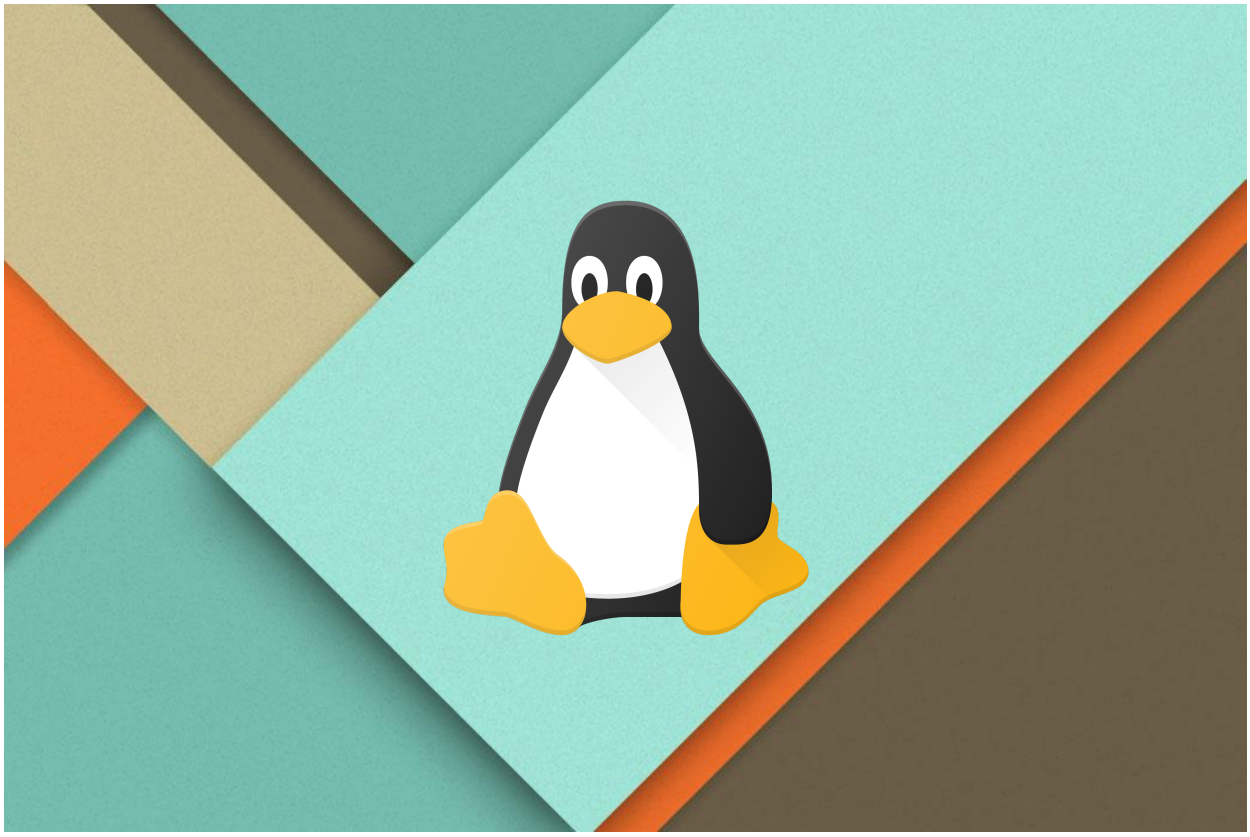


Operating Systems Lab

Experiment No. 6

22.02.2022

Professor - Dr. Shrinivas Khedkar.

Pratham Loya
201080068
IT
prloya_b20@it.vjti.ac.in

Aim

Implement classical synchronization problems using Semaphores in C programming. Write the program to solve the reader writer problem.

Theory

Suppose that a database is to be shared among several concurrent processes.

Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database.

We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Obviously, if two readers access the shared data simultaneously, no adverse effects will result.

However, if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.

To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database.

This synchronization problem is referred to as the readers-writers problem. The readers-writers problem has several variations, all involving priorities.

The simplest one, referred to as the first readers-writers problem, requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object.

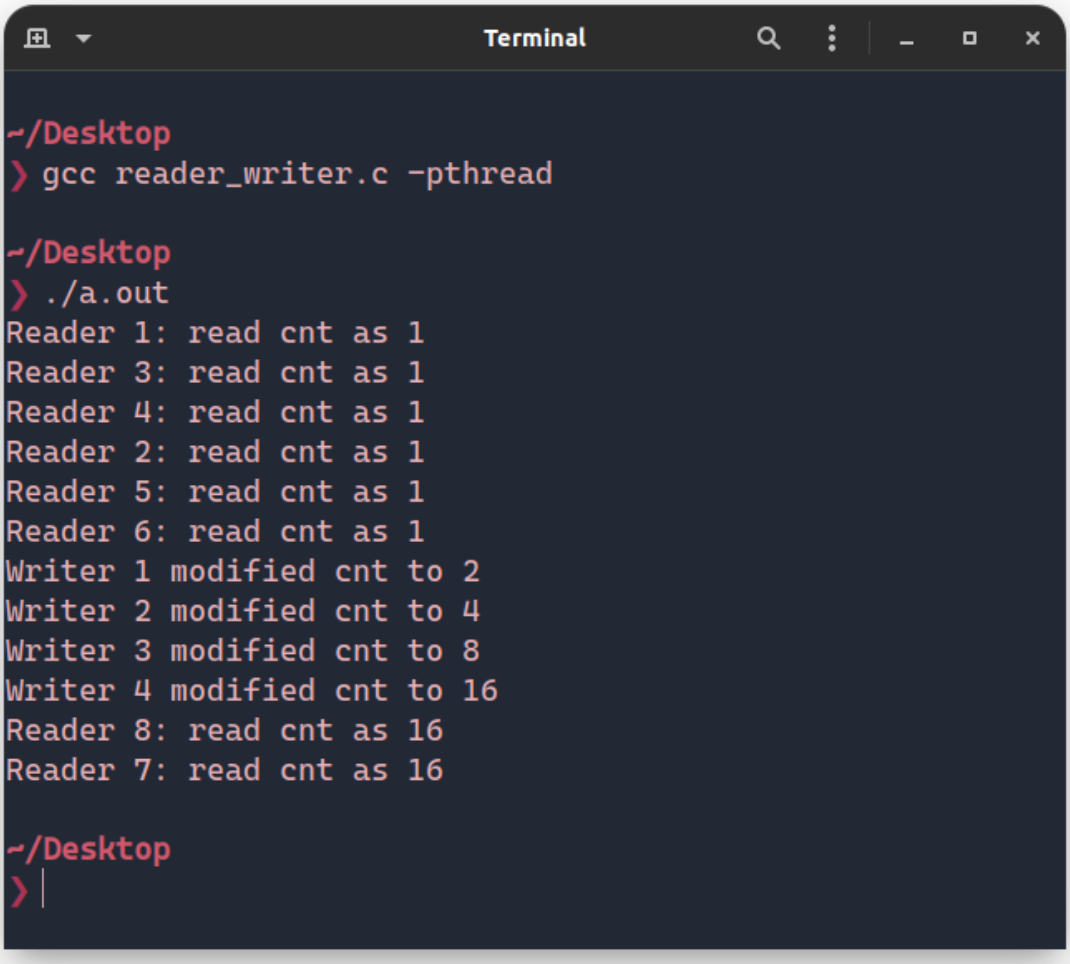
In other words, no reader should wait for other readers to finish simply because a writer is waiting.

Program

reader_writer.c

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

sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
    sem_wait(&wrt);
    cnt = cnt*2;
    printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
    sem_post(&wrt);

}
void *reader(void *rno)
{
    // Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader++;
    if(numreader == 1) {
        sem_wait(&wrt); // If this id the first reader, then it will block the writer
    }
    pthread_mutex_unlock(&mutex);
    // Reading Section
    printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);

    // Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader--;
    if(numreader == 0) {
        sem_post(&wrt); // If this is the last reader, it will wake up the writer.
    }
    pthread_mutex_unlock(&mutex);
}

int main()
{

    pthread_t read[8],write[4];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);

    int a[8] = {1,2,3,4,5,6,7,8}; //Just used for numbering the producer and consumer

    for(int i = 0; i < 8; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for(int i = 0; i < 4; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }

    for(int i = 0; i < 8; i++) {
        pthread_join(read[i], NULL);
    }
    for(int i = 0; i < 4; i++) {
        pthread_join(write[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;
}
```

Output


Terminal
~/Desktop
❯ gcc reader_writer.c -pthread
~/Desktop
❯ ./a.out
Reader 1: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 2: read cnt as 1
Reader 5: read cnt as 1
Reader 6: read cnt as 1
Writer 1 modified cnt to 2
Writer 2 modified cnt to 4
Writer 3 modified cnt to 8
Writer 4 modified cnt to 16
Reader 8: read cnt as 16
Reader 7: read cnt as 16
~/Desktop
❯