

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (FAST-NUCES)**

**OPERATING SYSTEM PROJECT REPORT (2021)**

**Group members:** Ashmal Anis (19k-0305)

Hasnain Somani (19k-0204)

**Project name:** System Call for Producer Consumer Problem

**Objective**

The objective behind this project was to make a practical implementation of the producer consumer problem using bounded buffers, in C language, using the Linux operating system (Ubuntu version). The project was implemented, and results were printed on console screen through a system call.

**Project Description**

Processes communicate through various sources of communication- a buffer with a limited number of data slots is one of the majorly used source of communication. In the project, producer writes to one end of the buffer, and the consumer reads from the other end. Using semaphores, various test cases have been taken care of- which include prohibiting writing to a full buffer, or reading from an empty buffer. The count is kept through wait() and signal() function calls. Data will be written by a producer, and a specific consumer will be reading the data off. The data written, read, as well as the producer and consumer numbers have been displayed in the program to ensure correct input and output of data.

This output is displayed through the printk() function in the system call made inside the program file. This system call has been made by making amendments to makefile, creating a new C file for the system call. The system call is invoked by creating a new directory named “ProducerConsumer”, and then creating a file inside the directory to run the c program file, which then calls the system call in it.

**Code results**

**Code:**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/syscall.h>

#include <linux/kernel.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;

struct data{

int \_id;

int input;

};

void \*producer(void \*pno)

{

int item;

for(int i = 0; i < MaxItems; i++) {

struct data d;

d = \*((struct data\*)pno);

item = d.input;

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

syscall(333,"Producer",d.\_id, "writes", 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];

struct data d;

d = \*((struct data\*)cno);

syscall(333,"Consumer",d.\_id, "reads", 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);

struct data d[5];

int i;

printf("Enter 5 data : ");

for(i=0; i<5; i++){

scanf("%d",&d[i].input);

d[i].\_id= i+1;

}

for(int i = 0; i < 5; i++) {

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&d[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&d[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);

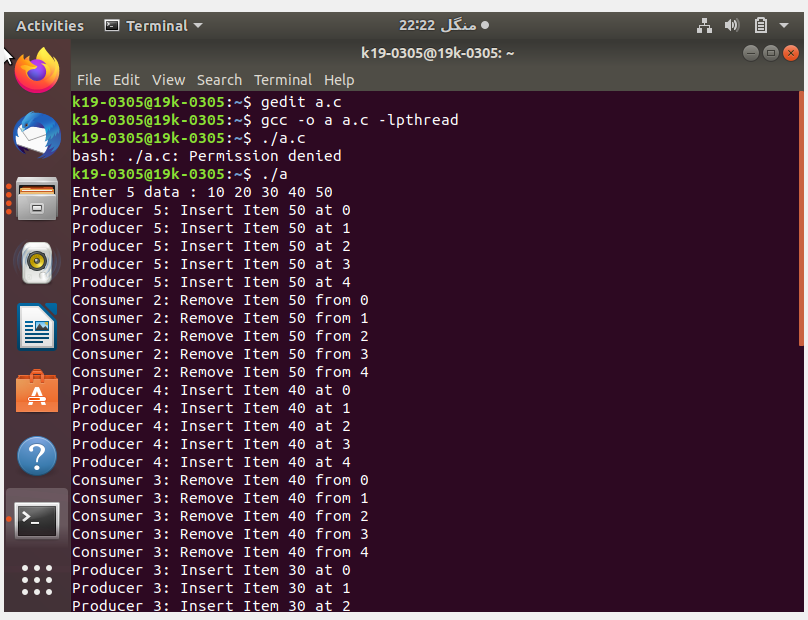
sem\_destroy(&empty);

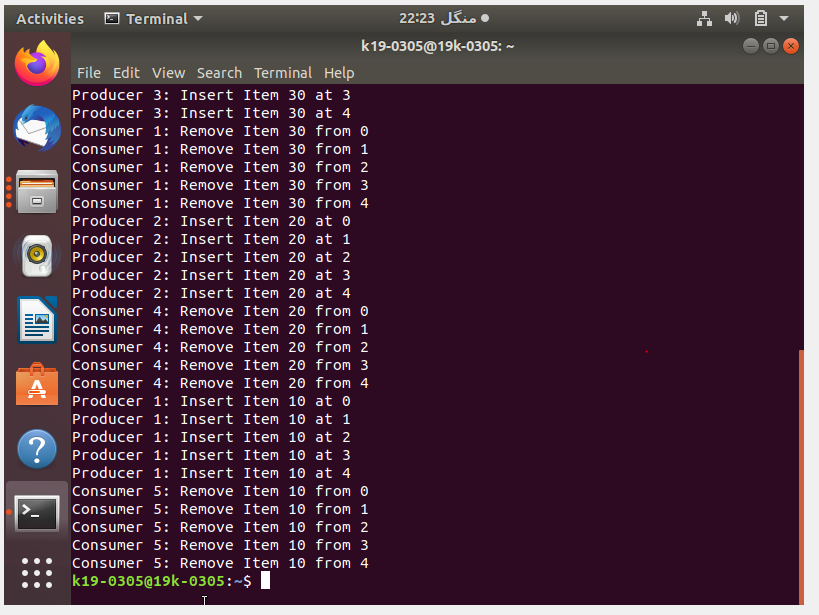
sem\_destroy(&full);

return 0;

}

**Results:**





**Conclusion**

Bounded buffers are one of the most efficient approaches to read and write data through processes. This is because it limits the data to be stored, and keeps a count of the empty and filled slots in the buffer using a counting semaphore. This easily accommodates a solution to the producer consumer problem, and prevents data inefficiency.