**Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп’ютерних систем**

ЛАБОРАТОРНА РОБОТА №4

*з дисципліни*

***«Паралельне програмування»***

**Тема: «Комплексне використання засобів взаємодії паралельних потоків операційної системи LINUX»**

Виконала: Сапожко В.Ю. Студентка групи КВ-31 Перевірив(ла):

Київ 2025

**Постановка задачі**

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.

2. 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.

3. 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).

4. Запуск усіх потоків повинен бути виконаний у головній програмі.

5. Кожен потік повинен бути організованим у вигляді нескінченного циклу.

6. Всі дії задані за варіантами, що вказані у таблиці, повинні бути виконані всередині цього нескінченного циклу.

7. Взаємне розташування операторів синхронізації та доступу до спільного ресурсу, якщо вони знаходяться у одному потоці, є довільним.

8. 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.

9. Закінчення програми можна виконати двома способами:

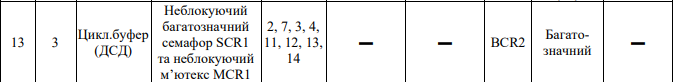
• примусовим перериванням за допомогою натиснення комбінації клавіш Ctrl+C;

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

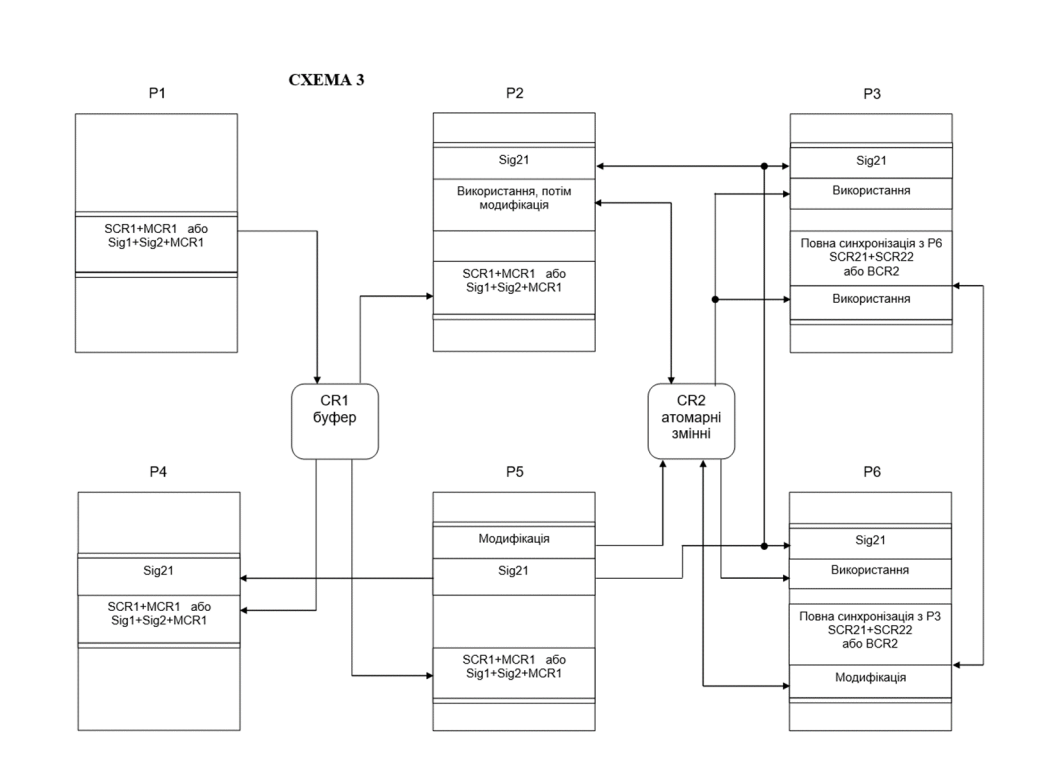
10.Якщо при реалізації паралельних потоків була використана функція usleep(), то передбачити режим запуску програми з «відключеними» функціями usleep().

11.Виконати налагодження написаної програми.

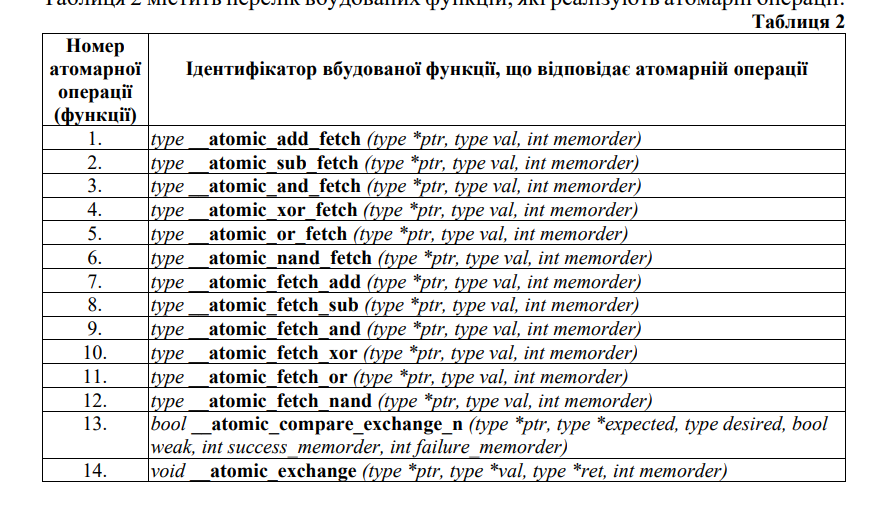
**Варіант №13**



**Схема**



**Таблиця 2**

****

**Код програми**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

FILE \*zapic;

sem\_t scr1;

pthread\_mutex\_t mcr1 = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_barrier\_t bcr2;

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

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

int flag2 = 0;

int flag3 = 0;

int flag4 = 0;

int flag6 = 0;

int max\_buff\_lengs=30;

struct t\_elem {

struct t\_elem\* next;

int number;

};

struct t\_elem\* last = NULL;

void add\_elem()

{

struct t\_elem\* NextIn = malloc(sizeof(struct t\_elem));

if (last == NULL)

{

NextIn->number = 0;

NextIn->next = NextIn;

last = NextIn;

}

else

{

NextIn->number = last->number + 1;

NextIn->next = last->next;

last->next = NextIn;

last = NextIn;

}

}

struct t\_elem\* get\_elem()

{

struct t\_elem\* NextOut = last->next;

if (NextOut == last)

{

last = NULL;

}

else

{

last->next = NextOut->next;

}

return NextOut;

}

int iatom1 = 0, iatom2 = 0;

unsigned uatom1 = 0, uatom2 = 0;

long latom1 = 0, latom2 = 0;

unsigned long luatom1 = 0, luatom2 = 0;

void Atomuse(int thread\_number) {

fprintf(zapic, "Untyped thread %d: Vars usage\n", thread\_number);

fprintf(zapic, "int: %d, %d\n", iatom1, iatom2);

fprintf(zapic, "unsigned: %u, %u\n", uatom1, uatom2);

fprintf(zapic, "long: %ld, %ld\n", latom1, latom2);

fprintf(zapic, "long unsigned: %lu, %lu\n", luatom1, luatom2);

}

void Atommod(int thread\_number) {

fprintf(zapic, "Untyped thread %d: Vars modification\n", thread\_number);

fprintf(zapic, "int sub fetch: %d\n", \_\_atomic\_sub\_fetch(&iatom1, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "int fetch sub: %d\n", \_\_atomic\_fetch\_add(&iatom2, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "unsigned xor fetch: %u\n", \_\_atomic\_and\_fetch(&uatom1, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "unsigned or fetch: %u\n", \_\_atomic\_xor\_fetch(&uatom2, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "long fetch and: %ld\n", \_\_atomic\_fetch\_or(&latom1, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "long fetch nand: %ld\n", \_\_atomic\_fetch\_nand(&latom2, thread\_number, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "int compare exchange n: %d\n",

\_\_atomic\_compare\_exchange\_n(&luatom1, &luatom2, thread\_number, 0, \_\_ATOMIC\_RELAXED, \_\_ATOMIC\_RELAXED));

fprintf(zapic, "int exchange: before %lu, %lu;", luatom2, luatom1);

\_\_atomic\_exchange(&luatom2, &luatom1, &luatom1, \_\_ATOMIC\_RELAXED);

fprintf(zapic, " after %lu, %lu\n", luatom2, luatom1);

}

pthread\_t pth1;

pthread\_t pth2;

pthread\_t pth3;

pthread\_t pth4;

pthread\_t pth5;

pthread\_t pth6;

void\* thread1(void\* thread\_number) {

int number = \*(int\*)thread\_number;

int sem\_value;

while (1) {

sem\_getvalue(&scr1,&sem\_value);

if (sem\_value < max\_buff\_lengs) {

while (pthread\_mutex\_trylock (&mcr1) != 0) {

fprintf(zapic, "Producer thread%d: mutex busy\n", number);

}

add\_elem();

printf("Producer thread%d: semaphore=%d; element %d creted; \n", number,sem\_value,last->number);

sem\_post(&scr1);

pthread\_mutex\_unlock (&mcr1);

}

}

return NULL;

}

void\* thread2(void\* thread\_number) {

int number = \*(int\*)thread\_number;

struct t\_elem\* curr\_elem=NULL;

int sem\_value;

while (1) {

pthread\_mutex\_lock(&mut21);

while (flag2 == 0) {

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

}

flag2 = 0;

pthread\_mutex\_unlock(&mut21);

Atomuse(number);

Atommod(number);

if(sem\_trywait (&scr1)!=0){

fprintf(zapic, "Producer thread%d: semaphor busy\n", number);

}else{

sem\_getvalue(&scr1,&sem\_value);

while (pthread\_mutex\_trylock (&mcr1) != 0) {

fprintf(zapic, "Producer thread%d: mutex busy\n", number);

}

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

printf("Consumer thread%d: semaphore=%d; element %d taken; \n",number,sem\_value,curr\_elem->number);

free (curr\_elem);

pthread\_mutex\_unlock(&mcr1);

}

}

return NULL;

}

void\* thread3(void\* thread\_number) {

int number = \*(int\*)thread\_number;

while (1) {

pthread\_mutex\_lock(&mut21);

while (flag3 == 0) {

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

}

flag3 = 0;

pthread\_mutex\_unlock(&mut21);

Atomuse(number);

pthread\_barrier\_wait(&bcr2);

Atomuse(number);

}

return NULL;

}

void\* thread4(void\* thread\_number) {

int number = \*(int\*)thread\_number;

struct t\_elem\* curr\_elem=NULL;

int sem\_value;

while (1) {

pthread\_mutex\_lock(&mut21);

while (flag4 == 0) {

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

}

flag4 = 0;

pthread\_mutex\_unlock(&mut21);

if(sem\_trywait (&scr1)!=0){

fprintf(zapic, "Producer thread%d: semaphor busy\n", number);

}else{

sem\_getvalue(&scr1,&sem\_value);

while (pthread\_mutex\_trylock (&mcr1) != 0) {

fprintf(zapic, "Producer thread%d: mutex busy\n", number);

}

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

printf("Consumer thread%d: semaphore=%d; element %d taken; \n",number,sem\_value,curr\_elem->number);

free (curr\_elem);

pthread\_mutex\_unlock(&mcr1);

}

}

return NULL;

}

void\* thread5(void\* thread\_number) {

int number = \*(int\*)thread\_number;

struct t\_elem\* curr\_elem=NULL;

int sem\_value;

while (1) {

Atommod(number);

pthread\_mutex\_lock(&mut21);

flag2 = 1;

flag3 = 1;

flag4 = 1;

flag6 = 1;

pthread\_cond\_broadcast(&sig21);

pthread\_mutex\_unlock(&mut21);

if(sem\_trywait (&scr1)!=0){

fprintf(zapic, "Producer thread%d: semaphor busy\n", number);

}else{

sem\_getvalue(&scr1,&sem\_value);

while (pthread\_mutex\_trylock (&mcr1) != 0) {

fprintf(zapic, "Producer thread%d: mutex busy\n", number);

}

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

printf("Consumer thread%d: semaphore=%d; element %d taken; \n",number,sem\_value,curr\_elem->number);

free (curr\_elem);

pthread\_mutex\_unlock(&mcr1);

}

}

return NULL;

}

void\* thread6(void\* thread\_number) {

int number = \*(int\*)thread\_number;

while (1) {

pthread\_mutex\_lock(&mut21);

while (flag6 == 0) {

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

}

flag6 = 0;

pthread\_mutex\_unlock(&mut21);

Atomuse(number);

pthread\_barrier\_wait(&bcr2);

Atommod(number);

}

return NULL;

}

int main() {

zapic = fopen("zapic.txt", "w");

int thread1\_number = 1;

int thread2\_number = 2;

int thread3\_number = 3;

int thread4\_number = 4;

int thread5\_number = 5;

int thread6\_number = 6;

sem\_init(&scr1, 0, 0);

pthread\_barrier\_init(&bcr2, NULL, 2);

pthread\_create(&pth1, NULL, &thread1, &thread1\_number);

pthread\_create(&pth2, NULL, &thread2, &thread2\_number);

pthread\_create(&pth3, NULL, &thread3, &thread3\_number);

pthread\_create(&pth4, NULL, &thread4, &thread4\_number);

pthread\_create(&pth5, NULL, &thread5, &thread5\_number);

pthread\_create(&pth6, NULL, &thread6, &thread6\_number);

pthread\_join(pth1, NULL);

pthread\_join(pth2, NULL);

pthread\_join(pth3, NULL);

pthread\_join(pth4, NULL);

pthread\_join(pth5, NULL);

pthread\_join(pth6, NULL);

sem\_destroy(&scr1);

fclose(zapic);

return 0;

}

**Результат програми**

Producer thread1: semaphore=0; element 0 creted;

Producer thread1: semaphore=1; element 1 creted;

Producer thread1: semaphore=2; element 2 creted;

Producer thread1: semaphore=3; element 3 creted;

Producer thread1: semaphore=4; element 4 creted;

Producer thread1: semaphore=5; element 5 creted;

Producer thread1: semaphore=6; element 6 creted;

Producer thread1: semaphore=7; element 7 creted;

Producer thread1: semaphore=8; element 8 creted;

Producer thread1: semaphore=9; element 9 creted;

Producer thread1: semaphore=10; element 10 creted;

Producer thread1: semaphore=11; element 11 creted;

Producer thread1: semaphore=11; element 12 creted;

Producer thread1: semaphore=12; element 13 creted;

Producer thread1: semaphore=13; element 14 creted;

Producer thread1: semaphore=14; element 15 creted;

Producer thread1: semaphore=15; element 16 creted;

Producer thread1: semaphore=16; element 17 creted;

Producer thread1: semaphore=17; element 18 creted;

Producer thread1: semaphore=18; element 19 creted;

Producer thread1: semaphore=19; element 20 creted;

Producer thread1: semaphore=20; element 21 creted;

Producer thread1: semaphore=21; element 22 creted;

Producer thread1: semaphore=22; element 23 creted;

Producer thread1: semaphore=23; element 24 creted;

Producer thread1: semaphore=24; element 25 creted;

Producer thread1: semaphore=25; element 26 creted;

Producer thread1: semaphore=26; element 27 creted;

Producer thread1: semaphore=27; element 28 creted;

Producer thread1: semaphore=28; element 29 creted;

Producer thread1: semaphore=29; element 30 creted;

Consumer thread2: semaphore=10; element 0 taken;

Consumer thread4: semaphore=29; element 1 taken;

Consumer thread5: semaphore=28; element 2 taken;

Producer thread1: semaphore=29; element 31 creted;

Producer thread1: semaphore=29; element 32 creted;

Producer thread1: semaphore=29; element 33 creted;

Consumer thread2: semaphore=28; element 3 taken;

Consumer thread4: semaphore=29; element 4 taken;

Consumer thread5: semaphore=28; element 5 taken;

Consumer thread2: semaphore=27; element 6 taken;

Consumer thread4: semaphore=26; element 7 taken;

Consumer thread5: semaphore=25; element 8 taken;

Producer thread1: semaphore=27; element 34 creted;

Producer thread1: semaphore=26; element 35 creted;

Consumer thread2: semaphore=25; element 9 taken;

Consumer thread4: semaphore=25; element 10 taken;

Consumer thread5: semaphore=24; element 11 taken;

Producer thread1: semaphore=26; element 36 creted;

Producer thread1: semaphore=25; element 37 creted;

Producer thread1: semaphore=26; element 38 creted;

Producer thread1: semaphore=27; element 39 creted;

Producer thread1: semaphore=27; element 40 creted;

Producer thread1: semaphore=28; element 41 creted;

Producer thread1: semaphore=29; element 42 creted;

Consumer thread2: semaphore=26; element 12 taken;

Consumer thread4: semaphore=29; element 13 taken;

Consumer thread5: semaphore=28; element 14 taken;

Producer thread1: semaphore=28; element 43 creted;

Producer thread1: semaphore=29; element 44 creted;

Consumer thread2: semaphore=29; element 15 taken;

Consumer thread4: semaphore=28; element 16 taken;

Consumer thread5: semaphore=27; element 17 taken;

Producer thread1: semaphore=29; element 45 creted;

Producer thread1: semaphore=28; element 46 creted;

Producer thread1: semaphore=29; element 47 creted;

Consumer thread2: semaphore=29; element 18 taken;

Consumer thread4: semaphore=28; element 19 taken;

Consumer thread5: semaphore=27; element 20 taken;

Producer thread1: semaphore=29; element 48 creted;

Producer thread1: semaphore=28; element 49 creted;

Producer thread1: semaphore=29; element 50 creted;

Consumer thread2: semaphore=29; element 21 taken;

Consumer thread4: semaphore=28; element 22 taken;

Consumer thread5: semaphore=27; element 23 taken;

Producer thread1: semaphore=29; element 51 creted;

Producer thread1: semaphore=28; element 52 creted;

Producer thread1: semaphore=29; element 53 creted;

Producer thread1: semaphore=29; element 54 creted;

Consumer thread2: semaphore=28; element 24 taken;

Consumer thread4: semaphore=29; element 25 taken;

Consumer thread5: semaphore=28; element 26 taken;

Producer thread1: semaphore=28; element 55 creted;

Producer thread1: semaphore=29; element 56 creted;

Consumer thread2: semaphore=29; element 27 taken;

Consumer thread4: semaphore=28; element 28 taken;

Consumer thread5: semaphore=27; element 29 taken;

Producer thread1: semaphore=29; element 57 creted;