Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій

Кафедра радіоелектронних
і комп'ютерних систем

Звіт
про виконання лабораторних робіт №5
«Керування процесами і потоками»

Виконав студент
групи ФЕІ-23
Ковальчук Д. М.
Перевірив Сінькевич О. О.

Львів — 2019

Мета: Вивчення та застосування програмних інтерфейсів ОС для керування процесами та потоками.

Завдання 1: Напишіть функцію, виклик якої приведе до знищення всіх процесів-зомбі. створених поточним процесом.

Результат роботи програми:

```
binchukchik@binchukchik-VirtualBox:~/oc$ gcc part_1.c -o zombie
binchukchik@binchukchik-VirtualBox:~/oc$ ./zombie
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
binchuk+  2558  0.0  0.0   2320   764 pts/1    S+   20:15   0:00 ./zombie

binchukchik@binchukchik-VirtualBox:~/oc$
```

Код програми :

```c
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void killZombie()
{
    pid_t pid;
    int status;
    printf("\n");

    while((pid= wait(&status))>0)
        printf("Zombie proces with PID %d terminated with status %d\n", pad,status);
}   printf("\n");

void zombieList()
{
    pid_t pid;
    if ((pid = fork())==-1) exit(-1);
    if (pid==0)
    {
        char *args[]={"ps","uf","-C","zombie",NULL};
        execvp("ps", args);
        exit(-1);
    }
    else
    {
        int status;
        waitpid(pid,&status,NULL);
        if(!WIFEXITED(status)) exit(-1);
    }
}
int int main(int argc, char const *argv[])
{
    zombieList();
    killZombie();
    return 0;
}
```

Завдання 2: Розробіть простий командний інтерпретатор для Linux і Windows. Він повинен видавати підказку (наприклад, «>»), обробляти введений користувачем командний рядок (що містить ім’я виконуваного файлу програми та її аргументи) і запускати задану

програму. Асинхронний запуск здійснюють уведенням «&» як останнього символу командного рядка. У разі завершення командного рядка будь-яким іншим символом програма запускається синхронно. Інтерпретатор завершує роботу після введення рядка «exit». Виконання програм, запущених інтерпретатором, може бути перерване натисканням клавіш Ctrl+C, однак воно не повинне переривати виконання інтерпретатора. Для запуску програмного коду в Linux рекомендовано використовувати функцію execvp(), що приймає два параметри prog і args, аналогічні до перших двох параметрів функції execve(), і використовує змінну оточення PATH для пошуку шляху до виконуваних файлів.

Результати роботи програми:

```
binchukchik@binchukchik-VirtualBox:~$ cd oc
binchukchik@binchukchik-VirtualBox:~/oc$ gcc part_2.c -o part_2
binchukchik@binchukchik-VirtualBox:~/oc$ ./part_2
myShell started
>pwd
/home/binchukchik/oc
>ls
part_1.c  part_1.o  part_2  part_2.c  part_2.o  zombie
```

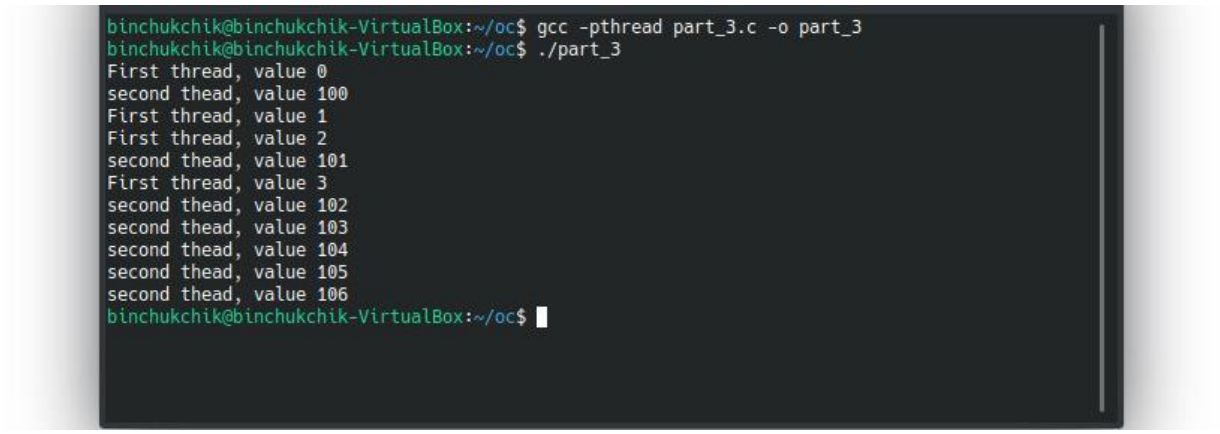
Код програми:

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

pid_t pid;

void exeCommand(char* command)
{
    char* argv[20];
    int beakground;

    char* token;
    token = strtok(command," ");
    int t = 0;
    for (; token != NULL; i++)
    {
        argv[i]=token;
        token=strtok(NULL," ");
    }
    if(!strcmp(argv[i-1],"&"))
    {
        beakground =1;
        argv[i-1]=NULL;
    }
    else
    {
        beakground=0;
        argv[i]=NULL;
    }
    if ((pid=fork())==-1) exit(1);
    if((pid==0))
    {
        execvp(argv[0],argv);
        exit(1);
    }
    if(!beakground) wait(NULL);

}
static void catch_function(int signal)
```

```
{
    kill(pid,SIGINT);
    printf("\n");
}
int int main(int argc, char const *argv[])
{
    printf("myShell started\n");
    signal(SIGINT,catch_function);
    char command[200];
    while(1)
    {
        printf(">");
        gets(command);

        if (command[0]=='\0') continue;
        if (!strcmp(command,"exit")) break;
        exeCommand(command);
    }
    printf("You are exit!\n");
    return 0;
}
```

Завдання 3: Розробіть застосування для Linux і Windows, що реалізує паралельне виконання коду двома потоками. Основний потік застосування T створює потік t Далі кожен із потоків виконує цикл (наприклад, до 30). На кожній ітерації циклу він збільшує значення локального лічильника на одиницю, відображає це значення з нового рядка і призупиняється на деякий час (потік T– на час w T , потік t – w t). Після завершення циклу потік T приєднує t. Як залежать результати виконання цього застосування від значень w T і w t ? Як зміняться ці результати, якщо потік t не буде приєднано?

Результат роботи програми:



Код програми:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *anotherThead(void *value)
{
    int i =0;
    for (int i = 100; i < 106; i++)
    {
        printf("second thead, value %d\n",i);
        sleep(2);
    }
}
int main(int argc, char const *argv[])
{
    pthread_t thead;
    pthread_create(&thead,NULL,anotherThead,NULL);

    int i =0;
    for (int i = 0; i < 4; i++)
    {
        printf("First thread, value %d\n",i);
        sleep(1);
    }
    pthread_join(thead,NULL);
    return 0;
}
```

Висновок: На даній лабораторній роботі я мав змогу вивчити та застосовати програмні інтерфейси ОС для керування процесами та потоками. Реалізував програми, які реалізовували завдання поставлені у лабораторній роботі. Результат роботи та код програм представив у звіті вище.