TRANSPORT AND
TELECOMMUNICATION
INSTITUTE

Operating Systems and System Programming

Topic: Process Management and Inter-Process Communication

Student’s name: Artjoms
Student’s surname: Voroncovs
St. code: st83663

RIGA

TABLE OF CONTENTS

Task Description

The purpose of this practical assignment is to demonstrate the use of operating system-level inter-process communication (IPC) mechanisms through the development of two separate applications. These applications must coordinate access to a shared resource (a text file) while displaying detailed information about process and thread operations, including process IDs, thread IDs, and synchronization steps. The synchronization between threads from different processes is achieved using POSIX named semaphores.

Reflection of the Solution

1. Description of Program Operation

Two separate applications were developed: writer.c and reader.c. Both interact with a shared text file ('shared.txt') while maintaining synchronization using a named POSIX semaphore ('/sync_sem'). The writer application spawns multiple threads that write data (process ID, thread ID, and message) into the file. The reader application spawns multiple threads that read the same file content. Both programs use the same semaphore to ensure mutual exclusion when accessing the file, thus preventing race conditions and data corruption.

2. Choice of Mechanism and Functions

The POSIX named semaphore was chosen for synchronization because it supports inter-process coordination, is easy to manage, and ensures mutual exclusion. The semaphore is created with sem_open(), synchronized using sem_wait() and sem_post(), and released with sem_close() and sem_unlink(). Threads are managed using the POSIX pthread library, which provides efficient multithreading support. The use of file I/O functions (fopen, fprintf, fclose) allows observing concurrent access in real time.

3. Algorithms

Algorithm for Writer Process:

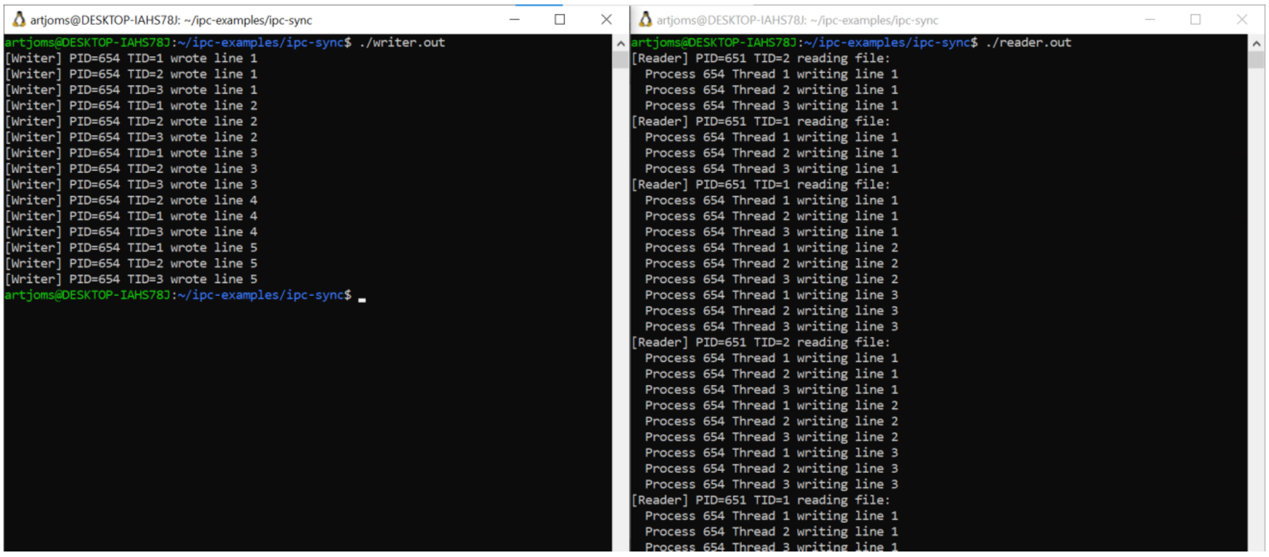
1. Create/open the named semaphore.
2. Spawn several threads.
3. Each thread:
   - Waits for semaphore (sem_wait).
   - Opens the file and appends a new line with its process and thread ID.
   - Releases the semaphore (sem_post).
   - Sleeps briefly to simulate concurrent access.
4. Join threads and close the semaphore.

Algorithm for Reader Process:

1. Open the same named semaphore.
2. Spawn several threads.
3. Each thread:
   - Waits for semaphore (sem_wait).
   - Reads the entire file content and displays it.
   - Releases the semaphore (sem_post).
   - Sleeps to simulate periodic reads.
4. Join threads and close the semaphore.

Proof of Program Operation

Both programs were compiled and executed concurrently in separate terminals on Ubuntu Linux. The output confirmed that access to the shared resource was properly synchronized. The writer process appended lines safely, while the reader process printed file contents without data corruption. The console output displayed detailed information such as PID, TID, and synchronization states:


artjoms@DESKTOP-IAHS78J: ~/ipc-examples/ipc-sync

```
artjoms@DESKTOP-IAHS78J:~/ipc-examples/ipc-sync$ ./writer.out
[Writer] PID=654 TID=1 wrote line 1
[Writer] PID=654 TID=2 wrote line 1
[Writer] PID=654 TID=3 wrote line 1
[Writer] PID=654 TID=1 wrote line 2
[Writer] PID=654 TID=2 wrote line 2
[Writer] PID=654 TID=3 wrote line 2
[Writer] PID=654 TID=1 wrote line 3
[Writer] PID=654 TID=2 wrote line 3
[Writer] PID=654 TID=3 wrote line 3
[Writer] PID=654 TID=2 wrote line 4
[Writer] PID=654 TID=1 wrote line 4
[Writer] PID=654 TID=3 wrote line 4
[Writer] PID=654 TID=1 wrote line 5
[Writer] PID=654 TID=2 wrote line 5
[Writer] PID=654 TID=3 wrote line 5
artjoms@DESKTOP-IAHS78J:~/ipc-examples/ipc-sync$
```

artjoms@DESKTOP-IAHS78J: ~/ipc-examples/ipc-sync

```
artjoms@DESKTOP-IAHS78J:~/ipc-examples/ipc-sync$ ./reader.out
[Reader] PID=651 TID=2 reading file:
  Process 654 Thread 1 writing line 1
  Process 654 Thread 2 writing line 1
  Process 654 Thread 3 writing line 1
[Reader] PID=651 TID=1 reading file:
  Process 654 Thread 1 writing line 1
  Process 654 Thread 2 writing line 1
  Process 654 Thread 3 writing line 1
[Reader] PID=651 TID=1 reading file:
  Process 654 Thread 1 writing line 1
  Process 654 Thread 2 writing line 1
  Process 654 Thread 3 writing line 1
  Process 654 Thread 1 writing line 2
  Process 654 Thread 2 writing line 2
  Process 654 Thread 3 writing line 2
  Process 654 Thread 1 writing line 3
  Process 654 Thread 2 writing line 3
  Process 654 Thread 3 writing line 3
[Reader] PID=651 TID=2 reading file:
  Process 654 Thread 1 writing line 1
  Process 654 Thread 2 writing line 1
  Process 654 Thread 3 writing line 1
  Process 654 Thread 1 writing line 2
  Process 654 Thread 2 writing line 2
  Process 654 Thread 3 writing line 2
  Process 654 Thread 1 writing line 3
  Process 654 Thread 2 writing line 3
  Process 654 Thread 3 writing line 3
[Reader] PID=651 TID=1 reading file:
  Process 654 Thread 1 writing line 1
  Process 654 Thread 2 writing line 1
  Process 654 Thread 3 writing line 1
```

Proof of Completion of All Tasks

The developed solution satisfies all requirements of the assignment:
• Two independent applications were implemented.
• Data was successfully transferred and synchronized through a shared file.
• POSIX named semaphore ensured correct synchronization between processes.
• The program provided detailed process/thread information and activity logs.
• The report describes the mechanism, operation, and justification in detail.

Source Code

writer.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <semaphore.h>

#define THREAD_COUNT 3
#define FILE_NAME "shared.txt"

sem_t *sem;

void* write_to_file(void* arg) {
   int thread_id = *(int*)arg;
   pid_t pid = getpid();

   for (int i = 0; i < 5; i++) {
      sem_wait(sem);
      FILE* f = fopen(FILE_NAME, "a");
      if (f) {
         fprintf(f, "Process %d Thread %d writing line %d\n", pid, thread_id, i+1);
         fclose(f);
      }
      printf("[Writer] PID=%d TID=%d wrote line %d\n", pid, thread_id, i+1);
      sem_post(sem);
      sleep(1);
   }
   return NULL;
}

int main() {
   sem = sem_open("/sync_sem", O_CREAT, 0644, 1);
   pthread_t threads[THREAD_COUNT];
   int ids[THREAD_COUNT];

   for (int i = 0; i < THREAD_COUNT; i++) {
      ids[i] = i + 1;
      pthread_create(&threads[i], NULL, write_to_file, &ids[i]);
   }
   for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
```

```
    sem_close(sem);
    return 0;
}
```

reader.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <semaphore.h>

#define THREAD_COUNT 2
#define FILE_NAME "shared.txt"

sem_t *sem;

void* read_file(void* arg) {
    int thread_id = *(int*)arg;
    pid_t pid = getpid();

    for (int i = 0; i < 5; i++) {
        sem_wait(sem);
        FILE* f = fopen(FILE_NAME, "r");
        if (f) {
            printf("[Reader] PID=%d TID=%d reading file:\n", pid, thread_id);
            char line[256];
            while (fgets(line, sizeof(line), f)) printf("  %s", line);
            fclose(f);
        }
        sem_post(sem);
        sleep(2);
    }
    return NULL;
}

int main() {
    sem = sem_open("/sync_sem", 0);
    pthread_t threads[THREAD_COUNT];
    int ids[THREAD_COUNT];

    for (int i = 0; i < THREAD_COUNT; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, read_file, &ids[i]);
    }
    for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
    sem_close(sem);
    return 0;
}
```

Conclusion

This work demonstrated inter-process communication and synchronization between two independent applications using a shared file. POSIX named semaphores ensured safe access to the resource, preventing race conditions and data corruption.

Both programs showed clear process and thread activity, proving correct synchronization. The assignment met all requirements and improved understanding of process coordination and thread safety in Linux systems.

Appendix

- https://github.com/ArtyomVorontsov/ipc-examples