Міністерство освіти та науки України
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»

Група: КВ-33

Виконав: Лашков Д. В.

Оцінка:

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.

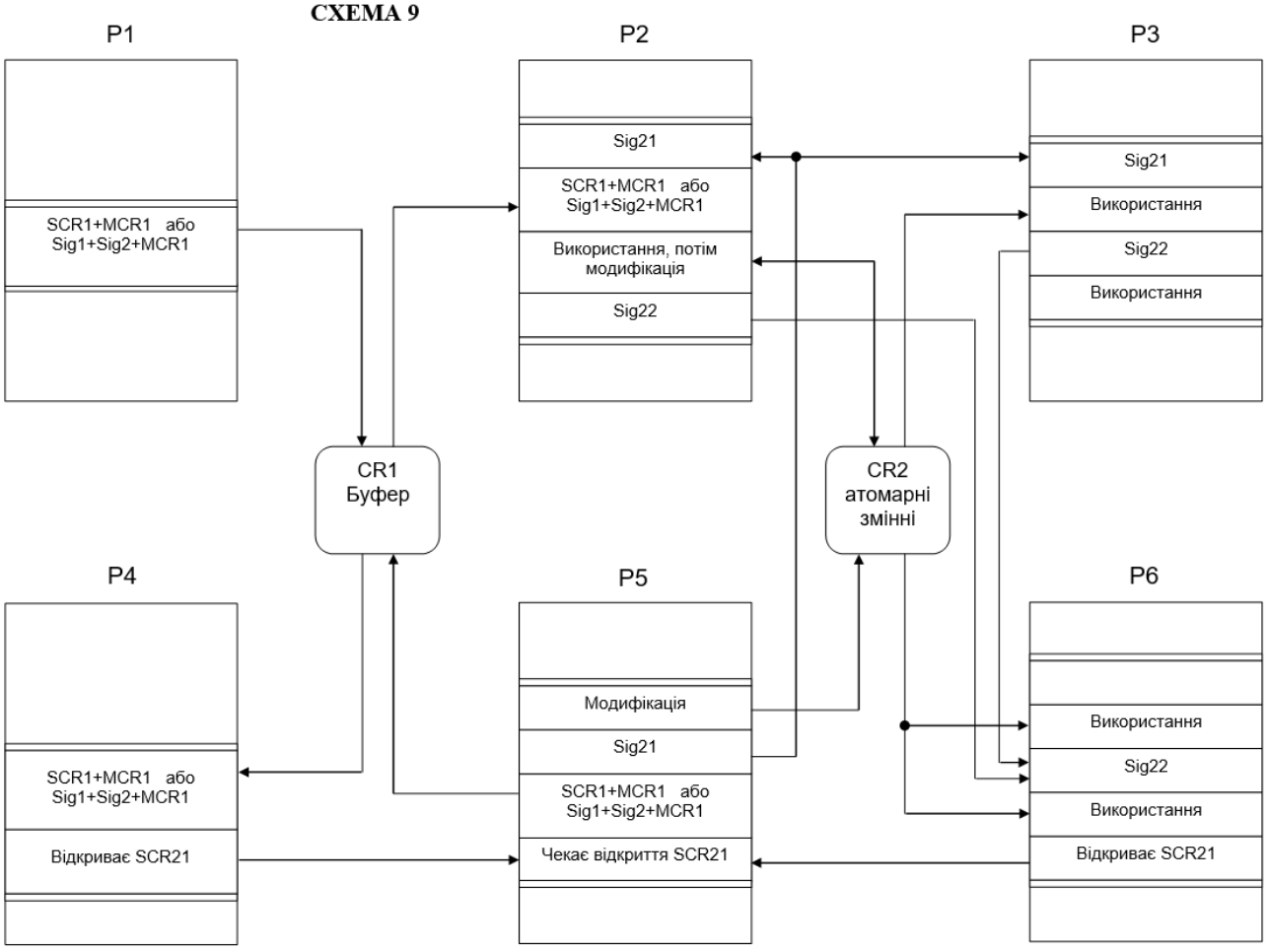
10. Якщо при реалізації паралельних потоків була використана функція usleep(), то передбачити режим запуску програми з «відключеними» функціями usleep().
11. Виконати налагодження написаної програми.

Варіант 9

9	9	Стек (Вектор)	Дві сигнальні (умовні) змінні Sig1 та Sig2 і неблокуючий м'ютекс MCR1	2, 8, 3, 6, 10, 11, 13, 14	Блокуючий	–	–	Багато-значний	Одинич-ний

2.	*type* **__atomic_sub_fetch** *(type *ptr, type val, int memorder)*
3.	*type* **__atomic_and_fetch** *(type *ptr, type val, int memorder)*
6.	*type* **__atomic_nand_fetch** *(type *ptr, type val, int memorder)*
8.	*type* **__atomic_fetch_sub** *(type *ptr, type val, int memorder)*
10.	*type* **__atomic_fetch_xor** *(type *ptr, type val, int memorder)*
11.	*type* **__atomic_fetch_or** *(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)*

СХЕМА 9



Текст програми

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>

FILE *f;

pthread_t thread1;
pthread_t thread2;
pthread_t thread3;
pthread_t thread4;
pthread_t thread5;
pthread_t thread6;

int buffer_clear_loops = 2;
int buffer_full_loops = 2;

int sem_value;
#define max_stack_length 20
int arr[max_stack_length];

int int_1=1, int_2=2;
unsigned uint_1=3, uint_2=4;
long long_1=5, long_2=6;
long unsigned ulong_1=7, ulong_2=8;

int curr_stack_elem =0;
int flag_sig21_p2=0,flag_sig21_p3=0, flag_sig22=0;
pthread_mutex_t  mcr1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mcr21 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mcr22 = PTHREAD_MUTEX_INITIALIZER;

sem_t scr21;

pthread_cond_t   sig21 = PTHREAD_COND_INITIALIZER;
pthread_cond_t   sig22  = PTHREAD_COND_INITIALIZER;

pthread_cond_t   sig1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t   sig2  = PTHREAD_COND_INITIALIZER;


int is_full()
{
    return curr_stack_elem >= max_stack_length-1;
}

int is_empty()
{
    return curr_stack_elem <0;
}

void add_elem()
{
    arr[curr_stack_elem] = curr_stack_elem + 1;
}

int get_elem()
{
    return arr[curr_stack_elem];
}
```

```
void actions_for_elem(int elem, int index)
{
    fprintf(f, "arr[%d]=%d\n\n\n",index, elem);
}

void use_cr2(int num)
{
    fprintf(f, "Thread %d using atomic variables: %d, %d, %d, %d, %ld, %ld,
%ld, %ld!\n", num, int_1, int_2, uint_1, uint_2, long_1, long_2, ulong_1,
ulong_2);
}

void mod_cr2(int num)
{
    fprintf(f, "Thread%d modificating atomic variables!\n", num);
    __atomic_sub_fetch(&int_1, 1, __ATOMIC_RELAXED);
    __atomic_and_fetch(&int_2, 2, __ATOMIC_RELAXED);
    __atomic_nand_fetch(&uint_1, 3, __ATOMIC_RELAXED);
    __atomic_fetch_sub(&uint_2, 4, __ATOMIC_RELAXED);
    __atomic_fetch_xor(&long_1, 5, __ATOMIC_RELAXED);
    __atomic_fetch_or(&long_2, 6, __ATOMIC_RELAXED);
    __atomic_compare_exchange_n(&ulong_1, &ulong_2, 7, 1, __ATOMIC_RELAXED,
__ATOMIC_RELAXED);
    __atomic_exchange(&long_1, &long_2, &ulong_2, __ATOMIC_RELAXED);
    fprintf(f, "Thread%d modified atomic variables!\n", num);
}

void* thread_producer_P1(void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Producer thread%d  created !!!\n",num);
    while (1)
    {
        if(pthread_mutex_trylock(&mcr1)==0)
        {
            while (is_full())
            {
                pthread_cond_wait (&sig2, &mcr1);
            }
            curr_stack_elem++;
            add_elem();

            fprintf(f, "Producer thread%d: element %d CREATED;\n",num,
curr_stack_elem);

            pthread_cond_signal(&sig1);

            if (is_full())
            {
                buffer_full_loops--;
                fprintf(f, "The buffer is full! buffer_full_loops= %d \n",
buffer_full_loops);
            }

            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
            {
                pthread_cancel(thread2);
                pthread_cancel(thread3);
                pthread_cancel(thread4);
                pthread_cancel(thread5);
                pthread_cancel(thread6);
                pthread_cond_broadcast(&sig21);
```

```
                fprintf(f, "Producer thread%d  stopped !!!\n",num);
                pthread_mutex_unlock (&mcr1);
                break;
            }
            pthread_mutex_unlock (&mcr1);
        }
        else
        {
            fprintf(f, "Thread%d doing some cool staff instead of some
usefull\n",num);
            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
                break;
        }

        //usleep(1);

    }  // while (1)


    return NULL;
}

void* thread_consumer_P2 (void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Consumer thread%d  created !!!\n",num);

    int curr_elem=1;
    int curr_index=0;

    while (1)
    {
        pthread_mutex_lock(&mcr21);
        fprintf(f, "\nThread%d wait sig21\n", num);
        while(flag_sig21_p2==0)
            pthread_cond_wait(&sig21, &mcr21);
        flag_sig21_p2=0;
        fprintf(f, "\nSignal sig21 is delivered in thread%d!\n", num);
        pthread_mutex_unlock (&mcr21);

        if(pthread_mutex_trylock (&mcr1)==0)
        {
            while (is_empty())
            {
                pthread_cond_wait (&sig1, &mcr1);
            }

            curr_elem = get_elem();
            curr_index= curr_stack_elem;
            curr_stack_elem--;
            fprintf(f, "Consumer thread%d: element %d TAKEN;\n",num,
curr_index);

            actions_for_elem (curr_elem, curr_index);

            arr[curr_index]=-2;
            pthread_cond_signal (&sig2);

            if (is_empty())
            {
                buffer_clear_loops--;
```

```
                fprintf(f, "The buffer is empty! buffer_clear_loop= %d \n",
buffer_clear_loops);
            }

            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
            {
                pthread_cancel(thread1);
                pthread_cancel(thread3);
                pthread_cancel(thread4);
                pthread_cancel(thread5);
                pthread_cancel(thread6);
                fprintf(f, "Consumer thread%d  stopped !!!\n",num);
                pthread_mutex_unlock (&mcr1);
                break;
            }
            pthread_mutex_unlock (&mcr1);

        }

        else
        {
            fprintf(f, "Thread%d doing some cool staff instead of some
usefull\n",num);
            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
                break;
        }
        use_cr2(num);
        mod_cr2(num);

        pthread_mutex_lock(&mcr22);
        flag_sig22=1;
        pthread_cond_signal(&sig22);
        fprintf(f, "\nSignal sig22 is sent!\n");
        pthread_mutex_unlock (&mcr22);
        // usleep(1);

    }  // while (1)


    return NULL;
}
void* thread_P3 (void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Thread%d  created !!!\n",num);

    while (1)
    {
        pthread_mutex_lock(&mcr21);
        fprintf(f, "\nThread%d wait sig21\n", num);
        while(flag_sig21_p3==0)
            pthread_cond_wait(&sig21, &mcr21);
        flag_sig21_p3=0;
        fprintf(f, "\nSignal sig21 is delivered in thread%d!\n", num);
        pthread_mutex_unlock (&mcr21);

        use_cr2(num);

        pthread_mutex_lock(&mcr22);
        flag_sig22=1;
        pthread_cond_signal(&sig22);
        fprintf(f, "\nSignal sig22 is sent!\n");
```

```
        pthread_mutex_unlock (&mcr22);

        use_cr2(num);
        // usleep(1);

    }  // while (1)

    return NULL;
}


void* thread_consumer_P4 (void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Consumer thread%d  created !!!\n",num);

    int curr_elem=0;
    int curr_index=0;

    while (1)
    {
        if(pthread_mutex_trylock(&mcr1)==0)
        {
            while (is_empty())
            {
                pthread_cond_wait (&sig1, &mcr1);
            }

            curr_elem = get_elem();
            curr_index= curr_stack_elem;
            curr_stack_elem--;
            fprintf(f, "Consumer thread%d: element %d TAKEN;\n",num,
curr_index);


            actions_for_elem (curr_elem, curr_index);

            arr[curr_index]=-2;
            pthread_cond_signal(&sig2);
            if (is_empty())
            {
                buffer_clear_loops--;
                fprintf
(f, "The buffer is empty! buffer_clear_loop= %d \n", buffer_clear_loops);
            }

            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
            {
                pthread_cancel(thread1);
                pthread_cancel(thread2);
                pthread_cancel(thread3);
                pthread_cancel(thread5);
                pthread_cancel(thread6);
                pthread_mutex_unlock (&mcr21);
                fprintf(f, "Concumer thread%d  stopped !!!\n",num);
                pthread_mutex_unlock (&mcr1);
                break;
            }
            pthread_mutex_unlock (&mcr1);
        }
        else
        {
```

```
            fprintf(f, "Thread%d doing some cool staff instead of some
usefull\n",num);
            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
                break;
        }

        sem_getvalue(&scr21, &sem_value);
        if (sem_value == 0)
        {
            sem_post(&scr21);
            fprintf(f, "\nSemaphore is open!:%d\n",sem_value );
        }

        // usleep(1);

    }  // while (1)


    return NULL;
}

void* thread_producer_P5(void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Producer thread%d  created !!!\n",num);
    while (1)
    {
        pthread_mutex_lock(&mcr21);
        flag_sig21_p2=1;
        flag_sig21_p3=1;
        pthread_cond_broadcast(&sig21);
        fprintf(f, "\nSignal sig21 is sent!\n");
        pthread_mutex_unlock (&mcr21);

        if(pthread_mutex_trylock(&mcr1)==0)
        {
            mod_cr2(num);

            while (is_full())
            {
                pthread_cond_wait (&sig2, &mcr1);
            }
            curr_stack_elem++;
            add_elem();

            fprintf(f, "Producer thread%d: element %d CREATED;\n",num,
curr_stack_elem);

            pthread_cond_signal(&sig1);
            if (is_full())
            {
                buffer_full_loops--;
                fprintf(f, "The buffer is full! buffer_full_loops= %d \n",
buffer_full_loops);
            }

            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
            {
                pthread_cancel(thread1);
                pthread_cancel(thread2);
                pthread_cancel(thread3);
                pthread_cancel(thread4);
```

```
                pthread_cancel(thread6);
                fprintf(f, "Producer thread%d  stopped !!!\n",num);
                pthread_mutex_unlock (&mcr1);
                break;
            }
            pthread_mutex_unlock (&mcr1);
        }
        else
        {
            fprintf(f, "Thread%d doing some cool staff instead of some
usefull\n",num);
            if(buffer_clear_loops<=0 && buffer_full_loops<=0)
                break;
        }
        fprintf(f, "\nWait scr21\n");
        sem_wait (&scr21);
        fprintf(f, "Semaphore scr21 is closed\n");

        //usleep(1);

    }  // while (1)

    return NULL;
}

void* thread_P6(void* arg)
{
    int num = *(int*)arg;
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    fprintf(f, "Producer thread%d  created !!!\n",num);
    while (1)
    {
        use_cr2(num);

        pthread_mutex_lock(&mcr22);
        fprintf(f, "\nThread%d wait sig22\n", num);
        while(flag_sig22==0)
            pthread_cond_wait(&sig21, &mcr22);
        flag_sig22=0;
        fprintf(f, "\nSignal sig22 is delivered in thread%d!\n", num);
        pthread_mutex_unlock (&mcr22);

        use_cr2(num);
        sem_getvalue(&scr21, &sem_value);
        if (sem_value == 0)
        {
            sem_post(&scr21);
            fprintf(f, "\nSemaphore is open!:%d\n",sem_value );
        }



        //usleep(1);

    }  // while (1)

    return NULL;
}

int main()
{
    if ((f = fopen("log.txt", "w")) == NULL)
    {
```

```
        perror("Could not open the file.");
        return 1;
    }
    sem_init(&scr21, 0, 0);



    int length_at_start=10;
    int i;

    for(i=0; i<length_at_start; i++)
    {
        arr[i] = i + 1;
        fprintf(f, "arr[%d]=%d\n", i, arr[i]);
    }
    curr_stack_elem =i-1;

    fprintf(f, "Stack with elements from 0-th to %d-th has been created
!!!\n", length_at_start-1);
    fprintf(f, "index_main = %d;\n",curr_stack_elem);

    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,&thread_producer_P1,(void*)&thread1_number);
    pthread_create
(&thread2,NULL,&thread_consumer_P2,(void*)&thread2_number);
    pthread_create (&thread3,NULL,&thread_P3,(void*)&thread3_number);
    pthread_create
(&thread4,NULL,&thread_consumer_P4,(void*)&thread4_number);
    pthread_create
(&thread5,NULL,&thread_producer_P5,(void*)&thread5_number);
    pthread_create (&thread6,NULL,&thread_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);
    fprintf(f, "All threads stopped!!!\n");

    pthread_mutex_destroy (&mcr1);
    pthread_mutex_destroy (&mcr21);
    pthread_mutex_destroy (&mcr22);

    pthread_cond_destroy(&sig1);
    pthread_cond_destroy(&sig2);
    pthread_cond_destroy(&sig21);
    pthread_cond_destroy(&sig22);

    fprintf(f, "END!!!\n");
    fclose(f);

    return 0;
}
```

Тестування програми

arr[0]=1

arr[1]=2

arr[2]=3

arr[3]=4

arr[4]=5

arr[5]=6

arr[6]=7

arr[7]=8

arr[8]=9

arr[9]=10

Stack with elements from 0-th to 9-th has been created !!!

index_main = 9;

Producer thread6 created !!!

Thread 6 using atomic variables: 1, 2, 3, 4, 5, 6, 7, 8!

Thread6 wait sig22

Producer thread5 created !!!

Signal sig21 is sent!

Thread5 modificating atomic variables!

Thread5 modified atomic variables!

Producer thread5: element 10 CREATED;

Wait scr21

Consumer thread4 created !!!

Consumer thread4: element 10 TAKEN;

arr[10]=11

Semaphore is open!

Consumer thread4: element 9 TAKEN;

arr[9]=10

Consumer thread4: element 8 TAKEN;

arr[8]=9

Consumer thread4: element 7 TAKEN;

arr[7]=8

Consumer thread4: element 6 TAKEN;

arr[6]=7

Consumer thread4: element 5 TAKEN;

arr[5]=6

Consumer thread4: element 4 TAKEN;

arr[4]=5

Consumer thread4: element 3 TAKEN;

arr[3]=4

Consumer thread4: element 2 TAKEN;

arr[2]=3

Consumer thread4: element 1 TAKEN;

arr[1]=2

Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= 1
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= 0

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -1

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -2

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -3

Semaphore is open!

Semaphore scr21 is closed

Signal sig21 is sent!

Thread5 modificating atomic variables!

Thread5 modified atomic variables!

Producer thread5: element 0 CREATED;

Wait scr21

Consumer thread4: element 0 TAKEN;

arr[0]=1

The buffer is empty! buffer_clear_loop= -4

Semaphore is open!

Semaphore scr21 is closed

Signal sig21 is sent!

Thread5 modificating atomic variables!

Thread5 modified atomic variables!

Producer thread5: element 0 CREATED;

Wait scr21

Consumer thread4: element 0 TAKEN;

arr[0]=1

The buffer is empty! buffer_clear_loop= -5

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -6

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -7

Semaphore is open!

Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -8

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -9

Semaphore is open!
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -10
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -11
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!

Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -12
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -13
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!

Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -14
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -15
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -16
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Semaphore is open!
Consumer thread4: element 0 TAKEN;
arr[0]=1

The buffer is empty! buffer_clear_loop= -17
Semaphore scr21 is closed

Signal sig21 is sent!
Thread5 modificating atomic variables!
Thread5 modified atomic variables!
Producer thread5: element 0 CREATED;

Wait scr21

Producer thread1 created !!!

Semaphore is open!

Consumer thread4: element 0 TAKEN;

arr[0]=1

The buffer is empty! buffer_clear_loop= -18

Semaphore scr21 is closed

Signal sig21 is sent!

Thread5 modificating atomic variables!

Thread5 modified atomic variables!

Producer thread5: element 0 CREATED;

Wait scr21

Producer thread1: element 1 CREATED;

Producer thread1: element 2 CREATED;

Producer thread1: element 3 CREATED;

Producer thread1: element 4 CREATED;

Producer thread1: element 5 CREATED;

Producer thread1: element 6 CREATED;

Producer thread1: element 7 CREATED;

Producer thread1: element 8 CREATED;

Producer thread1: element 9 CREATED;

Producer thread1: element 10 CREATED;

Semaphore is open!

Producer thread1: element 11 CREATED;

Producer thread1: element 12 CREATED;

Producer thread1: element 13 CREATED;

Producer thread1: element 14 CREATED;

Producer thread1: element 15 CREATED;

Producer thread1: element 16 CREATED;

Producer thread1: element 17 CREATED;

Producer thread1: element 18 CREATED;

Producer thread1: element 19 CREATED;

The buffer is full! buffer_full_loops= 1

Consumer thread4: element 19 TAKEN;

arr[19]=20

Semaphore scr21 is closed

Signal sig21 is sent!

Thread5 modificating atomic variables!

Thread5 modified atomic variables!

Producer thread5: element 19 CREATED;

The buffer is full! buffer_full_loops= 0

Consumer thread2 created !!!

Thread2 wait sig21

Signal sig21 is delivered in thread2!

Thread2 doing some cool staff instead of some usefull

Thread3 created !!!

Thread3 wait sig21

Signal sig21 is delivered in thread3!

Thread 3 using atomic variables: -21, 2, -1, -84, 6, 6, 7, 3!

Signal sig22 is sent!

Thread 3 using atomic variables: -21, 2, -1, -84, 6, 6, 7, 3!

Thread3 wait sig21

Producer thread5 stopped !!!

All threads stopped!!!

END!!!