

**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:**

Старший викладач ПЗ  
Грицай О.Д.

**Виконав:**

ст. гр. ПЗ-22  
Ясногордський Н.В.

**Прийняв:**

Старший викладач ПЗ  
Грицай О.Д.

« \_\_\_\_ » \_\_\_\_\_ 2022 р.

$\Sigma =$  \_\_\_\_

Львів – 2022

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

**Мета роботи:** Ознайомитися з багатопоточністю в ОС Linux. Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Linux. Навчитись використовувати різні механізми синхронізації потоків.

### 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

### Варіант завдання №2:

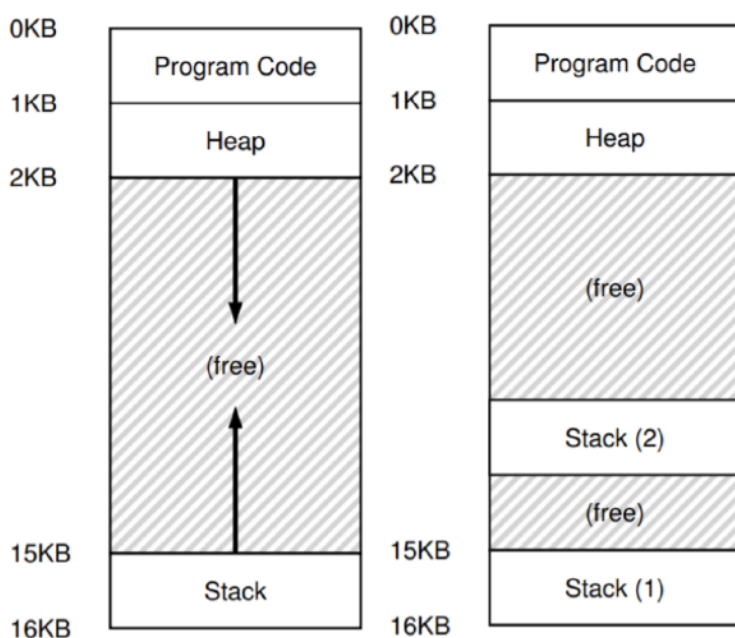
2. Обчислити суму елементів заданого масиву (кількість елементів >10000, елементи рандомні) (Синхронізація: семафор, спімблокування)

### Теоретичні відомості

Розглядаючи поняття процесу, визначають ще одну абстракцію для запущеного процесу: потік. У класичному уявленні існує єдина точка виконання в рамках програми (тобто єдиний потік контролю, на якому збираються та виконуються інструкції), багатопотокова програма має більш ніж одну точку виконання (тобто кілька потоків контролю, кожен з яких який отримується та виконується).

Кожен потік дуже схожий на окремий процес, за винятком однієї відмінності: вони мають спільний адресний простір і, отже, мають доступ до одних і тих же даних. Таким чином, стан одного потоку дуже подібний до стану процесу. Він має лічильник програм (PC), який відстежує, звідки програма отримує інструкції. Кожен потік має свій власний приватний набір реєстрів, який він використовує для обчислень; таким чином, якщо на одному процесорі працюють два потоки, при переході від запуску одного (T1) до запуску іншого (T2) має відбутися перемикання контексту. Контекстний перемикач між потоками дуже подібний до перемикання контекстів між процесами, оскільки перед запуском T2

необхідно зберегти реєстр стану T1 і відновити стан реєстру T2. За допомогою процесів ми зберегли стан до блоку управління процесами (PCB); тепер нам знадобиться один або кілька блоків управління потоками (TCB) для збереження стану кожного потоку процесу. Однак є одна істотна відмінність у перемиканні контексту, який ми виконуємо між потоками порівняно з процесами: адресний простір залишається незмінним (тобто немає необхідності змінювати, яку таблицю сторінок ми використовуємо). Ще одна істотна відмінність між потоками та процесами стосується стека. У простій моделі адресного простору класичного процесу (однопотокового) є єдиний стек, який зазвичай знаходиться внизу адресного простору. Однак у багатопотоковому процесі кожен потік працює окремо і, звичайно, може залучати різні підпрограми для виконання будь-якої роботи. Замість одного стека в адресному просторі буде по одному на кожен потік.



На цьому малюнку можна побачити два стеки, розповсюджені по адресному простору процесу. Таким чином, будь-які змінні, параметри, повернені значення, що виділяються стеком, та інші речі, які розміщуємо у стеку, будуть розміщені у тому, що іноді називають локальним сховищем потоків, тобто стеком відповідного потоку. Раніше стек і купа могли зростати незалежно, і проблеми виникали лише тоді, коли в адресному просторі вичерпалося місце. Тут немає такої приємної ситуації, оскільки стеки, як правило, не повинні бути дуже великими (виняток становлять програми, які дуже часто використовують рекурсію).

## Хід роботи

Спочатку напишу код для програми. Я виконуватиму її в консольному середовищі.

Також додам відлік часу, який рахуватиме, скільки потрібно чекати на результат Запрограмовую рішення.

### Код:

```
#include <pthread.h>
#include <semaphore.h>
#include <sys/resource.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

#include <algorithm>
#include <chrono>
#include <cmath>
#include <ctime>
#include <fstream>
#include <iostream>
#include <ratio>
#include <thread>

int duration;
int sched_policy = SCHED_RR;
int sleep_time;

std::vector<int> numbers;
long long resulting_sum = 0;

struct parameters {
    int from;
    int to;
    int method;
    bool with_sleep;
};

sem_t sum_semaphore;
pthread_spinlock_t sum_spinlock;
std::vector<pthread_t> thread_ids;

int thread_count = 0;
```

## Протокол роботи

Запусти 1 нитках, щоб порівняти роботу без та із синхронізаціями.

Порахуємо сумму 1000000 випадкових double

### Spinlock

```
Enter array size: 1000000
Enter thread amount: 1
Choose method:
0. Spinlock
1. Semaphore
0

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice:
-----
Computed sum: 5.00341e+08
Time taken: 75ms
-----
```

### Semaphore

```
Enter array size: 1000000
Enter thread amount: 1
Choose method:
0. Spinlock
1. Semaphore
1

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice:
-----
Computed sum: 5.00341e+08
Time taken: 83ms
-----
```

Час виконання приблизно однаковий

```
Enter array size: 1000000
Enter thread amount: 10
Choose method:
0. Spinlock
1. Semaphore
0

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice:
-----
Computed sum: 5.00341e+08
Time taken: 8ms
-----
```

```
Enter choice:
Enter array size: 1000000
Enter thread amount: 10
Choose method:
0. Spinlock
1. Semaphore
1

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice:
-----
Computed sum: 5.00341e+08
Time taken: 10ms
-----
```

Виконання в 10 нитках

Легко помітити прискорення виконання майже в 10 разів, що доводить ефективність паралельних обчислень. Семафор в порівнянні зі спінлоком витрачає більше процесорного часу, щоб не викликати race condition, в першу чергу це пов'язано з більшою к-стю системних викликів

```
Enter choice: F

Enter sleep time: 100
Enter array size: 10000
Enter thread amount: 10
Choose method:
0. Spinlock
1. Semaphore
0

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice: l

idx: 0 | tid: 274917410368 | prio: 0 | max prio: 0
idx: 1 | tid: 274925803072 | prio: 0 | max prio: 0
idx: 2 | tid: 274934195776 | prio: 0 | max prio: 0
idx: 3 | tid: 274942588480 | prio: 0 | max prio: 0
idx: 4 | tid: 274950981184 | prio: 0 | max prio: 0
idx: 5 | tid: 274959373888 | prio: 0 | max prio: 0
idx: 6 | tid: 274967766592 | prio: 0 | max prio: 0
idx: 7 | tid: 274976159296 | prio: 0 | max prio: 0
idx: 8 | tid: 274984552000 | prio: 0 | max prio: 0
idx: 9 | tid: 274992944704 | prio: 0 | max prio: 0

Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice: █
```

```
Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice: d
```

```
Enter idx: 0
thread has been detached
```

```
Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice: c
```

```
Enter idx: 8
thread has been canceled
```

```
Choose option:
f. Find array sum
F. Find array sum with sleep
p. Change thread prio
d. Detach thread
c. Cancel thread
l. List threads
q. Exit
Enter choice: p
```

```
Enter idx: 2
Enter priority:
10
couldn't change prio
```

Демонстрація операцій над потоками в Linux  
(detach, cancel, sche\_priority)

## **Висновок**

У цій лабораторній роботі я навчився працювати із потоками та методами їхньої синхронізації, використовуючи Linux. Я ознайомився із спін-локом та семафором, зрозумів, який метод швидший та чому. Я розбив завдання на окремі частинки та використав синхронізацію, щоб отримати потрібний результат.