K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: B2	**Roll No.: 16010121194**
Experiment No. 05	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Process synchronization algorithms using semaphore - producer consumer problem , reader-writers problem

---

AIM: Implementation of Process synchronization algorithms using semaphore - producer consumer problem, reader-writers problem

---

Expected Outcome of Experiment:

CO 3. To understand the concepts of process synchronization and deadlock.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Wiley Eighth edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Knowledge of Concurrency, Mutual Exclusion, Synchronization, Deadlock, Starvation.

Description of the chosen process synchronization algorithm:

- **Initialization**:

Initialize two semaphores, readMutex and writeMutex, and a counter, readCount. readMutex and writeMutex are binary semaphores (with values 0 or 1), while readCount is an integer initialized to 0.

- **Reader's Access:**

Before a reader accesses the shared resource, it must first acquire the readMutex semaphore. This semaphore ensures that only one reader can acquire the read access at a time, preventing a writer from entering the critical section. After acquiring the readMutex, the readCount is incremented, indicating that a reader is currently reading the resource. If it's the first reader, it also acquires the writeMutex semaphore to block any writers.

Once the reader has finished reading, it decrements readCount. If it was the last reader, it releases the writeMutex, allowing writers to access the resource.

- **Writer's Access:**

Before a writer accesses the shared resource, it must acquire the writeMutex semaphore. This ensures that only one writer can access the critical section at a time, and no readers can enter while a writer is active. Once the writer has acquired the writeMutex, it can perform its write operation on the shared resource.

After finishing writing, the writer releases the writeMutex, allowing other writers or readers to access the resource.

Implementation details: (printout of code)

```
#include<pthread.h>

#include<stdio.h>

#include<unistd.h>

#include<string.h>
```

```c
#include<semaphore.h>

#include<stdlib.h>

#define THREAD_NUM 10

static int rc = 0;

sem_t mutex;

sem_t database;

static int data = 0;

void * reader (void * args){

sem_wait(&mutex);

rc = rc + 1;

if(rc == 1)

{

sem_wait(&database);

}

sem_post(&mutex);

printf("Reading Value of shared data %d and value of mutex %d \n",data,mutex);

sleep(3);

sem_wait(&mutex);
```

```
rc = rc - 1;

if(rc == 0) {sem_post(&database);}

sem_post(&mutex);

free(args);

}

void * writer (void * args){

sem_wait(&database);

data = *(int*)args;

printf("Writing value of shared data %d and value of
database %d\n",*(int*)args,database);

sleep(1);

sem_post(&database);

free(args);

}

int main()

{

pthread_t th[THREAD_NUM];

sem_init(&mutex, 0, 1);
```

```
sem_init(&database, 0, 1);

int i;

for (i = THREAD_NUM/2; i < THREAD_NUM; i++) {

int* a =(int*) malloc(sizeof(int));

*a = i;

if (pthread_create(&th[i], NULL, &writer, a) != 0) {

perror("Failed to create thread");

}

}

for (i = 0; i < (THREAD_NUM/2); i++) {

int* a = (int*)malloc(sizeof(int));

*a = i;

if (pthread_create(&th[i], NULL, &reader, a) != 0) {

perror("Failed to create thread");

}

}

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

if (pthread_join(th[i], NULL) != 0) {
```

```
perror("Failed to join thread");


}


}


sem_destroy(&mutex);


return 0;


}
```

Output:

```
Writing value of shared data 5 and value of database 0
Writing value of shared data 9 and value of database 0
Writing value of shared data 6 and value of database 0
Writing value of shared data 7 and value of database 0
Reading Value of shared data 9 and value of mutex 0
Reading Value of shared data 9 and value of mutex 0
Reading Value of shared data 9 and value of mutex 0
Writing value of shared data 8 and value of database 0
Reading Value of shared data 9 and value of mutex 0
Reading Value of shared data 9 and value of mutex 0
```

Conclusion:

Successfully implemented Readers Writers Problem using binary semaphores.

Post Lab Objective Questions

1) A semaphore is a shared integer variable
a) That can't drop below zero
b) That can't be more than
c) That can't drop below one

Ans: a

2) Mutual exclusion can be provided by the

a) Mute locks
b) Binary semaphores
c) Both a and b
d) None of these

Ans: c

3) A monitor is a module that encapsulates
a) Shared data structures
b) Procedures that operate on shared data structure
c) Synchronization between concurrent procedure invocation
d) All of the above

Ans: d

4) To enable a process to wait within the monitor
a) A condition variable must be declared as condition
b) Condition Variables must be used as Boolean objects
c) Semaphore must be used
d) All of the above

Ans: a

Date: 10/10/2023 **Signature of faculty in-charge**