Experiment Number: 6

Problem Statement:

**a) Implementation of Classical problems Reader Writer  using Threads and Mutex.**

**b)  Implementation of Classical problems Reader Writer  using Threads and Samaphore.**

NAME: **Manoj Dhanraj Mule** ROLL NO: 71

CLASS: IT-B BATCH: B3

DATE OF PERFORMANCE: 12-09-2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Implementation of Classical problems Reader Writer using Threads and Mutex.**

#include <stdio.h>

#include <pthreadgcc.h>

#include <unistd.h>

pthread\_mutex\_t resource;

pthread\_mutex\_t r\_mutex;

int reader\_count = 0;

void\* reader(void\* arg) {

int reader\_id = \*((int\*)arg);

// Reader enters

pthread\_mutex\_lock(&r\_mutex);

reader\_count++;

if (reader\_count == 1) {

pthread\_mutex\_lock(&resource); // First reader locks the resource

}

pthread\_mutex\_unlock(&r\_mutex);

// Reading section

printf("Reader %d is reading the resource.\n", reader\_id);

sleep(1); // Simulate reading

// Reader exits

pthread\_mutex\_lock(&r\_mutex);

reader\_count--;

if (reader\_count == 0) {

pthread\_mutex\_unlock(&resource); // Last reader unlocks the resource

}

pthread\_mutex\_unlock(&r\_mutex);

return NULL;

}

void\* writer(void\* arg) {

int writer\_id = \*((int\*)arg);

// Writing section

pthread\_mutex\_lock(&resource); // Only one writer can write at a time

printf("Writer %d is writing to the resource.\n", writer\_id);

sleep(1); // Simulate writing

pthread\_mutex\_unlock(&resource);

return NULL;

}

int main() {

pthread\_t r\_threads[5], w\_threads[3];

pthread\_mutex\_init(&resource, NULL);

pthread\_mutex\_init(&r\_mutex, NULL);

int reader\_ids[5] = {1, 2, 3, 4, 5};

int writer\_ids[3] = {1, 2, 3};

// Create reader and writer threads

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

pthread\_create(&r\_threads[i], NULL, reader, &reader\_ids[i]);

}

for (int i = 0; i < 3; i++) {

pthread\_create(&w\_threads[i], NULL, writer, &writer\_ids[i]);

}

// Wait for threads to complete

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

pthread\_join(r\_threads[i], NULL);

}

for (int i = 0; i < 3; i++) {

pthread\_join(w\_threads[i], NULL);

}

pthread\_mutex\_destroy(&resource);

pthread\_mutex\_destroy(&r\_mutex);

return 0;

}

**Output:**

manoj@manoj-VirtualBox:~/shellscriptprogram$ gcc lab7.c -o lab7 -pthread

manoj@manoj-VirtualBox:~/shellscriptprogram$ ./lab7

Reader 3 is reading the resource.

Reader 5 is reading the resource.

Reader 2 is reading the resource.

Reader 1 is reading the resource.

Reader 4 is reading the resource.

Writer 1 is writing to the resource.

Writer 3 is writing to the resource.

Writer 2 is writing to the resource.

manoj@manoj-VirtualBox:~/shellscriptprogram$

**b)  Implementation of Classical problems Reader Writer  using Threads and Samaphore.**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t resource;

sem\_t r\_mutex;

int reader\_count = 0;

void\* reader(void\* arg) {

int reader\_id = \*((int\*)arg);

// Reader enters

sem\_wait(&r\_mutex);

reader\_count++;

if (reader\_count == 1) {

sem\_wait(&resource); // First reader locks the resource

}

sem\_post(&r\_mutex);

// Reading section

printf("Reader %d is reading the resource.\n", reader\_id);

sleep(1); // Simulate reading

// Reader exits

sem\_wait(&r\_mutex);

reader\_count--;

if (reader\_count == 0) {

sem\_post(&resource); // Last reader unlocks the resource

}

sem\_post(&r\_mutex);

return NULL;

}

void\* writer(void\* arg) {

int writer\_id = \*((int\*)arg);

// Writing section

sem\_wait(&resource); // Writer locks the resource

printf("Writer %d is writing to the resource.\n", writer\_id);

sleep(1); // Simulate writing

sem\_post(&resource);

return NULL;

}

int main() {

pthread\_t r\_threads[5], w\_threads[3];

sem\_init(&resource, 0, 1);

sem\_init(&r\_mutex, 0, 1);

int reader\_ids[5] = {1, 2, 3, 4, 5};

int writer\_ids[3] = {1, 2, 3};

// Create reader and writer threads

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

pthread\_create(&r\_threads[i], NULL, reader, &reader\_ids[i]);

}

for (int i = 0; i < 3; i++) {

pthread\_create(&w\_threads[i], NULL, writer, &writer\_ids[i]);

}

// Wait for threads to complete

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

pthread\_join(r\_threads[i], NULL);

}

for (int i = 0; i < 3; i++) {

pthread\_join(w\_threads[i], NULL);

}

sem\_destroy(&resource);

sem\_destroy(&r\_mutex);

return 0;

}

**Output:**

manoj@manoj-VirtualBox:~/shellscriptprogram$ gcc lab7b.c -o lab7b -pthread

manoj@manoj-VirtualBox:~/shellscriptprogram$ ./lab7b

Reader 2 is reading the resource.

Reader 3 is reading the resource.

Reader 1 is reading the resource.

Reader 4 is reading the resource.

Reader 5 is reading the resource.

Writer 1 is writing to the resource.

Writer 2 is writing to the resource.

Writer 3 is writing to the resource.

manoj@manoj-VirtualBox:~/shellscriptprogram$