {
```

```
                if (buf[index] == 0) {
                    int tempo = myrand(BSIZE * 3);
                    printf("Job %d has been produced\n",
tempo);
                    buf[index] = tempo;
                    break;
                }
                index++;
            }
            if (index == BSIZE)
                printf("Buffer is full, nothing can be
produced!!!\n");
            printf("Buffer: ");
            index = 0;
            while (index < BSIZE)
                printf("%d ", buf[index++]);
            printf("\n");

            flag[j] = false;
            if (*state == 0)
                break;
            wait_time = myrand(PWT);
            printf("Producer will wait for %d seconds\n\n",
wait_time);
            sleep(wait_time);
        }
        exit(0);
```

```
    }
    if (fork() == 0)
    {
        SHM1 = (bool*)shmat(shmid1, NULL, 0);
        SHM2 = (int*)shmat(shmid2, NULL, 0);
        SHM3 = (int*)shmat(shmid3, NULL, 0);
        if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 ==
(int*)-1) {
            perror("Consumer shmat error:");
            exit(1);
        }
        bool* flag = SHM1;
        int* turn = SHM2;
        int* buf = SHM3;
        int index = 0;
        flag[i] = false;
        sleep(5);
        while (*state == 1) {
            flag[i] = true;
            printf("Consumer is ready now.\n\n");
            *turn = j;
            while (flag[j] == true && *turn == j)
                ;

            if (buf[0] != 0) {
                printf("Job %d has been consumed\n", buf[0]);
```

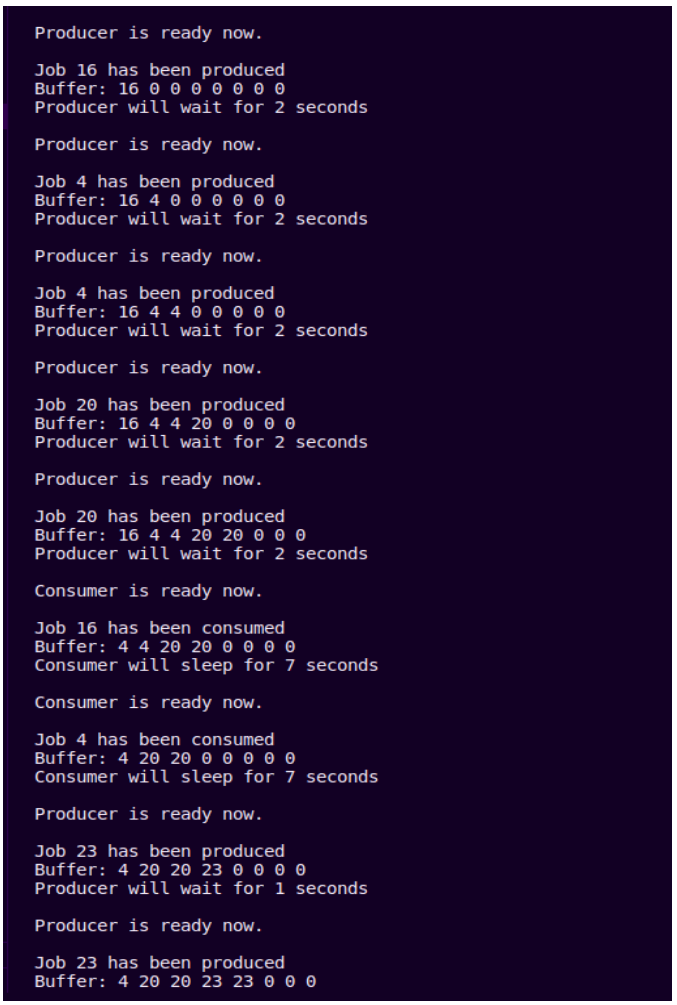
```
                buf[0] = 0;
                index = 1;
                while (index < BSIZE)
                {
                    buf[index - 1] = buf[index];
                    index++;
                }
                buf[index - 1] = 0;
            } else
                printf("Buffer is empty, nothing can be
consumed!!!\n");
            printf("Buffer: ");
            index = 0;
            while (index < BSIZE)
                printf("%d ", buf[index++]);
            printf("\n");

            flag[i] = false;
            if (*state == 0)
                break;
            wait_time = myrand(CWT);
            printf("Consumer will sleep for %d seconds\n\n",
wait_time);
            sleep(wait_time);
        }
        exit(0);
    }
```

```
   while (1) {
       gettimeofday(&t, NULL);
       t2 = t.tv_sec;
       if (t2 - t1 > RT)
       {
           *state = 0;
           break;
       }
   }

   wait();
   wait();
   printf("The clock ran out.\n");
   return 0;
}
```

RESULT



Producer Consumer Problem

We will be using Semaphores to solve this problem now.

A semaphore S is an integer variable that can be accessed only through two standard operations : wait() and signal().

The wait() operation reduces the value of semaphore by 1 and the signal() operation increases its value by 1.

CODE

```
#include <pthread.h>
```

```c
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>




#define MaxItems 5
#define BufferSize 5
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
   int item;
   for(int i = 0; i < MaxItems; i++) {
       item = rand();
       sem_wait(&empty);
       pthread_mutex_lock(&mutex);
       buffer[in] = item;
       printf("Producer %d: Insert Item %d at %d\n", *((int
*)pno),buffer[in],in);
       in = (in+1)%BufferSize;
       pthread_mutex_unlock(&mutex);
```

```
        sem_post(&full);
    }
}
void *consumer(void *cno)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int
*)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
int main()
{
    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);
    int a[5] = {1,2,3,4,5};
    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void
*)&a[i]);
    }
```

```
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void
*)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
    return 0;

}
```

RESULT

```
Producer 1: Insert Item 1804289383 at 0
Producer 1: Insert Item 846930886 at 1
Producer 1: Insert Item 1681692777 at 2
Producer 1: Insert Item 1714636915 at 3
Producer 1: Insert Item 1957747793 at 4
Consumer 1: Remove Item 1804289383 from 0
Consumer 1: Remove Item 846930886 from 1
Producer 2: Insert Item 424238335 at 0
Consumer 1: Remove Item 1681692777 from 2
Consumer 1: Remove Item 1714636915 from 3
Producer 3: Insert Item 719885386 at 1
Consumer 1: Remove Item 1957747793 from 4
Producer 4: Insert Item 1649760492 at 2
Producer 2: Insert Item 1189641421 at 3
Producer 5: Insert Item 596516649 at 4
Consumer 2: Remove Item 424238335 from 0
Consumer 2: Remove Item 719885386 from 1
Producer 3: Insert Item 1025202362 at 0
Producer 3: Insert Item 2044897763 at 1
Consumer 2: Remove Item 1649760492 from 2
Consumer 2: Remove Item 1189641421 from 3
Consumer 2: Remove Item 596516649 from 4
Producer 5: Insert Item 1102520059 at 2
Producer 2: Insert Item 783368690 at 3
Producer 3: Insert Item 1967513926 at 4
Consumer 3: Remove Item 1025202362 from 0
Consumer 3: Remove Item 2044897763 from 1
Consumer 3: Remove Item 1102520059 from 2
Producer 4: Insert Item 1350490027 at 0
Producer 2: Insert Item 1540383426 at 1
Consumer 3: Remove Item 783368690 from 3
Producer 5: Insert Item 1365180540 at 2
Producer 3: Insert Item 304089172 at 3
Consumer 3: Remove Item 1967513926 from 4
Producer 4: Insert Item 1303455736 at 4
Consumer 4: Remove Item 1350490027 from 0
Producer 2: Insert Item 35005211 at 0
Consumer 4: Remove Item 1540383426 from 1
Consumer 4: Remove Item 1365180540 from 2
Producer 5: Insert Item 521595368 at 1
Consumer 4: Remove Item 304089172 from 3
Consumer 4: Remove Item 1303455736 from 4
Producer 4: Insert Item 294702567 at 2
Producer 4: Insert Item 336465782 at 3
Producer 5: Insert Item 1726956429 at 4
Consumer 5: Remove Item 35005211 from 0
Consumer 5: Remove Item 521595368 from 1
Consumer 5: Remove Item 294702567 from 2
Consumer 5: Remove Item 336465782 from 3
Consumer 5: Remove Item 1726956429 from 4
[1] + Done                       "/usr/bin/gdb" --interpreter
vanitas@vinay:~/Documents/Code/OS/Thread$
```

Order of execution

We will be using semaphores to solve the problem of order of execution.

CODE

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
void* thread(void* arg)
{

   sem_wait(&mutex);
   printf("\nEntered thread\n");

   sleep(4);



   printf("\n Exit thread\n");
   sem_post(&mutex);
}
void* thread1(void* arg)
{
```

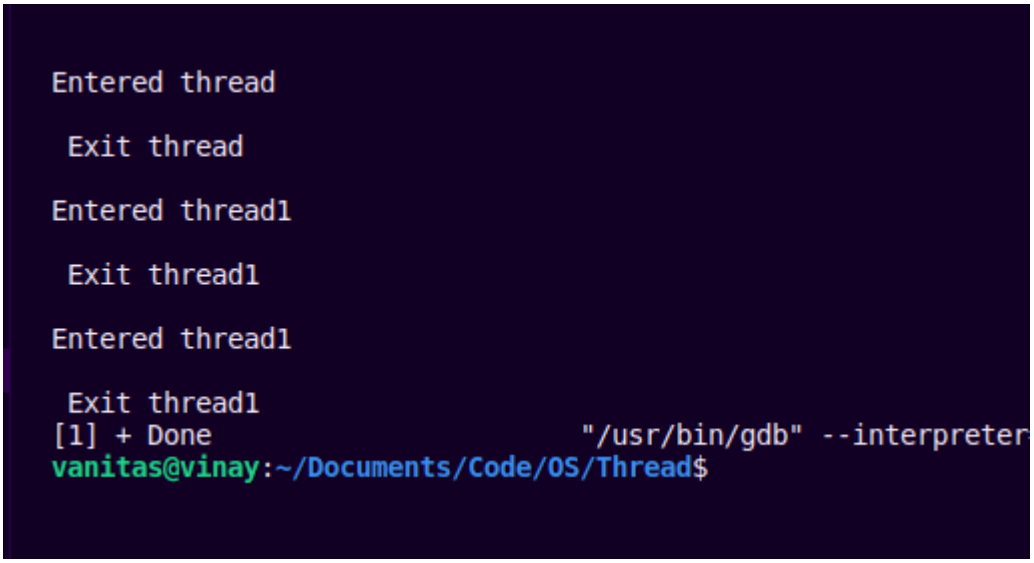
```
    sem_wait(&mutex);
    printf("\nEntered thread1\n");

    sleep(4);



    printf("\n Exit thread1\n");
    sem_post(&mutex);
}
int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread1,NULL);
    pthread_create(&t2,NULL,thread1,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```

RESULT:


Entered thread
Exit thread
Entered thread1
Exit thread1
Entered thread1
Exit thread1
[1] + Done "/usr/bin/gdb" --interpreter
vanitas@vinay:~/Documents/Code/OS/Thread$