МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи №4
на тему: «Програмне створення та керування процесами в операційній системі LINUX»
з дисципліни: «Операційні системи»

Лектор:
ст. викладач кафедри ПЗ
Грицай О. Д.

Виконав:
ст. гр. ПЗ-22
Чаус О. М.

Прийняла:
ст. викладач кафедри ПЗ
Грицай О. Д.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Програмне створення та керування процесами в операційній системі LINUX.

Мета роботи: Ознайомитися з багатопоточністю в ОС Linux. Навчитися працювати з процесами, в ОС Linux.

Теоретичні відомості

Процеси в ОС Linux створюються з допомогою системного виклику fork(). Цей виклик створює точну копію батьківського процесу. Після виконання fork() усі ресурси дочірнього процесу - це копія ресурсів батька. Копіювати процес з усіма виділеними сторінками пам'яті - справа дорога, тому в ядрі Linux використовується технологія Copy-On-Write. Всі сторінки пам'яті батька позначаються як read-only і стають доступні і батькові, і дитині. Як тільки один з процесів змінює дані на певній сторінці, ця сторінка не змінюється, а копіюється і змінюється вже копія. Оригінал при цьому «відв'язується» від даного процесу. Як тільки read-only оригінал залишається «прив'язаним» до одного процесу, сторінці знову призначається статус read-write.

Результат виклику fork() повертається і в батьківський і в дочірній процеси, які починають виконувати однакові інструкції.

Перероджження в іншу програму

Інколи існує необхідність, щоб дочірній процес виконував певну задачу, а батьківський процес лише делегував певні завдання. Якщо потрібно запустити іншу програму, то необхідно вдатися до системного виклику execve():

```
int execve (const char * filename, char * const argv [], char * const envp []);
```

або бібліотечним викликам execl (), execlp (), execle (), execv (), execvp (), execvpe ():

```
int execl (const char * path, const char * arg, ... / * (char *) NULL * /);
int execlp (const char * file, const char * arg, ... / * (char *) NULL * /);
int execle (const char * path, const char * arg, .../ *, (char *) NULL, char * const envp
[] * /);
int execv (const char * path, char * const argv []);
int execvp (const char * file, char * const argv []);
int execvpe (const char * file, char * const argv [], char * const envp []);
```

Всі з перерахованих викликів виконують програму, шлях до якої зазначений в першому аргументі. У разі успіху управління передається завантаженій програмі і в вихідну вже не повертається. При цьому у завантаженій програмі залишаються всі поля структури процесу, крім файлових дескрипторів,
позначених як O_CLOEXEC, вони закриються.

Отримати інформацію про пріоритет або встановити пріоритет процесу можна з допомогою

```
#include <sys/resource.h>
int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int prio);
```

Завдання для виконання лабораторної роботи

1. Виконати в окремому процесі табулювання функцій (Можна замінити алгоритмом заданим у лабораторній роботі No3).
2. Реалізувати табулювання функцій у 2-ох, 4-ох, 8-ох процесах. Виміряти час роботи процесів. Порівняти результати роботи в одному і в багатьох процесах.
3. Реалізувати можливість зміни пріоритету виконання процесу.

4. Реалізувати можливість зупинки і відновлення роботи процесу
5. Реалізувати можливість вбиття процесу.
6. Порівняти результати виконання програми під ОС Windows та Linux.
7. Результати роботи відобразити у звіті.

Хід роботи

1. Здійснив розв'язок індивідуальної задачі та створив окремий процес, який його виконав(зображення 1-2).

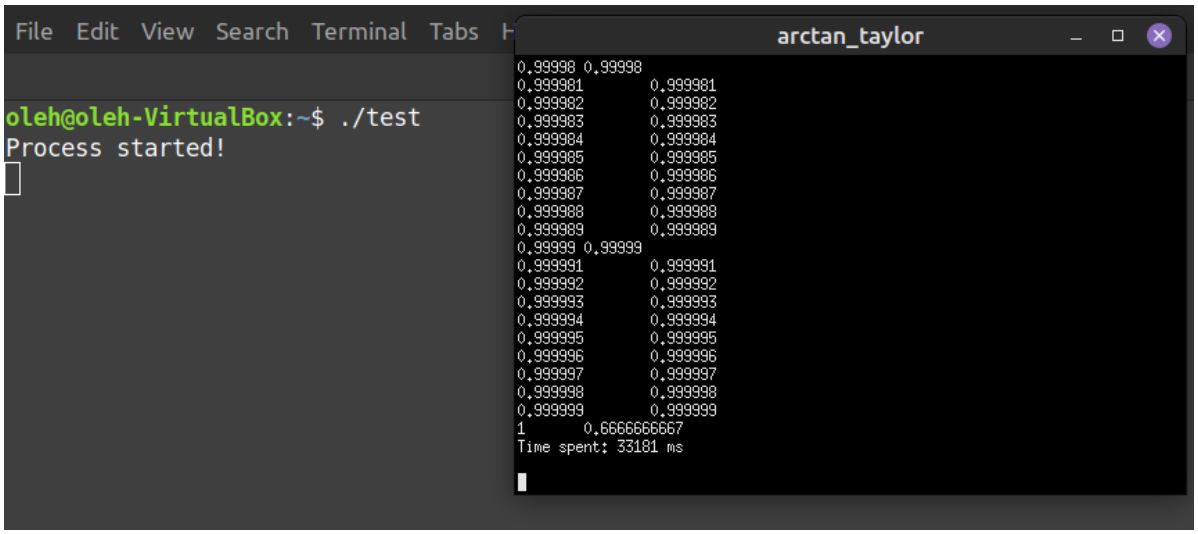
Код розв'язку індивідуальної задачі:

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <chrono>

#define EPS 0.0001

struct Arctan {
	double x;
	double arctanx;
};

int main(int argc, const char** argv) {
	auto start_time = std::chrono::system_clock::now();
	std::cout << std::setprecision(10);
	double A = atof(argv[1]);
	double B = atof(argv[2]);
	int steps = atoi(argv[3]);
	std::vector<Arctan> info(steps + 1);
	double step = (B - A) / steps;
	for (std::size_t i = 0; i < info.size(); i++) {
		info[i].x = A + (step * i);
		double sum = 0;
		double iter_value = EPS + 1;
		int k = 0;
		while (std::abs(iter_value) > EPS) {
			iter_value = (pow(-1, k) * pow(info[i].x, 1 + 2 * k)) / (1 + 2 * k);
			k++;
			sum += iter_value;
			if (k > 10) break;
		}
		info[i].arctanx = sum;
		std::cout << info[i].x << "\t" << info[i].arctanx << "\t" << "\n";
	}
	auto exit_time = std::chrono::system_clock::now();
	std::cout << "Time spent: " <<
std::chrono::duration_cast<std::chrono::milliseconds>(exit_time - start_time).count() << " ms\n";
	putchar('\n');
	getchar();
}
```

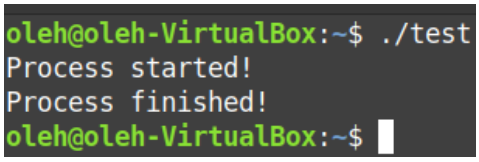
Код програми, що створює процес:

```
#include <unistd.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <iostream>

int main() {
    pid_t pid = fork();
    if (pid < 0)
        std::cout << "error creating process.\n";
    else if (pid == 0) {
          execl("/usr/bin/xterm", "xterm", "-e", "./arctan_taylor", "0", "1", "1000000",
(char*)NULL);
    }
    else {
```

```
        std::cout << "Process started!\n" << std::flush;

        wait(NULL);
        std::cout << "Process finished!\n";
    }
    return 0;
}
```

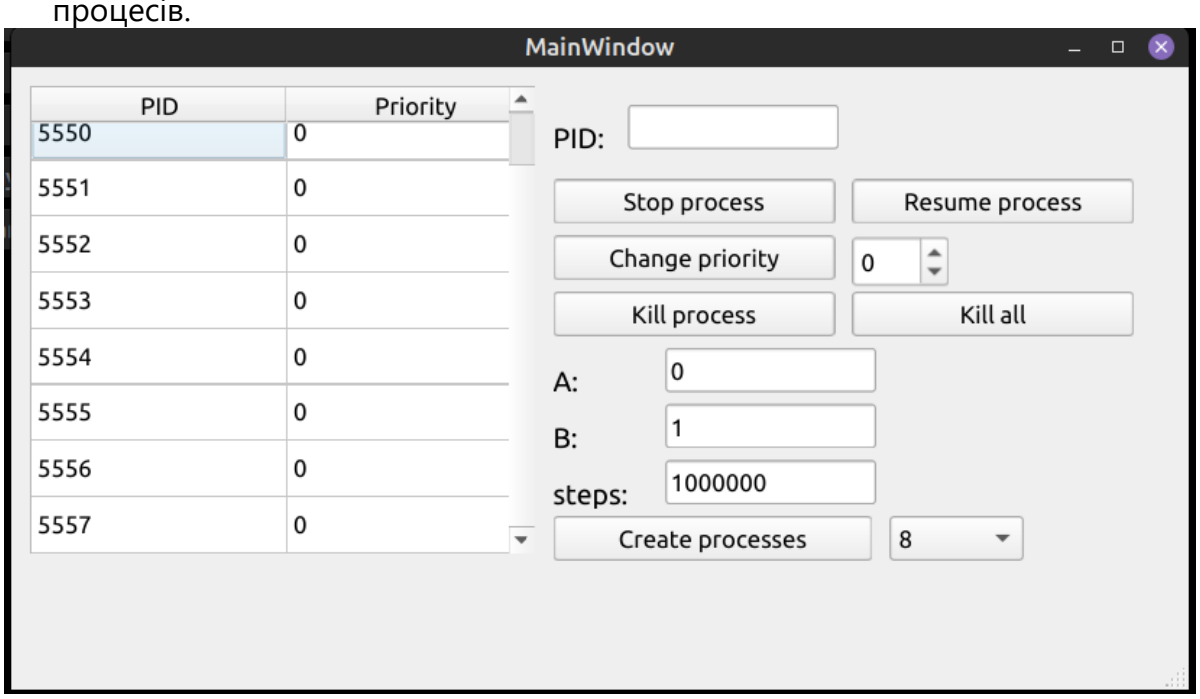


Зображення 1



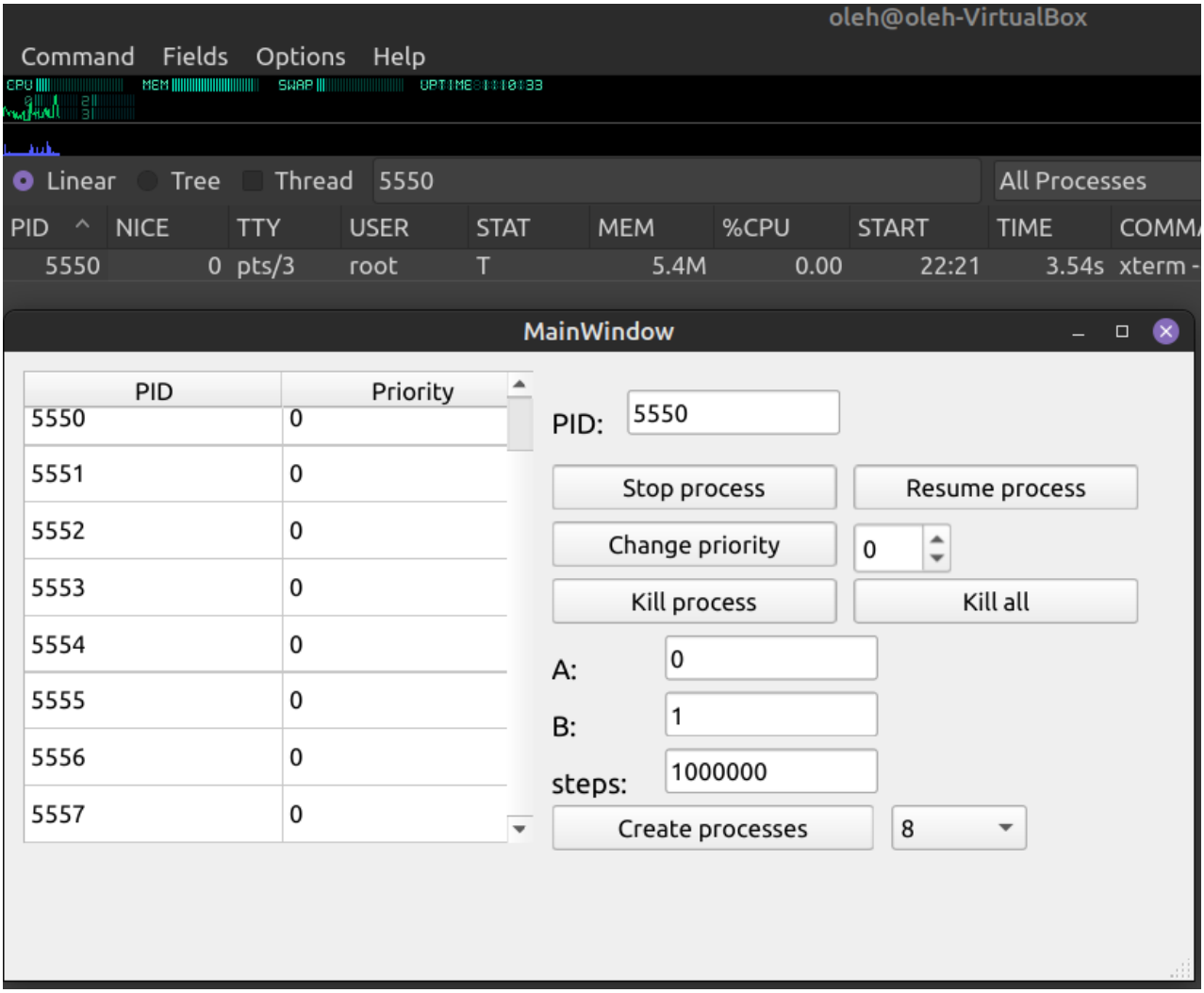
Зображення 2

2. Реалізував розв'язок задачі у 2-ох, 4-ох, 8-ох процесах. Виміряв час роботи процесів.

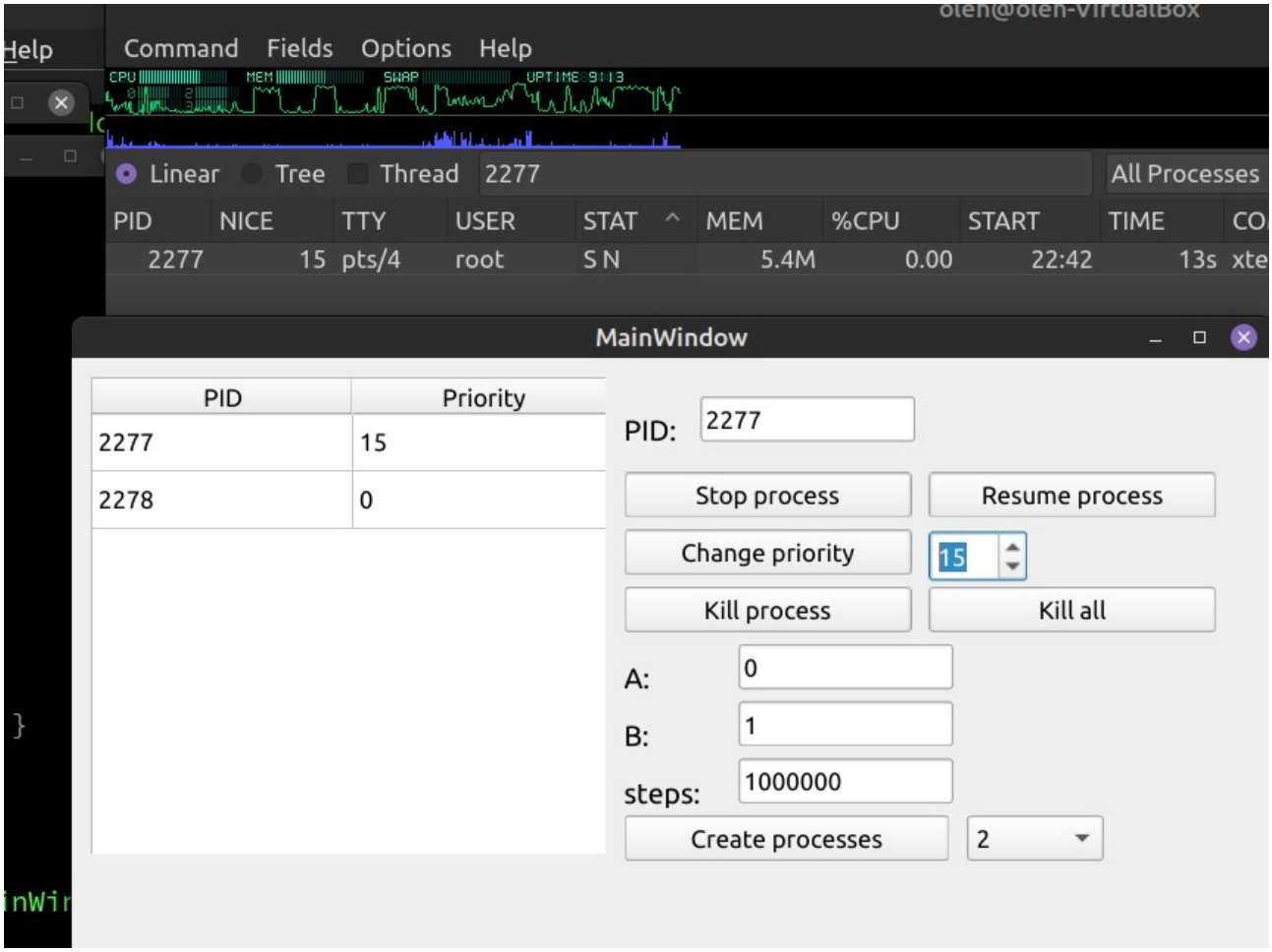


Інтерфейс програми

3. Для кожного процесу реалізував можливість його запуску, зупинення, та завершення. На зображенні показано зазначений функціонал.



4. Реалізував можливість зміни пріоритету виконання процесу.



Висновок: під час виконання лабораторної роботи ознайомився з процесами в системі Linux, навчився їх створювати, змінювати пріоритет і стан, завершувати. Провівши дослід із засікання часи роботи процесу розпаралеленого на 2, 4 та 8 процесів, дійшов до висновку, що найшвидше він виконався при розпаралеленні на 8 процесів(21 сек). Хоча розпаралелення краще реалізовано на Linux, все-ж таки процеси виконались швидше на Windows, адже Linux розташований у віртуальній машині, тоді як Windows може використовувати всі ресурси комп'ютера.