МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра Інформаційної Безпеки

Операційні системи

Комп'ютерний практикум

Робота №8. Засоби синхронізації потоків

Мета:

Оволодіння практичними навичками розроблення багатопотокових програм з підтримкою засобів синхронізації

Перевірив:
_____

Виконав:
студент II курсу
групи ФБ-01
Сахній Н.Р.

Київ 2022

Завдання до виконання:

1. Розминка. Стандартна задача виробник-споживач.

Задача була розглянута на лекції. Також детально розглянута в рекомендованих книжках [1, 5]. Розробіть програму, що демонструє рішення цієї задачі за допомогою семафорів. Для цього напишіть:

- функції виробника і споживача (наприклад, як на лекції, або як у Шеховцові, але так, щоби працювало);
- функції створення і споживання об'єктів (рекомендується "створювати" рядки тексту шляхом зчитування їх з файлу, хоча можливі й інші варіанти за вибором викладача або за вашою фантазією, наприклад розрахунки геш-функцій sha2 з рядків рандомних символів, а "споживати" їх шляхом роздрукування на екрані з додатковою інформацією такою як ідентифікатор потоку і мітка часу, причому і там, і там для моделювання складного характеру реального життя виробників і споживачів можна додавати рандомні затримки);
- функцію `main()`, що створює потоки-виробники і потоки-споживачі, при цьому треба передбачити введення з клавіатури або як параметри командного рядка кількості записів у буфері, кількості виробників і кількості споживачів для досліджень їх роботи;
- обов'язково передбачити коректне завершення усього цього господарства.

Продемонструвати викладачеві як воно працює (не менше двох виробників і двох споживачів) і код, що ви написали.

```
nazar@ubuntu:~$ cd OS; mkdir lab_8; cd lab_8
nazar@ubuntu:~/OS/lab_8$ nano semaphor.cpp
```

Код програми:

nazar@ubuntu: ~/OS/lab_8

GNU nano 4.8 semaphor.cpp

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6  #include <unistd.h>
7
8  using namespace std;
9
10
11 int rc;
12 pthread_t * cons;
13 pthread_t * prods;
14 int prodN, conN, buffN;
15 int * buffer, position = -1;
16 sem_t lock, empty, full;
17
```

```cpp
int produce_new_item() {  // Функція створення об’єктів
    // ↓Згенерувати рандомне число від 1 до 100, яке пізніше буде занесене до буфера
    int item = 1 + rand() % 100;
    return item;
}

void add_to_buffer(int item, pthread_t self) {  // Функція, яка додає новий об'єкт до буфера
    if (position + 1 < buffN) {
        position++;
        buffer[position] = item;
        int i = 0;
        while (!pthread_equal( * (prods + i), self) && i < prodN) {
            i++;
        }
        cout << "🔲Produser №" << i + 1 << " adds to the buffer new item→{" << item << "}\n";
    } else {
        cout << "Error" << endl;
    }
}

void * producer(void * data) {  // Функція виробника на семафорах
    while (1) {                                 // Працює постійно
        int item = produce_new_item();          // Створюємо об'єкт

        sem_wait( & empty);                     // Перевіряємо наявність місця
        sem_wait( & lock);                      // Входимо у критичну секцію

        add_to_buffer(item, pthread_self());    // Додаємо об'єкт до буфера

        sem_post( & lock);                      // Виходимо з критичної секції
        sem_post( & full);                      // Повідомляємо про новий об'єкт

        sleep(1 + rand() % 5);                  // Рандомні затримки від 1 до 5 секунд
    }
    return NULL;
}


int get_from_buffer() {  // Функція, яка забирає об'єкт з буфера
    int item;
    if (position != -1) {
        item = buffer[position];
        position--;
    } else {
        cout << "Error" << endl;
    }
    return item;
}

void consume_new_item(int item, pthread_t self) {  // Функція споживання об’єктів
    int i = 0;
    while (!pthread_equal( * (prods + i), self) && i < prodN) {
        i++;
    }
    cout << "►Consumer №" << i + 1 << " gets from the buffer this item→{" << item << "}\n\n";
}

void * consumer(void * data) {  // Функція споживача на семафорах
    while (1) {                                 // Працює постійно
        sem_wait( & full);                      // Перевіряємо наявність об'єкта
        sem_wait( & lock);                      // Входимо у критичну секцію

        int item = get_from_buffer();           // Забираємо об'єкт з буфера
```

```
81
82          sem_post( & lock);                      // Виходимо з критичної секції
83          sem_post( & empty);                     // Повідомляємо про вільне місце
84
85          consume_new_item(item, pthread_self());   // Споживаємо об'єкт
86          sleep(1 + rand() % 5);                  // Рандомні затримки від 1 до 5 секунд
87      }
88      return NULL;
89  }
90
91
92  int main() {    // Функція, що створює потоки-виробники і потоки-споживачі
93      srand(time(NULL));
94
95      sem_init( & lock, 0, 1);
96      sem_init( & full, 0, 0);
97
98      prodN = 2;  // Кількість виробників
99      prods = (pthread_t * ) malloc(prodN * sizeof(pthread_t));
100
101     conN = 3;   // Кількість споживачів
102     cons = (pthread_t * ) malloc(conN * sizeof(pthread_t));
103
104     buffN = 5;  // Кількість записів у буфері
105     buffer = new int[buffN];
106     sem_init( & empty, 0, buffN);
107
108     for (int i = 0; i < prodN; i++) {
109         rc = pthread_create(prods + i, NULL, & producer, NULL);
110         if (rc) {
111             // ↓Коректне завершення відповідно до {виробника}
112             cout << "\n ERROR: return code from pthread_create for producer number" << i + 1 << " is " << rc << endl;
113             exit(1);
114         }
115     }
116     for (int i = 0; i < conN; i++) {
117         rc = pthread_create(cons + i, NULL, & consumer, NULL);
118         if (rc) {
119             // ↓Коректне завершення відповідно до {споживача}
120             cout << "\n ERROR: return code from pthread_create for consumer number" << i + 1 << " is " << rc << endl;
121             exit(1);
122         }
123     }
124     for (int i = 0; i < prodN; i++) {
125         pthread_join( * (prods + i), NULL);
126     }
127     for (int i = 0; i < conN; i++) {
128         pthread_join( * (cons + i), NULL);
129     }
130     return 0;
131 }
132
```

```
nazar@ubuntu:~/OS/lab_8$ g++ -pthread semaphor.cpp -o semaphor
nazar@ubuntu:~/OS/lab_8$ ls -l
total 24
-rwxrwxr-x 1 nazar nazar 18624 May  6 15:40 semaphor
-rw-rw-r-- 1 nazar nazar  3414 May  6 14:41 semaphor.cpp
nazar@ubuntu:~/OS/lab_8$
```

Результат виконання програми:

```
nazar@ubuntu:~/OS/lab_8$ ./semaphor
■ Produser №1 adds to the buffer new item→{56}
► Consumer №3 gets from the buffer this item→{56}
```

```
■ Produser №2 adds to the buffer new item→{65}
► Consumer №3 gets from the buffer this item→{65}

■ Produser №1 adds to the buffer new item→{19}
► Consumer №3 gets from the buffer this item→{19}

■ Produser №1 adds to the buffer new item→{60}
► Consumer №3 gets from the buffer this item→{60}

■ Produser №2 adds to the buffer new item→{19}
► Consumer №3 gets from the buffer this item→{19}

■ Produser №1 adds to the buffer new item→{13}
► Consumer №3 gets from the buffer this item→{13}

■ Produser №2 adds to the buffer new item→{54}
► Consumer №3 gets from the buffer this item→{54}

^C
nazar@ubuntu:~/OS/lab_8$
```

2. Продовження розминки. Теж саме, але не на семафорах, а на м’ютексі і умовних змінних.

Модифікуйте програму п. 1 так, щоби використовувати м’ютекс і умовну змінну.

```
nazar@ubuntu:~/OS/lab_8$ nano mutex_and_condvar.cpp
```

Код програми:

/* Далі буде продемонтровано лише зміни в програмному коді порівняно з попереднім, який реалізований на семафорах (так як код схожий на попередній)*/

Зробимо зміни в оголошенні глобальних змінних:

```
10
11 int rc;
12 pthread_t * cons;
13 pthread_t * prods;
14 int prodN, conN, buffN;
15 int * buffer, position = -1;
16 pthread_cond_t var[2];
17 pthread_mutex_t lock;
18
```

А також реалізуємо функцію виробника на м’ютексі і умовних змінних

```
38
39 void * producer(void * data) {  // Функція виробника на м’ютексі і умовних змінних
40     while (1) {
41         int item = produce_new_item();          // Створюємо об'єкт
42         pthread_mutex_lock( & lock);            // Займаємо м'ютекс
43
44         while (position + 1 == buffN) {         // Перевіряємо умову в циклі
45             // ↓ Викликається, коли потік у критичній секції не може продовжувати роботу через невиконання певної умови
46             pthread_cond_wait( & var[0], & lock);
47         }
48         add_to_buffer(item, pthread_self());    // Додаємо об'єкт до буфера
49
50         /*  Потік, що виконав дії з даними у критичній секції, перевіряє, чи не очікують на умовній змінній інші потоки,
51          ↓   і якщо очікують – переводить один з них у стан готовності (потік буде поновлено після звільнення м’ютексу) */
52         pthread_cond_signal( & var[1]);
53         pthread_mutex_unlock( & lock);          // Звільняємо м'ютекс
54
55         sleep(1 + rand() % 5);          // Рандомні затримки від 1 до 5 секунд
56     }
57     return NULL;
58 }
```

І так само змінимо функцію споживача

```
80 void * consumer(void * data) {  // Функція споживача на м’ютексі і умовних змінних
81     while (1) {
82         pthread_mutex_lock( & lock);            // Займаємо м'ютекс
83
84         while (position + 1 == buffN) {         // Перевіряємо умову в циклі
85             // ↓ Викликається, коли потік у критичній секції не може продовжувати роботу через невиконання певної умови
86             pthread_cond_wait( & var[1], & lock);
87         }
88         int item = get_from_buffer();           // Забираємо об'єкт з буфера
89
90         /*  Потік, що виконав дії з даними у критичній секції, перевіряє, чи не очікують на умовній змінній інші потоки,
91          ↓   і якщо очікують – переводить один з них у стан готовності (потік буде поновлено після звільнення м’ютексу) */
92         pthread_cond_signal( & var[0]);
93         pthread_mutex_unlock( & lock);          // Звільняємо м'ютекс
94
95         consume_new_item(item, pthread_self());  // Споживаємо об'єкт
96         sleep(1 + rand() % 5);                   // Рандомні затримки від 1 до 5 секунд
97     }
98     return NULL;
99 }
100
```

```
nazar@ubuntu:~/OS/lab_8$ g++ -pthread mutex_and_condvar.cpp -o mutex_and_condvar
nazar@ubuntu:~/OS/lab_8$ ls -l
total 48
-rwxrwxr-x 1 nazar nazar 18688 May  6 15:56 mutex_and_condvar
-rw-rw-r-- 1 nazar nazar  2963 May  6 15:55 mutex_and_condvar.cpp
-rwxrwxr-x 1 nazar nazar 18624 May  6 15:41 semaphor
-rw-rw-r-- 1 nazar nazar  3416 May  6 15:55 semaphor.cpp
nazar@ubuntu:~/OS/lab_8$
```

Результат виконання програми:

```
nazar@ubuntu:~/OS/lab_8$ ./mutex_and_condvar
■ Produser №1 adds to the buffer new item→{48}
► Consumer №3 gets from the buffer this item→{48}

Error
► Consumer №3 gets from the buffer this item→{0}
```

```
■ Produser №1 adds to the buffer new item→{93}
► Consumer №3 gets from the buffer this item→{93}

■ Produser №2 adds to the buffer new item→{78}
► Consumer №3 gets from the buffer this item→{78}

^C
nazar@ubuntu:~/OS/lab_8$
```

3. Продовження розминки для тих, хто шукає пригод. Взаємне блокування

Модифікуйте програму п. 1 так, щоби викликати взаємне блокування. Для цього поміняйте місцями семафори. Переконайтесь у факті взаємного блокування і отримайте задоволення.

```
nazar@ubuntu:~/OS/lab_8$ nano deadlock_with_semaphor.cpp
```

Код програми:

/* Далі буде продемонтровано лише зміни в програмному коді порівняно з попереднім, який реалізований на семафорах (так як код схожий на попередній)*/

Поміняємо місцями лише семафори очікування у функції виробника та споживача:

```
38 void * producer(void * data) {  // Функція виробника на семафорах
39     while (1) {                                  // Працює постійно
40         int item = produce_new_item();           // Створюємо об'єкт
41
42         sem_wait( & empty);                      // Перевіряємо наявність місця  ↤ Поміняли місцями
43         sem_wait( & lock);                       // Входимо у критичну секцію    ↤       семафори
44
45         add_to_buffer(item, pthread_self());     // Додаємо об'єкт до буфера
46
47         sem_post( & lock);                       // Виходимо з критичної секції
48         sem_post( & full);                       // Повідомляємо про новий об'єкт
49
50         sleep(1 + rand() % 5);                   // Рандомні затримки від 1 до 5 секунд
51     }
52     return NULL;
53 }
54
```

```
74
75 void * consumer(void * data) { // Функція споживача на семафорах
76     while (1) {                                 // Працює постійно
77         sem_wait( & lock);                      // Входимо у критичну секцію          ↤ Поміняли місцями
78         sem_wait( & full);                      // Перевіряємо наявність об'єкта      ↤       семафори
79
80         int item = get_from_buffer();           // Забираємо об'єкт з буфера
81
82         sem_post( & lock);                      // Виходимо з критичної секції
83         sem_post( & empty);                     // Повідомляємо про вільне місце
84
85         consume_new_item(item, pthread_self()); // Споживаємо об'єкт
86         sleep(1 + rand() % 5);                  // Рандомні затримки від 1 до 5 секунд
87     }
88     return NULL;
89 }
90
```

```
nazar@ubuntu:~/OS/lab_8$ g++ -pthread deadlock_with_semaphor.cpp -o deadlock_with_semaphor
nazar@ubuntu:~/OS/lab_8$ ls -l
total 72
-rwxrwxr-x 1 nazar nazar 18640 May  6 16:11 deadlock_with_semaphor
-rw-rw-r-- 1 nazar nazar  3419 May  6 16:09 deadlock_with_semaphor.cpp
-rwxrwxr-x 1 nazar nazar 18688 May  6 15:56 mutex_and_condvar
-rw-rw-r-- 1 nazar nazar  2963 May  6 15:55 mutex_and_condvar.cpp
-rwxrwxr-x 1 nazar nazar 18624 May  6 15:41 semaphor
-rw-rw-r-- 1 nazar nazar  3416 May  6 15:55 semaphor.cpp
nazar@ubuntu:~/OS/lab_8$
```

Результат виконання програми:

↓Пояснення щодо факту взаємного блокування // Взято із презентації наших лекцій

- Перевірка умови з можливим очікуванням здійснюється ***всередині*** критичної секції. Можлива така послідовність дій:
  1. Виробник входить у критичну секцію, закриваючи семафор **lock**
  2. Виробник перевіряє семафор **empty** і очікує на ньому (буфер повний)
  3. Споживач намагається ввійти у критичну секцію і блокується на семафорі **lock**
- Така ситуація називається ***взаємне блокування*** або ***тупик*** (***deadlock***)

↓На фото програма “зависла”, через що й була зупинена

```
nazar@ubuntu:~/OS/lab_8$ ./deadlock_with_semaphor
■ Produser №1 adds to the buffer new item→{52}
► Consumer №3 gets from the buffer this item→{52}

^C
nazar@ubuntu:~/OS/lab_8$
```

4. Індивідуальне завдання.

Пором у Парку культури та відпочинку

У Парку культури та відпочинку десантники та морські піхотинці одночасно відмічають свята відповідних родів військ. Повертатись з парку вони мають на поромі, що вміщає 2N вояків. Пором можна відправляти лише тоді, коли він повністю заповнений і на ньому або представники лише одного роду військ, або рівна кількість десантників і морських піхотинців. Якщо сили будуть нерівними, неминуча жорстока бійка. Переправа займає рандомний час. На березі вояки поводять себе культурно, бо там чергує озброєний контингент “Беркуту”, що контролює посадку на пором. Вояки підходять з парку поодинці у випадкові моменти часу і стають у чергу.

Змоделювати роботу такої переправи, створюючи для “обслуговування” кожного вояка окремий потік. Програма має синхронно (тобто, у тому ж порядку, як воно і відбувалося) писати у журнал події приходу вояків, посадки їх на пором, відправлення і повернення порому

```
nazar@ubuntu:~/OS/lab_8$ nano military_ferry.cpp
```

Код програми:

nazar@ubuntu: ~/OS/lab_8

GNU nano 4.8 military_ferry.cpp

```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <queue>

using namespace std;


pthread_t * air_paratrooper;  // Повітряний десантник
pthread_t * marine_infantryman;  // Морський піхотинець
int buffN, airN, marN;
int * buffer, number, position, air_on_ferry = 0, mar_on_ferry = 0;
sem_t lock_air, lock_mar, empty, full;
int ferry_type = 1;


int stand_in_line(const char * military) {
    int military_number = 1 + rand() % 1000;  // Нехай кожен військовий має свій номер
    cout << "(" << military <<" №"<< military_number << " став у чергу)" << endl;
    return military_number;
}

void add_to_buffer(int military_number, const char * military) {
    if (position + 1 <= buffN) {
        position++;
        buffer[position] = military_number;

        cout << "\n[" << military <<" №"<< military_number << " сів на паром і чекає відправлення]" << endl;
        if (military == "Десантник") {
            air_on_ferry++;  // +1 десантник на поромі
        } else if (military == "Морський піхотинець") {
            mar_on_ferry++;  // +1 морський піхотинець на поромі
        }
    } else {
        cout << "Error";
    }
}

void * ferry() {  // Функція перевірки заповненості парома та його відправки
    if (position == buffN) {
        ferry_type++;
        if (ferry_type > 3) {
            ferry_type = 1;
        }

        sem_wait( & lock_air);
        sem_wait( & lock_mar);

        cout << "\n ◘Паром відправився на інший берег ►\n\n" << endl;
        while (position > 0) {
            sem_post( & empty);
            position--;
        }
        air_on_ferry = 0;
        mar_on_ferry = 0;

        /* Переправа парома із парку на інший берег та назад займає рандомний час із певного проміжку.
            ↓ Інші в цей час можуть займати чергу, щоб сісти на паром, коли він повернеться назад у парк */
        sleep(10 + rand() % 4);

```

```
        sem_post( & lock_mar);
        sem_post( & lock_air);

        if (ferry_type == 1) {
            cout << "\n\n ◄Паром повернувся у парк. Можна проводити посадку для всіх військових ◘\n" << endl;
        } else if (ferry_type == 2) {
            cout << "\n\n ◄Паром повернувся у парк. Можна проводити посадку тільки для десантників ◘\n" << endl;
        } else if (ferry_type == 3) {
            cout << "\n\n ◄Паром повернувся у парк. Можна проводити посадку тільки для морських піхотинців ◘\n" << endl;
        }
    }
    return NULL;
}

void * airborne(void * data) {  // Функція морської піхоти
    bool switch_line = true;
    const char * air = "Десантник";
    queue <int> military_numbers;
    if (switch_line) {
        sleep(1 + rand() % 100);  // Десантники підходять з парку поодинці у випадкові моменти часу

        int military_number = stand_in_line(air);  // Десантники стають в чергу
        military_numbers.push(military_number);  // Додаємо його військовий номер до черги

        switch_line = false;;
    }

    while (1) {
        if ((ferry_type == 1 and air_on_ferry == number) or ferry_type == 3) {
            sleep(2);
        } else if (!military_numbers.empty()) {
            sem_wait( & empty);
            sem_wait( & lock_air);

            add_to_buffer(military_numbers.front(), air);  // Десантник сідає на паром і чкає відпрвлення
            military_numbers.pop();  // Видаляємо його військовий номер із черги

            sem_post( & lock_air);
            sem_post( & full);

            cout << "••• К-ть військових на паромі→{" << position << "} •••" << endl;
            cout << "\t Морпіхи " << mar_on_ferry << ":" << air_on_ferry << " Десантники \n\n" << endl;

            ferry();
        }
    }
    return NULL;
}

void * marine_infantry(void * data) {  // Функція повітряного десанту
    bool switch_line = true;
    const char * mar = "Морський піхотинець";
    queue <int> military_numbers;
    if (switch_line) {
        sleep(1 + rand() % 100);  // Морпіхи підходять з парку поодинці у випадкові моменти часу

        int military_number = stand_in_line(mar);  // Морпіхи стають в чергу
        military_numbers.push(military_number);  // Додаємо його військовий номер до черги

        switch_line = false;
    }

    while (1) {
        if ((ferry_type == 1 and mar_on_ferry == number) or ferry_type == 2) {
            sleep(2);
        } else if (!military_numbers.empty()) {
            sem_wait( & empty);
            sem_wait( & lock_mar);

            add_to_buffer(military_numbers.front(), mar);  // Морпіх сідає на паром і чкає відпрвлення
            military_numbers.pop();  // Видаляємо його військовий номер із черги

            sem_post( & lock_mar);
            sem_post( & full);

```

```
            cout << "••• К-ть військових на паромі→{" << position << "} •••" << endl;
            cout << "\t Морпіхи " << mar_on_ferry << ":" << air_on_ferry << " Десантники \n\n" << endl;

            ferry();
        }
    }
    return NULL;
}


int main() {
    cout << "Так як паром вміщує 2*N військових, тому, щоб розпочати процес переправи, \n \
            Введіть цілочисельне значення N → ";
    cin >> number; cout << "\n";

    srand(time(NULL));

    airN = 16;  // Кількість повітряних десантників
    air_paratrooper = (pthread_t * ) malloc(airN * sizeof(pthread_t));
    sem_init( & lock_air, 0, 1);

    marN = 16;  // Кількість морських піхотинців
    marine_infantryman = (pthread_t * ) malloc(marN * sizeof(pthread_t));
    sem_init( & lock_mar, 0, 1);

    buffN = 2 * number; // Паром вміщує 2*N військових
    buffer = new int[buffN];
    sem_init( & empty, 0, buffN);
    sem_init( & full, 0, 0);

    for (int i = 0; i < airN; i++) {
        pthread_create(air_paratrooper + i, NULL, & airborne, NULL);  // Кожен десантник - це окремий потік
    }

    for (int i = 0; i < marN; i++) {
        pthread_create(marine_infantryman + i, NULL, & marine_infantry, NULL);  // Кожен морпіх - це окремий потік
    }

    pthread_exit(NULL);  // To exit from main thread
    return 0;
}

```

```
nazar@ubuntu:~/OS/lab_8$ g++ -pthread military_ferry.cpp -o military_ferry
nazar@ubuntu:~/OS/lab_8$ ls -l
total 104
-rwxrwxr-x 1 nazar nazar 18640 May  6 16:11 deadlock_with_semaphor
-rw-rw-r-- 1 nazar nazar  4443 May  7 10:11 deadlock_with_semaphor.cpp
-rwxrwxr-x 1 nazar nazar 17040 May  7 10:41 military_ferry
-rw-rw-r-- 1 nazar nazar   163 May  7 10:29 military_ferry.cpp
-rwxrwxr-x 1 nazar nazar 18688 May  6 15:56 mutex_and_condvar
-rw-rw-r-- 1 nazar nazar  2906 May  6 22:59 mutex_and_condvar.cpp
-rwxrwxr-x 1 nazar nazar 18624 May  6 15:41 semaphor
-rw-rw-r-- 1 nazar nazar  4381 May  7 10:40 semaphor.cpp
nazar@ubuntu:~/OS/lab_8$
```

Результат виконання програми:

```
nazar@ubuntu:~/OS/lab_8$ ./military_ferry
Так як паром вміщує 2*N військових, тому, щоб розпочати процес переправи,
            Введіть цілочисельне значення N → 3
```

// Спочатку посадка на паром здійсюється як для десантників, так і для морпіхів. Військові стають якби в дві черги, щоб сісти на паром, і переправа розпочнеться тільки тоді, коли буде рівна кількість десантників та піхотинців на паромі.

```
(Морський піхотинець №222 став у чергу)

[Морський піхотинець №222 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{1} •••
        Морпіхи 1:0 Десантники


(Десантник №947 став у чергу)

[Десантник №947 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{2} •••
        Морпіхи 1:1 Десантники


(Морський піхотинець №826 став у чергу)

[Морський піхотинець №826 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{3} •••
        Морпіхи 2:1 Десантники


(Морський піхотинець №589 став у чергу)

[Морський піхотинець №589 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{4} •••
        Морпіхи 3:1 Десантники


(Морський піхотинець №458 став у чергу)
(Морський піхотинець №478 став у чергу)
(Десантник №93 став у чергу)

[Десантник №93 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{5} •••
        Морпіхи 3:2 Десантники


(Десантник №651 став у чергу)

[Десантник №651 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{6} •••
        Морпіхи 3:3 Десантники
```

// Отож паром відправився, проте військові продовжують ставати в чергу

```
 ◘ Паром відправився на інший берег ►


(Десантник №536 став у чергу)
(Морський піхотинець №299 став у чергу)
(Десантник №843 став у чергу)
(Морський піхотинець №223 став у чергу)
(Десантник №117 став у чергу)
```

// Тепер розпочнемо посадку на паром лише повітряних десантників. У цей момент морпіхи продовжують ставати в чергу, але сісти зможуть лише пізніше.

```
◄Паром повернувся у парк. Можна проводити посадку тільки для десантників ◘


[Десантник №536 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{1} •••
        Морпіхи 0:1 Десантники



[Десантник №843 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{2} •••
        Морпіхи 0:2 Десантники



[Десантник №117 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{3} •••
        Морпіхи 0:3 Десантники


(Десантник №831 став у чергу)

[Десантник №831 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{4} •••
        Морпіхи 0:4 Десантники


(Морський піхотинець №112 став у чергу)
(Десантник №876 став у чергу)

[Десантник №876 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{5} •••
        Морпіхи 0:5 Десантники


(Десантник №196 став у чергу)

[Десантник №196 сів на паром і чекає відправлення]
••• К-ть військових на паромі→{6} •••
        Морпіхи 0:6 Десантники
```

// Паром лише з десантниками відправився. Військові продовжують займати чергу

```
◘Паром відправився на інший берег ►


(Десантник №480 став у чергу)
(Морський піхотинець №664 став у чергу)
(Десантник №791 став у чергу)
(Десантник №982 став у чергу)
```

// Можемо розпочати посадку на паром лише морських піхотинців.

```
◄Паром повернувся у парк. Можна проводити посадку тільки для морських піхотинців ◘


[Морський піхотинець №478 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{1} •••
        Морпіхи 1:0 Десантники



[Морський піхотинець №458 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{2} •••
        Морпіхи 2:0 Десантники



[Морський піхотинець №299 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{3} •••
        Морпіхи 3:0 Десантники



[Морський піхотинець №223 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{4} •••
        Морпіхи 4:0 Десантники



[Морський піхотинець №112 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{5} •••
        Морпіхи 5:0 Десантники



[Морський піхотинець №664 сів на паром і чекає відправлення]
••• К-ть військових на поромі→{6} •••
        Морпіхи 6:0 Десантники
```

// Ну, а далі паром з морпіхами відправляється, військові продовжують ставати в чергу. І коли паром повернеться назад, знову можна буде сідати усім військовим.

```
◘Паром відправився на інший берег ►


(Морський піхотинець №106 став у чергу)
(Десантник №869 став у чергу)
(Морський піхотинець №246 став у чергу)
^C
nazar@ubuntu:~/OS/lab_8$ nano military_ferry.cpp
nazar@ubuntu:~/OS/lab_8$
```

Висновки:

У результаті виконання комп’ютерного практикуму, мною було здобуто навички із синхронізації потоків під час розроблення багатопотокових програм.