VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SWE3001 – Operating Systems Laboratory Manual

Lab - 05

Student Name	Sam Prince Franklin
Reg Number	20MIS1115
Subject Code	SWE3001
Slot	L37,38
Faculty	Dr.M.Braveen

SWE3001 – Operating Systems

Lab – 05– Synchronization

---

1. In a Company, goods are manufactured and stored in a sharable warehouse. Goods are distributed to the distributor by the company. Goods can't be manufactured when warehouse is full, and at the same time, Goods can't be distributed when warehouse is empty. Implement the above scenario using appropriate operating system concept. Write a C-Program to provide a Solution to the above problem.

```
#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;
    }
  }
}
```

```
samprincefranklin@Sams-MacBook-Air Lab % gcc pc.c -o pc
samprincefranklin@Sams-MacBook-Air Lab % ./pc

  1. Press 1 for Producer
  2. Press 2 for Consumer
  3. Press 3 for Exit
  Enter your choice:1

  Producer producesitem 1
  Enter your choice:1

  Producer producesitem 2
  Enter your choice:1

  Producer producesitem 3
  Enter your choice:2

  Consumer consumes item 3
  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:
```

2. We have a database which many users are allowed to read simultaneously, but to which only one user may write at a time (to avoid race conditions in the database). So it is identical to the mutual exclusion problem, except that we wish to permit more flexibility in the system by letting more than one simultaneous reader. Naturally there is no danger of a race condition when many tasks are *reading* from a shared data structure.
Write a C-Program to provide a Solution to the above problem.

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

/*
This program provides a possible solution for first readers writers problem using mutex and semaphore.
I have used 10 readers and 5 producers to demonstrate the solution. You can always play with these values.
*/

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[10], write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt, 0, 1);

    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // Just used for numbering the producer and consumer

    for (int i = 0; i < 10; i++)
    {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for (int i = 0; i < 5; i++)
```

```
    {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }

    for (int i = 0; i < 10; i++)
    {
        pthread_join(read[i], NULL);
    }
    for (int i = 0; i < 5; i++)
    {
        pthread_join(write[i], NULL);
    }

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

    return 0;
}
```

Output :

```
samprincefranklin@Sams-MacBook-Air Lab % ./rw
Reader 1: read cnt as 1
Reader 2: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 5: read cnt as 1
Reader 7: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Reader 10: 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
Writer 5 modified cnt to 32
Reader 6: read cnt as 1
samprincefranklin@Sams-MacBook-Air Lab %
```