CSB 302: Operating System

Lab4: Process Synchronization

Submitted By:

Name: **UMANG KUMAR**

Roll No: **201210051**

Branch: **CSE**

Semester: **5th Sem**

Submitted To: ***Dr. Rishav Singh***

Release Date: 12/09/2022 **Submitted Date:** 20/09/2022

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

Department of Computer Science and Engineering

2022

Q1. Write a program in C to implement Peterson's solution for process synchronization.

CODE:

```
#include <stdbool.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#define _BSD_SOURCE
#include <stdio.h>
#include <sys/time.h>
#include <sys/wait.h>

#define BSIZE 8  // Buffer size
#define PWT 2    // Producer wait time limit
#define CWT 10   // Consumer wait time limit
#define RT 10    // Program run-time in seconds

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)  // Returns a random number between 1 and n
{
   time_t t;

   srand((unsigned)time(&t));

   return (rand() % n + 1);
}

int main() {
   shmid1 = shmget(k1, sizeof(bool) * 2, IPC_CREAT | 0660);  // flag

   shmid2 = shmget(k2, sizeof(int) * 1, IPC_CREAT | 0660);  // turn
```

```
shmid3 = shmget(k3, sizeof(int) * BSIZE, IPC_CREAT | 0660);  // buffer

shmid4 = shmget(k4, sizeof(int) * 1, IPC_CREAT | 0660);  // time stamp

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)  // Initializing buffer

    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;  // Consumer

int j = 1;  // Producer

if (fork() == 0)  // Producer code

{
    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)

        ;

    // Critical Section Begin

    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");

        // Critical Section End

        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)  // Consumer code

{
    SHM1 = (bool*)shmat(shmid1, NULL, 0);

    SHM2 = (int*)shmat(shmid2, NULL, 0);

    SHM3 = (int*)shmat(shmid3, NULL, 0);
```

```c
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)  // Shifting remaining jobs forward

        {
            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);
}

// Parent process will now for RT seconds before causing child to terminate

while (1) {
    gettimeofday(&t, NULL);

    t2 = t.tv_sec;

    if (t2 - t1 > RT)  // Program will exit after RT seconds

    {
        *state = 0;

        break;
        }
    }

    // Waiting for both processes to exit

    wait();

    wait();

    printf("The clock ran out.\n");

    return 0;
}
```

Output:

```
$ gcc test.c
$ ./a.out
Producer is ready now.

Job 1 has been produced
Buffer: 1 0 0 0 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 5 has been produced
Buffer: 1 5 0 0 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 17 has been produced
Buffer: 1 5 17 0 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 8 has been produced
Buffer: 1 5 17 8 0 0 0 0
Producer will wait for 2 seconds

Consumer is ready now.
```

```
Job 1 has been consumed
Buffer: 5 17 8 0 0 0 0 0
Consumer will sleep for 9 seconds

Producer is ready now.

Job 11 has been produced
Buffer: 5 17 8 11 0 0 0 0
Producer will wait for 1 seconds

Producer is ready now.

Job 12 has been produced
Buffer: 5 17 8 11 12 0 0 0
Producer will wait for 2 seconds

Producer is ready now.

Job 14 has been produced
Buffer: 5 17 8 11 12 14 0 0
Producer will wait for 2 seconds

Producer is ready now.

Job 4 has been produced
Buffer: 5 17 8 11 12 14 4 0
Producer will wait for 2 seconds

The clock ran out.
```

Q2. Write a program in C to implement Producer-Consumer problem.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;

void producer() {
   --mutex;
   ++full;
   --empty;
   x++;
   printf(
       "\nProducer produces"
       "item %d",
       x);
   ++mutex;
}

void consumer() {
   --mutex;
   --full;
   ++empty;
   printf(
       "\nConsumer consumes "
       "item %d",
       x);
   x--;
   ++mutex;
}

int main() {
   int n, i;
   printf(
       "\n1. Press 1 for Producer"
       "\n2. Press 2 for Consumer"
       "\n3. Press 3 for Exit");
#pragma omp critical
   for (i = 1; i > 0; i++) {
       printf("\nEnter your choice:");
       scanf("%d", &n);
```

```
        switch (n) {
            case 1:
                if ((mutex == 1) && (empty != 0)) {
                    producer();
                } else {
                    printf("Buffer is full!");
                }
                break;
            case 2:
                if ((mutex == 1) && (full != 0)) {
                    consumer();
                } else {
                    printf("Buffer is empty!");
                }
                break;
            case 3:
                exit(0);
                break;
        }
    }
}
```

Output:

```
1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:1

Producer producesitem 2
Enter your choice:2
```

```
Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:^C
```