

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

#### Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №4**

з дисципліни

**«Паралельні та розподілені обчислення»**

#### Тема «Засоби взаємодії паралельних потоків операційної системи лінукс»

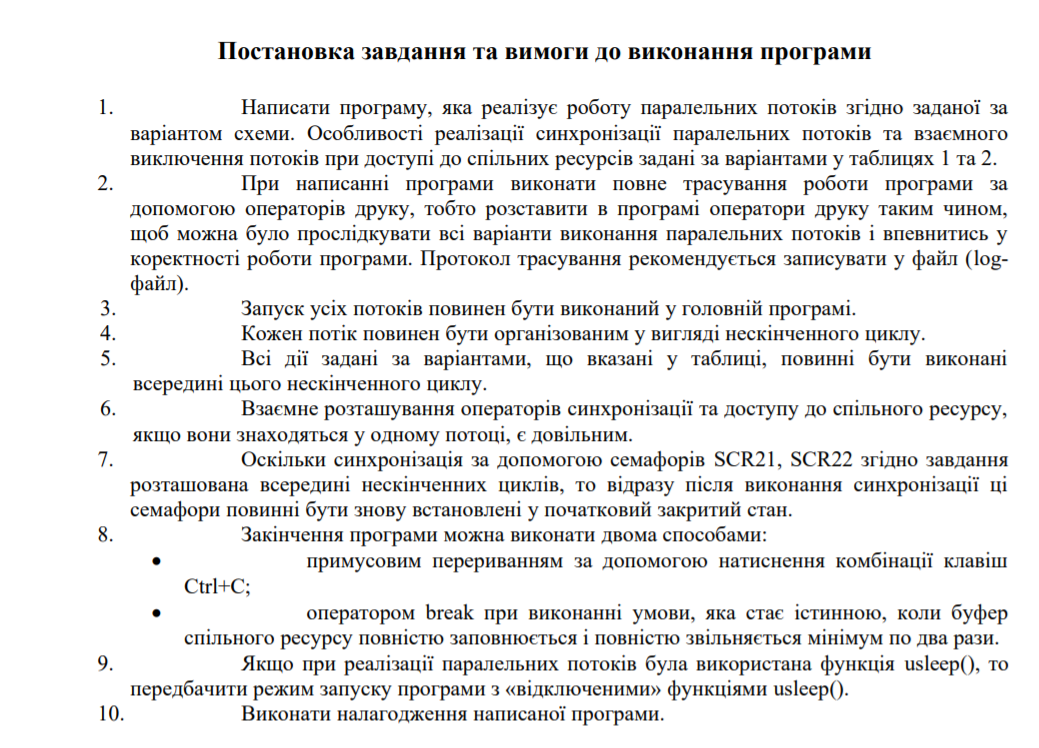
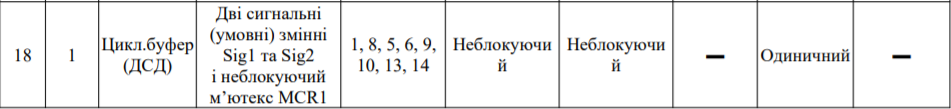
Виконав: студент ІІ курсу

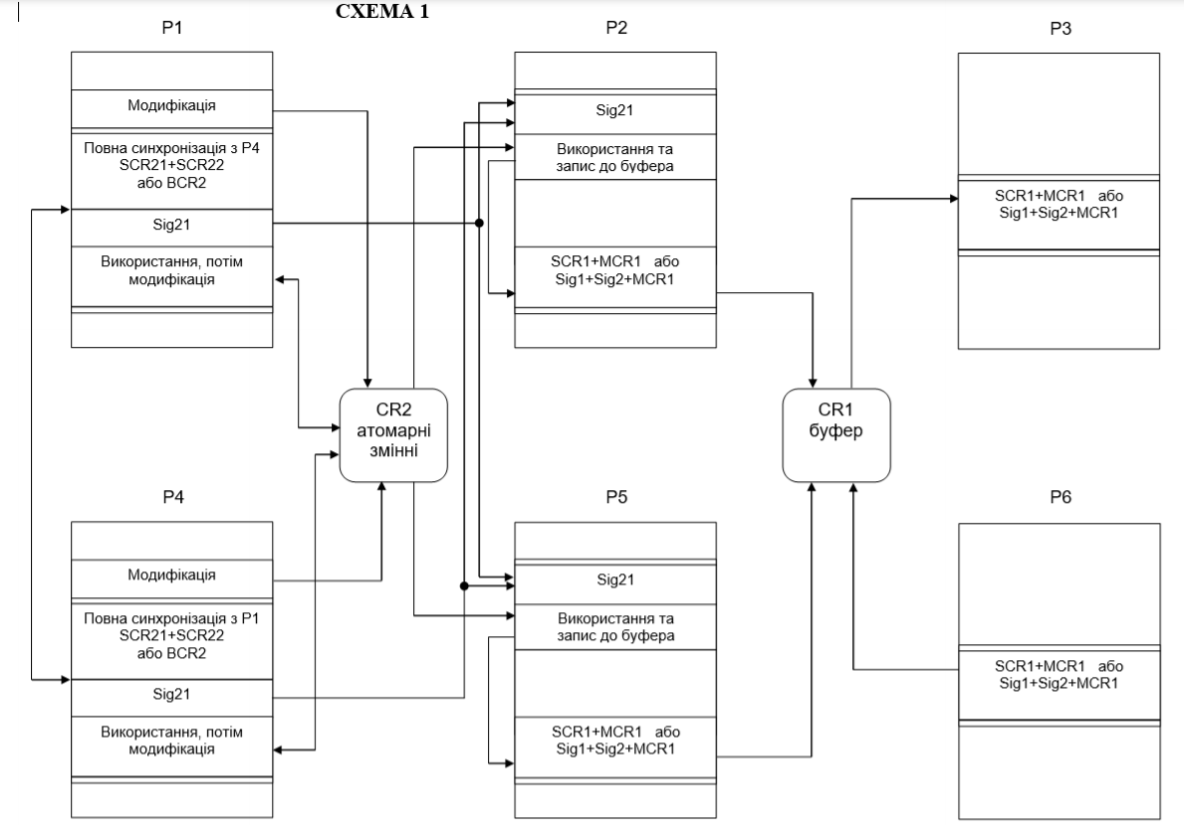
ФПМ групи КВ-91

Селетков В. Р.

Перевірив: Марченко О. О.

Київ – 2021

**Варіант 18.**

****

**Текст програми**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <fcntl.h>

#include <stdlib.h>

//threads

pthread\_t p1;

pthread\_t p2;

pthread\_t p3;

pthread\_t p4;

pthread\_t p5;

pthread\_t p6;

//semaphores

sem\_t scr21;

sem\_t scr22;

//mutex init

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

pthread\_mutex\_t mutex\_for\_sig21 = PTHREAD\_MUTEX\_INITIALIZER;

//conditional signals

pthread\_cond\_t sig1\_not\_empty = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t sig2\_not\_full = PTHREAD\_COND\_INITIALIZER;

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

//file

FILE \*f = NULL;

//dsd bufer values

struct cr1 {

struct cr1 \*next;

int numb;

};

struct cr1 \*next\_in = NULL;

struct cr1 \*next\_out = NULL;

struct cr1 \*begin\_buf = NULL;

struct cr1 \*end\_buf = NULL;

unsigned int n = 200;

unsigned int el\_numb = 0;

unsigned int new\_buf = 0;

//atomic values

int int1\_cr2 = 0;

int int2\_cr2 = 0;

unsigned uns1\_cr2 = 0;

unsigned uns2\_cr2 = 0;

long long1\_cr2 = 0;

long long2\_cr2 = 0;

long unsigned luns1\_cr2 = 0;

long unsigned luns2\_cr2 = 0;

//control bufer

int empty\_bufer = 0;

int full\_bufer = 0;

//bufer function declarations

void DeleteBuf();

void OutputBuf();

int IsBufEmpty();

int IsBufFull();

void AddElemToBuf();

struct cr1 \*GetElemFromBuf();

//atomic function declarations

void AddAtomicToBuf(int value);

void AtomicInput();

void AtomicModification();

void AtomicUsing();

void AtomicOutput();

//threads function declarations

void \*P1thread(void \*p\_number);

void \*P2thread(void \*p\_number);

void \*P3thread(void \*p\_number);

void \*P4thread(void \*p\_number);

void \*P5thread(void \*p\_number);

void \*P6thread(void \*p\_number);

int main() {

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

if (f == NULL) {

printf("\nerror file\n");

exit(1);

}

srand(time(NULL));

for(unsigned int i = 0; i < n / 2; i++)

AddElemToBuf();

fprintf(f, "Cycle buffer with elements from 0-th to %d-th has been created !!!\n", (n / 2));

printf("Cycle buffer with elements from 0-th to %d-th has been created !!!\n", (n / 2));

AtomicInput();

int p1\_number = 1;

int p2\_number = 2;

int p3\_number = 3;

int p4\_number = 4;

int p5\_number = 5;

int p6\_number = 6;

int p1\_join = -1;

int p2\_join = -1;

int p3\_join = -1;

int p4\_join = -1;

int p5\_join = -1;

int p6\_join = -1;

sem\_init(&scr21, 0, 0);

sem\_init(&scr22, 0, 0);

pthread\_create(&p1, NULL, &P1thread, (void\*)&p1\_number);

pthread\_create(&p2, NULL, &P2thread, (void\*)&p2\_number);

pthread\_create(&p3, NULL, &P3thread, (void\*)&p3\_number);

pthread\_create(&p4, NULL, &P4thread, (void\*)&p4\_number);

pthread\_create(&p5, NULL, &P5thread, (void\*)&p5\_number);

pthread\_create(&p6, NULL, &P6thread, (void\*)&p6\_number);

p1\_join = pthread\_join(p1, NULL);

fprintf(f, "\npthread\_join p%d = %d", p1\_number, p1\_join);

printf("\npthread\_join p%d = %d", p1\_number, p1\_join);

p2\_join = pthread\_join(p2, NULL);

fprintf(f, "\npthread\_join p%d = %d", p2\_number, p2\_join);

printf("\npthread\_join p%d = %d", p2\_number, p2\_join);

p3\_join = pthread\_join(p3, NULL);

fprintf(f, "\npthread\_join p%d = %d", p3\_number, p3\_join);

printf("\npthread\_join p%d = %d", p3\_number, p3\_join);

p4\_join = pthread\_join(p4, NULL);

fprintf(f, "\npthread\_join p%d = %d", p4\_number, p4\_join);

printf("\npthread\_join p%d = %d", p4\_number, p4\_join);

p5\_join = pthread\_join(p5, NULL);

fprintf(f, "\npthread\_join p%d = %d", p5\_number, p5\_join);

printf("\npthread\_join p%d = %d", p5\_number, p5\_join);

p6\_join = pthread\_join(p6, NULL);

fprintf(f, "\npthread\_join p%d = %d\n", p6\_number, p6\_join);

printf("\npthread\_join p%d = %d\n", p6\_number, p6\_join);

fprintf(f, "\nAll information was written to the log file\n");

printf("\nAll information was written to the log file\n");

fclose(f);

return 0;

}

void DeleteBuf() {

struct cr1 \*temp = begin\_buf;

struct cr1 \*temp\_free = NULL;

if (temp == NULL)

return;

while(temp != end\_buf) {

temp\_free = temp;

temp = temp->next;

free(temp\_free);

}

if (temp != NULL) {

free(temp);

}

else {

printf("\nCycle buffer error delete\n\n");

fprintf(f, "\nCycle buffer error delete\n\n");

}

fprintf(f, "\nCycle buffer was deleted\n\n");

printf("\nCycle buffer was deleted\n\n");

}

int IsBufEmpty() {

return el\_numb <= 0;

}

int IsBufFull() {

return el\_numb >= n;

}

void AddElemToBuf() {

if (new\_buf < n) {

struct cr1 \*temp = (struct cr1 \*)malloc(sizeof(struct cr1));

temp->next = NULL;

if (next\_in == NULL) {

//temp->numb = rand() % 50;

temp->numb = new\_buf;

next\_in = temp;

next\_out = temp;

begin\_buf = temp;

end\_buf = temp;

new\_buf++;

fprintf(f, "Cycle bufer was created\n");

printf("Cycle bufer was created\n");

}

else {

if (el\_numb > n) {

fprintf(f, "\nerror el\_numb > %d\n", n);

printf("\nerror el\_numb > %d\n", n);

free(temp);

return;

}

//temp->numb = rand() % 50;

temp->numb = new\_buf;

next\_in->next = temp;

next\_in = temp;

new\_buf++;

if (new\_buf == n)

temp->next = begin\_buf;

end\_buf = temp;

}

el\_numb++;

}

else {

next\_in->next->numb = rand() % 50;

if (next\_in == end\_buf)

next\_in = begin\_buf;

else

next\_in = next\_in->next;

el\_numb++;

}

}

struct cr1 \*GetElemFromBuf() {

struct cr1 \*temp = NULL;

if (el\_numb < 0) {

fprintf(f, "\nerror el\_numb < 0\n");

printf("\nerror el\_numb < 0\n");

return NULL;

}

temp = next\_out;

if (temp == end\_buf)

next\_out = begin\_buf;

else

next\_out = next\_out->next;

el\_numb--;

return temp;

}

void OutputBuf() {

struct cr1 \*temp = begin\_buf;

if (temp == NULL)

return;

while (temp != end\_buf) {

fprintf(f, "%d ", temp->numb);

printf("%d ", temp->numb);

temp = temp->next;

}

if (temp != NULL) {

fprintf(f, "%d ", temp->numb);

printf("%d ", temp->numb);

fprintf(f, "\n");

printf("\n");

}

else {

printf("\nCycle buffer error output\n\n");

fprintf(f, "\nCycle buffer error output\n\n");

}

}

void \*P1thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

AtomicModification();

fprintf(f, "\nthread p[%d]: modificated atomic values\n", p\_num);

printf("\nthread p[%d]: modificated atomic values\n", p\_num);

while (1) {

fprintf(f, "thread p[%d]: opens semaphore scr22 for the thread p[4]\n", p\_num);

printf("thread p[%d]: opens semaphore scr22 for the thread p[4]\n", p\_num);

sem\_post(&scr22);

fprintf(f, "\nthread p[%d]: semaphore scr22 is opened\n", p\_num);

printf("\nthread p[%d]: semaphore scr22 is opened\n", p\_num);

fprintf(f, "\nthread p[%d]: waits for the opening of the semaphore scr21\n", p\_num);

printf("\nthread p[%d]: waits for the opening of the semaphore scr21\n", p\_num);

trywait:

if (sem\_trywait(&scr21) == 0) {

sem\_close(&scr21);

sem\_open("scr21", O\_CREAT, 0, 0);

fprintf(f, "thread p[%d]: works after receiving semaphore scr21\n", p\_num);

printf("thread p[%d]: works after receiving semaphore scr21\n", p\_num);

pthread\_mutex\_lock(&mutex\_for\_sig21);

pthread\_cond\_signal(&sig21);

pthread\_mutex\_unlock(&mutex\_for\_sig21);

fprintf(f, "thread p[%d]: sent sig21\n", p\_num);

printf("thread p[%d]: sent sig21\n", p\_num);

AtomicUsing();

fprintf(f, "thread p[%d]: used atomic values\n", p\_num);

printf("thread p[%d]: used atomic values\n", p\_num);

AtomicModification();

fprintf(f, "\nthread p[%d]: modificated atomic values\n", p\_num);

printf("\nthread p[%d]: modificated atomic values\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

goto trywait;

}

}

fprintf(f, "thread p[%d]: stopped\n", p\_num);

printf("thread p[%d]: stopped\n", p\_num);

return NULL;

}

void \*P2thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

int flag = 0;

int old\_state = 0;

while(1) {

fprintf(f, "thread p[%d]: is waiting for sig21\n", p\_num);

printf("thread p[%d]: is waiting for sig21\n", p\_num);

pthread\_mutex\_lock(&mutex\_for\_sig21);

pthread\_cond\_wait(&sig21, &mutex\_for\_sig21);

pthread\_mutex\_unlock(&mutex\_for\_sig21);

fprintf(f, "thread p[%d]: received sig21\n", p\_num);

printf("thread p[%d]: received sig21\n", p\_num);

AtomicUsing();

fprintf(f, "thread p[%d]: used atomic values\n", p\_num);

printf("thread p[%d]: used atomic values\n", p\_num);

if (el\_numb == 0) {

fprintf(f, "thread p[%d]: bufer is empty\n", p\_num);

printf("thread p[%d]: bufer is empty\n", p\_num);

empty\_bufer++;

if (empty\_bufer == 2)

break;

}

if (pthread\_mutex\_trylock(&mcr1) == 0) {

while(IsBufFull())

pthread\_cond\_wait(&sig2\_not\_full, &mcr1);

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, &old\_state);

if (flag == 0)

AddAtomicToBuf((int)int1\_cr2);

else if (flag == 1)

AddAtomicToBuf((int)int2\_cr2);

else if (flag == 2)

AddAtomicToBuf((int)uns1\_cr2);

else if (flag == 3)

AddAtomicToBuf((int)uns2\_cr2);

else if (flag == 4)

AddAtomicToBuf((int)long1\_cr2);

else if (flag == 5)

AddAtomicToBuf((int)long2\_cr2);

else if (flag == 6)

AddAtomicToBuf((int)luns1\_cr2);

else if (flag == 7)

AddAtomicToBuf((int)luns2\_cr2);

pthread\_setcancelstate(old\_state, NULL);

flag++;

if (flag == 8)

flag = 0;

fprintf(f, "\nthread p[%d]: element %d ATOMIC CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

printf("\nthread p[%d]: element %d ATOMIC CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

pthread\_mutex\_unlock(&mcr1);

pthread\_cond\_signal(&sig1\_not\_empty);

fprintf(f, "thread p[%d]: sent sig1\_not\_empty\n", p\_num);

printf("thread p[%d]: sent sig1\_not\_empty\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

}

}

pthread\_cancel(p1);

pthread\_cancel(p3);

pthread\_cancel(p4);

pthread\_cancel(p5);

pthread\_cancel(p6);

DeleteBuf();

fprintf(f, "thread p[%d]: stopped other threads and was stopped too\n", p\_num);

printf("thread p[%d]: stopped other threads and was stopped too\n", p\_num);

return NULL;

}

void \*P3thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

struct cr1 \*temp = NULL;

int old\_state = 0;

while(1) {

temp = 0;

if (el\_numb == n) {

fprintf(f, "thread p[%d]: bufer is full\n", p\_num);

printf("thread p[%d]: bufer is full\n", p\_num);

full\_bufer++;

if (full\_bufer == 2)

break;

}

if (pthread\_mutex\_trylock(&mcr1) == 0) {

while(IsBufEmpty()) {

fprintf(f, "thread p[%d]: is waiting for sig2\_not\_empty\n", p\_num);

printf("thread p[%d]: is waiting for sig2\_not\_empty\n", p\_num);

pthread\_cond\_wait(&sig1\_not\_empty, &mcr1);

}

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, &old\_state);

temp = GetElemFromBuf();

pthread\_setcancelstate(old\_state, NULL);

fprintf(f, "\nthread p[%d]: element %d TAKEN; current el\_numb = %d;\n", p\_num, temp->numb, el\_numb);

printf("\nthread p[%d]: element %d TAKEN; current el\_numb = %d;\n", p\_num, temp->numb, el\_numb);

pthread\_mutex\_unlock(&mcr1);

pthread\_cond\_broadcast(&sig2\_not\_full);

fprintf(f, "thread p[%d]: sent sig2\_not\_full\n", p\_num);

printf("thread p[%d]: sent sig2\_not\_full\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

}

}

pthread\_cancel(p1);

pthread\_cancel(p2);

pthread\_cancel(p4);

pthread\_cancel(p5);

pthread\_cancel(p6);

DeleteBuf();

fprintf(f, "thread p[%d]: stopped other threads and was stopped too\n", p\_num);

printf("thread p[%d]: stopped other threads and was stopped too\n", p\_num);

return NULL;

}

void \*P4thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

AtomicModification();

fprintf(f, "\nthread p[%d]: modificated atomic values\n", p\_num);

printf("\nthread p[%d]: modificated atomic values\n", p\_num);

while (1) {

fprintf(f, "thread p[%d]: opens semaphore scr21 for the thread p[1]\n", p\_num);

printf("thread p[%d]: opens semaphore scr21 for the thread p[1]\n", p\_num);

sem\_post(&scr21);

fprintf(f, "\nthread p[%d]: semaphore scr21 is opened\n", p\_num);

printf("\nthread p[%d]: semaphore scr21 is opened\n", p\_num);

fprintf(f, "\nthread p[%d]: waits for the opening of the semaphore scr22\n", p\_num);

printf("\nthread p[%d]: waits for the opening of the semaphore scr22\n", p\_num);

trywait:

if (sem\_trywait(&scr22) == 0) {

sem\_close(&scr22);

sem\_open("scr22", O\_CREAT, 0, 0);

fprintf(f, "thread p[%d]: works after receiving semaphore scr22\n", p\_num);

printf("thread p[%d]: works after receiving semaphore scr22\n", p\_num);

pthread\_mutex\_lock(&mutex\_for\_sig21);

pthread\_cond\_signal(&sig21);

pthread\_mutex\_unlock(&mutex\_for\_sig21);

fprintf(f, "thread p[%d]: sent sig21\n", p\_num);

printf("thread p[%d]: sent sig21\n", p\_num);

AtomicUsing();

fprintf(f, "thread p[%d]: used atomic values\n", p\_num);

printf("thread p[%d]: used atomic values\n", p\_num);

AtomicModification();

fprintf(f, "\nthread p[%d]: modificated atomic values\n", p\_num);

printf("\nthread p[%d]: modificated atomic values\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

goto trywait;

}

}

fprintf(f, "thread p[%d]: stopped\n", p\_num);

printf("thread p[%d]: stopped\n", p\_num);

return NULL;

}

void \*P5thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

int flag = 0;

int old\_state = 0;

while(1) {

fprintf(f, "thread p[%d]: is waiting for sig21\n", p\_num);

printf("thread p[%d]: is waiting for sig21\n", p\_num);

pthread\_mutex\_lock(&mutex\_for\_sig21);

pthread\_cond\_wait(&sig21, &mutex\_for\_sig21);

pthread\_mutex\_unlock(&mutex\_for\_sig21);

fprintf(f, "thread p[%d]: received sig21\n", p\_num);

printf("thread p[%d]: received sig21\n", p\_num);

AtomicUsing();

fprintf(f, "thread p[%d]: used atomic values\n", p\_num);

printf("thread p[%d]: used atomic values\n", p\_num);

if (el\_numb == 0) {

fprintf(f, "thread p[%d]: bufer is empty\n", p\_num);

printf("thread p[%d]: bufer is empty\n", p\_num);

empty\_bufer++;

if (empty\_bufer == 2)

break;

}

if (pthread\_mutex\_trylock(&mcr1) == 0) {

while(IsBufFull())

pthread\_cond\_wait(&sig2\_not\_full, &mcr1);

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, &old\_state);

if (flag == 0)

AddAtomicToBuf((int)int1\_cr2);

else if (flag == 1)

AddAtomicToBuf((int)int2\_cr2);

else if (flag == 2)

AddAtomicToBuf((int)uns1\_cr2);

else if (flag == 3)

AddAtomicToBuf((int)uns2\_cr2);

else if (flag == 4)

AddAtomicToBuf((int)long1\_cr2);

else if (flag == 5)

AddAtomicToBuf((int)long2\_cr2);

else if (flag == 6)

AddAtomicToBuf((int)luns1\_cr2);

else if (flag == 7)

AddAtomicToBuf((int)luns2\_cr2);

pthread\_setcancelstate(old\_state, NULL);

flag++;

if (flag == 8)

flag = 0;

fprintf(f, "\nthread p[%d]: element %d ATOMIC CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

printf("\nthread p[%d]: element %d ATOMIC CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

pthread\_mutex\_unlock(&mcr1);

pthread\_cond\_signal(&sig1\_not\_empty);

fprintf(f, "thread p[%d]: sent sig1\_not\_empty\n", p\_num);

printf("thread p[%d]: sent sig1\_not\_empty\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

}

}

pthread\_cancel(p1);

pthread\_cancel(p2);

pthread\_cancel(p3);

pthread\_cancel(p4);

pthread\_cancel(p6);

DeleteBuf();

fprintf(f, "thread p[%d]: stopped other threads and was stopped too\n", p\_num);

printf("thread p[%d]: stopped other threads and was stopped too\n", p\_num);

return NULL;

}

void \*P6thread(void \*p\_number) {

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

int p\_num = \*((int\*)p\_number);

int old\_state = 0;

while(1) {

if (el\_numb == 0) {

fprintf(f, "thread p[%d]: bufer is empty\n", p\_num);

printf("thread p[%d]: bufer is empty\n", p\_num);

empty\_bufer++;

if (empty\_bufer == 2)

break;

}

if (pthread\_mutex\_trylock(&mcr1) == 0) {

while(IsBufFull()) {

fprintf(f, "thread p[%d]: is waiting for sig2\_not\_full\n", p\_num);

printf("thread p[%d]: is waiting for sig2\_not\_full\n", p\_num);

pthread\_cond\_wait(&sig2\_not\_full, &mcr1);

}

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, &old\_state);

AddElemToBuf();

pthread\_setcancelstate(old\_state, NULL);

fprintf(f, "\nthread p[%d]: element %d CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

printf("\nthread p[%d]: element %d CREATED; current el\_numb = %d;\n", p\_num, next\_in->numb, el\_numb);

pthread\_mutex\_unlock(&mcr1);

pthread\_cond\_signal(&sig1\_not\_empty);

fprintf(f, "thread p[%d]: sent sig1\_not\_empty\n", p\_num);

printf("thread p[%d]: sent sig1\_not\_empty\n", p\_num);

}

else {

//printf("\nthread p[%d]: does some useful work\n", p\_num);

}

}

pthread\_cancel(p1);

pthread\_cancel(p2);

pthread\_cancel(p3);

pthread\_cancel(p4);

pthread\_cancel(p5);

DeleteBuf();

fprintf(f, "thread p[%d]: stopped other threads and was stopped too\n", p\_num);

printf("thread p[%d]: stopped other threads and was stopped too\n", p\_num);

return NULL;

}

void AtomicInput() {

int1\_cr2 = (rand() % 6) - 3;

int2\_cr2 = (rand() % 6) - 3;

uns1\_cr2 = (rand() % 6) - 3;

uns2\_cr2 = (rand() % 6) - 3;

long1\_cr2 = (rand() % 6) - 3;

long2\_cr2 = (rand() % 6) - 3;

luns1\_cr2 = (rand() % 6) - 3;

luns2\_cr2 = (rand() % 6) - 3;

}

void AtomicOutput() {

fprintf(f, "\n");

fprintf(f, "cr2: (1) int = %d\n", int1\_cr2);

fprintf(f, "cr2: (2) int = %d\n", int2\_cr2);

fprintf(f, "cr2: (1) unsigned = %d\n", uns1\_cr2);

fprintf(f, "cr2: (2) unsigned = %d\n", uns2\_cr2);

fprintf(f, "cr2: (1) long = %ld\n", long1\_cr2);

fprintf(f, "cr2: (2) long = %ld\n", long2\_cr2);

fprintf(f, "cr2: (1) long unsigned = %lu\n", luns1\_cr2);

fprintf(f, "cr2: (2) long unsigned = %lu\n", luns2\_cr2);

fprintf(f, "\n");

printf("\n");

printf("cr2: (1) int = %d\n", int1\_cr2);

printf("cr2: (2) int = %d\n", int2\_cr2);

printf("cr2: (1) unsigned = %d\n", uns1\_cr2);

printf("cr2: (2) unsigned = %d\n", uns2\_cr2);

printf("cr2: (1) long = %ld\n", long1\_cr2);

printf("cr2: (2) long = %ld\n", long2\_cr2);

printf("cr2: (1) long unsigned = %lu\n", luns1\_cr2);

printf("cr2: (2) long unsigned = %lu\n", luns2\_cr2);

printf("\n");

}

void AtomicModification() {

\_\_sync\_fetch\_and\_add(&int1\_cr2, (rand() % 6) - 3);

\_\_sync\_sub\_and\_fetch(&int2\_cr2, (rand() % 6) - 3);

\_\_sync\_fetch\_and\_xor(&uns1\_cr2, (rand() % 6) - 3);

\_\_sync\_fetch\_and\_nand(&uns2\_cr2, (rand() % 6) - 3);

\_\_sync\_or\_and\_fetch(&long1\_cr2, (rand() % 6) - 3);

\_\_sync\_and\_and\_fetch(&long2\_cr2, (rand() % 6) - 3);

\_\_sync\_bool\_compare\_and\_swap (&luns1\_cr2, (rand() % 6) - 3, (rand() % 6) - 3);

\_\_sync\_val\_compare\_and\_swap (&luns2\_cr2, (rand() % 6) - 3, (rand() % 6) - 3);

}

void AtomicUsing() {

\_\_sync\_fetch\_and\_add(&int1\_cr2, 0);

\_\_sync\_sub\_and\_fetch(&int2\_cr2, 0);

\_\_sync\_fetch\_and\_xor(&uns1\_cr2, 0);

\_\_sync\_fetch\_and\_nand(&uns2\_cr2, uns2\_cr2);

\_\_sync\_fetch\_and\_nand(&uns2\_cr2, uns2\_cr2);

\_\_sync\_or\_and\_fetch(&long1\_cr2, 0);

\_\_sync\_and\_and\_fetch(&long2\_cr2, long2\_cr2);

\_\_sync\_bool\_compare\_and\_swap (&luns1\_cr2, 0, 0);

\_\_sync\_val\_compare\_and\_swap (&luns2\_cr2, 0, 0);

}

void AddAtomicToBuf(int value) {

if (new\_buf < n) {

struct cr1 \*temp = (struct cr1 \*)malloc(sizeof(struct cr1));

temp->next = NULL;

if (next\_in == NULL) {

temp->numb = value;

next\_in = temp;

next\_out = temp;

begin\_buf = temp;

end\_buf = temp;

new\_buf++;

fprintf(f, "Cycle bufer was created\n");

printf("Cycle bufer was created\n");

}

else {

if (el\_numb > n) {

fprintf(f, "\nerror el\_numb > %d\n", n);

printf("\nerror el\_numb > %d\n", n);

free(temp);

return;

}

temp->numb = value;

next\_in->next = temp;

next\_in = temp;

new\_buf++;

if (new\_buf == n)

temp->next = begin\_buf;

end\_buf = temp;

}

el\_numb++;

}

else {

next\_in->next->numb = value;

if (next\_in == end\_buf)

next\_in = begin\_buf;

else

next\_in = next\_in->next;

el\_numb++;

}

}

**Тестування програми**

**№1**

Cycle bufer was created

Cycle buffer with elements from 0-th to 16-th has been created !!!

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[1]: semaphore scr22 is opened

thread p[1]: waits for the opening of the semaphore scr21

thread p[6]: element 42 CREATED; current el\_numb = 17;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 47 CREATED; current el\_numb = 18;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 19 CREATED; current el\_numb = 19;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 7 CREATED; current el\_numb = 20;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 29 CREATED; current el\_numb = 21;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 33 CREATED; current el\_numb = 22;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 5 CREATED; current el\_numb = 23;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 18 CREATED; current el\_numb = 24;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 32 CREATED; current el\_numb = 25;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 29 CREATED; current el\_numb = 26;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 1 CREATED; current el\_numb = 27;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 43 CREATED; current el\_numb = 28;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 28 CREATED; current el\_numb = 29;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 6 CREATED; current el\_numb = 30;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 36 CREATED; current el\_numb = 31;

thread p[6]: sent sig1\_not\_empty

thread p[6]: element 16 CREATED; current el\_numb = 32;

thread p[6]: sent sig1\_not\_empty

thread p[6]: is waiting for sig2\_not\_full

thread p[5]: is waiting for sig21

thread p[4]: modificated atomic values

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[4]: semaphore scr21 is opened

thread p[4]: waits for the opening of the semaphore scr22

thread p[1]: works after receiving semaphore scr21

thread p[1]: sent sig21

thread p[1]: used atomic values

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[5]: received sig21

thread p[1]: semaphore scr22 is opened

thread p[2]: is waiting for sig21

thread p[5]: used atomic values

thread p[4]: works after receiving semaphore scr22

thread p[4]: sent sig21

thread p[4]: used atomic values

thread p[4]: modificated atomic values

thread p[2]: received sig21

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[2]: used atomic values

thread p[3]: bufer is full

thread p[3]: element 8 TAKEN; current el\_numb = 31;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 10 TAKEN; current el\_numb = 30;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 36 TAKEN; current el\_numb = 29;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 45 TAKEN; current el\_numb = 28;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 29 TAKEN; current el\_numb = 27;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 48 TAKEN; current el\_numb = 26;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 15 TAKEN; current el\_numb = 25;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 30 TAKEN; current el\_numb = 24;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 14 TAKEN; current el\_numb = 23;

thread p[3]: sent sig2\_not\_full

thread p[2]: element -7 ATOMIC CREATED; current el\_numb = 24;

thread p[4]: semaphore scr21 is opened

thread p[2]: sent sig1\_not\_empty

thread p[6]: element 3 CREATED; current el\_numb = 25;

thread p[2]: is waiting for sig21

thread p[3]: element 45 TAKEN; current el\_numb = 24;

thread p[6]: sent sig1\_not\_empty

thread p[3]: sent sig2\_not\_full

thread p[4]: waits for the opening of the semaphore scr22

thread p[4]: works after receiving semaphore scr22

thread p[4]: sent sig21

thread p[4]: used atomic values

thread p[4]: modificated atomic values

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[4]: semaphore scr21 is opened

thread p[4]: waits for the opening of the semaphore scr22

thread p[2]: received sig21

thread p[2]: used atomic values

thread p[2]: element 7 ATOMIC CREATED; current el\_numb = 25;

thread p[2]: sent sig1\_not\_empty

thread p[2]: is waiting for sig21

thread p[1]: waits for the opening of the semaphore scr21

thread p[1]: works after receiving semaphore scr21

thread p[1]: sent sig21

thread p[1]: used atomic values

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[1]: semaphore scr22 is opened

thread p[1]: waits for the opening of the semaphore scr21

thread p[4]: works after receiving semaphore scr22

thread p[1]: works after receiving semaphore scr21

thread p[2]: received sig21

thread p[2]: used atomic values

thread p[2]: element -1 ATOMIC CREATED; current el\_numb = 26;

thread p[2]: sent sig1\_not\_empty

thread p[2]: is waiting for sig21

thread p[3]: element 39 TAKEN; current el\_numb = 25;

thread p[3]: sent sig2\_not\_full

thread p[6]: element 30 CREATED; current el\_numb = 26;

thread p[6]: sent sig1\_not\_empty

thread p[3]: element 48 TAKEN; current el\_numb = 25;

thread p[1]: sent sig21

thread p[1]: used atomic values

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[1]: semaphore scr22 is opened

thread p[1]: waits for the opening of the semaphore scr21

thread p[4]: sent sig21

thread p[4]: used atomic values

thread p[4]: modificated atomic values

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[4]: semaphore scr21 is opened

thread p[4]: waits for the opening of the semaphore scr22

thread p[4]: works after receiving semaphore scr22

thread p[4]: sent sig21

thread p[4]: used atomic values

thread p[4]: modificated atomic values

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[4]: semaphore scr21 is opened

thread p[4]: waits for the opening of the semaphore scr22

thread p[3]: sent sig2\_not\_full

thread p[3]: element 24 TAKEN; current el\_numb = 24;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 30 TAKEN; current el\_numb = 23;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 9 TAKEN; current el\_numb = 22;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 47 TAKEN; current el\_numb = 21;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 42 TAKEN; current el\_numb = 20;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 47 TAKEN; current el\_numb = 19;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 19 TAKEN; current el\_numb = 18;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 7 TAKEN; current el\_numb = 17;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 29 TAKEN; current el\_numb = 16;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 33 TAKEN; current el\_numb = 15;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 5 TAKEN; current el\_numb = 14;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 18 TAKEN; current el\_numb = 13;

thread p[2]: received sig21

thread p[3]: sent sig2\_not\_full

thread p[3]: element 32 TAKEN; current el\_numb = 12;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 29 TAKEN; current el\_numb = 11;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 1 TAKEN; current el\_numb = 10;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 43 TAKEN; current el\_numb = 9;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 28 TAKEN; current el\_numb = 8;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 6 TAKEN; current el\_numb = 7;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 36 TAKEN; current el\_numb = 6;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 16 TAKEN; current el\_numb = 5;

thread p[3]: sent sig2\_not\_full

thread p[3]: element -7 TAKEN; current el\_numb = 4;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 3 TAKEN; current el\_numb = 3;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 7 TAKEN; current el\_numb = 2;

thread p[3]: sent sig2\_not\_full

thread p[3]: element -1 TAKEN; current el\_numb = 1;

thread p[3]: sent sig2\_not\_full

thread p[3]: element 30 TAKEN; current el\_numb = 0;

thread p[3]: sent sig2\_not\_full

thread p[3]: is waiting for sig2\_not\_empty

thread p[2]: used atomic values

thread p[2]: is waiting for sig21

thread p[1]: works after receiving semaphore scr21

thread p[2]: received sig21

thread p[2]: used atomic values

thread p[2]: is waiting for sig21

thread p[1]: sent sig21

thread p[1]: used atomic values

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[1]: semaphore scr22 is opened

thread p[1]: waits for the opening of the semaphore scr21

thread p[1]: works after receiving semaphore scr21

thread p[2]: received sig21

thread p[2]: used atomic values

thread p[2]: is waiting for sig21

thread p[1]: sent sig21

thread p[1]: used atomic values

thread p[1]: modificated atomic values

thread p[1]: opens semaphore scr22 for the thread p[4]

thread p[1]: semaphore scr22 is opened

thread p[1]: waits for the opening of the semaphore scr21

thread p[5]: element -11 ATOMIC CREATED; current el\_numb = 1;

thread p[5]: sent sig1\_not\_empty

thread p[5]: is waiting for sig21

thread p[6]: element 9 CREATED; current el\_numb = 2;

thread p[6]: sent sig1\_not\_empty

thread p[3]: element -11 TAKEN; current el\_numb = 1;

thread p[4]: works after receiving semaphore scr22

thread p[3]: sent sig2\_not\_full

thread p[3]: element 9 TAKEN; current el\_numb = 0;

thread p[3]: sent sig2\_not\_full

thread p[3]: is waiting for sig2\_not\_empty

thread p[4]: sent sig21

thread p[2]: received sig21

thread p[6]: bufer is empty

thread p[4]: used atomic values

thread p[4]: modificated atomic values

thread p[4]: opens semaphore scr21 for the thread p[1]

thread p[4]: semaphore scr21 is opened

thread p[1]: works after receiving semaphore scr21

thread p[4]: waits for the opening of the semaphore scr22

thread p[4]: works after receiving semaphore scr22

thread p[6]: element 12 CREATED; current el\_numb = 1;

thread p[6]: sent sig1\_not\_empty

thread p[3]: element 12 TAKEN; current el\_numb = 0;

thread p[6]: bufer is empty

thread p[5]: received sig21

thread p[4]: sent sig21

thread p[3]: sent sig2\_not\_full

thread p[3]: is waiting for sig2\_not\_empty

thread p[2]: used atomic values

thread p[2]: bufer is empty

Cycle buffer was deleted

thread p[2]: stopped other threads and was stopped too

pthread\_join p1 = 0

pthread\_join p2 = 0

pthread\_join p3 = 0

pthread\_join p4 = 0

pthread\_join p5 = 0

pthread\_join p6 = 0

All information was written to the log file