RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science & Engineering

LAB REPORT

Topic: Thread Creation

Course No: CSE 3202

Course Name: Sessional Based on Operating Systems

Submitted By

Saifur Rahman
Roll No: 1703018
Section: A
CSE, RUET

Submitted To

Mohiuddin Ahmed
Lecturer
CSE, RUET

Date of Lab: May 30, 2022

Date of Submission: June 13, 2022

Contents

Theory

Thread is an instruction stream. It is basically lightweight process (sub-process) which is the smallest unit of processing that can be performed in an Operating System. Thread exists within a process. A single process can have multiple threads.

Thread allows processes to run sub-processes concurrently. This is called multithreading. A single program with multithreading capabilities allows individual threads to run seemingly at the same time. For example, downloading a video while playing games. Also in a browser, multiple tabs can be different threads.

POSIX Threads (or Pthreads) is a POSIX standard for threads. GCC compiler supports pthread.

Function Description

In Unix operating system, C/C++ languages provide the POSIX thread standard API for all thread related functions. The basic pthread library functions:

pthread_create:

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- thread: pointer to an unsigned integer value that returns the thread id of the thread created.
- attr: pointer to a structure that is used to define thread attributes. Set to NULL for default thread attributes.
- start_routine: pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *.
- arg: pointer to void that contains the arguments to the function defined in the earlier argument

pthread_join:

```
int pthread_join(pthread_t th,
                 void **thread_return);
```

- th: thread id of the thread.
- thread_return: pointer to the location where the exit status of the thread mentioned in th is stored.

pthread_exit:

```
void pthread_exit(void *retval);
```

- retval: pointer to an integer that stores the return status of the thread terminated.

Program

Implementing Producer Consumer (Bound buffer) problem using pthread.

CODE

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

int n = 50;
int a[70];
void *producer(void *ptr);
void *consumer(void *ptr);

int main()
{
        int a[n];
        for(int i = 0; i < n; i++) {
                a[i] = 0;
        }
        pthread_t ptid1, ptid2;
        pthread_create(&ptid1, NULL, producer, NULL);
        pthread_create(&ptid2, NULL, consumer, NULL);
        pthread_join(ptid1, NULL);
        pthread_join(ptid1, NULL);

        for(int i = 0; i < n; i++) {
                printf("%d ", a[i]);
        }
        printf("\n");

        printf("Thread ended"
            " \n");
        pthread_exit(NULL);

        return 0;
}

void *producer(void *ptr)
{

        printf("Inside producer thread\n");

        int count = 0;
        for(int i = 0; i < n; i++) {
                a[i] = rand() % 100;
                count++;
        }
        for(int i = 0; i < n; i++) {
                printf("p%d ", a[i]);
        }
        printf("\n");
        if(count == n) {
                printf("Producing completed\n");
                sleep(1);
                //wakeup(consumer);

        }
}
```

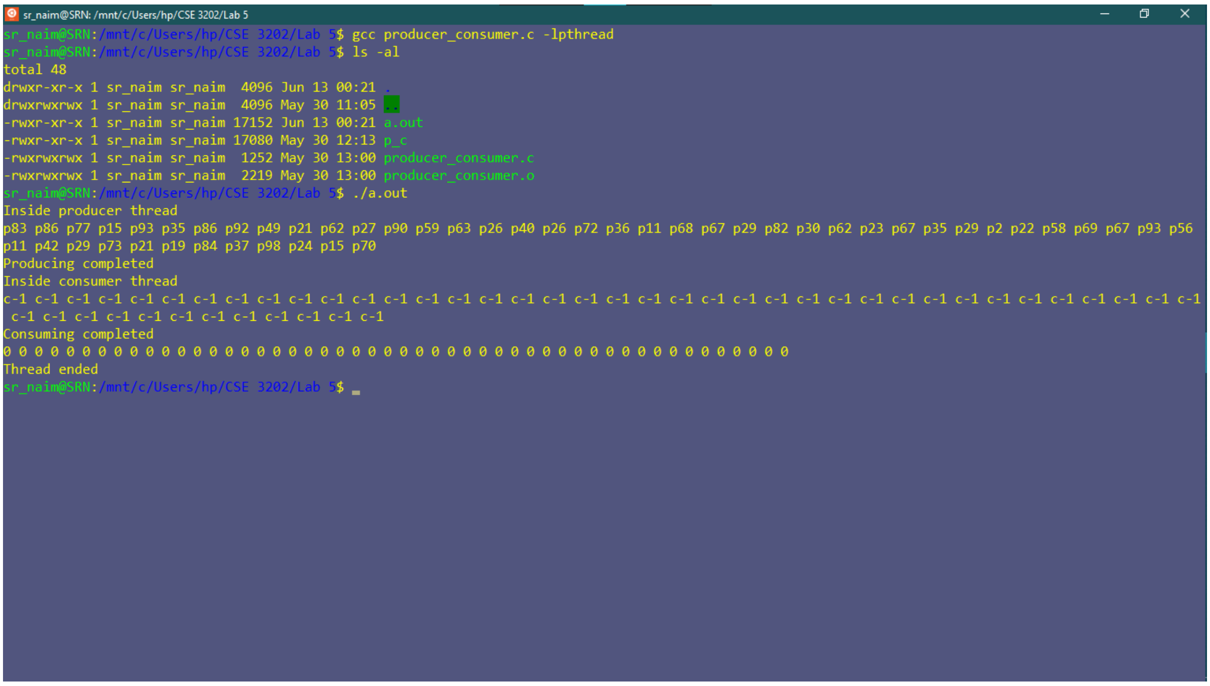
```
void *consumer(void *ptr)
{

        printf("Inside consumer thread\n");

        int count = n;
        for(int i = 0; i < n; i++) {
                a[i] = -1;
                count--;
        }
        for(int i = 0; i < n; i++) {
                printf("c%d ", a[i]);
        }
        printf("\n");
        if(count == 0) {
                printf("Consuming completed\n");
                sleep(1);
                //wakeup(producer());

        }
}
```

OUTPUT



Discussion

Here two threads are created for producer and consumer. The order of the lines of output are interchanged depending upon which thread is processed earlier. Here two threads are running concurrently. So, processor may switch effecting the output to be changed. That's why *pthread_join()* is used to wait for the termination of a thread. *pthread_exit()* is used to terminate a thread but here it has no use because thread had already ended.