Department of Computer Engineering

Batch E-2 **Roll No.: 16010123325**
Experiment No. 07
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

TITLE: Readers-Writers problem using semaphore/monitors.

AIM: Implementation of Process synchronization algorithm using semaphore to solve 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", Willey Eight 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:

The readers-writer problem is about managing access to shared data. It allows multiple readers to read data at the same time without issues but ensures that only one writer can write at a time, and no reader can read while the writer is writing.

This helps prevent data corruption and ensures smooth concurrent operation of multiple processes.

The readers/writers problem is stated as follows:

There is a data area shared among a number of processes.

The data area could be a file, a block of main memory,or even a bank of processor registers.

There are a number of processes that only read the data area (readers) and a number that only write to the data area (writers).

The conditions that must be satisfied are as follows:

1. Any number of readers may simultaneously read the file.

2. Only one writer at a time may write to the file.

3. If a writer is writing to the file, no reader may read it.

The reader-writer problem can be thought of for two different scenarios; priority to reader or priority to writer. The solution to the problem can be proposed using semaphore or monitors.

Implementation details:

DATA STRUCTURES Used/ Code

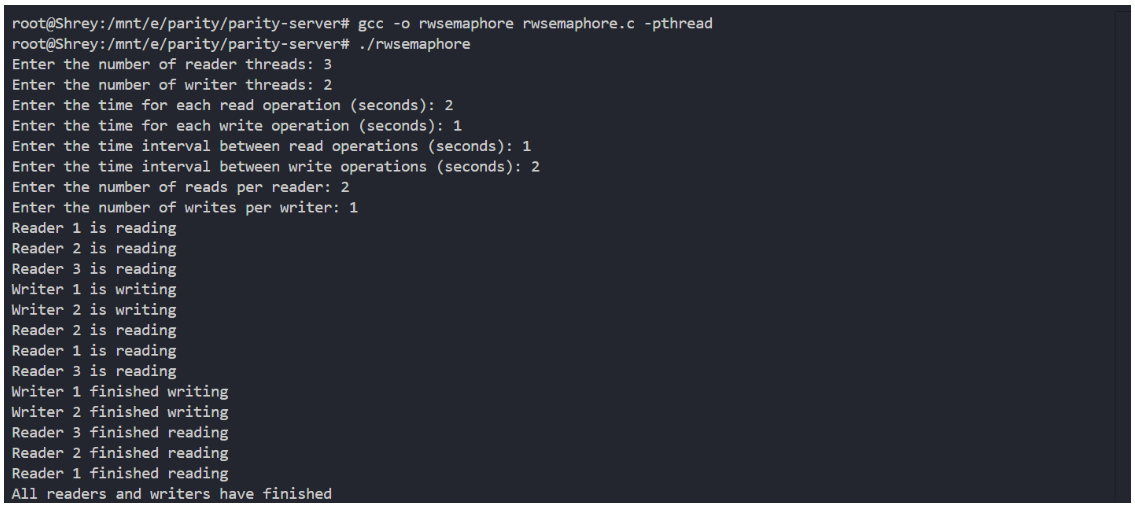
Data Structures-

- **thread_data struct** – Stores thread-specific data like ID, time, intervals, and operation counts.
- **sem_t mutex** – Semaphore to protect access to read_count (used for reader synchronization).
- **sem_t wrt** – Semaphore to provide exclusive access to writers.
- **pthread_t r_threads[]** – Array to store thread IDs for all reader threads.
- **pthread_t w_threads[]** – Array to store thread IDs for all writer threads.
- **thread_data r_data[]** – Array to store data for each reader thread.
- **thread_data w_data[]** – Array to store data for each writer thread.

Code-

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

typedef struct {
    int id;
    int rt;
    int wt;
    int ri;
    int wi;
    int nr;
    int nw;
} td;

sem_t mtx;
sem_t wrt;
int rc = 0;
int fr = 0;
int fw = 0;

void *reader(void *arg) {
    td *d = (td *)arg;
    for (int i = 0; i < d->nr; i++) {
        sem_wait(&mtx);
        rc++;
        if (rc == 1) {
            sem_wait(&wrt);
        }
        sem_post(&mtx);

        printf("Reader %d is reading\n", d->id);
        sleep(d->rt);

        sem_wait(&mtx);
        rc--;
        if (rc == 0) {
            sem_post(&wrt);
        }
        sem_post(&mtx);

        sleep(d->ri);
    }
    printf("Reader %d finished reading\n", d->id);
    fr++;

    return NULL;
}

void *writer(void *arg) {
    td *d = (td *)arg;
    for (int i = 0; i < d->nw; i++) {
        sem_wait(&wrt);

        printf("Writer %d is writing\n", d->id);
        sleep(d->wt);

        sem_post(&wrt);

        sleep(d->wi);
    }

    printf("Writer %d finished writing\n", d->id);
    fw++;

    return NULL;
}

int main() {
    int nr, nw, rt, wt, ri, wi, npr, npw;

    printf("Enter the number of reader threads: ");
    scanf("%d", &nr);

    printf("Enter the number of writer threads: ");
    scanf("%d", &nw);

    printf("Enter the time for each read operation (seconds): ");
    scanf("%d", &rt);

    printf("Enter the time for each write operation (seconds): ");
    scanf("%d", &wt);

    printf("Enter the time interval between read operations (seconds):
");
    scanf("%d", &ri);
    printf("Enter the time interval between write operations (seconds): 
");
    scanf("%d", &wi);

    printf("Enter the number of reads per reader: ");
    scanf("%d", &npr);

    printf("Enter the number of writes per writer: ");
    scanf("%d", &npw);

    pthread_t rt[nr], wt_arr[nw];
    td rd[nr], wd[nw];

    sem_init(&mtx, 0, 1);
    sem_init(&wrt, 0, 1);

    for (int i = 0; i < nr; i++) {
        rd[i].id = i + 1;
        rd[i].rt = rt;
        rd[i].ri = ri;
        rd[i].nr = npr;
        pthread_create(&rt[i], NULL, reader, &rd[i]);
    }

    for (int i = 0; i < nw; i++) {
        wd[i].id = i + 1;
        wd[i].wt = wt;
        wd[i].wi = wi;
        wd[i].nw = npw;
        pthread_create(&wt_arr[i], NULL, writer, &wd[i]);
    }

    for (int i = 0; i < nr; i++) {
        pthread_join(rt[i], NULL);
    }
    for (int i = 0; i < nw; i++) {
        pthread_join(wt_arr[i], NULL);
    }

    printf("All readers and writers have finished\n");

    sem_destroy(&mtx);
    sem_destroy(&wrt);
    return 0;
}
```

Output-



```
root@Shrey:/mnt/e/parity/parity-server# gcc -o rwsemaphore rwsemaphore.c -pthread
root@Shrey:/mnt/e/parity/parity-server# ./rwsemaphore
Enter the number of reader threads: 3
Enter the number of writer threads: 2
Enter the time for each read operation (seconds): 2
Enter the time for each write operation (seconds): 1
Enter the time interval between read operations (seconds): 1
Enter the time interval between write operations (seconds): 2
Enter the number of reads per reader: 2
Enter the number of writes per writer: 1
Reader 1 is reading
Reader 2 is reading
Reader 3 is reading
Writer 1 is writing
Writer 2 is writing
Reader 2 is reading
Reader 1 is reading
Reader 3 is reading
Writer 1 finished writing
Writer 2 finished writing
Reader 3 finished reading
Reader 2 finished reading
Reader 1 finished reading
All readers and writers have finished
```

Conclusion:

This experiment demonstrated how semaphores and thread synchronization techniques like mutexes ensure safe access to shared resources. By simulating reader and writer processes, we observed how concurrency issues can be managed using binary semaphores to maintain mutual exclusion and prevent race conditions effectively.

Post Lab Objective Questions

1) A semaphore which takes only binary values is called
a) Counting semaphore
b) Mutex
c) Weak semaphore
d) None of the above
Ans: b

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

Post Lab Subjective Questions

1. What is Monitor? Explain how the monitor supports synchronization by the use of condition variables.

A **monitor** is a high-level synchronization construct that allows safe access to shared data by encapsulating the data along with the procedures that operate on it. It ensures that only one process/thread can be active in the monitor at a time, thus providing **mutual exclusion** automatically.

Monitors support **synchronization** using **condition variables**. These are special variables associated with the monitor and allow threads to wait for some condition to be true before continuing execution. There are two main operations on condition variables:

- `wait()` – This operation suspends the execution of the calling thread and places it in the waiting queue associated with the condition.
- `signal()` – This wakes up one of the threads waiting on the condition variable (if any).

By combining mutual exclusion and condition variables, monitors make thread synchronization more structured and easier to manage.

Date: _____________ **Signature of faculty in-charge**