Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

Звіт

Про виконання лабораторної роботи №8

З курсу «Операційні системи та системне програмування»

«Керування процесами і потоками»

Виконала:

Студентка групи ФеС-21

Шавало А.А.

Львів – 2024

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

Хід роботи:

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

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

void destroy_zombie_processes() {
    pid_t pid;
    int status;

    while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
        printf("Знищено зомбі процес: %d\n", pid);
    }
}

int main() {
    pid_t pid;

    for (int i = 0; i < 5; i++) {
        pid = fork();
        if (pid == 0) {
            printf("Дочірній процес %d запущено.\n", getpid());
            exit(0);
        } else if (pid < 0) {
            perror("fork помилка");
            exit(1);
```

```c
        }
    }
    sleep(2);
    destroy_zombie_processes();

    return 0;
}
```

```
andriy@andriy-VirtualBox:~/lab8$ gcc -o zombie_processes zombie_
processes.c
andriy@andriy-VirtualBox:~/lab8$ ./zombie_processes
Дочірній процес 9753 запущено.
Дочірній процес 9757 запущено.
Дочірній процес 9756 запущено.
Дочірній процес 9754 запущено.
Дочірній процес 9755 запущено.
Знищено зомбі процес: 9753
Знищено зомбі процес: 9754
Знищено зомбі процес: 9755
Знищено зомбі процес: 9756
Знищено зомбі процес: 9757
```

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.

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

#define MAX_CMD_LENGTH 1024

    command(char *cmd) {
    pid_t pid = fork();
    if (pid == 0) {
        char *args[100]; /
        int i = 0;

        char *token = strtok(cmd, " ");
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;

        execvp(args[0], args);
        perror("execvp помилка");
        exit(1);
```

App Center

```
    } else if (pid < 0) {
        perror("fork помилка");
    } else {
        int status;
        waitpid(pid, &status, 0);
    }
}

int main() {
    char cmd[MAX_CMD_LENGTH];

    while (1) {
        printf("> ");
        fgets(cmd, MAX_CMD_LENGTH, stdin);

        cmd[strcspn(cmd, "\n")] = 0;

        if (strcmp(cmd, "exit") == 0) {
            break;
        }
```

```
        int async = (cmd[strlen(cmd) - 1] == '&');
        if (async) {
            cmd[strlen(cmd) - 1] = '\0';
            run_command(cmd);
        } else {
            run_command(cmd);
        }
    }

    return 0;
}
```

```
andriy@andriy-VirtualBox:~/lab8$ gcc -o simple_shell simple_shell.c
andriy@andriy-VirtualBox:~/lab8$ ./simple_shell
> ls
kill.sh   simple_shell    zombie_processes
kill_z.c  simple_shell.c  zombie_processes.c
> ls -l
total 52
-rw-rw-r-- 1 andriy andriy   482 Oct 23 00:03 kill.sh
-rw-rw-r-- 1 andriy andriy   482 Oct 22 23:59 kill_z.c
-rwxrwxr-x 1 andriy andriy 16520 Oct 23 00:09 simple_shell
-rw-rw-r-- 1 andriy andriy  1691 Oct 23 00:08 simple_shell.c
-rwxrwxr-x 1 andriy andriy 16320 Oct 23 00:05 zombie_processes
-rw-rw-r-- 1 andriy andriy  1146 Oct 23 00:04 zombie_processes.c
>
```

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 wT, потік t – wt). Після завершення циклу потік T приєднує t. Як залежать результати виконання цього застосування від значень wT і wt? Як зміняться ці результати, якщо потік t не буде приєднано?

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

#define MAX_COUNT 30

void *thread_function(void *arg) {
    int *count = (int *)arg;

    for (int i = 0; i < MAX_COUNT; i++) {
        printf("Потік %ld: %d\n", pthread_self(), i + 1);
        (*count)++;
        usleep(100000);
    }
    return NULL;
}

int main() {
    pthread_t thread;
    int count1 = 0, count2 = 0;

    pthread_create(&thread, NULL, thread_function, &count2);
```

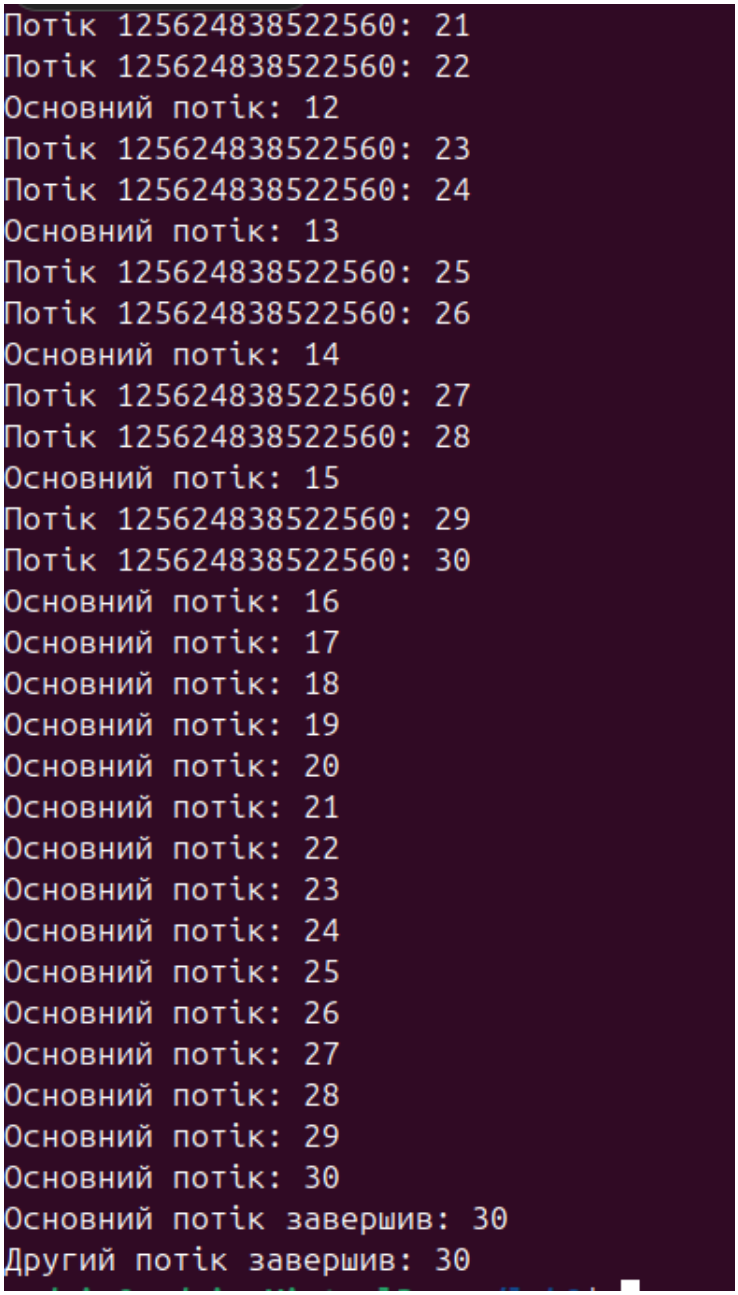
```
    for (int i = 0; i < MAX_COUNT; i++) {
        printf("Основний потік: %d\n", i + 1);
        count1++;
        usleep(200000);
    }

    pthread_join(thread, NULL);

    printf("Основний потік завершив: %d\n", count1);
    printf("Другий потік завершив: %d\n", count2);

    return 0;
}
```

```
andriy@andriy-VirtualBox:~/lab8$ ./threads_example
Основний потік: 1
Потік 125624838522560: 1
Потік 125624838522560: 2
Основний потік: 2
Потік 125624838522560: 3
Потік 125624838522560: 4
Основний потік: 3
Потік 125624838522560: 5
Потік 125624838522560: 6
Основний потік: 4
Потік 125624838522560: 7
Основний потік: 5
Потік 125624838522560: 8
Потік 125624838522560: 9
Основний потік: 6
Потік 125624838522560: 10
Потік 125624838522560: 11
Основний потік: 7
Потік 125624838522560: 12
Потік 125624838522560: 13
Основний потік: 8
Потік 125624838522560: 14
Потік 125624838522560: 15
Потік 125624838522560: 16
Основний потік: 9
Потік 125624838522560: 17
Потік 125624838522560: 18
Основний потік: 10
Потік 125624838522560: 19
Потік 125624838522560: 20
Основний потік: 11
Потік 125624838522560: 21
```



Висновок: виконуючи лабораторну роботу 8, я вивчиння та застосування програмних інтерфейсів ОС для керування процесами та потоками