МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи № 6
на тему: «Багатопоточність в операційній системі Linux. Створення, керування та синхронізація потоків»
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: Багатопоточність в операційній системі Linux. Створення, керування та синхронізація потоків

Мета роботи: 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.

Створення потоків.

Для створення потоку використовують функцію phread_create():

```
int pthread_create(pthread_t *thread,
                   pthread_attr_t *attr,
                   void*(*thread_func)(void *),
                   void *arg);
```

thread - вказівник на структуру pthead_t, дескриптор потоку;
attr - вказівник на структуру атрибутів (розмір стеку, пріоритет, ...)
thread_func - вказівник на функцію потоку
arg- дані, що передаються у функцію.
Функція потоку має особливу сигнатуру
`void*(*thread_func)(void *)`, що дозволяє передавати в неї аргументи будь-якого типу і повертати результат також будь-якого типу.

Очікування завершення потоків.

Аналогічно, як у Windows можна дочекатися завершення дочірнього потоку. Це можна зробити за допомогою функції pthread_join.

```
int pthread_join(pthread_t thread, void **value_ptr);
```

thread - дескриптор потоку;
value_ptr – вказівник на повернений через pthread_exit або return результат виконання потоку; може бути NULL – тоді не очікуємо результат.
`void pthread_exit(void * retval)`- функція, що дозволяє завершити виконання потоку з функції виконання потоку де retval - код повернення.
Виклик цієї функції можливий лише для приєднаних потоків (joinable). За замовчуванням всі новостворені потоки приєднанні.
кщо нема необхідності очікувати повернення результату від потоку, то потік можна позначити як відокремлений. Тоді система отримає вказівку звільнити виділенні ресурси після завершення потоку. Зробити це можна з допомогою функції pthread_detach.
`int pthread_detach(pthread_t thread);` - функція позначає потік, ідентифікований потоком, як відокремлений. Коли відокремлений потік завершується, його ресурси автоматично звільняються назад до системи без необхідності приєднання іншого потоку до завершеного потоку. Задати, що потік відокремлений можна задавши

параметри структури атрибутів потоку.

Структура атрибутів потоку.

Параметри потоку можна змінити через структуру атрибутів потоку.

```
pthread_attr_t attr;
```

Ініціалізувати структуру можна з допомогою функції

```
pthread_attr_init(&attr);
```

Призначення атрибутів відбувається у наступному порядку:

1) Створити структуру атрибутів
2) Ініціалізувати її стандартними значеннями
3) Записати необхідні значення атрибутів (функції наведенні нище)
4) Передати вказівник на структуру при створенні потоків
5) Викликати pthread_attr_destroy() – для видалення об’єкту структури атрибутів з пам’яті, але сам об’єкт не видаляється, його можна проініціалізувати знову.

Condition Variables

Умовна змінна дозволяє потокам очікувати виконання деякої умови (події), пов'язаної з даними що розділяються. Над умовними змінними визначено дві основні операції: інформування про настання події і очікування події. При виконанні операції «інформування» один з потоків, які очікують на умовній змінній, відновлює свою роботу. Умовна змінна завжди використовується спільно з м'ютексом. Перед виконанням операції «очікування» потік повинен заблокувати м'ютекс. При виконанні операції «очікування» зазначений м'ютекс автоматично розблокується. Перед поновленням що очікує потоку виконується автоматичне блокування мьютекса, що дозволяє потоку увійти в критичну секцію, після критичної секції рекомендується розблокувати м'ютекс. При подачі сигналу іншим потокам рекомендується так само функцію «сигналізації» захистити м'ютексів.

Прототипи функцій для роботи з умовними змінними містяться в файлі pthread. H. Нижче наводяться прототипи функцій разом з поясненням їх синтаксису і виконуваних ними дій.

```
pthread_cond_init(pthread_cond_t * cond,
              const pthread_condattr_t* attr);
```

ініціалізує умовну змінну cond із зазначеними атрибутами attr або з атрибутами за замовчуванням (при вказуванні 0 в якості attr).

`int pthread_cond_destroy(pthread_cond_t* cond);` - знищує умовну змінну cond.

`int pthread_cond_signal(pthread_cond_t*cond);` - інформування про настання події потоків, які очікують на умовній змінній cond.

`int pthread_cond_broadcast(pthread_cond_t* cond);` - інформування про настання події потоків, які очікують на умовній змінній cond. При цьому відновлені будуть всі очікують потоки.

```
int pthread_cond_wait(pthread_cond_t* cond,
                      pthread_mutex_t* mutex);
```

– очікування події на умовній змінній cond.

Бар’єри.

Ідея бар’єру полягає у блокуванні потоків до того часу, поки кількість потоків що очікують не досягне певного значення лічильнику бар’єру, після чого потоки продовжують свою роботу.

Оголошення бар’єру

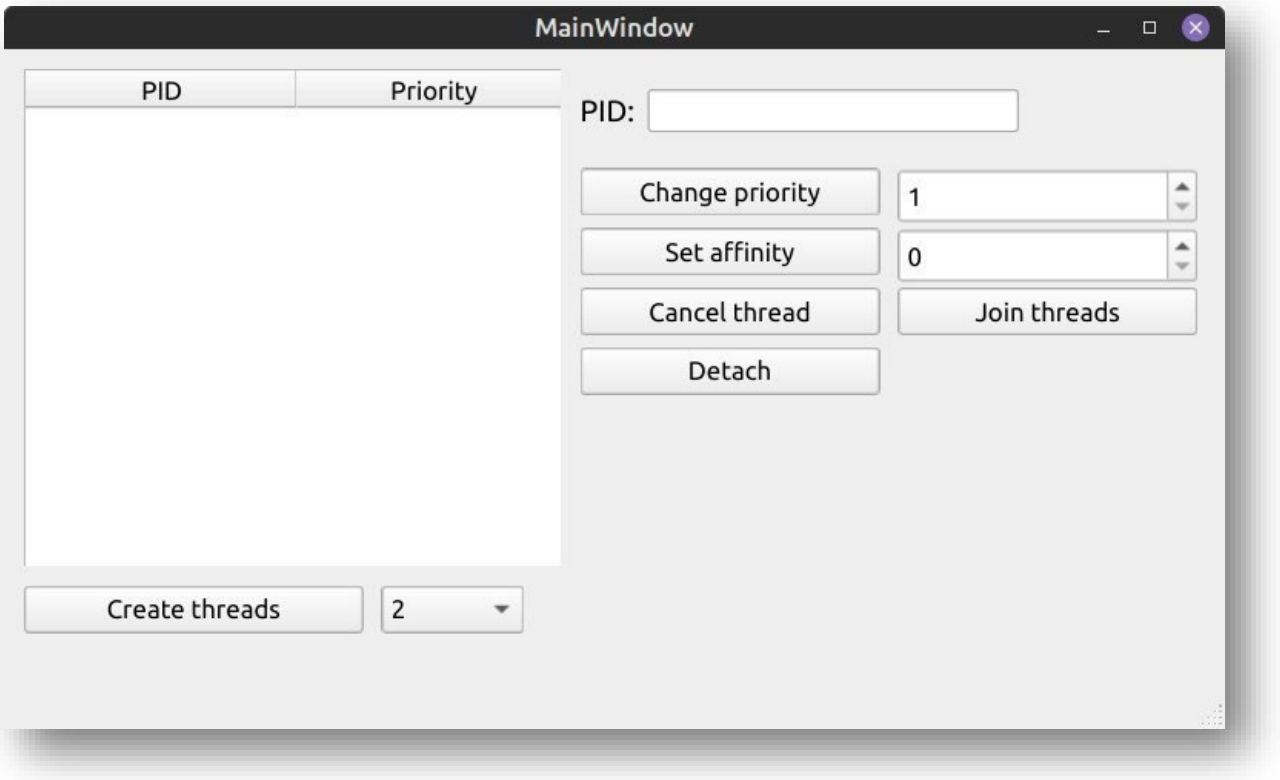
```
barrier_t barrier_cs;
```
Ініціалізація бар'єру
```
barrier_init (barrier_t barrier_cs, int n) ;
```
Очікування на бар'єрі barrier_wait(barrier_cs); В результатті виконання значення лічильника збільується на 1. Якщо він менший від n, то цей потік блокується. Якщо досягаються максимальне значення, то робота всіх потоків відновлюється. Значення лічильника для всіх потоків стає 0, а для одного з них-1. В результаті, цей потік, що отримає -1, робить підсумкову роботу.

Завдання для виконання лабораторної роботи

1. Реалізувати заданий 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
13. Вивід найменшого слова в кожному рядку. (кількість рядків у файлі > 1000, текст довільна наукова стаття) Синхронізація: умовні змінні, бар'єри.

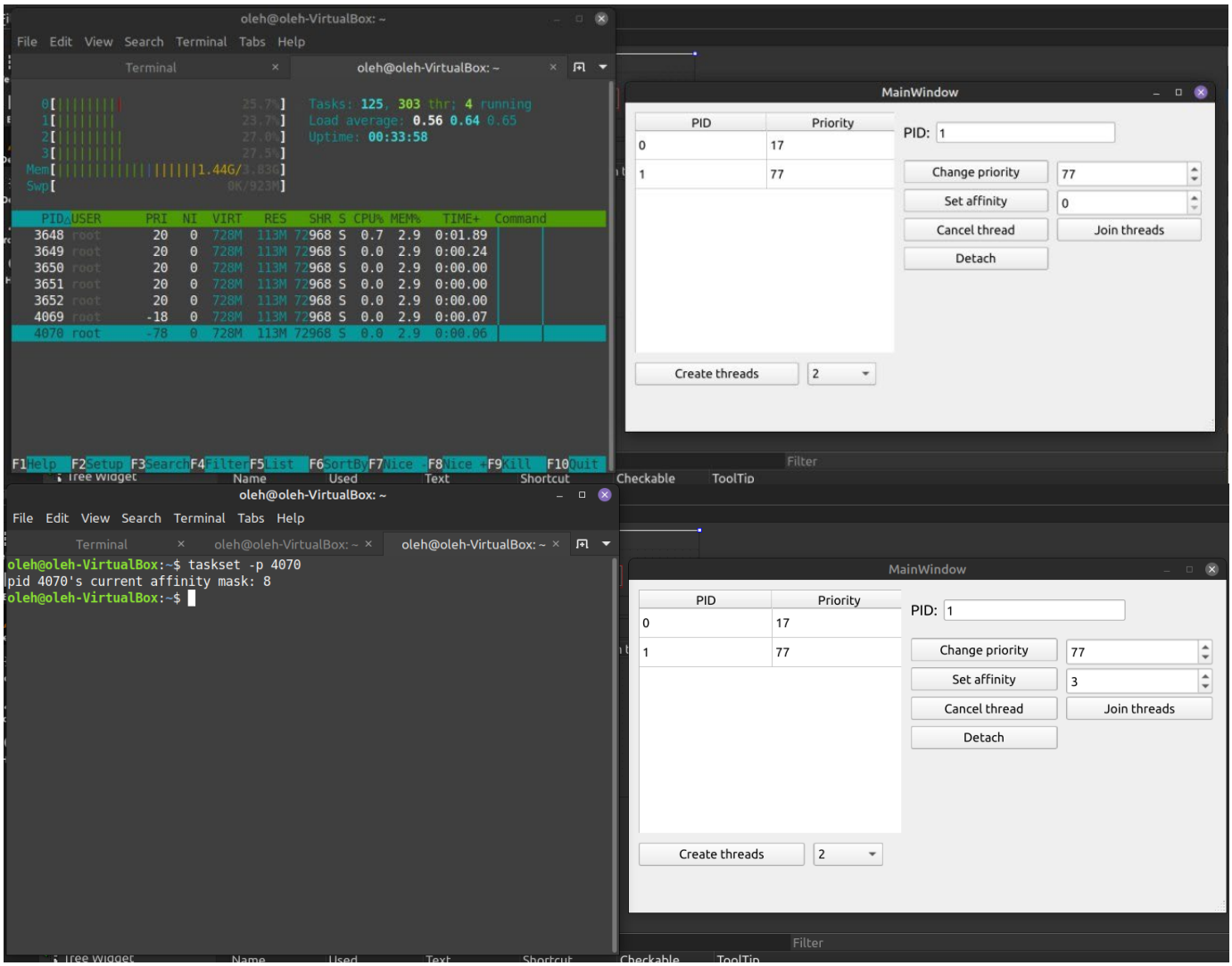
Хід роботи

1. Реалізував розв'язок задачі у 1-му, 2-ох, 4-ох, 8-ох та 16-ох потоках із вказаною синхронізацією. Виміряв час роботи потоків.

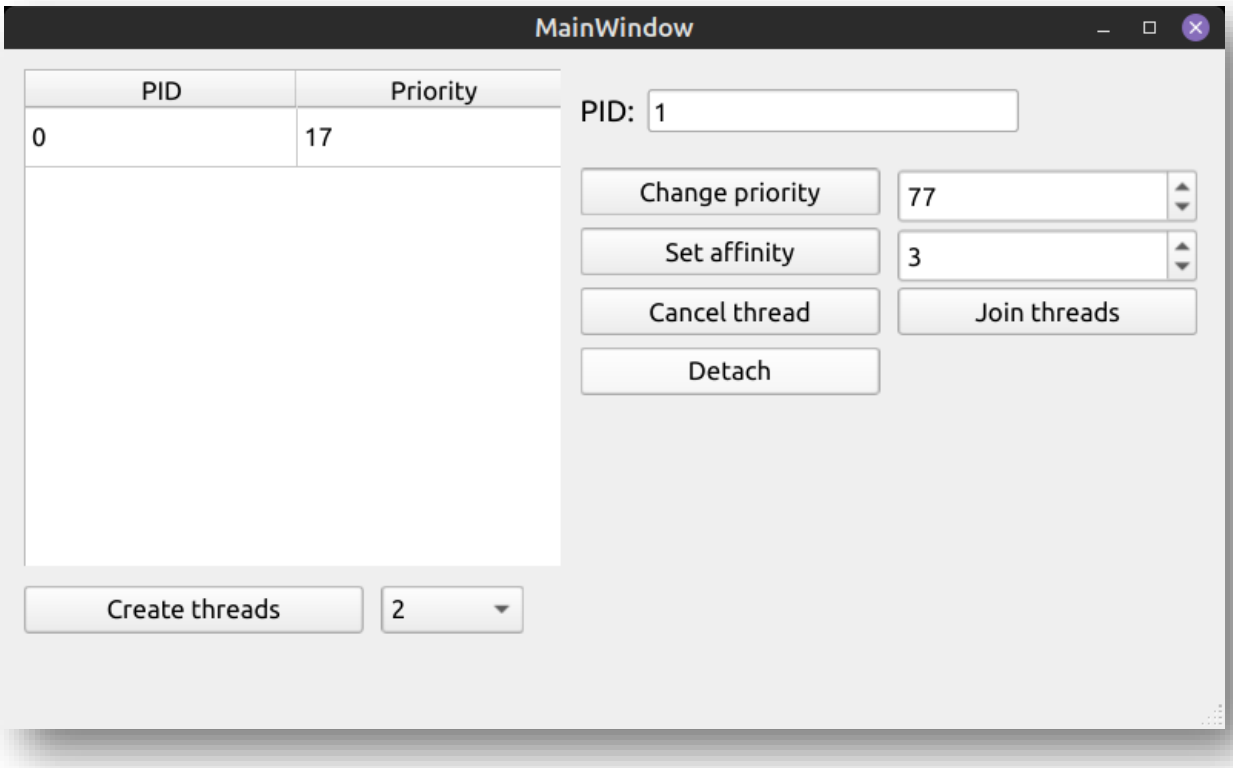


Інтерфейс програми

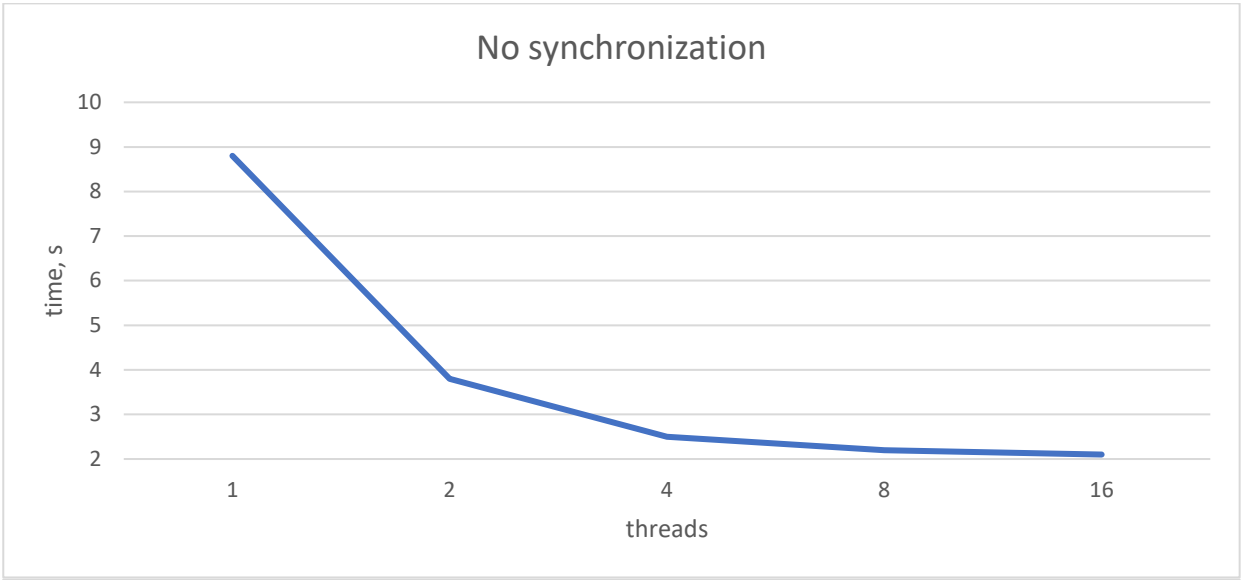
2. Для кожного потоку реалізував можливість зміни/встановлення пріоритету та встановлення відповідності виконання на ядрі. На зображеннях показано зазначений функціонал.

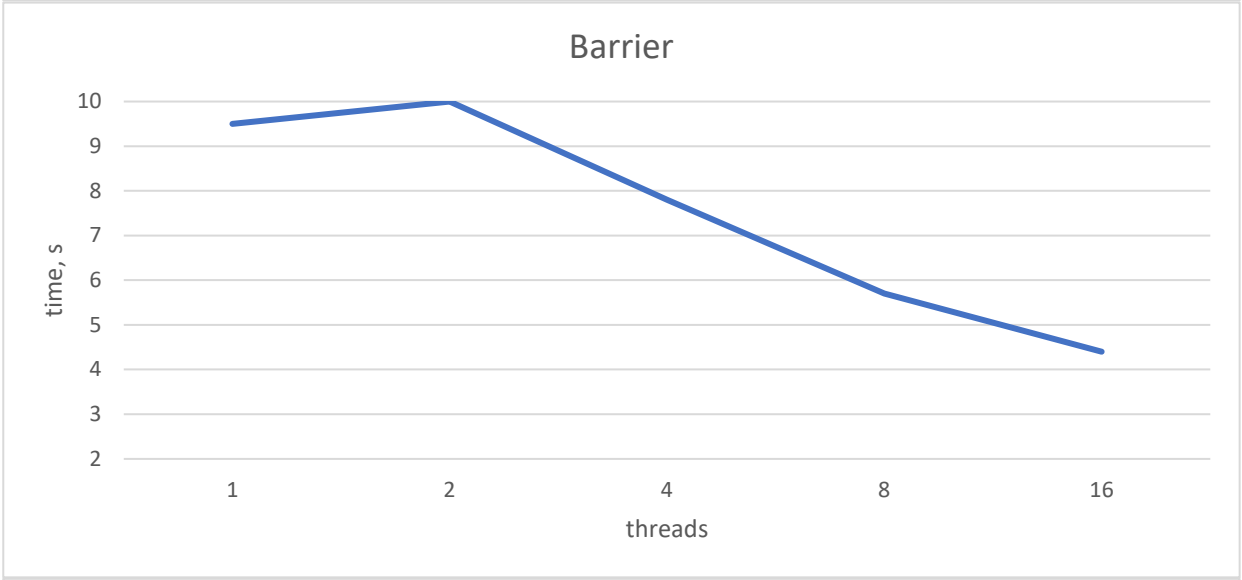


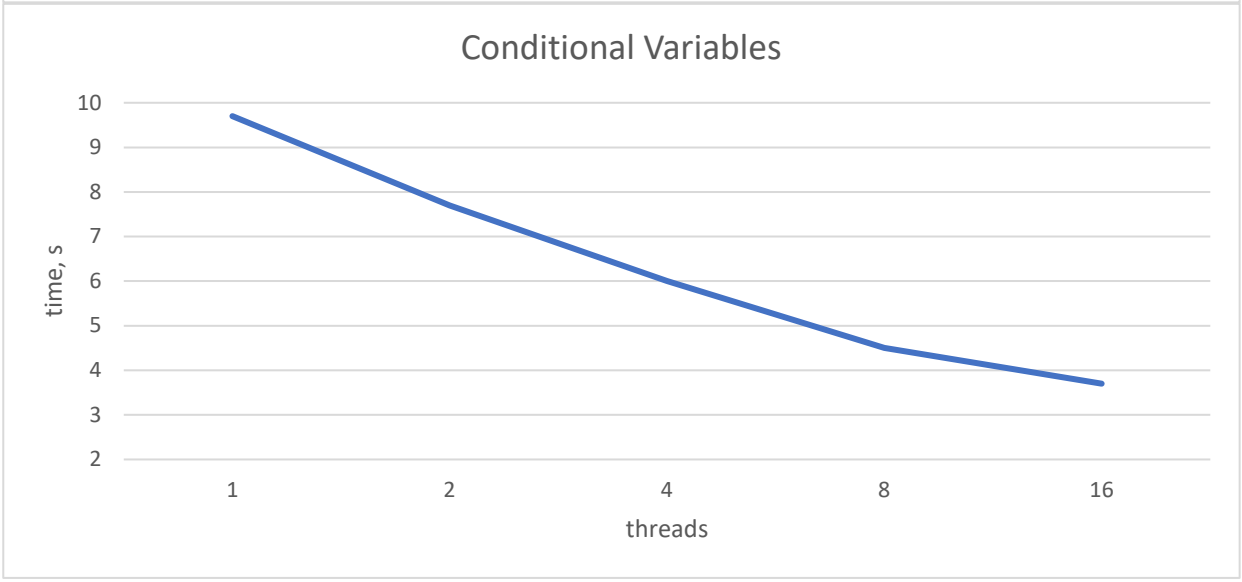
3. Реалізував можливість зробити потік від’єднаним та можливість його скасувати.

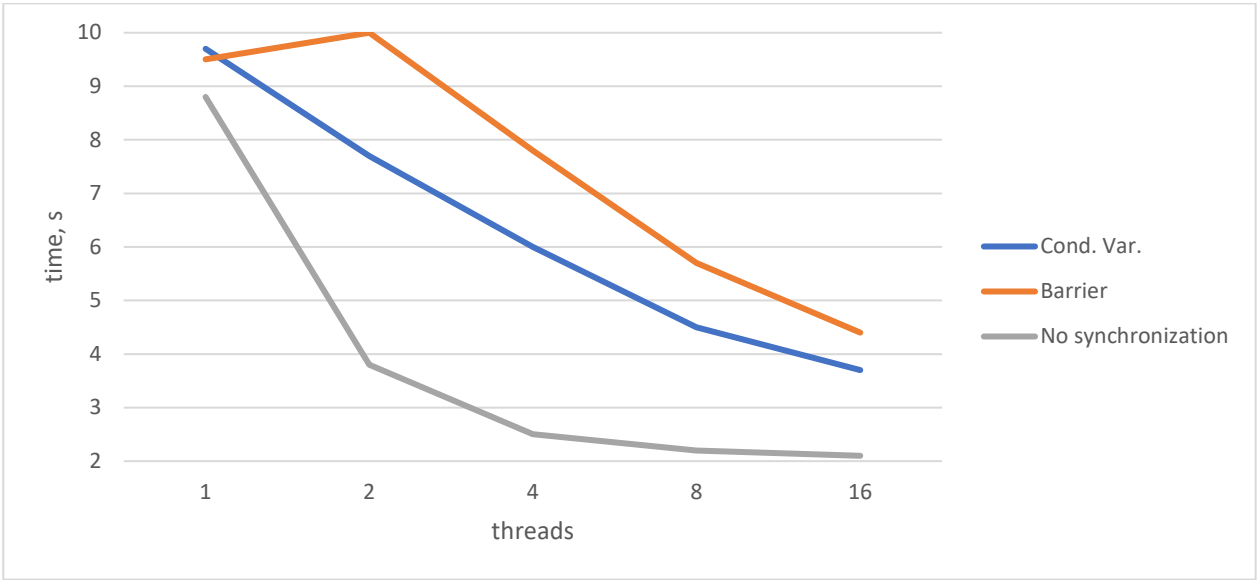


4. Виміряв час виконання для 1, 2, 4, 8 та 16 потоків та зобразив залежність кількість потоків-час для кожного виду синхронізації.


No synchronization
time, s
10
9
8
7
6
5
4
3
2
1
2
4
8
16
threads


Barrier
time, s
10
9
8
7
6
5
4
3
2
1
2
4
8
16
threads


Conditional Variables
time, s
10
9
8
7
6
5
4
3
2
1
2
4
8
16
threads



Висновок: під час виконання лабораторної роботи ознайомився з потоками в системі Linux, навчився їх створювати, змінювати пріоритет, відповідність процесорам, скасовувати. Навчився синхронізувати потоки використовуючи бар'єри та умовні змінні.