# Звіт до лабораторної 5 (Inter-Process Communications (IPC) + Threads)

Студента групи ТТП-32

Остренка Олександра

## 1. Умова лабораторної:

## Изображение выглядит как текст, Шрифт, белый, чек Автоматически созданное описание

## 2. Реалізація:

### 2.1. Допоміжна програма f.cpp

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Програма провертає 1, якщо число парне; 0 – якщо непарне.

### 2.2. Допоміжна програма g.cpp

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Програма повертає 1, якщо число більше п’яти; 0 – якщо менше.

### 2.3. Основна програма main.cpp

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

На початку іде перевірка валідності вводу. Вхідний рядок має бути числом, відповідно кожен символ – цифрою.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Далі, щоб забезпечити двосторонній канал зв’язку між основним процесом та двома допоміжними (які виконують програми f.cpp та g.cpp), створюємо пару сокетів за допомогою socketpair():

* AF\_UNIX вказує на те, що буде використовуватися Unix-домен сокетів, який призначений для спілкування між процесами на одному і тому ж комп’ютері;
* SOCK\_STREAM означає, що буде використовуватися потоковий сокет, який забезпечує надійне, двостороннє, послідовне та з’єднане спілкування.
* 0: Цей параметр вказує на протокол, який буде використовуватися. Для AF\_UNIX та SOCK\_STREAM, зазвичай вказується 0, оскільки існує лише один протокол потокового спілкування в Unix-домені, тому немає необхідності вказувати конкретний протокол.
* sv: Це масив з двох цілих чисел, який функція socketpair() заповнює дескрипторами сокетів. Після виклику функції, sv[0] та sv[1] будуть дескрипторами двох сокетів, які можуть використовуватися для спілкування між процесами. Все, що написано в один сокет, може бути прочитано з іншого, і навпаки.

Далі створюємо дочірній процес для виконання файлу f – «pid\_f = fork()». Якщо fork() повертає 0, це означає, що ми знаходимося у дочірньому процесі. Якщо повертається позитивне число, це PID дочірнього процесу, і ми все ще знаходимося у батьківському процесі. Якщо повертається негативне значення, це означає, що сталася помилка при створенні дочірнього процесу. Після перевірки закриваємо сокет, що використовується батьківським процесом – «close(sv[0])». «execlp("./f", "f", argv[1], NULL)» замінює поточний дочірній процес новою програмою, яка виконується з файлу ./f. Перший аргумент execlp - це шлях до виконуваного файлу, другий аргумент – це ім’я, яке з’явиться як argv[0] у виконуваній програмі, а argv[1] передається як аргумент цій програмі. NULL вказує на кінець списку аргументів. «exit(1)» виконується тільки у випадку, якщо execlp не вдається. execlp зазвичай не повертає управління, якщо вона успішно запускає нову програму. Якщо ж вона повертає управління, це означає, що сталася помилка, і тому дочірній процес викликає exit(1), щоб завершити себе з кодом помилки 1.

Далі вертаємось в основний процес. «waitpid(pid\_f, &status\_f, 0);» використовується для очікування завершення конкретного дочірнього процесу, статус завершення процесу запишеться у відповідну змінну. Третій аргумент, який в цьому випадку дорівнює 0, вказує на те, що основний процес не використовує жодних спеціальних опцій і просто чекає завершення дочірнього процесу. WIFEXITED(status): перевіряє, чи дочірній процес завершився нормально (тобто через виклик exit() або завершення main(). WEXITSTATUS(status) повертає код завершення, який дочірній процес передав у exit(), якщо WIFEXITED повернула true

## 3. Тестування.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание