

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут» ім. Ігоря Сікорського

# Лабораторна робота №4

***з дисципліни «Паралельне програмування»***

**“*КОМПЛЕКСНЕ ВИКОРИСТАННЯ ЗАСОБІВ ВЗАЄМОДІЇ ПАРАЛЕЛЬНИЙ ПОТОКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX*”**

Виконала студентка групи: КВ-31

Кузьменко Олександра

**Київ 2025**

**Постановка задачі**

1. Опрацювати всі надані лектором приклади коду паралельних потоків по темі «Засоби взаємодії паралельних потоків операційної системи Linux», що знаходяться в директоріях **04\_Common\_Resource** та **05\_Atomic\_Operations**, тобто:

-вміти запускати всі ці приклади і отримувати результати на захисті лабораторної роботи;

-знати які структури даних та конструкції взаємодії паралельних потоків описані в коді кожного прикладу та як вони працюють, а також вміти це пояснити на захисті лабораторної роботи;

-розібратися з теоретичними ситуаціями, які відображують дані приклади, а також вміти їх розказати та пояснити на захисті лабораторної роботи;

-бути готовими до виконання модифікацій будь-яких з цих прикладів на захисті лабораторної роботи.

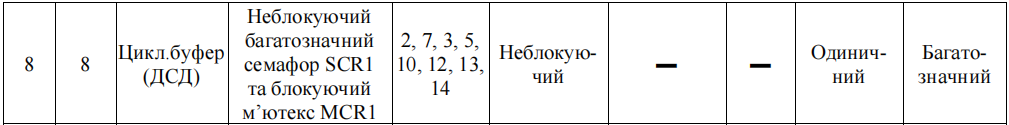
1. Написати програму, яка реалізує роботу паралельних потоків згідно заданої за варіантом схеми. Особливості реалізації синхронізації паралельних потоків та взаємного виключення потоків при доступі до спільних ресурсів задані за варіантами у таблицях 1 та 2.
2. При написанні програми виконати повне трасування роботи програми за допомогою операторів друку, тобто розставити в програмі оператори друку таким чином, щоб можна було прослідкувати всі варіанти виконання паралельних потоків і впевнитись у коректності роботи програми. Протокол трасування рекомендується записувати у файл (log-файл).
3. Запуск усіх потоків повинен бути виконаний у головній програмі.
4. Кожен потік повинен бути організованим у вигляді нескінченного циклу.
5. Всі дії задані за варіантами, що вказані у таблиці, повинні бути виконані всередині цього нескінченного циклу.
6. Взаємне розташування операторів синхронізації та доступу до спільного ресурсу, якщо вони знаходяться у одному потоці, є довільним.
7. Оскільки синхронізація за допомогою семафорів SCR21, SCR22 згідно завдання розташована всередині нескінченних циклів, то відразу після виконання синхронізації ці семафори повинні бути знову встановлені у початковий закритий стан.
8. Закінчення програми можна виконати двома способами:

-примусовим перериванням за допомогою натиснення комбінації клавіш Ctrl+C;

-оператором break при виконанні умови, яка стає істинною, коли буфер спільного ресурсу повністю заповнюється і повністю звільняється мінімум по два рази.

1. Якщо при реалізації паралельних потоків була використана функція usleep(), то передбачити режим запуску програми з «відключеними» функціями usleep().
2. Виконати налагодження написаної програми.

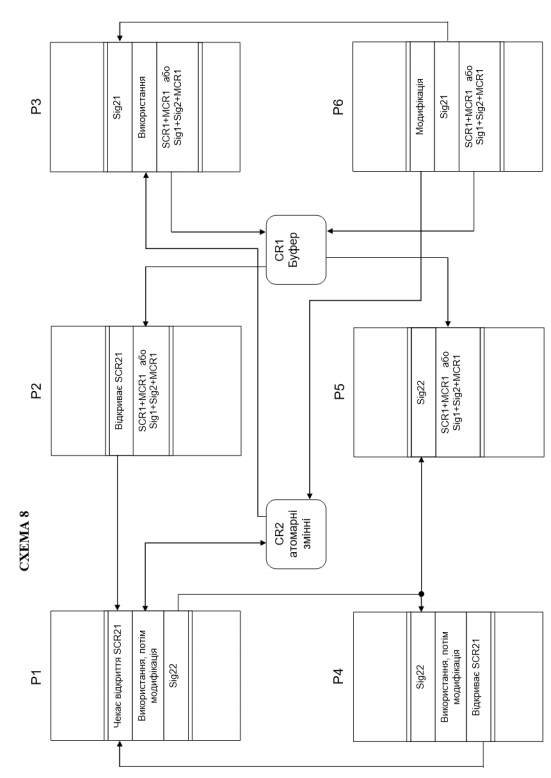
**Завдання 8 варіанту:**



Таблиця 2 містить перелік вбудованих функцій, які реалізують атомарні операції.

**Таблиця 2**

|  |  |
| --- | --- |
| **Номер атомарної операції**  **(функції)** | **Ідентифікатор вбудованої функції, що відповідає атомарній операції** |
| 2. | *type* **\_\_atomic\_sub\_fetch** *(type \*ptr, type val, int memorder)* |
| 3. | *type* **\_\_atomic\_and\_fetch** *(type \*ptr, type val, int memorder)* |
| 5. | *type* **\_\_atomic\_or\_fetch** *(type \*ptr, type val, int memorder)* |
| 7. | *type* **\_\_atomic\_fetch\_add** *(type \*ptr, type val, int memorder)* |
| 10. | *type* **\_\_atomic\_fetch\_xor** *(type \*ptr, type val, int memorder)* |
| 12. | *type* **\_\_atomic\_fetch\_nand** *(type \*ptr, type val, int memorder)* |
| 13. | *bool* **\_\_atomic\_compare\_exchange\_n** *(type \*ptr, type \*expected, type desired, bool weak, int success\_memorder, int failure\_memorder)* |
| 14. | *void* **\_\_atomic\_exchange** *(type \*ptr, type \*val, type \*ret, int memorder)* |



**Код програми**

#include stdio.h

#include malloc.h

#include unistd.h

#include pthread.h

#include semaphore.h

#include stdbool.h

#define MAX\_COUNT 20

pthread\_mutex\_t mut\_q = PTHREAD\_MUTEX\_INITIALIZER;

sem\_t sem\_q;

pthread\_t thread1;

pthread\_t thread2;

pthread\_t thread3;

pthread\_t thread4;

pthread\_t thread5;

pthread\_t thread6;

int sem\_value;

struct t\_elem {

int number;

struct t\_elem next;

};

struct t\_elem tail = NULL; останній елемент (tail-next = head)

int count = 0;

bool was\_full = false;

bool was\_empty = true;

int full\_transitions = 0;

int empty\_transitions = 0;

int buffer\_clear\_loops = 2;

---------

pthread\_cond\_t sig21 = PTHREAD\_COND\_INITIALIZER;for sig21

pthread\_cond\_t sig22 = PTHREAD\_COND\_INITIALIZER;

int flag21 = 0;for sig21

int flag22 = 0;

pthread\_mutex\_t mut21 = PTHREAD\_MUTEX\_INITIALIZER;for sig21

pthread\_mutex\_t mut22 = PTHREAD\_MUTEX\_INITIALIZER;

sem\_t SCR21;

-------

void add\_elem(int number) {

if (count = MAX\_COUNT) {

printf([Add] Cannot add %d list is full (max %d elements)n, number, MAX\_COUNT);

return;

}

struct t\_elem new\_elem = malloc(sizeof(struct t\_elem));

if (!new\_elem) {

printf(Allocation failed!n);

return;

}

new\_elem-number = number;

if (tail == NULL) {

перший елемент

new\_elem-next = new\_elem;

tail = new\_elem;

} else {

new\_elem-next = tail-next; next вказує на голову

tail-next = new\_elem;

tail = new\_elem;

}

count++;

printf([Add] Added %d (current count %d)n, number, count);

}

struct t\_elem get\_elem() {

if(count == 0)

buffer\_clear\_loops--;

if ( buffer\_clear\_loops == 0) {

printf([Buffer] EMPTY! Cannot get.n);

return NULL;

}

struct t\_elem head = tail-next;

if (head == tail) {

tail = NULL;

} else {

tail-next = head-next;

}

count--;

printf([Get] Got %dn, head-number);

return head;

}

int ati1 = 0, ati2 = 0;

unsigned atu1 = 0, atu2 = 0;

long atl1 = 0, atl2 = 0;

long unsigned atlu1 = 0, atlu2 = 0;

void use\_atom(int tnum)

{

printf(Thread%d use atom START n, tnum);

printf(int %d, %dn, ati1, ati2);

printf(unsigned %u, %un, atu1, atu2);

printf(long %ld, %ldn, atl1, atl2);

printf(long unsigned %lu, %lun, atlu1, atlu2);

}

void mod\_atom(int tnum)

{

printf(Thread%d mod\_atom START nnn, tnum);

printf(int sub fetch %dn, \_\_atomic\_sub\_fetch(&ati1, tnum, \_\_ATOMIC\_RELAXED));

printf(int xor fetch %dn, \_\_atomic\_and\_fetch(&ati2, tnum, \_\_ATOMIC\_RELAXED));

printf(unsigned nand fetch %un, \_\_atomic\_or\_fetch(&atu1, tnum, \_\_ATOMIC\_RELAXED));

printf(unsigned fetch add %un, \_\_atomic\_fetch\_add(&atu2, tnum, \_\_ATOMIC\_RELAXED));

printf(long fetch and %ldn, \_\_atomic\_fetch\_xor(&atl1, tnum, \_\_ATOMIC\_RELAXED));

printf(long fetch or %ldn, \_\_atomic\_fetch\_nand(&atl2, tnum, \_\_ATOMIC\_RELAXED));

printf(long unsigned compare exchange\_n %dn, \_\_atomic\_compare\_exchange\_n(&atlu1, &atlu2, tnum, 0, \_\_ATOMIC\_RELAXED, \_\_ATOMIC\_RELAXED));

printf(long unsigned compare exchange\_n before %lu, %lu;n, atlu2, atlu1);

\_\_atomic\_exchange(&atlu2, &atlu1, &atlu2, \_\_ATOMIC\_RELAXED);

printf( after useg compare exchange %lu, %lun, atlu2, atlu1);

}

void p1 (void arg)

{

int num = (int)arg;

while (1)

{

---------------------------------------

if(sem\_trywait(&SCR21) == 0)

{

printf(Thread%d semaphore SCR21 semaphore is openn, num);

}------------------------------------------------

use\_atom(num);

mod\_atom(num);

--------

pthread\_mutex\_lock(&mut22);

printf(Thread%d in mutex22! n,num);

flag22++;

pthread\_cond\_signal(&sig22);

printf(Thread%d send signal22! outside mutex22 n,num);

pthread\_mutex\_unlock(&mut22);

--------

}

}

void p2 (void arg)

{

int num = (int)arg;

struct t\_elem curr\_elem=NULL;

while (1) {

--------

if(sem\_trywait(&SCR21) == 0)

{

printf(Thread%d semaphore`s(SCR21) signal was just sent n, num);

sem\_post(&SCR21);

}

else

{

printf(Thread%d semaphor`s(SCR21) signal has already been sent before n, num);

}

----------------------

if ( sem\_trywait (&sem\_q) == 0 ) {

sem\_getvalue(&sem\_q,&sem\_value);

pthread\_mutex\_lock(&mut\_q);

curr\_elem = (struct t\_elem)get\_elem();

printf(Consumer thread%d semaphore=%d; element %d TAKEN; n,

num,sem\_value,curr\_elem-number);

free (curr\_elem);

pthread\_mutex\_unlock (&mut\_q);

}

----------------------

}

}

void p3(void arg)

{

int num = (int)arg;

while (1) {

--------

pthread\_mutex\_lock(&mut21);

printf(Thread%d in mutex21! n,num);

while (flag21==0)

{

pthread\_cond\_wait(&sig21,&mut21);

}

flag21=0;

printf(Thread%d take signal1! outside mutex1 n,num);

pthread\_mutex\_unlock(&mut21);

--------

use\_atom(num);

----------------------

sem\_getvalue(&sem\_q,&sem\_value);

if (sem\_value MAX\_COUNT) {

pthread\_mutex\_lock(&mut\_q);

add\_elem(8);

printf(Producer thread%d semaphore=%d; element CREATED; n,

num,sem\_value);

sem\_post (&sem\_q);

pthread\_mutex\_unlock (&mut\_q);

}

----------------------

}

}

void p4 (void arg)

{

int num = (int)arg;

while (1)

{

--------

pthread\_mutex\_lock(&mut22);

printf(Thread%d in mutex22! n,num);

while (flag22==0)

{

pthread\_cond\_wait(&sig22,&mut22);

}

flag22--;

printf(Thread%d take signal22! outside mutex22 n,num);

pthread\_mutex\_unlock(&mut22);

--------

use\_atom(num);

mod\_atom(num);

--------

if(sem\_trywait(&SCR21) == 0)

{

printf(Thread%d semaphore`s(SCR21) signal was just sent n, num);

sem\_post(&SCR21);

}

else

{

printf(Thread%d semaphor`s(SCR21) signal has already been sent beforen, num);

}

-------

}

}

void p5 (void arg)

{

int num = (int)arg;

struct t\_elem curr\_elem=NULL;

while (1) {

--------

pthread\_mutex\_lock(&mut22);

printf(Thread%d in mutex22! n,num);

while (flag22==0)

{

pthread\_cond\_wait(&sig22,&mut22);

}

flag22--;

printf(Thread%d take signal22! outside mutex22 n,num);

pthread\_mutex\_unlock(&mut22);

----------------------

if ( sem\_trywait (&sem\_q) == 0 ) {

sem\_getvalue(&sem\_q,&sem\_value);

pthread\_mutex\_lock(&mut\_q);

curr\_elem = (struct t\_elem)get\_elem();

printf(Consumer thread%d semaphore=%d; element %d TAKEN; n,

num,sem\_value,curr\_elem-number);

free (curr\_elem);

pthread\_mutex\_unlock (&mut\_q);

}

----------------------

}

}

void p6(void arg)

{

int num = (int)arg;

while (1) {

mod\_atom(num);

--------

pthread\_mutex\_lock(&mut21);

printf(Thread%d in mutex21! n,num);

flag21 = 1; запам'ятовується факт посилання сигнала sig21

pthread\_cond\_signal(&sig21);

printf(Thread%d send signal21! outside mutex21 n,num);

pthread\_mutex\_unlock(&mut21);

----------------------

sem\_getvalue(&sem\_q,&sem\_value);

if (sem\_value MAX\_COUNT) {

pthread\_mutex\_lock(&mut\_q);

add\_elem(8);

printf(Producer thread%d semaphore=%d; element CREATED; n,

num,sem\_value);

sem\_post (&sem\_q);

pthread\_mutex\_unlock (&mut\_q);

}

----------------------

}

}

int main()

{

sem\_init (&sem\_q, 0, 0);

sem\_init(&SCR21,0,0);

int sem\_value;

sem\_getvalue(&sem\_q,&sem\_value);

printf(semaphore=%dn,sem\_value);

int length\_at\_start=10;

int i;

for(i=0;ilength\_at\_start;i++) {

add\_elem(i);

sem\_post(&sem\_q);

}

printf(Queue with elements from 0-th to %d-th has been created !!!n,length\_at\_start-1);

sem\_getvalue(&sem\_q,&sem\_value);

printf(semaphore=%dn,sem\_value);

int thread1\_number=1;

int thread2\_number=2;

int thread3\_number=3;

int thread4\_number=4;

int thread5\_number=5;

int thread6\_number=6;

pthread\_create (&thread1,NULL,&p1,(void)&thread1\_number);

pthread\_create (&thread2,NULL,&p2,(void)&thread2\_number);

pthread\_create (&thread3,NULL,&p3,(void)&thread3\_number);

pthread\_create (&thread4,NULL,&p4,(void)&thread4\_number);

pthread\_create (&thread5,NULL,&p5,(void)&thread5\_number);

pthread\_create (&thread6,NULL,&p6,(void)&thread6\_number);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

pthread\_join(thread3,NULL);

pthread\_join(thread4,NULL);

pthread\_join(thread5,NULL);

pthread\_join(thread6,NULL);

printf(All threads stopped !!!n);

pthread\_mutex\_destroy (&mut\_q);

pthread\_mutex\_destroy (&mut21);

pthread\_mutex\_destroy (&mut22);

sem\_destroy (&sem\_q);

sem\_destroy (&SCR21);

pthread\_cond\_destroy(&sig21);

pthread\_cond\_destroy(&sig22);

return 0;

}