

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут КНІТ
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 кафедри ПЗ
Грицай О.Д.

Виконав:

студент групи ПЗ-22
Коваленко Д.М.

Прийняла:

старший викладач кафедри ПЗ
Грицай О.Д.

«_____» _____ 2022 р.
 Σ =

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.

Лабораторне завдання

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

Хід роботи

```
#include "task.h"
#include <unistd.h>
#include "QDebug"

using namespace std;

int numCpus = sysconf(_SC_NPROCESSORS_ONLN);

void* thread_task(void* args) {
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, 0);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, 0);

    ThreadArgs* targ = (ThreadArgs*)args;
    int start = targ->start;
    int end = targ->end;
    int* sum = targ->sum;
    int* arr = targ->arr;
    sem_t* sem = targ->sem;
    pthread_spinlock_t* spin = targ->spin;
    Task* task = (Task*)targ->task;

    task->setStatus(Running);
    int value;
    sem_getvalue(sem, &value);
    if (value == 0) {
        task->setStatus(Waiting);
    }
    sem_wait(sem);
    task->setStatus(Running);
    std::srand(static_cast<unsigned int>(std::time(nullptr)));
    for (int i = start; i < end; i++) {
```

```

        arr[i] = rand() % 5;
    }

    for (int i = start; i < end; i++) {
        pthread_spin_lock(spin);
        *sum += arr[i];
        pthread_spin_unlock(spin);
    }
    sem_post(sem);
    task->setStatus(Done);
    pthread_exit(NULL);
}

Task::Task(int threadCount, int threadIndex, int arrLen, int* sum, int* arr, sem_t* sem, pthread_t* spin) {
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    this->attr = attr;
    pthread_attr_setschedpolicy(&this->attr, SCHED_FIFO);
    ThreadArgs* args = new ThreadArgs(threadCount, threadIndex, arrLen, sum, arr, sem, spin);
    if (pthread_create(&this->thread, &attr, &thread_task, args)) {
        qDebug() << "pthread_create error!";
        return;
    }
    this->threadIndex = threadIndex;
    this->setAffinity(this->threadIndex % numCpus);
}

void Task::detach() {
    pthread_detach(this->thread);
    if (this->status != Done) this->status = Detached;
}

void Task::cancel() {
    pthread_cancel(this->thread);
    if (this->status != Done) this->status = Canceled;
}

void Task::setPriority(int n) {
    int policy;
    struct sched_param param;

    pthread_getschedparam(this->thread, &policy, &param);
    param.sched_priority = n;
    pthread_setschedparam(this->thread, policy, &param);
}

int Task::getPriority() {
    int policy;
    struct sched_param param;
    pthread_getschedparam(this->thread, &policy, &param);
    return param.sched_priority;
}

void Task::setAffinity(int cpu) {
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(cpu, &cpuset);
    if (pthread_setaffinity_np(this->thread, sizeof(cpuset), &cpuset)) {
        qDebug() << "pthread_setaffinity_np() error";
    }
}

```

```

int Task::getAffinity() {
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    if (pthread_getaffinity_np(this->thread, sizeof(cpuset), &cpuset)) {
        qDebug() << "pthread_attr_getaffinity_np() error";
    }
    for (int i = 0; i < numCpus; i++) {
        if (CPU_ISSET(i, &cpuset)) return i;
    }
    return -1;
}

```

The screenshot shows a Qt application window with a light gray background. On the left side, there are several input fields and buttons. At the top left, there is a 'Start' button. Below it are three input fields containing the values '5', '123123', and '2'. To the right of these fields are two buttons: 'Set Priority' and 'Set Affinity'. The 'Set Priority' button is next to an input field containing '0'. The 'Set Affinity' button is next to an input field containing '2'. Below these buttons are two more buttons: 'Detach' and 'Cancel'. At the bottom left, there is a 'Sum' label next to an input field containing '245849'. Below that is a 'Correct sum' label next to an empty input field. To the right of the 'Correct sum' field is a 'Check' button. On the right side of the window, there is a table with four columns: 'Index', 'Status', 'Priority', and 'Affinity'. The table has five rows of data, all with 'Done' status and '32765' priority. The 'Index' column contains values 0, 1, 2, 3, and 4. The 'Affinity' column contains the value 0 for all rows.

Index	Status	Priority	Affinity
0	Done	32765	0
1	Done	32765	0
2	Done	32765	0
3	Done	32765	0
4	Done	32765	0

Рис. 1: Виконання програми

Висновок

Під час виконання лабораторної роботи я навчився створювати потоки та керувати ними в операційній системі Linux. Ознайомився з методами синхронізації потоків в операційній системі Linux. Навчився реалізовувати багатопоточний алгоритм розв'язку задачі з використанням синхронізації в операційній системі Linux.