МІНИСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Дніпровський національний університет
залізничного транспорту ім. академіка В. Лазаряна

Кафедра КІТ

ЛАБОРАТОРНА РОБОТА № 7

з дисципліни: «Операційні системи»

на тему: «Вивчення механізму сигналів UNIX»

Виконав: студент групи ПЗ1712

Нікольський О.В.

Приняла: Нежуміра О.І.

Дніпро, 2020

Тема. Вивчення механізму сигналів UNIX.

Мета:

- ознайомитися з механізмом сигналів UNIX (види сигналів, способи і засоби посилки сигналів);
- отримати практичні навички програмування передачі і обробки сигналів;
- отримати практичні навички використання сигналів.

Опис специфікацій

main.cpp	Основна програма; створює дочірні процеси, які, в свою чергу, запускають потоки

```
#include <iostream>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include "parent.cpp"

int main(int argc, char*argv[])
{
// 1. Создать файл для передачи данных
// Имя файла передается как параметр при вызове программы
const char* path = argv[1];
// разрешения для создаваемого файла:
// Текущий пользователь имеет право на чтение
// Текущий пользователь имеет право на запись
// Группа (с правами, как у текущего пользователя) имеет право на чтение
// Остальные пользователи имеют право на чтение
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
int file = creat(path, mode);
// 2. Запустить второй процесс
std::string num;
int pid;
int *tpid;
for(int i = 1; i <= 3; ++i){
pid = fork();
// 3. Во втором процессе запустить поток выполнения
// Отделяем код процесса-потомка от кода процесса-родителя
if(pid == 0){
// Запускаем приложение записи строки в файл в отдельном потоке
execl("./child", path, "\nThread #", std::to_string(i).c_str(), NULL);
}
// 4. В родительском процессе считать данные из файла
// Махинация с указателем нужна для передачи идентификатора дочернего
процессора в функцию ожидания в родительском
// Ожидание успешного заверешения дочернего процесса
tpid = new int(pid);
```

```
wait(tpid);
}
delete tpid;
// Работа родительского процесса
Parent(path);
return 0;
}
```

parent.cpp	Містить метод, що зчитує дані з файлу; На вхід подається ім'я файлу, з якого проводитиметься зчитування

```
#include <iostream>
#include <fcntl.h>
#include <string>
#include <cstring>
#include <unistd.h>

void Parent(std::string filename){
// Открытие файла с именем, полученным как параметр при вызове программы из процесса в режиме "чтение и запись"
int file = open(filename.c_str(), O_RDONLY);
char buf;
std::string mes;
// Побайтово считываем содержимое файла в переменную
while(read(file, &buf, 1) > 0){
mes += buf;
}
// Выводим считанные данные на экран
std::cout << "Parent: " << mes << std::endl;
close(file);
}
```

child.cpp	Побічна програма; запускається з програми main, записує дані у файл; На вхід подається назва файлу та дані, що будуть записані у файл

```
#include <fcntl.h>
#include <string>
#include <cstring>
#include <unistd.h>
#include <iostream>

int main(int argc, char*argv[]){
// Открытие файла с именем, полученным как параметр при вызове программы из процесса; режим дозаписи
int file = open(argv[0], O_WRONLY | O_APPEND);
std::string mfc = std::strcat(argv[1], argv[2]);
// Запись сообщения в файл и его закрытие
```

```
write(file, mfc.c_str(), strlen(mfc.c_str()));
close(file);
return 0;
}
```

Makefile	Файл з описом порядку дій для компілятора

```
all: start
# Порядок действий Makefile
start: clear child parent.o makelib launch
clear:
# Очистка консоли
clear
child:
# Создание исполняемого файла из child.cpp, нужен для вызова execl()
g++ child.cpp -o child
parent.o:
# Создание объектного файла, нужен для дальнейшей работы main-a
g++ -c parent.cpp
makelib:
# Создание библиотеки, содержащей объектные файлы
ar rc lib.a parent.o
ranlib lib.a
launch:
# Компилируем main.cpp в исполняемый файл main
g++ main.cpp -o main
# Запускаем его
clear
./main mf.txt
```

Висновки

Ознайомилися з процессом створення та запуску потоків у Linux.

У ході лабораторної роботи реалізували програму, яка створює новий процес, що є фактично повною копією бітьківського, у якому запускаємо новий поток виконання. Керування повертається до батьківського процесу за допомогою сигналу від дочірнього процесу, а дані — через файл.