Міністерство освіти України
Харківський національний університет радіоелектроніки

Звіт до лабораторної роботи № 5
з курсу «Операційні системи UNIX»

Виконав: ст. гр. ПЗПІ-16-3
Губар С.О.
Прийняв: Сокорчук І.П.

Харків 2020

Мета роботи: познайомитися з процесами, сигналами у операційних система UNIX. Написати програму на C, який запускає ще один процес, та взаємодіє з ним за допомогою сигналів. Навчитися використовувати Makefile.

Хід виконання роботи

Створимо файл lab5.c. Підключимо усі необхідні залежності.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
```

Запустимо дочірній процес за допомогою виклику fork(). Додамо обробку сигналів за допомогою signal. Також для SIGCHLD вкажемо SIG_IGN щоб дочірній процес не закінчував роботу з батьківським автоматично.

```
int pid = fork();
if (pid == 0) {
    signal(SIGUSR1, child_usr1_handler);
    signal(SIGTERM, child_term_handler);
    signal(SIGUSR2, child_ping_back_handler);
    while (1) {

    }
} else {
    signal(SIGUSR2, parent_ping_back_handler);
    signal(SIGCHLD, SIG_IGN);
```

Додамо взаємодію з користувачем

```
printf("Enter the action:\n");
char text_choice[100];
int choice;

scanf("%s", text_choice);
if (strcmp(text_choice, "ping") == 0)
{
    choice = PING_CHOICE;
}
if (strcmp(text_choice, "kill") == 0)
{
    choice = KILL_CHOICE;
}
if (strcmp(text_choice, "back") == 0)
{
    choice = PING_BACK_CHOICE;
}
if (strcmp(text_choice, "quit") == 0)
{
    choice = EXIT_CHOICE;
}
```

Реалізуємо функцію журналювання. Перевіримо, що директорія для логування існує, або створимо її якщо треба

```
void log_message(char *str, int pid, int signal, char* signal_str)
{
    FILE *file;
    const char *home = getenv("HOME");

    struct stat st = {0};
```

```
    char *log_dir = concat(home, "/log");

    if (stat(log_dir, &st) == -1) {
        printf("Directory fail %s", log_dir);
        mkdir(log_dir, 0777);
    }
```

Отформатуємо дату та час

```
   char *path = concat(log_dir, "/pzpi-16-3-hubar-serhii-lab5.log");
    time_t rawtime;
    struct tm *info;
    char buffer[80];

    time(&rawtime);

    info = localtime(&rawtime);

    strftime(buffer,80,"%a, %d %b %Y %X %z", info);
```

Сформуємо лог та запишемо до файла

```
char* timestamp = (char*)malloc(20 * sizeof(char));
    sprintf(timestamp, "%d",(int)time(NULL));
    file = fopen(path, "a+");

    char* message = (char*)malloc(100 * sizeof(char));
     sprintf(message, "%s; %s; %d; %d; %s; %s\n", buffer, timestamp, pid, signal, signal_str,
str);
    fprintf(file, "%s", message);
    fclose(file);
```

Також запишемо до logger

```
 char* logger_formatted_message = (char*)malloc(100 * sizeof(char));
```

```
    sprintf(logger_formatted_message, "\"%s\"", message);
    system(concat("logger ", logger_formatted_message));
```

Напишемо функції, які обробляють сигнали

```
void parent_ping_back_handler() {
    printf("parent ping back handler\n");
    log_message("Nothing - child pinged back signal", getpid(), 17, "SIGUSR2");
}

void child_usr1_handler() {
    printf("child usr1 handler\n");
    log_message("Nothing - ping signal", getpid(), 16, "SIGUSR1");
}

void child_ping_back_handler() {
    printf("child ping back handler\n");
    log_message("Nothing - child pinged back signal", getpid(), 17, "SIGUSR2");
    ping_back(getppid());
}

void child_term_handler() {
    printf("child term handler\n");
    log_message("Child termination", getpid(), 15, "SIGTERM");
    exit(1);
}

void parent_child_died_handler() {
    printf("parent child die handler\n");
    log_message("Child process die", getpid(), 18, "SIGCHLD");
}
```

Прив’яжемо дані, які вводить користувач до відсилання сигналів

```
switch (choice)
        {
        case 10:
           ping_child_process(pid);
           break;
        case 11:
           kill_child_process(pid);
           break;
        case 12:
           ping_back(pid);
           break;
        case 13:
           exit = 1;
           break;
        default:
           printf("Try again\n");
           break;
        }
```

Зробимо Makefile для зручної вхаємодії з кодом. Для цього створимо файл з назвою Makefile, та додамо до нього наступні команди:

All - збирає наш код за допомогою gcc

```
all:
        gcc -o pzpi-16-3-hubar-serhii-lab5.out pzpi-16-3-hubar-serhii-lab5.c
```

Install - запускає скрипт перевірки залежностей, та в разі успішного виконання збирає проект та запускає скрипт встановлення у ~/bin

```
install:
        chmod +x ./install.sh
        chmod +x ./configure.sh
        ./configure.sh
        gcc -o pzpi-16-3-hubar-serhii-lab5.out pzpi-16-3-hubar-serhii-lab5.c
```

```
	./install.sh
```

Uninstall визиває clean, та видаляє встановлений до bin файл. Clean видаляє усі out файли та логи у поточній директорії (якщо вони є).

```
uninstall:
	make clean
	rm ${HOME}/bin/pzpi-16-3-hubar-serhii-lab5.out

clean:
	rm *.out || true
	rm *.log || true
```

Також для зручності додамо ціль archive, щоб зібрати проект у архів

```
archive:
	tar -czvf pzpi-16-3-hubar-serhii-lab5.tar.gz Makefile configure.sh install.sh pzpi-16-3-hubar-serhii-lab5.c README.md
```

Для того, щоб впевнитися, що проект можна зібрати, зробимо скрипт configure.sh, який буде перевіряти наявність gcc

```
#!/bin/bash

if ! [ -x "$(command -v gcc)" ]; then
  echo 'Error: gcc is not installed.' >&2
  exit 1
fi

echo 'All components are available, continue installation'
```

Висновки: у ході виконання лабораторної роботи було отримано практичні навички з взаємодії з процесами та сигналами у ОС Unix. Також було створено програму з використанням мови програмування С, яка демонструє можливості взаємодії між процесами.