National University of Computer and
Emerging Sciences

Lab Manual 9
Operating System Lab

Course Instructor	Ms. Namra Absar
Lab Instructor(s)	Saba Tariq
Section	BCS 4A
Semester	Spring 2024

Department of Computer Science
FAST-NU, Lahore, Pakistan

A semaphore is a synchronization primitive that controls access to shared resources by multiple processes or threads. It maintains a counter and supports two fundamental operations:

- **Wait (P) Operation**: Decrements the semaphore value. If the semaphore value is non-negative, the decrement proceeds, and the process continues. If the semaphore value becomes negative (i.e., no more resources available), the process is blocked until another process increments the semaphore.
- **Signal (V) Operation**: Increments the semaphore value. If any processes were blocked waiting for this semaphore, one of them is allowed to proceed.

POSIX Semaphores

POSIX (Portable Operating System Interface) semaphores are a standardized form of semaphores available on Unix and Linux systems. They provide a way to synchronize processes (including unrelated processes) using named semaphores. POSIX semaphores are part of the POSIX Threads (pthreads) library (**libpthread**).

Key Functions for POSIX Semaphores

1. sem_open

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
```

- Opens or creates a named semaphore.
- **name**: Name of the semaphore (must start with a **/** character).
- **oflag**: Flags indicating the mode of operation (**O_CREAT** for creating if not existing, **O_EXCL** to ensure creation fails if the semaphore already exists).
- **mode**: Permissions for the semaphore if created (**0644** is commonly used).
- **value**: Initial value of the semaphore.

2. sem_wait

```
#include <semaphore.h>

 int sem_wait(sem_t *sem);
```

- Waits (P operation) on the semaphore.
- Decrements the semaphore value.
- Blocks if the semaphore value is zero (no resources available).

3. sem_post

```
#include <semaphore.h>

 int sem_post(sem_t *sem);
```

- Signals (V operation) on the semaphore.
- Increments the semaphore value.

- Unblocks one of the waiting processes (if any).

4. sem_close

#include <semaphore.h>

int sem_close(sem_t *sem);

- Closes the named semaphore.
- Releases the associated resources.
- After closing, the semaphore can no longer be used by the process.

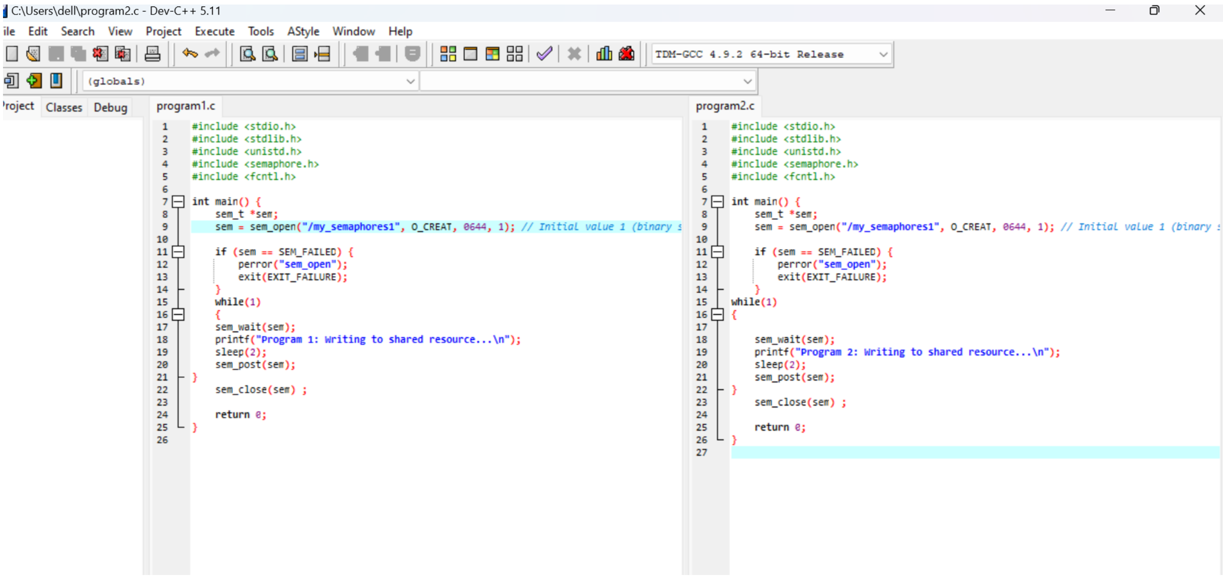
5. sem_unlink

#include <semaphore.h>

int sem_unlink(const char *name);

- Removes a named semaphore from the system.
- The semaphore can no longer be opened or used after unlinking.
- This is typically done after all processes using the semaphore have finished.

Example 1:



C:\Users\dell\program2.c - Dev-C++ 5.11
ile Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
'roject Classes Debug

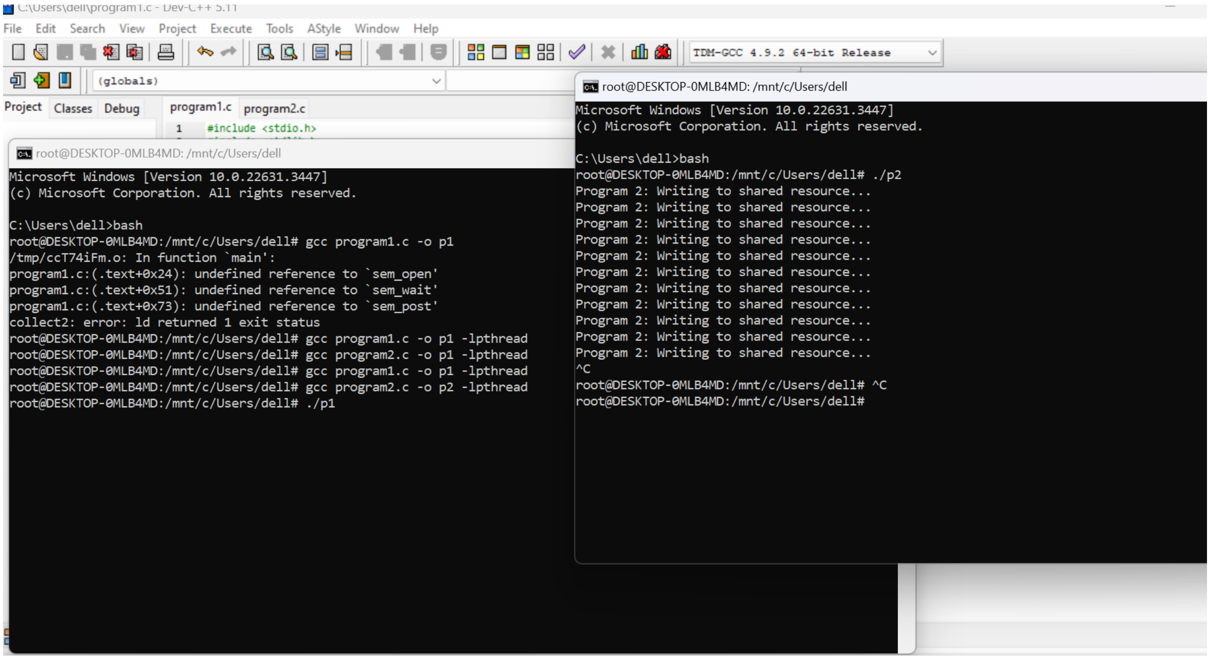
program1.c

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

int main() {
    sem_t *sem;
    sem = sem_open("/my_semaphores1", O_CREAT, 0644, 1); // Initial value 1 (binary s

    if (sem == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }
    while(1)
    {
    sem_wait(sem);
    printf("Program 1: Writing to shared resource...\n");
    sleep(2);
    sem_post(sem);
}
    sem_close(sem) ;

    return 0;
}
```

program2.c

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

int main() {
    sem_t *sem;
    sem = sem_open("/my_semaphores1", O_CREAT, 0644, 1); // Initial value 1 (binary s

    if (sem == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }
while(1)
{

    sem_wait(sem);
    printf("Program 2: Writing to shared resource...\n");
    sleep(2);
    sem_post(sem);
}
    sem_close(sem) ;

    return 0;
}

```

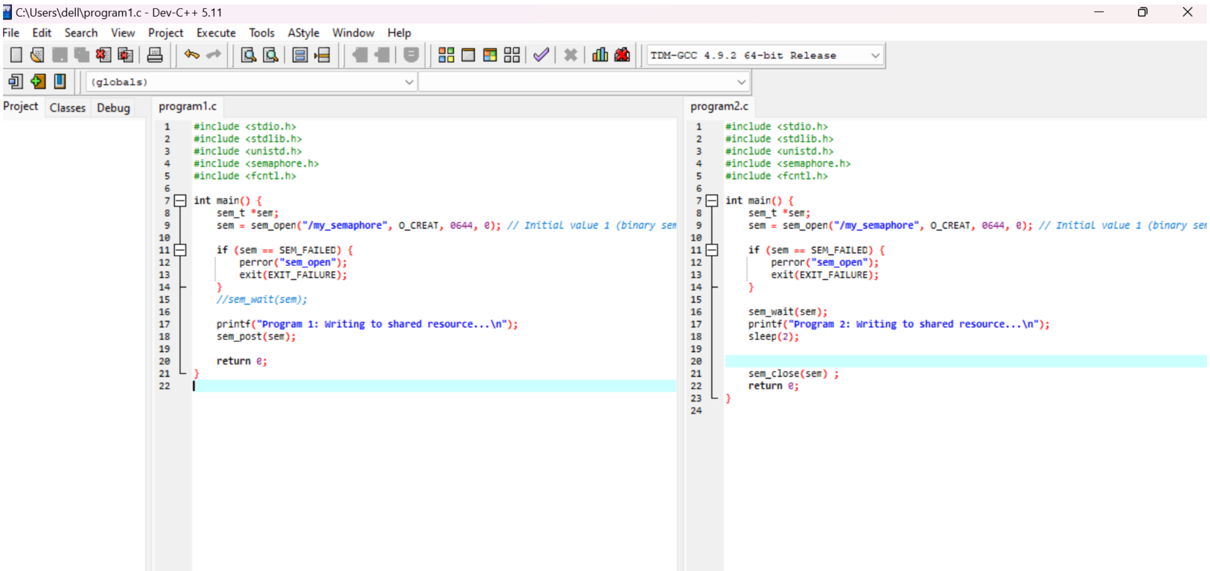
Output:


C:\Users\dell\program1.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
program1.c program2.c
root@DESKTOP-0MLB4MD: /mnt/c/Users/dell
root@DESKTOP-0MLB4MD: /mnt/c/Users/dell

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>bash
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program1.c -o p1
/tmp/ccT74iFm.o: In function `main':
program1.c:(.text+0x24): undefined reference to `sem_open'
program1.c:(.text+0x51): undefined reference to `sem_wait'
program1.c:(.text+0x73): undefined reference to `sem_post'
collect2: error: ld returned 1 exit status
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program1.c -o p1 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program2.c -o p1 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program1.c -o p1 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program2.c -o p2 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# ./p1
```

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>bash
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# ./p2
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
Program 2: Writing to shared resource...
^C
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# ^C
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell#
```

Example 2:


C:\Users\dell\program1.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug

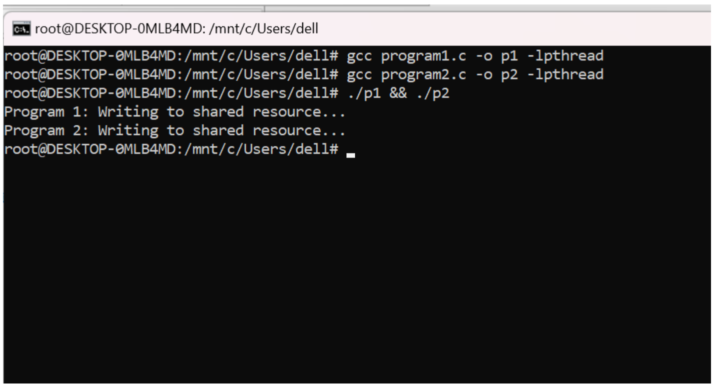
program1.c

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

int main() {
    sem_t *sem;
    sem = sem_open("/my_semaphore", O_CREAT, 0644, 0); // Initial value 1 (binary sem

    if (sem == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }
    //sem_wait(sem);

    printf("Program 1: Writing to shared resource...\n");
    sem_post(sem);

    return 0;
}

```

program2.c

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

int main() {
    sem_t *sem;
    sem = sem_open("/my_semaphore", O_CREAT, 0644, 0); // Initial value 1 (binary sem

    if (sem == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    sem_wait(sem);
    printf("Program 2: Writing to shared resource...\n");
    sleep(2);


    sem_close(sem) ;
    return 0;
}

```

Output 2:



```
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program1.c -o p1 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# gcc program2.c -o p2 -lpthread
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# ./p1 && ./p2
Program 1: Writing to shared resource...
Program 2: Writing to shared resource...
root@DESKTOP-0MLB4MD:/mnt/c/Users/dell# _
```

Question: Consider a scenario where three processes (**P1**, **P2**, **P3**) need to execute in a specific order while sharing a common resource (file) that requires mutual exclusion. Use semaphores to ensure that the processes execute in the sequence **P1 -> P2 -> P3** and have exclusive access to the shared resource when needed.

- P1 opens a file having all integers and calculate their sum and write it into the same file.
- After that P2 counts the integers and also write it in the same file.
- P3 reads sum and count from this file calculated by P1 and P2 and calculates average and print it on the screen.