**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №6

На тему: «Багатопоточність в операційній системі LINUX. Створення, керування та синхронізація потоків.»

З дисципліни: «Операційні системи»

**Лектор** : ст.викл каф.ПЗ

Грицай О.Д.

**Виконала:** ст.гр.ПЗ-23

Кохман О.В.

**Прийняла:** ст.викл каф.ПЗ

Грицай О.Д.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 р.

\_ \_ \_ \_ \_ \_ \_ \_ .

Львів – 2022

**Тема:** Багатопоточність в операційній системі LINUX. Створення, керування та синхронізація потоків.

**Мета:** Навчитися створювати потоки та керувати ними в операційній системі Linux. Ознайомитися з методами синхронізації потоків в операційній системі Linux. Навчитися реалізовувати багатопоточний алгоритм розв’язку задачі з використанням синхронізації в операційній системі Linux.

**Теоретичні відомості**

**Потоки в операційній системі Linux.** Поняття потоку, як одиниці виконання процесу в операційній системі (ОС) Linux, аналогічне як і у ОС Windows. Кожен процес можна представити як контейнер ресурсів і послідовність інструкцій виконання. Таким чином, можна сказати, що кожен процес містить хоча б один потік виконання, якому надані лічильник інструкцій, регістри і стек. Крім того, у кожному процесі можна створити додаткові гілки виконання – потоки, тоді такий процес називають багатопоточним.

Різниця між потоками в ОС Linux і в ОС Windows полягає у їх представленні в ядрі операційних систем. В ОС Windows потоки виконання у режимі користувача зіставляються з потоками у режим ядра. У перших версіях ядра Linux потоки користувача зіставлялись з процесами у ядрі. Створення потоку відбувалось з допомогою системного виклику clone(). Виклик clone(), як і fork() дозволяє створювати новий процес, але з певними відмінностями :

- одразу повне копіювання батьківського процесу

- створення власного стеку

- необхідно вказати спеціальний набір прапорців успадкування для того, щоб визначити, як будуть розподілятися ресурси (адресний простір, відкритті файли, обробники сигналів) між батьківським і дочірнім процесом.

Таким чином, створювався новий потік у режимі користувача, який відобрається у процес ядра. Відображення здійснюється за моделлю 1:1. Оскільки керуючий блок процесу в Linux представлений в ядрі структурою *task\_struct*, то і представлення потоку здійснювалось через *task\_struct.* Фактично у ядрі потоки і процеси не розрізнялися. Але системний виклик clone() не підтримувався стандартом POSIX і тому розроблялись бібліотеки потоків, що дозволяли працювати з потоками, використовуючи clone() з різними атрибутами виконання.

У сучасних версіях Linux підтримуються спеціальні об'єкти ядра - потоки ядра, побудованні на зміненному і розширеному системному виклику clone(). Підтримка потоків здійснюється через POSIX-сумісну бібліотеку NPTL (Native POSIX Threads Library). Типи даних і функції, що застосовуються до потоків POSIX, мають префікс pthread\_ і доступні через підключення phread.h.

**Індивідуальне завдання**

1. Реалізувати заданий (згідно варіанту) алгоритм в окремому потоці.

2. Виконати розпаралелення заданого алгоритму на 2, 4, 8, 16 потоків.

3. Реалізувати можливість зміни/встановлення пріоритету потоку (для планування потоків) або встановлення відповідності виконання на ядрі.

4. Реалізувати можливість зробити потік від’єднаним.

5. Реалізувати можливість відміни потоку.

6. Реалізувати синхронізацію потоків за допомогою вказаних методів (згідно варіанту)

7. Порівняти час виконання задачі відповідно до кількості потоків і методу синхронізації (чи без синхронізації).

8. Результати виконання роботи оформити у звіт

9 варіант : Вивід слів з файлу, що розпочинаються на задану літеру (кількість рядків у файлі > 1000, текст довільна наукова стаття) (Синхронізація: спінлок, умовні змінні).

**Код програми**

Назва файлу: main.cpp

#include <pthread.h>

#include <iostream>

#include <fstream>

#include <chrono>

#include <unistd.h>

#include <cstring>

#include <sched.h>

using namespace std;

using namespace std::chrono;

int countThread;

pthread\_spinlock\_t \*spinLock;

pthread\_mutex\_t mutexLock = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t mutexCond = PTHREAD\_COND\_INITIALIZER;

pthread\_t\* threads;

int readyCondition = 0;

void \* value\_ptr;

typedef struct {

string\* array;

int size;

} th\_param; //структура передачі параметрів

void asyncJob(th\_param params);

void spinLockJob(th\_param params);

void conditionalVariablesJob(th\_param params);

void setPriorityJob(th\_param params);

void otherThings(th\_param params);

void search(string\* array, int start, int end, char value) {

for (int i = start; i < end; i++) {

if (array[i][0] == value || array[i][0] == toupper(value) || array[i][0] == tolower(value)) {

cout << array[i] << " ";

}

}

cout << "END SEARCH\n";

}

void\* th\_func(void\* args) {

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

return NULL;

}

void\* sleep\_th\_func(void\* args) {

sleep(1000000);

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

return NULL;

}

void\* cond\_th\_func(void\* args) {

pthread\_mutex\_lock(&mutexLock);

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

readyCondition = 1;

pthread\_cond\_signal(&mutexCond);

pthread\_mutex\_unlock(&mutexLock);

return NULL;

}

void\* spin\_th\_func(void\* args) {

pthread\_spin\_lock(spinLock);

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

pthread\_spin\_unlock(spinLock);

return NULL;

}

int main() {

fstream file;

string word, filename;

filename = "myfile.txt";

file.open(filename.c\_str());

int size = 0;

while (file >> word) {

size++;

}

file.close();

string\* array = new string[size];

file.open(filename.c\_str());

int i = 0;

while (file >> word) {

array[i] = word;

i++;

}

th\_param params;

params.size = size;

params.array = array;

threads = new pthread\_t[countThread];

int choice = 0;

while (true) {

cout << "Enter number:\n[1] - ASYNCHRONIZATION\n[2] - SPINLOCK\n[3] - CONDITIONAL VARIABLES\n[4] - MEASURE TIME\n[5] - OTHER\n[6] - EXIT\n";

cin >> choice;

if (choice == 1) {

cout << "\nEnter number of threads to be created:" << endl;

cin >> countThread;

asyncJob(params);

}

else if (choice == 2) {

cout << "\nEnter number of threads to be created:" << endl;

cin >> countThread;

spinLockJob(params);

}

else if (choice == 3) {

cout << "\nEnter number of threads to be created:" << endl;

cin >> countThread;

conditionalVariablesJob(params);

}

else if (choice == 4) {

int countThreadArray[] = { 1,2,4 };

for (int i = 0; i < 3; i++) {

countThread = countThreadArray[i];

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "\t\t\t\t\t\tNUMBER OF THREADS - " << countThread << endl;

auto start = high\_resolution\_clock::now();

asyncJob(params);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by ASYNCHRONIZATION: " << duration.count() << " microseconds" << endl;

auto start2 = high\_resolution\_clock::now();

spinLockJob(params);

auto stop2 = high\_resolution\_clock::now();

auto duration2 = duration\_cast<microseconds>(stop2 - start2);

cout << "Time taken by SPIN LOCK: " << duration2.count() << " microseconds" << endl;

auto start3 = high\_resolution\_clock::now();

conditionalVariablesJob(params);

auto stop3 = high\_resolution\_clock::now();

auto duration3 = duration\_cast<microseconds>(stop3 - start3);

cout << "Time taken by CONDITIONAL VARIABLES: " << duration3.count() << " microseconds" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

else if (choice == 5) {

otherThings(params);

}

else if(choice == 6) {

break;

}

else {

continue;

}

}

return 0;

}

void setPriorityJob(th\_param params) {

cout << "Enter thread priority: " << endl;

int priority = 10;

int dPolicy = 0;

cin >> priority;

cout << "Enter the number of sched policy: \n1 - FIFO\n2 - RR\n3 - BATCH\n0 - OTHER\n";

std::cin >> dPolicy;

pthread\_t thread;

pthread\_attr\_t thAttr;

pthread\_attr\_init(&thAttr);

int inh = PTHREAD\_EXPLICIT\_SCHED;

pthread\_attr\_setinheritsched(&thAttr, inh);

// int policy = SCHED\_FIFO;

struct sched\_param p;

p.sched\_priority = priority;

pthread\_attr\_setschedpolicy(&thAttr, dPolicy);

pthread\_attr\_getschedpolicy(&thAttr, &dPolicy);

pthread\_attr\_setschedparam(&thAttr, &p);

pthread\_attr\_getschedpolicy(&thAttr, &dPolicy);

pthread\_attr\_getschedparam(&thAttr, &p);

pthread\_create(&thread, &thAttr, &sleep\_th\_func, &params);

pthread\_join(thread, NULL);

pthread\_attr\_destroy(&thAttr);

cout << "policy: " << dPolicy << endl;

cout << "priority " << priority << endl;

}

void conditionalVariablesJob(th\_param params) {

for (int i = 0; i < countThread; ++i) {

pthread\_create(&threads[i], NULL, &cond\_th\_func, &params);

while (readyCondition == 0) {

pthread\_cond\_wait(&mutexCond, &mutexLock);

}

pthread\_mutex\_unlock(&mutexLock);

readyCondition = 0;

}

}

void spinLockJob(th\_param params) {

spinLock = (pthread\_spinlock\_t\*)malloc(sizeof(pthread\_spinlock\_t));

pthread\_spin\_init(spinLock, 0);

for (int i = 0; i < countThread; ++i) {

pthread\_create(&threads[i], NULL, &spin\_th\_func, &params);

}

for (int i = 0; i < countThread; ++i) {

pthread\_join(threads[i], NULL);

}

pthread\_spin\_destroy(spinLock);

}

void\* threadFn(void\* args) {

pthread\_detach(pthread\_self());

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

pthread\_exit(NULL);

}

void\* calls(void\* args) {

search((string\*)((th\_param\*)args)->array, 0, (int)((th\_param\*)args)->size, 'z');

pthread\_cancel(pthread\_self());

pthread\_join(pthread\_self(), NULL);

pthread\_exit(NULL);

}

void asyncJob(th\_param params) {

for (int i = 0; i < countThread; ++i) {

pthread\_create(&threads[i], NULL, &th\_func, &params);

}

for (int i = 0; i < countThread; ++i) {

pthread\_join(threads[i], NULL);

}

}

void otherThings(th\_param params) {

while (true) {

int number = 0;

cout << "Enter number:\n[1] - SET PRIORITY\n[2] - DETACH\n[3] - CANCEL\n[4] - EXIT\n";

cin >> number;

if (number == 1) {

setPriorityJob(params);

}

else if (number == 2) {

pthread\_t threadId;

pthread\_create(&threadId, NULL, threadFn, &params);

cout << "thread was detached" << endl;

}

else if (number == 3) {

pthread\_t threadId;

pthread\_create(&threadId, NULL, calls, &params);

cout << "thread was cancelled" << endl;

}

else if (number == 4) {

break;

}

}

}

**Протокол роботи**

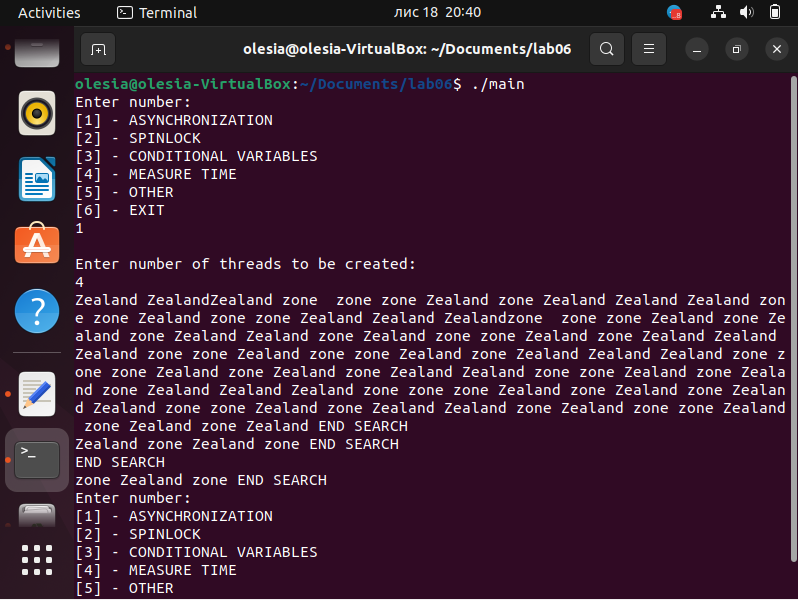
****

Рис. 1 Результат виконання програми при асинхронізації.

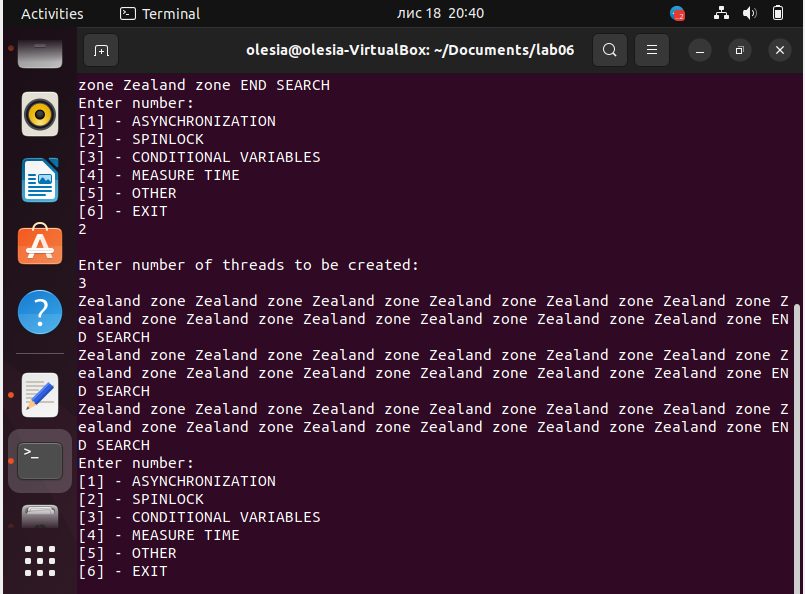


Рис. 2 Результат виконання програми при синхронізації за допомогою спінлок.

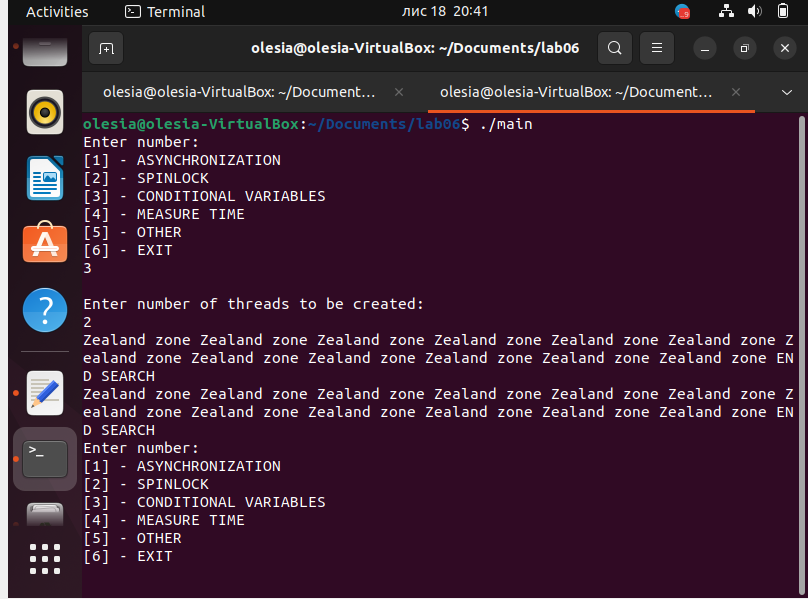


Рис. 3 Результат виконання програми при синхронізації за допомогою умовних змінних.

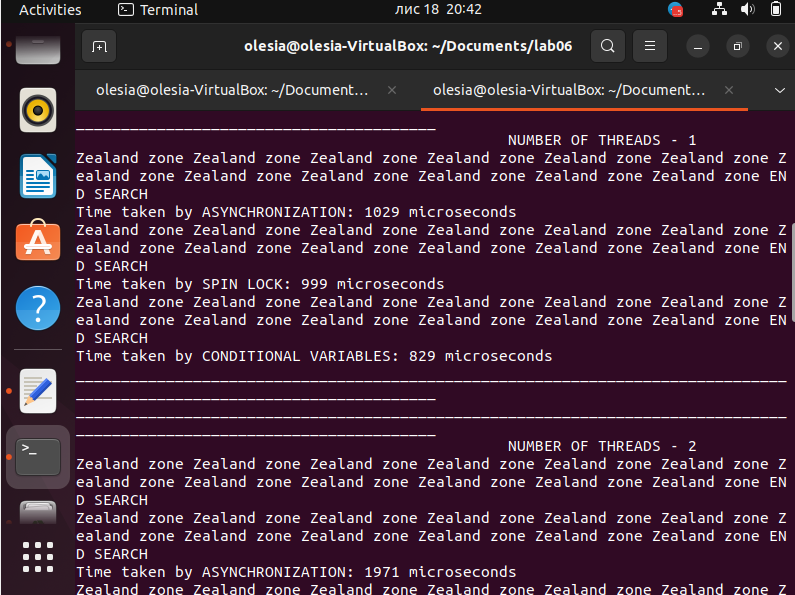


Рис. 4 Вимірювання часу виконання одного потоку при асинхронізації, синхронізації за допомогою спінлоку та синхронізації за допомогою умовних змінних.

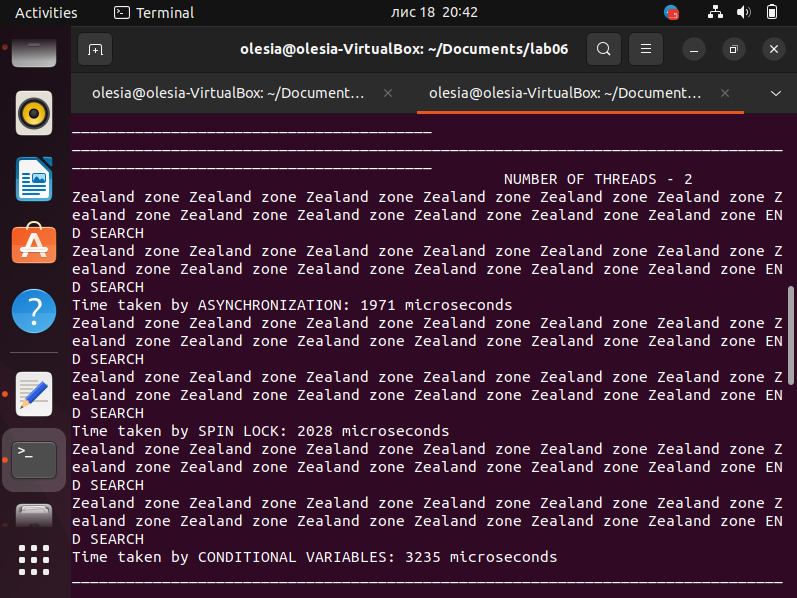


Рис. 5 Вимірювання часу виконання двох потоків при асинхронізації, синхронізації за допомогою спінлоку та синхронізації за допомогою умовних змінних.

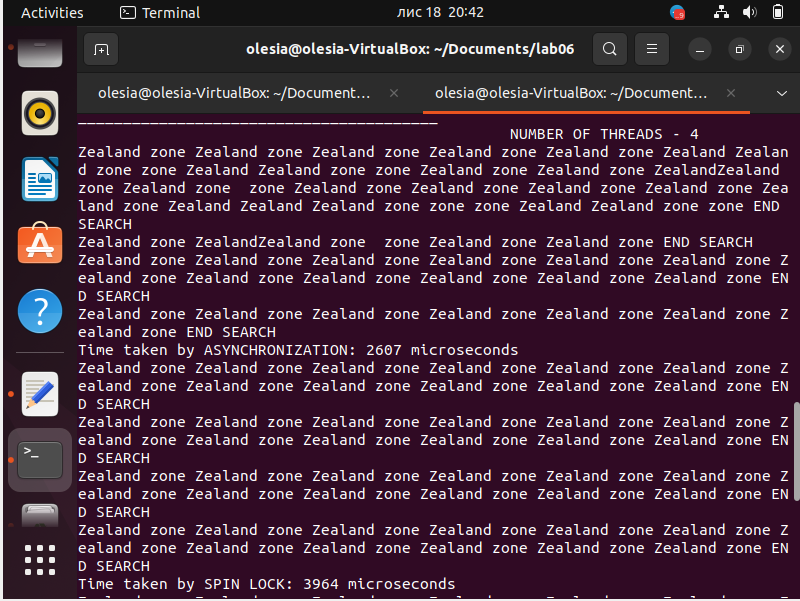


Рис. 6 Вимірювання часу виконання чотирьох потоків при асинхронізації, синхронізації за допомогою спінлоку та синхронізації за допомогою умовних змінних.



Рис. 7 Вимірювання часу виконання чотирьох потоків при асинхронізації, синхронізації за допомогою спінлоку та синхронізації за допомогою умовних змінних.

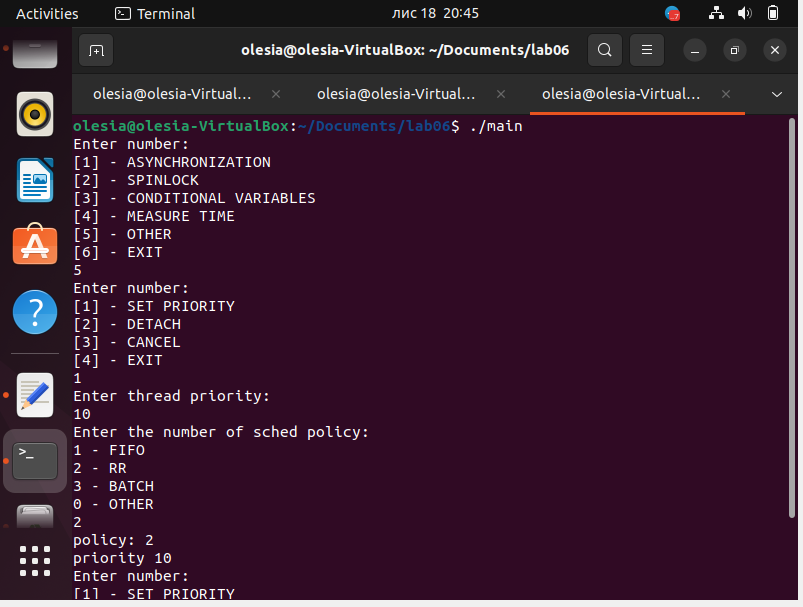


Рис. 8 Встановлення пріоритету та policy.

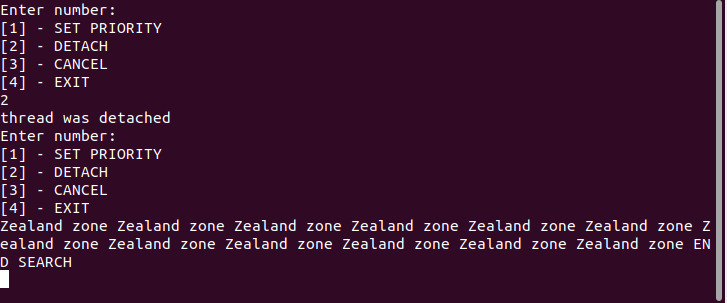


Рис. 9 Від’єднання потоку

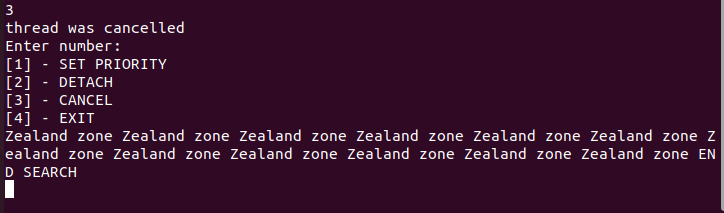


Рис. 10 Скасування потоку.

|  |  |  |  |
| --- | --- | --- | --- |
| amount/type | asynchronous | spinlock | Conditional vars |
| 1 | 1029 | 1971 | 2607 |
| 2 | 999 | 2028 | 3964 |
| 4 | 829 | 3235 | 4532 |

Таблиця 1 Порівняння часу виконання при 1,2 та 4 потоках та при асинхронізації, синхронізації за допомогою спінлок та умовних змінних.

**Висновок**

На цій лабораторній роботі я дізналась про багатопоточність в лінуксі та реалізувала програму за допомогою POSIX, де створила декілька потоків , зробила синхронізацію за допомогою умовних змінних та за допомогою спінлоку, виміряла час виконання при різній кількості потоків, , зробила таблицю, де порівняла результати .від’єднала потік, відмінила потік , зробила можливість встановлення пріоритету та policy.