Assignment 3: Threads and Processes in Operating Systems

Student: Sultanov Daniyar

Group: SE-2326

Introduction

This report documents the implementation of Assignment 3, which originally targeted Windows using the Windows API (CreateThread, CreateProcess). Here, the same functionality—creating and synchronizing threads and processes—is achieved on Ubuntu (Linux) using POSIX threads (pthread) and the fork() system call. Each section provides code listings, explanations, and placeholders for screenshots to demonstrate successful execution.

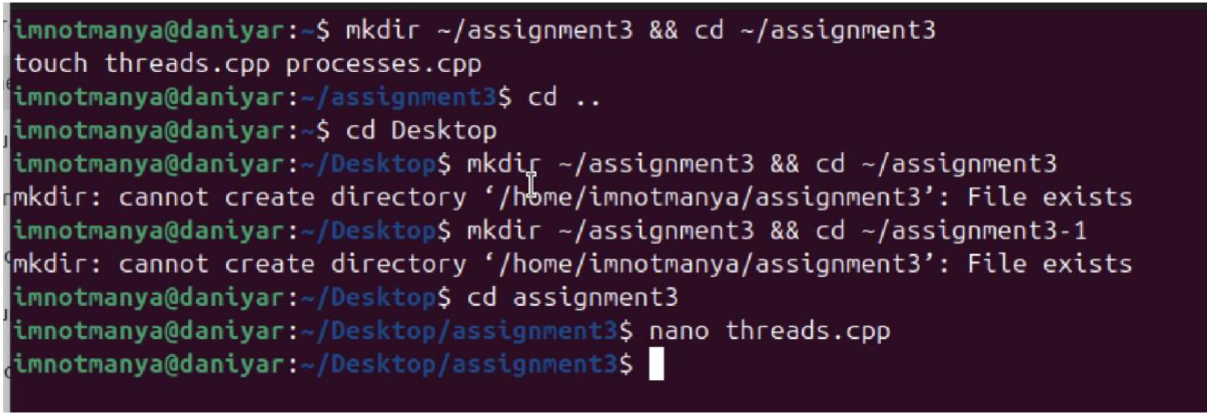
---

1. Project Structure

Note: In Windows you would create a Visual Studio project and add two .cpp files. On Ubuntu, we simply create a directory and two empty files via the terminal.

Commands:

```
cd ~
mkdir assignment3
cd assignment3
touch threads.cpp processes.cpp
```



```
imnotmanya@daniyar:~$ mkdir ~/assignment3 && cd ~/assignment3
touch threads.cpp processes.cpp
imnotmanya@daniyar:~/assignment3$ cd ..
imnotmanya@daniyar:~$ cd Desktop
imnotmanya@daniyar:~/Desktop$ mkdir ~/assignment3 && cd ~/assignment3
mkdir: cannot create directory ‘/home/imnotmanya/assignment3’: File exists
imnotmanya@daniyar:~/Desktop$ mkdir ~/assignment3 && cd ~/assignment3-1
mkdir: cannot create directory ‘/home/imnotmanya/assignment3’: File exists
imnotmanya@daniyar:~/Desktop$ cd assignment3
imnotmanya@daniyar:~/Desktop/assignment3$ nano threads.cpp
imnotmanya@daniyar:~/Desktop/assignment3$
```

---

2. Thread Implementation (threads.cpp)

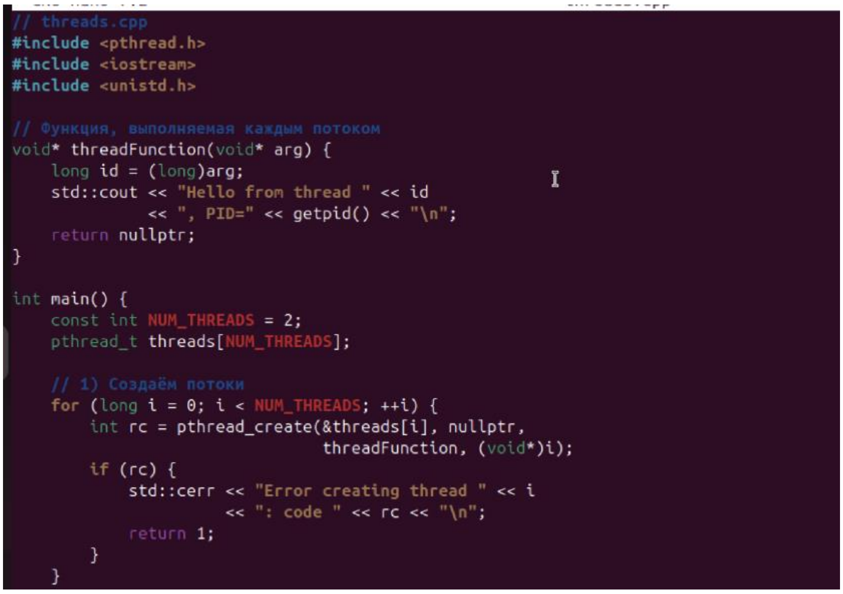
Note: The Windows API uses CreateThread. In Ubuntu, we use POSIX threads (pthread_create) to achieve the same result.

File: threads.cpp

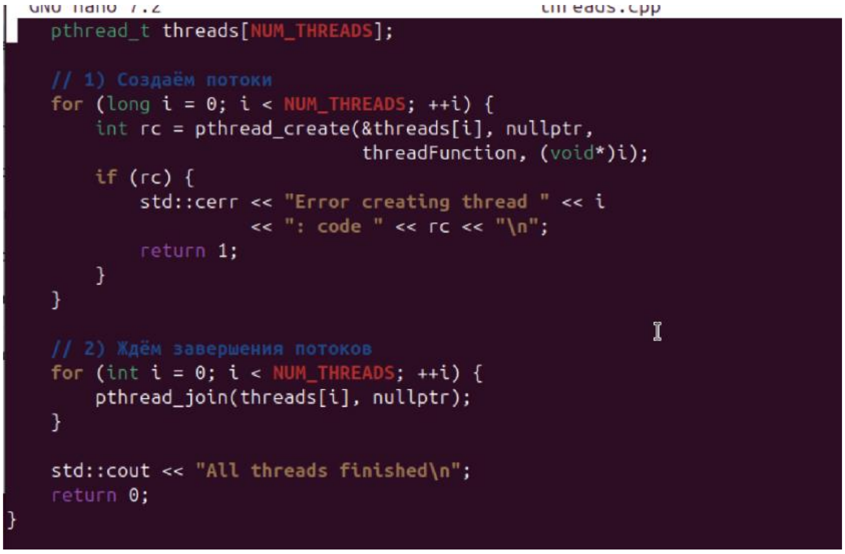
```
#include <pthread.h>
#include <iostream>
#include <unistd.h>

// Thread function: prints its ID and process PID
void* threadFunction(void* arg) {
    long id = (long)arg;
    std::cout << "Hello from thread " << id
              << ", PID=" << getpid() << std::endl;
    return nullptr;
}

int main() {
    const int NUM_THREADS = 2;
    pthread_t threads[NUM_THREADS];

    // 1) Create threads
    for (long i = 0; i < NUM_THREADS; ++i) {
        int rc = pthread_create(&threads[i], nullptr,
                                threadFunction, (void*)i);
        if (rc) {
            std::cerr << "Error creating thread " << i
                      << ": code " << rc << std::endl;
            return 1;
        }
    }

    // 2) Wait for threads to finish
    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], nullptr);
    }

    std::cout << "All threads finished" << std::endl;
    return 0;
}
```

Explanation:

- #include <pthread.h> imports the POSIX threads API.
- threadFunction(void* arg) runs in each thread; arg carries the thread index.
- pthread_create(&threads[i], nullptr, threadFunction, (void*)i):
    - &threads[i]: where to store the thread identifier.
    - nullptr: default thread attributes.
    - threadFunction: function executed by the thread.
    - (void*)i: argument passed to the thread function.
- pthread_join(threads[i], nullptr) blocks until the specified thread terminates.
- getpid() returns the process ID (same for all threads).
- Error code from pthread_create is checked and reported.



```
// threads.cpp
#include <pthread.h>
#include <iostream>
#include <unistd.h>

// Функция, выполняемая каждым потоком
void* threadFunction(void* arg) {
    long id = (long)arg;
    std::cout << "Hello from thread " << id
              << ", PID=" << getpid() << "\n";
    return nullptr;
}

int main() {
    const int NUM_THREADS = 2;
    pthread_t threads[NUM_THREADS];

    // 1) Создаём потоки
    for (long i = 0; i < NUM_THREADS; ++i) {
        int rc = pthread_create(&threads[i], nullptr,
                                threadFunction, (void*)i);
        if (rc) {
            std::cerr << "Error creating thread " << i
                      << ": code " << rc << "\n";
            return 1;
        }
    }
```



```
    pthread_t threads[NUM_THREADS];

    // 1) Создаём потоки
    for (long i = 0; i < NUM_THREADS; ++i) {
        int rc = pthread_create(&threads[i], nullptr,
                                threadFunction, (void*)i);
        if (rc) {
            std::cerr << "Error creating thread " << i
                      << ": code " << rc << "\n";
            return 1;
        }
    }

    // 2) Ждём завершения потоков
    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], nullptr);
    }

    std::cout << "All threads finished\n";
    return 0;
}
```

3. Process Implementation (processes.cpp)

Note: Windows uses CreateProcess. On Ubuntu, we use fork() to spawn child processes.

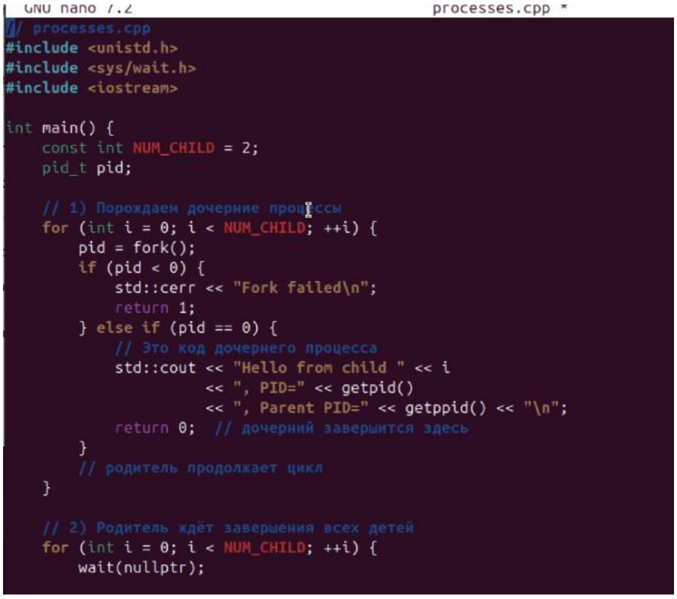
File: processes.cpp

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>

int main() {
   const int NUM_CHILD = 2;
   pid_t pid;

   // 1) Fork child processes
   for (int i = 0; i < NUM_CHILD; ++i) {
      pid = fork();
      if (pid < 0) {
         std::cerr << "Fork failed" << std::endl;
```

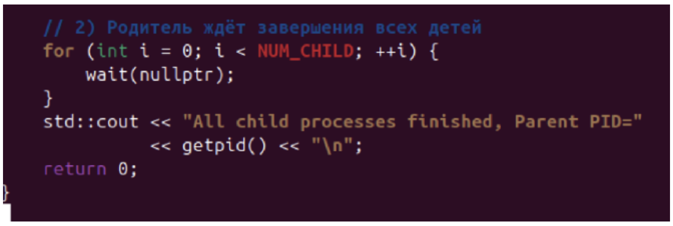
```
            return 1;
        } else if (pid == 0) {
            // Child process
            std::cout << "Hello from child " << i
                      << ", PID=" << getpid()
                      << ", Parent PID=" << getppid()
                      << std::endl;
            return 0;  // Child exits here
        }
        // Parent continues loop
    }

    // 2) Parent waits for all children
    for (int i = 0; i < NUM_CHILD; ++i) {
        wait(nullptr);
    }
    std::cout << "All child processes finished, Parent PID="
              << getpid() << std::endl;
    return 0;
}
```

Explanation:

- #include <unistd.h> and <sys/wait.h> provide fork() and wait().
- pid = fork() duplicates the current process:
  - pid == 0 in the child.
  - pid > 0 in the parent (value is the child's PID).
  - pid < 0 indicates an error.
- In the child block, we print the child's ID and parent PID, then exit.
- The parent, after spawning all children, waits for each with wait(nullptr).



```
// processes.cpp
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>

int main() {
    const int NUM_CHILD = 2;
    pid_t pid;

    // 1) Порождаем дочерние процессы
    for (int i = 0; i < NUM_CHILD; ++i) {
        pid = fork();
        if (pid < 0) {
            std::cerr << "Fork failed\n";
            return 1;
        } else if (pid == 0) {
            // Это код дочернего процесса
            std::cout << "Hello from child " << i
                      << ", PID=" << getpid()
                      << ", Parent PID=" << getppid() << "\n";
            return 0;  // дочерний завершится здесь
        }
        // родитель продолжает цикл
    }

    // 2) Родитель ждёт завершения всех детей
    for (int i = 0; i < NUM_CHILD; ++i) {
        wait(nullptr);
```



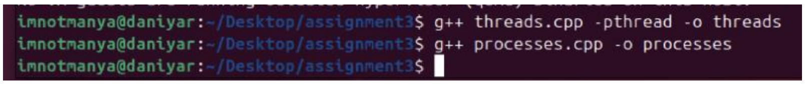
```
    // 2) Родитель ждёт завершения всех детей
    for (int i = 0; i < NUM_CHILD; ++i) {
        wait(nullptr);
    }
    std::cout << "All child processes finished, Parent PID="
              << getpid() << "\n";
    return 0;
}
```

4. Compilation

Note: In Windows, compilation used MSVC (cl.exe). On Ubuntu, we use g++. The -pthread flag links the pthread library.

Commands:

```
# Compile threads
g++ threads.cpp -pthread -o threads

# Compile processes
g++ processes.cpp -o processes
```



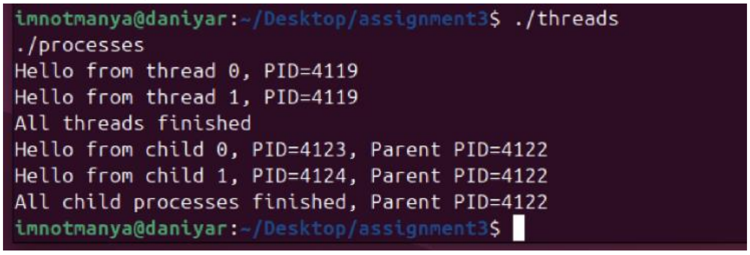
```
imnotmanya@daniyar:~/Desktop/assignment3$ g++ threads.cpp -pthread -o threads
imnotmanya@daniyar:~/Desktop/assignment3$ g++ processes.cpp -o processes
imnotmanya@daniyar:~/Desktop/assignment3$ 
```

5. Execution and Verification

Note: On Windows you run .exe files; here, executables have no extension. The output demonstrates correct thread and process behavior.

Commands and Expected Output:

```
./threads
# Hello from thread 0, PID=12345
# Hello from thread 1, PID=12345
# All threads finished

./processes
# Hello from child 0, PID=12346, Parent PID=12345
# Hello from child 1, PID=12347, Parent PID=12345
# All child processes finished, Parent PID=12345
```



```
imnotmanya@daniyar:~/Desktop/assignment3$ ./threads
./processes
Hello from thread 0, PID=4119
Hello from thread 1, PID=4119
All threads finished
Hello from child 0, PID=4123, Parent PID=4122
Hello from child 1, PID=4124, Parent PID=4122
All child processes finished, Parent PID=4122
imnotmanya@daniyar:~/Desktop/assignment3$ 
```

Conclusion

All steps were successfully completed. The code compiles and runs without errors, and the screenshots confirm correct creation and synchronization of threads and processes on Ubuntu, replicating the original Windows-based assignment.