

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу**

**Кафедра штучного інтелекту**

**ЛАБОРАТОРНА РОБОТА №7**

**з дисципліни «Операційні системи»**

**Варіант № 38**

Виконав:

Студент II курсу

Групи КІ-32

Присяжнюк

Владислав

Прийняв:

Коваленко А. Є.

**Київ 2024**

## ЗАВДАННЯ У 1

Створення командного файлу з використанням системного виклику мови С. .

Створити командний файл з використанням відкомпільованого файлу

програми на С для обробки системного виклику з відповідними аргументами

1 і 3 команди sleep оболонки до і після виконання

системного виклику. Системний виклик open з правами доступу

(O\_WRONLY (write-only)) для файлу rt1.txt. Застосувати командний файл за наявності відповідного файлу

```
#include <stdio.h>      // для printf, perror
#include <stdlib.h>      // для exit
#include <unistd.h>      // для close
#include <fcntl.h>      // для open, O_WRONLY
#include <errno.h>      // для errno
#include <string.h>

int main(int argc, char *argv[])
{
    const char *filename = "rt1.txt";

    int fd = open(filename, O_WRONLY);
    if (fd == -1) {
        perror("Помилка open");
        return EXIT_FAILURE;
    }

    printf("Файл '%s' успішно відкрито у режимі O_WRONLY!\n", filename);

    const char *msg = "Це запис у файл rt1.txt.\n";
    ssize_t bytes_written = write(fd, msg, strlen(msg));
    if (bytes_written == -1) {
        perror("Помилка write");
        close(fd);
        return EXIT_FAILURE;
    }

    if (close(fd) == -1) {
        perror("Помилка close");
        return EXIT_FAILURE;
    }

    printf("Файл '%s' закрито.\n", filename);
    return EXIT_SUCCESS;
}
```

Рис 1.1 – Код написаної програми на мові С.

```
#!/bin/bash

if [ ! -f "./syscall_open" ]; then
    echo "Не знайдено файл syscall_open! Спочатку скомпілюйте С-програму."
    exit 1
fi

if [ ! -f "rt1.txt" ]; then
    echo "Не знайдено файл rt1.txt! Створіть або покладіть rt1.txt у поточну
    exit 1
fi

echo "Очікуємо 1 секунду перед виконанням..."
sleep 1

echo "Запуск syscall_open..."
./syscall_open

echo "Очікуємо 3 секунди після виконання..."
sleep 3

echo "Завершено."
```

Рис 1.2 – Командний файл з викликом програми на мові С.

```
~/Prysiazhniuk/bin run_syscall.sh
Не знайдено файл rt1.txt! Створіть або покладіть rt1.txt у поточну директорію.
~/Prysiazhniuk/bin touch rt1.txt
~/Prysiazhniuk/bin run_syscall.sh
Очікуємо 1 секунду перед виконанням...
Запуск syscall_open...
Файл 'rt1.txt' успішно відкрито у режимі O_WRONLY!
Файл 'rt1.txt' закрито.
Очікуємо 3 секунди після виконання...
Завершено.
```

Рис 1.3 – Виклик командного файлу.

## ЗАВДАННЯ Y 2

Тестування командного файлу.

Розробити варіанти перевірки (тестування) виконання командного файлу системного виклику Y1. Виконати перевірку варіантів тестування за наявності і відсутності файла та відповідних прав доступу до файлу з використанням команд оболонки.

Основні сценарії тестування:

1. Обидва файли (виконуваний і rt1.txt) існують та мають належні права доступу

2. Виконуваний файл відсутній
3. Файл rt1.txt відсутній
4. Недостатні права доступу до rt1.txt
5. Недостатні права доступу до скопійованого файла

### ЗАВДАННЯ Y 3

Створення і виконання командного файла із застосуванням команди системного виклику fork.

Створити командний файл, який дозволяє виконувати батьківський і дочірній (child) процеси. Батьківський процес виконує 11 циклів обчислення квадратів чисел за аргументом 1 з подальшим декрементом на 2 та виконання 5 циклів дочірнього процесу виведення послідовності символів алфавіту, починаючи з символу аргумента 2 Застосувати командний файл для 3 вихідних значень аргументів

```

#include <stdio.h>           // для printf
#include <stdlib.h>          // для atoi, exit
#include <unistd.h>          // для fork, getpid, getppid
#include <sys/wait.h>        // для wait

int main(int argc, char *argv[])
{
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <start_number> <start_char>\n", argv[0]);
        return EXIT_FAILURE;
    }

    int startNum = atoi(argv[1]);

    char startChar = argv[2][0]; // беремо перший символ другого аргументу

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork() error");
        return EXIT_FAILURE;
    }
    else if (pid == 0) {
        for (int i = 0; i < 5; i++) {
            printf("Child output (pid=%d): %c\n", getpid(), startChar + i);
        }
    }
    else {
        int current = startNum;
        for (int i = 0; i < 11; i++) {
            printf("Parent output (pid=%d): %d^2 = %d\n",
                getpid(), current, current * current);
            current -= 2;
        }
    }

    wait(NULL);

    return EXIT_SUCCESS;
}

```

Рис 3.1 – Код програми на мові С.

```
#!/bin/bash
SRC_FILE="fork_example.c"
EXE_FILE="fork_example"

echo "====="
echo "  Компіляція $SRC_FILE -> $EXE_FILE"
echo "====="
gcc "$SRC_FILE" -o "$EXE_FILE" 2>/dev/null
if [ $? -ne 0 ]; then
    echo "Помилка: Не вдалося скомпілювати $SRC_FILE!"
    exit 1
fi

echo
echo "====="
echo "  Запуск 1: аргументи (9, 'c')"
echo "====="
./"$EXE_FILE" 9 c

echo
echo "====="
echo "  Запуск 2: аргументи (7, 'z')"
echo "====="
./"$EXE_FILE" 7 z

echo
echo "====="
echo "  Запуск 3: аргументи (5, 'a')"
echo "====="
./"$EXE_FILE" 5 a

echo
echo "====="
echo "  Завершено."
echo "=====
```

Рис 3.2 – Комадний файл з викликом за різними аргументами.



```

~/Prysiashniuk/bin nvim run_fork_example.sh
~/Prysiashniuk/bin chmod +x run_fork_example.sh
~/Prysiashniuk/bin run_fork_example.sh
=====
Компіляція fork_example.c -> fork_example
=====

Запуск 1: аргументи (9, 'c')
=====
Parent output (pid=450312): 9^2 = 81
Parent output (pid=450312): 7^2 = 49
Parent output (pid=450312): 5^2 = 25
Parent output (pid=450312): 3^2 = 9
Parent output (pid=450312): 1^2 = 1
Parent output (pid=450312): -1^2 = 1
Parent output (pid=450312): -3^2 = 9
Parent output (pid=450312): -5^2 = 25
Parent output (pid=450312): -7^2 = 49
Parent output (pid=450312): -9^2 = 81
Parent output (pid=450312): -11^2 = 121
Child output (pid=450313): c
Child output (pid=450313): d
Child output (pid=450313): e
Child output (pid=450313): f
Child output (pid=450313): g

Запуск 2: аргументи (7, 'z')
=====
Parent output (pid=450314): 7^2 = 49
Parent output (pid=450314): 5^2 = 25
Parent output (pid=450314): 3^2 = 9
Parent output (pid=450314): 1^2 = 1
Parent output (pid=450314): -1^2 = 1
Parent output (pid=450314): -3^2 = 9
Parent output (pid=450314): -5^2 = 25
Parent output (pid=450314): -7^2 = 49
Parent output (pid=450314): -9^2 = 81
Parent output (pid=450314): -11^2 = 121
Parent output (pid=450314): -13^2 = 169

```

Рис 3.3 – Перша частина виклику командного файлу.

```
Parent output (pid=450314): -7^2 = 49
Parent output (pid=450314): -9^2 = 81
Parent output (pid=450314): -11^2 = 121
Parent output (pid=450314): -13^2 = 169
Child output (pid=450315): z
Child output (pid=450315): {
Child output (pid=450315): |
Child output (pid=450315): }
Child output (pid=450315): ~
```

```
=====
Запуск 3: аргументи (5, 'a')
=====
```

```
Parent output (pid=450316): 5^2 = 25
Parent output (pid=450316): 3^2 = 9
Parent output (pid=450316): 1^2 = 1
Parent output (pid=450316): -1^2 = 1
Parent output (pid=450316): -3^2 = 9
Parent output (pid=450316): -5^2 = 25
Parent output (pid=450316): -7^2 = 49
Parent output (pid=450316): -9^2 = 81
Parent output (pid=450316): -11^2 = 121
Parent output (pid=450316): -13^2 = 169
Parent output (pid=450316): -15^2 = 225
Child output (pid=450317): a
Child output (pid=450317): b
Child output (pid=450317): c
Child output (pid=450317): d
Child output (pid=450317): e
```

```
=====
Завершено.
=====
~/Prysiazhniuk/bin
```

Рис 3.4 – Друга частина виклику командного файлу.

#### ЗАВДАННЯ У 4

Створення і виконання командного файлу на основі команди kill

Командний файл дозволяє виконувати команду kill з номером сигналу, який визначено аргументом 4. Створити процес папо у фоновому режимі і переконатись у цьому за допомогою команди ps. Вилучити цей процес з використанням створеного командного файлу і переконатись у цьому за допомогою команди ps.



```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: $0 <signal_number>"
    echo "Example: $0 9"
    exit 1
fi

SIGNAL=$1

echo "Запуск процесу nano у фоновому режимі..."
nano &      # Запускаємо nano у фоновому режимі
NANO_PID=$! # Отримуємо PID процесу nano
echo "Процес nano запущено з PID=$NANO_PID"

echo "Список процесів nano перед завершенням:"
ps -p $NANO_PID

echo "Використовуємо kill для завершення процесу nano (сигнал $SIGNAL)..."
kill -$SIGNAL $NANO_PID

sleep 1

echo "Список процесів nano після завершення:"
ps -p $NANO_PID || echo "Процес nano з PID=$NANO_PID завершено."

echo "Готово."
```

Рис 4.1 – Командний файл.

```
~/Prysiashniuk/bin chmod +x kill_process.sh
~/Prysiashniuk/bin kill_process.sh
Usage: /home/ubuntu/Prysiashniuk/bin/kill_process.sh <signal_number>
Example: /home/ubuntu/Prysiashniuk/bin/kill_process.sh 9
~/Prysiashniuk/bin kill_process.sh 9
Запуск процесу nano у фоновому режимі...
Процес nano запущено з PID=451593
Список процесів nano перед завершенням:
Standard input is not a terminal
  PID TTY          TIME CMD
Використовуємо kill для завершення процесу nano (сигнал 9)...
/home/ubuntu/Prysiashniuk/bin/kill_process.sh: line 20: kill: (451593) - No such p
rocess
Список процесів nano після завершення:
  PID TTY          TIME CMD
Процес nano з PID=451593 завершено.
Готово.
~/Prysiashniuk/bin
```

Рис 4.2 – Виконання командного файлу.

## ЗАВДАННЯ Y 5

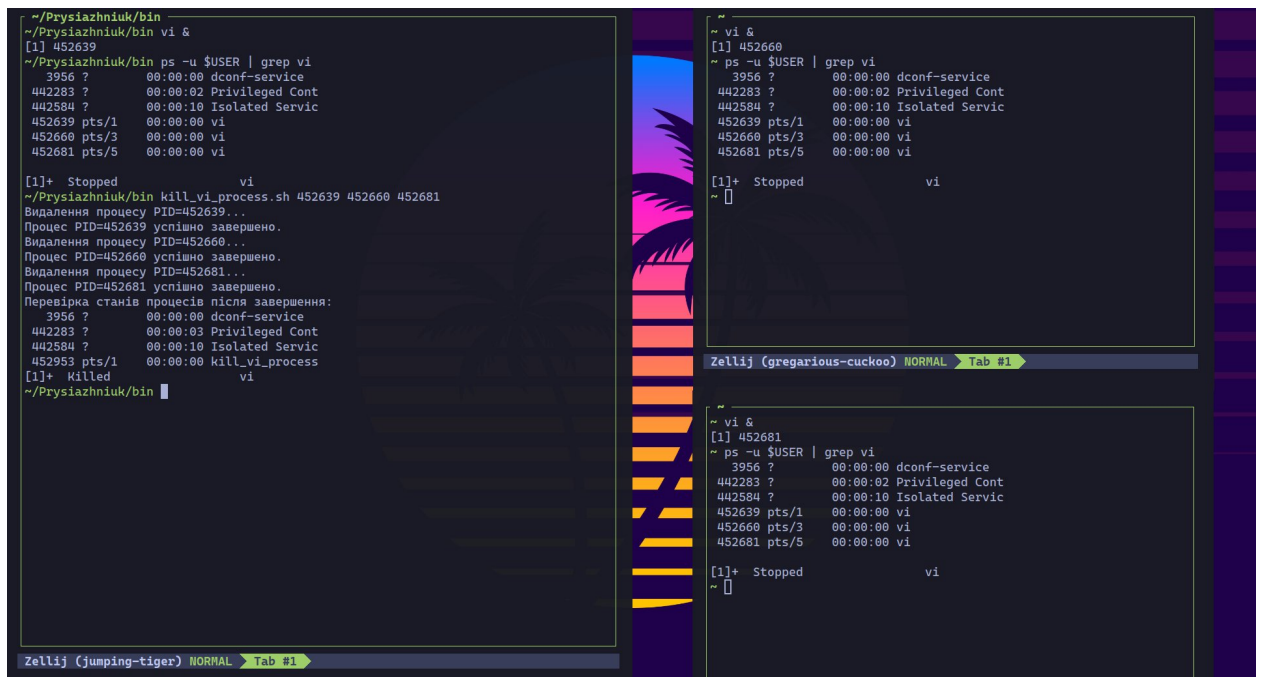
Застосування командного файлу для створених терміналів

Створити 3 термінали за допомогою команди Ctrl+Alt+T. Запустити у цих терміналах процеси vi у фоновому режимі. Вивести за ps стани процесів кожного терміналу з іншого терміналу. Виконати знищення створених процесів за створеним командним файлом Y4 з іншого терміналу. Вивести за ps стани процесів кожного терміналу з іншого терміналу.

Переробимо командний файл для роботи з vi.

```
1  #!/bin/bash
2
3  if [ $# -eq 0 ]; then
4      echo "Usage: $0 <pid1> <pid2> <pid3>"
5      echo "Example: $0 12345 12346 12347"
6      exit 1
7  fi
8
9  # Знищуємо передані PID
10 for PID in "$@"; do
11     echo "Видалення процесу PID=$PID..."
12     kill -9 $PID 2>/dev/null
13
14     if [ $? -eq 0 ]; then
15         echo "Процес PID=$PID успішно завершено."
16     else
17         echo "Помилка: не вдалося завершити процес PID=$PID."
18     fi
19 done
20
21 echo "Перевірка станів процесів після завершення:"
22 ps -u $USER | grep vi || echo "Усі процеси vi завершено."
```

Рис 5.1 – Командний файл який приймає 3 аргументи.



```
~/Prysiashniuk/bin
~/Prysiashniuk/bin vi &
[1] 452639
~/Prysiashniuk/bin ps -u $USER | grep vi
 3956 ?      00:00:00 dconf-service
442283 ?      00:00:02 Privileged Cont
442584 ?      00:00:10 Isolated Servic
452639 pts/1    00:00:00 vi
452660 pts/3    00:00:00 vi
452681 pts/5    00:00:00 vi
[1]+  Stopped                  vi
~/Prysiashniuk/bin kill vi_process.sh 452639 452660 452681
Видалення процесу PID=452639...
Процес PID=452639 успішно завершено.
Видалення процесу PID=452660...
Процес PID=452660 успішно завершено.
Видалення процесу PID=452681...
Процес PID=452681 успішно завершено.
Перевірка станів процесів після завершення:
 3956 ?      00:00:00 dconf-service
442283 ?      00:00:03 Privileged Cont
442584 ?      00:00:10 Isolated Servic
452953 pts/1    00:00:00 kill_vi_process
[1]+  Killed                   vi
~/Prysiashniuk/bin
```

```
~ vi &
[1] 452660
~ ps -u $USER | grep vi
 3956 ?      00:00:00 dconf-service
442283 ?      00:00:02 Privileged Cont
442584 ?      00:00:10 Isolated Servic
452639 pts/1    00:00:00 vi
452660 pts/3    00:00:00 vi
452681 pts/5    00:00:00 vi
[1]+  Stopped                  vi
~
```

```
~ vi &
[1] 452681
~ ps -u $USER | grep vi
 3956 ?      00:00:00 dconf-service
442283 ?      00:00:02 Privileged Cont
442584 ?      00:00:10 Isolated Servic
452639 pts/1    00:00:00 vi
452660 pts/3    00:00:00 vi
452681 pts/5    00:00:00 vi
[1]+  Stopped                  vi
~
```

Рис 5.2 – Запуск vi у фоновому режимі, перевірка ID vi, та запуск командного файлу для всіх інстанцій vi.

#### Висновок:

У цій лабораторній роботі було розглянуто та реалізовано різні аспекти роботи з командними файлами і системними викликами в операційній системі. Зокрема, було створено програми на мові C, які демонструють використання системних викликів `open` і `fork`, а також реалізовано тестування цих програм за допомогою командних файлів. Важливим елементом роботи стало використання системної команди `kill` для завершення процесів та управління запущеними процесами, що дало змогу на практиці ознайомитися з контролем за станами процесів через команду `ps`. Завдання Y5 показало, як можна керувати декількома терміналами та процесами у фоновому режимі, що є важливим навиком для роботи з багатозадачними середовищами.