Operating Systems(CSL3030)

Lab Assignment 8 - Report

Soumen Kumar
Roll No-B22ES006

1 Introduction

The objective of this assignment was to implement a multithreaded C program using the `pthread` library. The program uses two threads to compute sums of separate pairs of numbers provided by the main process. Each thread writes its result to a shared memory segment, and the main process retrieves these results to compute and display the total sum.

2 Problem Breakdown

2.1 Thread Creation and Execution

Two threads, T1 and T2, are created to compute the sums of pairs of integers passed from the main process. T1 and T2 each calculate the sum of one pair of numbers and store the result in a shared memory space, which the main process later accesses.

2.2 Shared Memory Setup

To share data between threads and the main process, a shared memory segment is created. System calls `shmget`, `shmat`, and `shmdt` are utilized to allocate, attach, and detach shared memory. This allows data to persist across process boundaries.

2.3 Synchronization with Semaphores

A semaphore is employed to handle race conditions when threads write to the shared memory, ensuring that only one thread accesses the shared resource at any given time. The semaphore is initialized with a value of 1, allowing each thread exclusive access to the shared memory while performing its computations.

3 Code Explanation

The following sections describe the main components of the code:

3.1 Initializing Shared Memory and Semaphores

The program begins by initializing a semaphore to control access to the shared memory, ensuring only one thread writes to it at a time. It then creates a shared memory segment to store the results of the computations.

3.2 Thread Function `compute_sum`

Each thread executes the `compute_sum` function, which receives a pair of numbers, calculates their sum, and writes the result to a designated position in the shared memory. The semaphore protects this write operation to prevent race conditions.

3.3 Creating Threads

The main function creates two threads and passes each a pair of numbers to compute. `pthread_create` is used to initialize each thread, and `pthread_join` waits for the threads to complete.

3.4 Retrieving Results and Calculating the Total Sum

After the threads finish execution, the main process retrieves the computed sums from shared memory. It then calculates the total sum and outputs the results.

3.5 Cleaning Up Resources

Finally, the program detaches the shared memory segment from the process and destroys both the semaphore and the shared memory to free up resources.

4 Execution and Results

The program takes input as pairs of numbers for each thread and outputs the computed sum from each thread and the total sum as calculated by the main process.

4.1 Test Cases

- **Test Case 1**
  - Input: Pair 1: (3, 5), Pair 2: (7, 2)
  - Expected Output:
    - Sum from T1: 8
    - Sum from T2: 9
    - Total Sum: 17
- **Test Case 2**
  - Input: Pair 1: (10, 20), Pair 2: (15, 25)
  - Expected Output:
    - Sum from T1: 30
    - Sum from T2: 40
    - Total Sum: 70
- **Test Case 3**
  - Input: Pair 1: (-5, 10), Pair 2: (-15, 5)
  - Expected Output:
    - Sum from T1: 5
    - Sum from T2: -10
    - Total Sum: -5

5 Screenshots of Test Cases

```
assi8.c:(.text+0x20): undefined reference to `sem_wait'
/usr/bin/ld: assi8.c:(.text+0x42): undefined reference to `sem_post'
/usr/bin/ld: /tmp/cch74mZE.o: in function `main':
assi8.c:(.text+0x7d): undefined reference to `sem_init'
/usr/bin/ld: assi8.c:(.text+0x18c): undefined reference to `pthread_create'
/usr/bin/ld: assi8.c:(.text+0x1af): undefined reference to `pthread_create'
/usr/bin/ld: assi8.c:(.text+0x1c0): undefined reference to `pthread_join'
/usr/bin/ld: assi8.c:(.text+0x1d1): undefined reference to `pthread_join'
/usr/bin/ld: assi8.c:(.text+0x262): undefined reference to `sem_destroy'
collect2: error: ld returned 1 exit status
(base) soumen_kr@Soumen:~/os_lab$ gcc assi8.c -o assi8 -pthread -lrt
(base) soumen_kr@Soumen:~/os_lab$ ./assi8
Enter first pair (num1 num2): 3 5
Enter second pair (num1 num2): 7 2
Sum from T1: 8
Sum from T2: 9
Total Sum: 17
(base) soumen_kr@Soumen:~/os_lab$ ./assi8
Enter first pair (num1 num2): 10 20
Enter second pair (num1 num2): 15 25
Sum from T1: 30
Sum from T2: 40
Total Sum: 70
(base) soumen_kr@Soumen:~/os_lab$ ./assi8
Enter first pair (num1 num2): -5 10
Enter second pair (num1 num2): -15 5
Sum from T1: 5
Sum from T2: -10
Total Sum: -5
(base) soumen_kr@Soumen:~/os_lab$
```

Figure 1: Screenshot of Test Cases Output

6 Conclusion

This project demonstrates a practical implementation of multithreading and shared memory in C using semaphores for synchronization. It efficiently utilizes system calls to allocate and manage shared memory, allowing concurrent threads to communicate and share results with the main process.