****

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”**

**Факультет прикладної математики**

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

**Лабораторна робота №4**

**з дисципліни “Паралельні та розподільні обчислення”**

**Варіант 5**

**Виконав:**

**студент II-го курсу**

**групи КВ-41**

**Горпинич-Радуженко Іван**

**Київ 2016**

**Постановка завдання та вимоги до виконання програми**

1. Написати програму, яка реалізує роботу паралельних потоків згідно заданого варіанту.

Особливості реалізації синхронізації паралельних потоків та взаємного виключення потоків при доступі до спільного ресурсу задані за варіантами у таблиці завдань.

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 Sem1, Sem2 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:

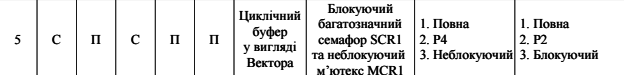
• примусовим перериванням за допомогою натиснення комбінації клавіш Ctrl+C;

• оператором виходу з циклу break при виконанні умови, яка стає істинною, коли буфер

спільного ресурсу повністю заповнюється і повністю звільняється мінімум по два рази.

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

10. Виконати налагодження написаної програми.



**Текст програми мовою С:**

**#include** <stdio.h>

**#include** <unistd.h>

**#include** <pthread.h>

**#include** <semaphore.h>

pthread\_t P1, P2, P3, P4, P5;

**#define** N 10

**int** CR1[N];

**int** inp = 0, out = 0;

**int** prod\_breakflag = 0, cons\_breakflag = 0;

pthread\_mutex\_t MCR1 = PTHREAD\_MUTEX\_INITIALIZER;

sem\_t SCR1;

sem\_t Sem1, Sem2;

FILE\* f;

**void**\* **thread1\_func** (**void**\* unused) {

**int** sem\_value, read\_value;

**fprintf**(f, "Consumer thread 1 started!\n");

**pthread\_setcanceltype**(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

**while** (1) {

**if** (cons\_breakflag == 2) **break**;

**sem\_wait**(&SCR1);

**if** (**pthread\_mutex\_trylock**(&MCR1)==0)

{

read\_value = CR1[out];

**if** (out == N-1) {

out = 0;

++cons\_breakflag;

}

**else** ++out;

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Thread 1 read %d from the array; semaphore = %d\n",

read\_value, sem\_value);

**pthread\_mutex\_unlock**(&MCR1);

}

**usleep**(1);

}

**pthread\_cancel**(P3);

**fprintf**(f, "Consumer threads 1 and 3 stopped!\n");

**return** NULL;

}

**void**\* **thread2\_func** (**void**\* arg) {

**int** sem\_value, write\_value;

**fprintf**(f, "Producer thread 2 started!\n");

**pthread\_setcanceltype**(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

**while** (1) {

**if** (prod\_breakflag == 2) **break**;

**sem\_getvalue**(&SCR1, &sem\_value);

**if** (sem\_value < N) {

**sem\_post**(&Sem1);

**sem\_wait**(&Sem2);

**if** (**pthread\_mutex\_trylock**(&MCR1)==0)

{

write\_value = inp;

CR1[inp] = write\_value;

**if** (inp == N-1) {

inp = 0;

++prod\_breakflag;

}

**else** ++inp;

**sem\_post**(&SCR1);

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Thread 2 wrote %d into the array; semaphore = %d\n",

write\_value, sem\_value);

**pthread\_mutex\_unlock**(&MCR1);

}

}

**usleep**(1);

}

**pthread\_cancel**(P4);

**pthread\_cancel**(P5);

**fprintf**(f, "Producer threads 2, 4 and 5 stopped!\n");

**return** NULL;

}

**void**\* **thread3\_func** (**void**\* unused) {

**int** sem\_value, read\_value;

**fprintf**(f, "Consumer thread 3 started!\n");

**pthread\_setcanceltype**(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

**while** (1) {

**if** (cons\_breakflag == 2) **break**;

**sem\_wait**(&SCR1);

**if** (**pthread\_mutex\_trylock**(&MCR1)==0)

{

read\_value = CR1[out];

**if** (out == N-1) {

out = 0;

++cons\_breakflag;

}

**else** ++out;

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Thread 3 read %d from the array; semaphore = %d\n",

read\_value, sem\_value);

**pthread\_mutex\_unlock**(&MCR1);

}

**usleep**(1);

}

**pthread\_cancel**(P1);

**fprintf**(f, "Consumer threads 1 and 3 stopped!\n");

**return** NULL;

}

**void**\* **thread4\_func** (**void**\* arg) {

**int** sem\_value, write\_value;

**fprintf**(f, "Producer thread 4 started!\n");

**pthread\_setcanceltype**(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

**while** (1) {

**if** (prod\_breakflag == 2) **break**;

**sem\_getvalue**(&SCR1, &sem\_value);

**if** (sem\_value < N) {

**sem\_post**(&Sem2);

**sem\_wait**(&Sem1);

**if** (**pthread\_mutex\_lock**(&MCR1)==0)

{

write\_value = inp;

CR1[inp] = write\_value;

**if** (inp == N-1) {

inp = 0;

++prod\_breakflag;

}

**else** ++inp;

**sem\_post**(&SCR1);

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Thread 4 wrote %d into the array; semaphore = %d\n",

write\_value, sem\_value);

**pthread\_mutex\_unlock**(&MCR1);

}

}

**usleep**(1);

}

**pthread\_cancel**(P2);

**pthread\_cancel**(P5);

**fprintf**(f, "Producer threads 2, 4 and 5 stopped!\n");

**return** NULL;

}

**void**\* **thread5\_func** (**void**\* arg) {

**int** sem\_value, write\_value;

**fprintf**(f, "Producer thread 5 started!\n");

**pthread\_setcanceltype**(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

**while** (1) {

**if** (prod\_breakflag == 2) **break**;

**sem\_getvalue**(&SCR1, &sem\_value);

**if** (sem\_value < N) {

**if** (**pthread\_mutex\_lock**(&MCR1)==0)

{

write\_value = inp;

CR1[inp] = write\_value;

**if** (inp == N-1) {

inp = 0;

++prod\_breakflag;

}

**else** ++inp;

**sem\_post**(&SCR1);

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Thread 5 wrote %d into the array; semaphore = %d\n",

write\_value, sem\_value);

**pthread\_mutex\_unlock**(&MCR1);

}

}

**usleep**(1);

}

**pthread\_cancel**(P2);

**pthread\_cancel**(P4);

**fprintf**(f, "Producer threads 2, 4 and 5 stopped!\n");

**return** NULL;

}

**int** **main** () {

**int** sem\_value, p1\_numb = 1, p2\_numb = 2, p3\_numb = 3, p4\_numb = 4, p5\_numb = 5;

f = **fopen**("log.txt", "w");

**sem\_init**(&SCR1, 0, 0);

**sem\_init**(&Sem1, 0, 1);

**sem\_init**(&Sem2, 0, 0);

**sem\_getvalue**(&SCR1, &sem\_value);

**fprintf**(f, "Initial installation: semaphore = %d\n", sem\_value);

**pthread\_create**(&P1, NULL, &thread1\_func, NULL);

**pthread\_create**(&P2, NULL, &thread2\_func, NULL);

**pthread\_create**(&P3, NULL, &thread3\_func, NULL);

**pthread\_create**(&P4, NULL, &thread4\_func, NULL);

**pthread\_create**(&P5, NULL, &thread5\_func, NULL);

**pthread\_join**(P1, NULL);

**pthread\_join**(P2, NULL);

**pthread\_join**(P3, NULL);

**pthread\_join**(P4, NULL);

**pthread\_join**(P5, NULL);

**fclose**(f);

**printf**("Program finished its work!");

**return** 0;

}

**Протоколи програми**

1. Initial installation: semaphore = 0

Producer thread 5 started!

Thread 5 wrote 0 into the array; semaphore = 1

Producer thread 4 started!

Thread 4 wrote 1 into the array; semaphore = 2

Consumer thread 3 started!

Thread 3 read 0 from the array; semaphore = 1

Producer thread 2 started!

Thread 2 wrote 2 into the array; semaphore = 2

Consumer thread 1 started!

Thread 1 read 1 from the array; semaphore = 1

Thread 5 wrote 3 into the array; semaphore = 2

Thread 4 wrote 4 into the array; semaphore = 3

Thread 3 read 2 from the array; semaphore = 2

Thread 2 wrote 5 into the array; semaphore = 3

Thread 1 read 3 from the array; semaphore = 2

Thread 5 wrote 6 into the array; semaphore = 3

Thread 4 wrote 7 into the array; semaphore = 4

Thread 5 wrote 8 into the array; semaphore = 5

Thread 3 read 4 from the array; semaphore = 4

Thread 2 wrote 9 into the array; semaphore = 5

Thread 4 wrote 0 into the array; semaphore = 6

Thread 1 read 5 from the array; semaphore = 5

Thread 5 wrote 1 into the array; semaphore = 6

Thread 3 read 6 from the array; semaphore = 5

Thread 2 wrote 2 into the array; semaphore = 6

Thread 4 wrote 3 into the array; semaphore = 7

Thread 1 read 7 from the array; semaphore = 6

Thread 5 wrote 4 into the array; semaphore = 7

Thread 3 read 8 from the array; semaphore = 6

Thread 2 wrote 5 into the array; semaphore = 7

Thread 4 wrote 6 into the array; semaphore = 8

Thread 1 read 9 from the array; semaphore = 7

Thread 5 wrote 7 into the array; semaphore = 8

Thread 3 read 0 from the array; semaphore = 7

Thread 2 wrote 8 into the array; semaphore = 8

Thread 4 wrote 9 into the array; semaphore = 9

Thread 1 read 1 from the array; semaphore = 8

Producer threads 2, 4 and 5 stopped!

Thread 3 read 2 from the array; semaphore = 7

Thread 1 read 3 from the array; semaphore = 6

Thread 3 read 4 from the array; semaphore = 5

Thread 1 read 5 from the array; semaphore = 4

Thread 3 read 6 from the array; semaphore = 3

Thread 1 read 7 from the array; semaphore = 2

Thread 3 read 8 from the array; semaphore = 1

Thread 1 read 9 from the array; semaphore = 0

Consumer threads 1 and 3 stopped!

2. Initial installation: semaphore = 0

Producer thread 2 started!

Consumer thread 3 started!

Producer thread 4 started!

Thread 2 wrote 0 into the array; semaphore = 1

Consumer thread 1 started!

Thread 1 read 0 from the array; semaphore = 0

Producer thread 5 started!

Thread 5 wrote 1 into the array; semaphore = 1

Thread 3 read 1 from the array; semaphore = 0

Thread 4 wrote 2 into the array; semaphore = 1

Thread 3 read 2 from the array; semaphore = 0

Thread 2 wrote 3 into the array; semaphore = 1

Thread 1 read 3 from the array; semaphore = 0

Thread 4 wrote 4 into the array; semaphore = 1

Thread 5 wrote 5 into the array; semaphore = 2

Thread 1 read 4 from the array; semaphore = 1

Thread 3 read 5 from the array; semaphore = 0

Thread 5 wrote 6 into the array; semaphore = 1

Thread 4 wrote 7 into the array; semaphore = 2

Thread 1 read 6 from the array; semaphore = 1

Thread 5 wrote 8 into the array; semaphore = 2

Thread 3 read 7 from the array; semaphore = 1

Thread 4 wrote 9 into the array; semaphore = 2

Thread 1 read 8 from the array; semaphore = 1

Thread 2 wrote 0 into the array; semaphore = 2

Thread 5 wrote 1 into the array; semaphore = 3

Thread 4 wrote 2 into the array; semaphore = 4

Thread 3 read 9 from the array; semaphore = 3

Thread 1 read 0 from the array; semaphore = 2

Thread 5 wrote 3 into the array; semaphore = 3

Thread 4 wrote 4 into the array; semaphore = 4

Thread 1 read 1 from the array; semaphore = 3

Thread 3 read 2 from the array; semaphore = 2

Thread 5 wrote 5 into the array; semaphore = 3

Thread 2 wrote 6 into the array; semaphore = 4

Thread 4 wrote 7 into the array; semaphore = 5

Thread 1 read 3 from the array; semaphore = 4

Thread 5 wrote 8 into the array; semaphore = 5

Thread 3 read 4 from the array; semaphore = 4

Thread 1 read 5 from the array; semaphore = 3

Thread 2 wrote 9 into the array; semaphore = 4

Thread 3 read 6 from the array; semaphore = 3

Thread 1 read 7 from the array; semaphore = 2

Thread 3 read 8 from the array; semaphore = 1

Thread 1 read 9 from the array; semaphore = 0

Producer threads 2, 4 and 5 stopped!

Consumer threads 1 and 3 stopped!