NAME: ADITYA RAJ PANDIT

REG NO: 23BRS1157

PROCESS SYNCHRONIZATION

Q1) Develop a readers and writers problem with minimum 2 readers and 2 writers, ensure that synchronisation is done with semaphore and satisfy the 4 below mentioned conditions. Shared data as an integer variable and let the writers do the increment operations and readers do the shared variable read operation.

Case	Process 1	Process 2	Allowed/Not Allowed
Case 1	Writing	Writing	Not Allowed
Case 2	Writing	Reading	Not Allowed
Case 3	Reading	Writing	Not Allowed
Case 4	Reading	Reading	Allowed

SOLUTION

To solve this problem we used two semaphores – **mutex**, **writeBlock** and two integers **sharedData**, **readCount**.

- **readCount** is used to keep track of the number of readers currently reading.
- **sharedData** variable represents the shared resource.
- A semaphore **mutex** is used to update readCount.
- Another semaphore **writeBlock** ensures that only one writer can access the shared data at any time, preventing data inconsistency.

C ReadersWriters.c > main()

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

sem_t mutex, writeBlock;
int sharedData = 0;
int readCount = 0;

void *reader(void *arg) {
    int readerId = *((int *)arg);

    sem_wait(&mutex);
    readCount++;
    if (readCount == 1) {
        sem_wait(&writeBlock);
    }
    sem_post(&mutex);

    printf("Reader %d reads shared data: %d\n", readerId, sharedData);
    sleep(1);

    sem_wait(&mutex);
    readCount--;
    if (readCount == 0) {
        sem_post(&writeBlock);
    }
    sem_post(&mutex);
    return NULL;
}

void *writer(void *arg) {
    int writerId = *((int *)arg);

    sem_wait(&writeBlock);
    sharedData++;
    printf("Writer %d increments shared data to: %d\n", writerId, sharedData);
```

```c
    sem_wait(&writeBlock);
    sharedData++;
    printf("Writer %d increments shared data to: %d\n", writerId, sharedData);
    sleep(1);
    sem_post(&writeBlock);
    return NULL;
}

int main() {
    pthread_t r1, r2, w1, w2;
    int r1Id = 1, r2Id = 2, w1Id = 1, w2Id = 2;

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

    pthread_create(&r1, NULL, reader, &r1Id);
    pthread_create(&w1, NULL, writer, &w1Id);
    pthread_create(&r2, NULL, reader, &r2Id);
    pthread_create(&w2, NULL, writer, &w2Id);

    pthread_join(r1, NULL);
    pthread_join(w1, NULL);
    pthread_join(r2, NULL);
    pthread_join(w2, NULL);

    sem_destroy(&mutex);
    sem_destroy(&writeBlock);

    return 0;
}

```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters.c -o ReadersWriters && "c:\Codes\OS\"ReadersWriters
Writer 1 increments shared data to: 1
Writer 2 increments shared data to: 2
Reader 2 reads shared data: 2
Reader 1 reads shared data: 2

c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters.c -o ReadersWriters && "c:\Codes\OS\"ReadersWriters
Reader 1 reads shared data: 0
Reader 2 reads shared data: 0
Writer 1 increments shared data to: 1
Writer 2 increments shared data to: 2

c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters.c -o ReadersWriters && "c:\Codes\OS\"ReadersWriters
Writer 2 increments shared data to: 1
Reader 2 reads shared data: 1
Reader 1 reads shared data: 1
Writer 1 increments shared data to: 2
```

Q2) Modify the question 1 as 1 writer performs increment operation and another writer performs decrement operation of the same account. Readers read and display the shared variable.

C ReadersWriters2.c > ...

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

sem_t mutex, writeBlock;
int sharedData = 0;
int readCount = 0;

void *reader(void *arg) {
    int readerId = *((int *)arg);

    sem_wait(&mutex);
    readCount++;
    if (readCount == 1) {
        sem_wait(&writeBlock);
    }
    sem_post(&mutex);

    printf("Reader %d reads shared data: %d\n", readerId, sharedData);
    sleep(1);

    sem_wait(&mutex);
    readCount--;
    if (readCount == 0) {
        sem_post(&writeBlock);
    }
    sem_post(&mutex);
    return NULL;
}

void *writerIncrement(void *arg) {
    int writerId = *((int *)arg);

    sem_wait(&writeBlock);
    sharedData++;
    printf("Writer %d increments shared data to: %d\n", writerId, sharedData);
    sleep(1);
    sem_post(&writeBlock);
    return NULL;
}

void *writerDecrement(void *arg) {
    int writerId = *((int *)arg);

```

```c
void *writerDecrement(void *arg) {
    int writerId = *((int *)arg);

    sem_wait(&writeBlock);
    sharedData--;
    printf("Writer %d decrements shared data to: %d\n", writerId, sharedData);
    sleep(1);
    sem_post(&writeBlock);
    return NULL;
}

int main() {
    pthread_t r1, r2, w1, w2;
    int r1Id = 1, r2Id = 2, w1Id = 1, w2Id = 2;

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

    pthread_create(&r1, NULL, reader, &r1Id);
    pthread_create(&w1, NULL, writerIncrement, &w1Id);
    pthread_create(&r2, NULL, reader, &r2Id);
    pthread_create(&w2, NULL, writerDecrement, &w2Id);

    pthread_join(r1, NULL);
    pthread_join(w1, NULL);
    pthread_join(r2, NULL);
    pthread_join(w2, NULL);

    sem_destroy(&mutex);
    sem_destroy(&writeBlock);

    return 0;
}
```

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters2.c -o ReadersWriters2 && "c:\Codes\OS\"ReadersWriters2
Reader 1 reads shared data: 0
Reader 2 reads shared data: 0
Writer 2 decrements shared data to: -1
Writer 1 increments shared data to: 0

c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters2.c -o ReadersWriters2 && "c:\Codes\OS\"ReadersWriters2
Reader 1 reads shared data: 0
Reader 2 reads shared data: 0
Writer 1 increments shared data to: 1
Writer 2 decrements shared data to: 0

c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters2.c -o ReadersWriters2 && "c:\Codes\OS\"ReadersWriters2
Writer 2 increments shared data to: 1
Reader 1 reads shared data: 1
Reader 2 reads shared data: 1
Writer 1 decrements shared data to: 0

c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp ReadersWriters2.c -o ReadersWriters2 && "c:\Codes\OS\"ReadersWriters2
Writer 2 increments shared data to: 1
Writer 1 decrements shared data to: 0
Reader 1 reads shared data: 0
Reader 2 reads shared data: 0
```

Q3) Write C program for Dining philosopher and Producer-Consumer

DINING PHILOSOPHER

We can solve this problem using semaphores by associating each fork with a semaphore. Each philosopher will pick up the two semaphores (representing forks) on either side of them before eating and release them afterward.

- Each fork is represented by a semaphore initialized to 1, allowing only one philosopher to hold it at a time.
- Each philosopher goes through an infinite loop of thinking, picking up forks, eating, and then putting the forks back down.
- sem_wait(&forks[rightFork]) and sem_wait(&forks[leftFork]) are used to acquire forks.
- After eating, sem_post(&forks[rightFork]) and sem_post(&forks[leftFork]) release the forks.
- To avoid deadlocks a condition is added that a philosopher can only pick up the chopsticks only if both the chopsticks are available

C DiningPhilosopher.c > ...

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

#define NUM_PHILOSOPHERS 5

sem_t forks[NUM_PHILOSOPHERS];
pthread_mutex_t mutex;

void *philosopher(void *arg) {
    int id = *((int *)arg);
    int leftFork = id;
    int rightFork = (id + 1) % NUM_PHILOSOPHERS;

    while (1) {
        printf("Philosopher %d is thinking.\n", id);
        sleep(1);

        pthread_mutex_lock(&mutex);

        sem_wait(&forks[leftFork]);
        sem_wait(&forks[rightFork]);

        pthread_mutex_unlock(&mutex);

        printf("Philosopher %d is eating.\n", id);
        sleep(1);

        sem_post(&forks[leftFork]);
        sem_post(&forks[rightFork]);

        printf("Philosopher %d has finished eating and put down forks %d and %d.\n", id, leftFork, rightFork);
    }
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int philosopherIds[NUM_PHILOSOPHERS];

    pthread_mutex_init(&mutex, NULL);
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_init(&forks[i], 0, 1);
        philosopherIds[i] = i;
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_create(&philosophers[i], NULL, philosopher, &philosopherIds[i]);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_destroy(&forks[i]);
    }
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp DiningPhilosopher.c -o DiningPhilosopher && "c:\Codes\OS\"DiningPhilosopher
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is thinking.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 4 has finished eating and put down forks 4 and 0.
Philosopher 2 is eating.
Philosopher 1 has finished eating and put down forks 1 and 2.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 0 is eating.
Philosopher 2 has finished eating and put down forks 2 and 3.
Philosopher 2 is thinking.
Philosopher 0 has finished eating and put down forks 0 and 1.
Philosopher 0 is thinking.
Philosopher 4 is eating.
Philosopher 2 is eating.
Philosopher 3 has finished eating and put down forks 3 and 4.
Philosopher 3 is thinking.
Philosopher 2 has finished eating and put down forks 2 and 3.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 4 has finished eating and put down forks 4 and 0.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 0 is eating.
Philosopher 1 has finished eating and put down forks 1 and 2.
Philosopher 1 is thinking.
Philosopher 3 has finished eating and put down forks 3 and 4.
Philosopher 3 is thinking.
Philosopher 2 is eating.
^C
c:\Codes\OS>
```

PRODUCER-CONSUMER

C Producer-Consumer.c > consumer(void *)

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t emptySlots;
sem_t fullSlots;
pthread_mutex_t mutex;

void *producer(void *arg) {
    int item;
    while (1) {
        item = rand() % 100;

        sem_wait(&emptySlots);
        pthread_mutex_lock(&mutex);

        buffer[in] = item;
        printf("Producer produced item %d at index %d\n", item, in);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&fullSlots);

        sleep(rand() % 4);
    }
}

void *consumer(void *arg) {
    int item;
    while (1) {
        sem_wait(&fullSlots);
        pthread_mutex_lock(&mutex);

        item = buffer[out];
        printf("Consumer consumed item %d from index %d\n", item, out);
        out = (out + 1) % BUFFER_SIZE;

```

```
45
46          pthread_mutex_unlock(&mutex);
47          sem_post(&emptySlots);
48
49          sleep(rand() % 3);
50      }
51  }
52
53  int main() {
54      pthread_t producerThread, consumerThread;
55
56      sem_init(&emptySlots, 0, BUFFER_SIZE);
57      sem_init(&fullSlots, 0, 0);
58      pthread_mutex_init(&mutex, NULL);
59
60      srand(time(NULL));
61
62      pthread_create(&producerThread, NULL, producer, NULL);
63      pthread_create(&consumerThread, NULL, consumer, NULL);
64
65      pthread_join(producerThread, NULL);
66      pthread_join(consumerThread, NULL);
67
68      sem_destroy(&emptySlots);
69      sem_destroy(&fullSlots);
70      pthread_mutex_destroy(&mutex);
71
72      return 0;
73  }
74
```

OUTPUT :

```
c:\Codes\OS>cd "c:\Codes\OS\" && gcc -fopenmp Producer-Consumer.c -o Producer-Consumer && "c:\Codes\OS\"Producer-Consumer
Producer produced item 41 at index 0
Consumer consumed item 41 from index 0
Producer produced item 34 at index 1
Consumer consumed item 34 from index 1
Producer produced item 69 at index 2
Producer produced item 78 at index 3
Producer produced item 62 at index 4
Consumer consumed item 69 from index 2
Producer produced item 5 at index 0
Producer produced item 81 at index 1
Consumer consumed item 78 from index 3
Consumer consumed item 62 from index 4
Producer produced item 61 at index 2
Consumer consumed item 5 from index 0
Consumer consumed item 81 from index 1
Consumer consumed item 61 from index 2
Producer produced item 95 at index 3
Consumer consumed item 95 from index 3
Producer produced item 27 at index 4
Consumer consumed item 27 from index 4
Producer produced item 91 at index 0
Producer produced item 2 at index 1
Producer produced item 92 at index 2
Consumer consumed item 91 from index 0
Producer produced item 21 at index 3
Producer produced item 18 at index 4
Consumer consumed item 2 from index 1
Consumer consumed item 92 from index 2
Producer produced item 47 at index 0
Consumer consumed item 21 from index 3
Consumer consumed item 18 from index 4
Producer produced item 71 at index 1
Consumer consumed item 47 from index 0
^C
```