**Experiment 4**

**Title:** Process Management: Synchronization

Write a C program to implement solution of Producer consumer problem through Semaphore

**Estimated time to complete this experiment:** 2 hours

**Objective:** Learning about Process synchronization using Mutex. Implementing a program in Cfor solving Producer Consumer problem.

**Expected Outcome of Experiment:** To perform Enqueue Dequeue operations by Producer and Consumer respectively on a shared queue and ensure that the data as received by Consumer is consistent with the data as inserted by Producer.

**Books/ Journals/ Websites referred:**

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0

**Pre Lab/ Prior Concepts:** C Programming platform, Queue operations, File handling in C.

**Brief description:**

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.

Any solution to the critical section problem must satisfy three requirements:

* **Mutual Exclusion:** If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
* **Progress:** If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
* **Bounded Waiting:** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**New Concepts to be learned:** Process Synchronization using semaphores

**Requirements:** PC with C programming platform.

**Theory:**

Producer consumer problem is a classical synchronization problem. We can solve this problem by using semaphores.

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.

wait(S){

while(S<=0); // busy waiting

S--;

}

signal(S){

S++;

}

Semaphores are of two types:

**Binary Semaphore** – This is similar to mutex lock but not the same thing. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.

**Counting Semaphore** – Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

**Problem Statement** – We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define Queue\_size 10

int queue[Queue\_size]; *// Shared queue*

int in = 0, out = 0; *// queue pointers*

sem\_t empty, full, mutex; *// Semaphores for synchronization*

*// Producer function*

void \*producer(void \*arg) {

    FILE \*fp = fopen("Product.txt", "r"); *// Open the input file*

    int val;

    while (fscanf(fp, "%d", &val) == 1) { *// Read from the file*

        sem\_wait(&empty); *// Wait until the queue is not full*

        sem\_wait(&mutex); *// Wait to acquire the queue lock*

        queue[in] = val; *// Insert the new item into the queue*

        in = (in + 1) % Queue\_size; *// Increment the queue pointer*

        printf("Produced: %d\n", val); *// Print the produced value*

        sem\_post(&mutex); *// Release the queue lock*

        sem\_post(&full); *// Signal that the queue is not empty*

    }

    fclose(fp); *// Close the input file*

    pthread\_exit(NULL); *// Exit the thread*

}

*// Consumer function*

void \*consumer(void \*arg) {

    int val;

    FILE \*fp = fopen("Consumed\_product.txt", "w"); *// Open the output file*

    while (1) { *// Run forever*

        sem\_wait(&full); *// Wait until the queue is not empty*

        sem\_wait(&mutex); *// Wait to acquire the queue lock*

        val = queue[out]; *// Retrieve an item from the queue*

        out = (out + 1) % Queue\_size; *// Increment the queue pointer*

        fprintf(fp, "%d\n", val); *// Write the consumed value to the output file*

        printf("Consumed: %d\n", val);*//consumed*

        sem\_post(&mutex); *// Release the queue lock*

        sem\_post(&empty); *// Signal that the queue is not full*

    }

    fclose(fp); *// Close the output file*

    pthread\_exit(NULL); *// Exit the thread*

}

int main() {

    sem\_init(&empty, 0, Queue\_size); *// Initialize the empty semaphore to Queue\_size*

    sem\_init(&full, 0, 0); *// Initialize the full semaphore to 0*

    sem\_init(&mutex, 0, 1); *// Initialize the mutex semaphore to 1*

    pthread\_t producer\_thread, consumer\_thread;

    pthread\_create(&producer\_thread, NULL, producer, NULL); *// Create the producer thread*

    pthread\_create(&consumer\_thread, NULL, consumer, NULL); *// Create the consumer thread*

    pthread\_join(producer\_thread, NULL); *// Wait for the producer thread to finish*

    pthread\_join(consumer\_thread, NULL); *// Wait for the consumer thread to finish*

    return 0;

}

**Output:**

**Text

Description automatically generated**

**Conclusion:** Hence we have understoodthe method to achieve consistency through Process synchronization and applied the same on Producer-Consumer Problem. The solution involves using binary and counting semaphore.

**Real Life Application:** Incorporating consistency policies in multiprocessing / multithreading applications such as an application controlling a shared network printer that receives files for printing from multiple users.