Лабораторна робота “Спільна пам’ять”

Мета: Створити високорівневий інтерфейс роботи зі спільною пам’ятью.

Задачи:

1. Розробити шаблон класу **shared_memory** та продемострувати його функціональність. Опис полів та методів надані далі у лабораторній роботі.

2. Створити програму **server** та **client** для демонстрації можливостей методів класу.

Номер індивідуального завдання обчислюється за формулою:

```
(< номер завдання >) = (< свій номер у списку > - 1) % 4 + 1
```

Свій номер у списку дивись на останній сторінці поточного документу (Додаток 1).

Завдання:

1. У вхідному потоці надходять **n** натуральних чисел. Сервер зчитує ці значення та записує їх у спільну пам’ять. Після завершення завантаження даних клієнт починає їх обробку — сортує числа у порядку неспадання. Після завершення обробки відсортований масив даних виводиться у вихідний потік.

2. У вхідному потоці надходять n натуральних чисел. Потрібно реалізувати обмін даними між двома окремими процесами (сервером і клієнтом) за допомогою спільної пам’яті, із синхронізацією на основі сигнальної змінної:

Ролі процесів:

Сервер:

- зчитує n натуральних чисел із вхідного потоку;
- створює сегмент спільної пам’яті (**shared_memory**);
- записує в пам’ять:
    - сигнальну змінну (початкове значення — 0);
    - значення **n**;
    - масив із **n** чисел;
- після запису переходить у режим активного очікування (**busy waiting**), перевіряючи сигнальну змінну;

- після того, як сигнальна змінна набуває значення 1, зчитує відсортований масив безпосередньо зі спільної пам’яті та виводить його у вихідний потік.

Клієнт:

- підключається до вже створеної сервером спільної пам’яті;
- отримує доступ до масиву чисел через типізований вказівник;
- виконує сортування масиву безпосередньо у спільній пам’яті (тобто без копіювання даних у локальний буфер);
- після завершення сортування змінює значення сигнальної змінної на 1, що сигналізує серверу про завершення обробки.

3. Реалізація служби повідомлень між двома користувачами через спільну пам’ять із використанням сигнальних змінних.

Постановка задачі:

Необхідно реалізувати службу обміну повідомленнями між двома окремими користувачами (процесами) через спільну пам’ять. Кожен користувач може відправляти текстові повідомлення іншому, а також читати повідомлення, адресовані йому.

Обмін повідомленнями має тривати доти, доки будь-який з користувачів не надішле спеціальне повідомлення "**bay**", яке означає завершення сесії.

Принцип взаємодії:

1. Початкова ініціалізація: обидві сигнальні змінні встановлено в 0, що означає "немає нового повідомлення".
2. Надсилання повідомлення:
   - Користувач вводить повідомлення.
   - Записує його у відповідне поле.
   - Встановлює відповідний флаг у 1, сигналізуючи іншому користувачу, що нове повідомлення готове.
3. Читання повідомлення:
   - Користувач перевіряє флаг, що вказує на наявність нового повідомлення.
   - Якщо значення флагу — 1, він читає повідомлення з відповідного буфера та скидає флаг у 0.
4. Завершення сеансу:

- Якщо будь-який з користувачів отримує або надсилає повідомлення "**bay**", обидва процеси завершують роботу.

4. **Гра Баше між двома процесами.**

Постановка задачі:

Є натуральне число **n** — початкове значення гри. Два гравці (процеси) ходять по черзі. На своєму ході гравець може зменшити значення спільної змінної (число) на будь-яке натуральне число від 1 до **m** (включно), де **m** — задане максимальне допустиме зменшення.

Гравець, який у свій хід зробить число рівним 0, виграє.

Опис процесів:

- Два незалежні процеси — гравці 1 і 2.
- Вони мають спільний доступ до змінної n у спільній пам’яті.
- Процеси ходять по черзі, кожен у свій хід:
  - Зчитує поточне значення **n**.
  - Обирає (можна випадково або за стратегією) число **k**, таке що **1≤k≤min(m,n)**.
  - Зменшує **n** на **k**.
  - Виводить на екран своє ім’я, значення **k**, та нове значення **n**.
  - Чекає 3 секунди перед тим, як передати хід іншому гравцю.

Рекомендації:

- Для генерації **k** можна використовувати випадкове число або просту стратегію.
- Активне очікування варто реалізувати з короткими паузами (**usleep(100)**), щоб не перевантажувати процесор.
- Після завершення гри — процеси коректно закривають спільну пам’ять.
- Вивід повинен бути синхронізований по черзі, що гарантується зміною **turn**.

Підсумок:

Це класична гра із простим протоколом між двома процесами, які використовують спільну пам’ять і сигнальні змінні для синхронізації ходів. Завдяки 3-секундній затримці вивід буде зрозумілим і повільним, що полегшить спостереження за процесом.

Спільна пам’ять

Сегмент пам’яті, який одночасно доступен декільком процессам називається спільною пам’ятью. По замовченню процеси изольовані один від одного, і не мають загальної пам’яті. Але інколи виникає потреба забеспечити одночасний доступ декількох процесів до одного і того ж сегменту пам’яті, наприклад при суміной обробці загальних даних.

Операційна система дозволяє створення такого сегменту пам’яті. Розглянемо усі етапи сворення та використання такої пам’яті.

Один з процесів, ми будемо називати його сервером, повинен створити сегмент пам’яті необхідного розміру, а по завершенню робиті з ним звільнити, повернувши операційній системі.

Створення сегменту пам’яті:

```
int shmget(key, size, flags);
```

Повертає ідентификатор (дескриптор) сегменту, або **-1** у разі не успіху.

key – ключ доступу, довільне ціле число більше ніж 0 (однакове для усіх взаімодіючих процесів), **size** – розмір сегменту, у байтах, **flags** – набір флагів, які визначають доступ до сегменту та дію над. По суті **flags** це ціле число - **1** (2^0), **2** (2^1), **4** (2^2), **8** (2^3), **16** (2^4), **32** (2^5) и т.д. або їх комбінація). Кожет біт має свое значення та відповідає за ті чи інши властивості. Кожен біт має своє им’я, наприклад: **IPC_CREAT** (2^9) - створити, **IPC_EXCL** (2^8) – використовувати.

Наприклад, для сервера:

```
int mid;
if ((mid = shmget(512, 1024, IPC_CREAT | 0666)) < 0) {
  printf(“shmget error\n”);
  exit(1);
}
```

… для кліента:

```
int mid;
if ((mid = shmget(512, 1024, IPC_EXCL)) < 0) {
```

```
  printf(“shmget error\n”);
  exit(1);
}
```

Флаг **0666** – це вісімкове ціле число, яке вказує дозвіл на операції читання та запису по їх власникам та групам власників процесів. Детальніше розглянемо це пізніше.

Наступний етап – це підключення сегменту пам’яті до процесу.

```
void* shmat(mid, NULL, 0);
```

Повертає вказівник на сегмент пам’яті або NULL невдачі.

Далі, сегмент пам’яті може бути інтерпритован як звичайний масив. Наприклад:

```
int mid;
if ((mid = shmget(512, 1024, IPC_CREAT | 0666)) < 0) {
  printf(“shmget error\n”);
  exit(1);
}
int* buf;
if ((buf = (int*) shmat(mid, NULL, 0)) == NULL) {
  printf(“shmat error\n”);
  exit(1);
}
int n = 1024 / sizeof(int); // Кількість елементів типу int
for (int i = 0; i < n; ++i) {
  scanf(“%d ”, &buf[i]);
}
```

По закінчені роботи необхідно відключити сегмент пам’яті від процесу, це робить кожен з взаімодіючих процесів.

```
int shmdt(buf);
```

Повертає **0** у випадку успіху, або **-1** при невдачі.

І остаточне: звільнити пам’ть, видалити створений сегмент пам’яті. Це робить тільки сервер:

```
int shmctl(mid, IPC_RMID, NULL);
```

Повертає 0 у випадку успіху, або -1 при невдачі.

Зауваження: Щоб зменьшити верогідність втручання в дані сегменту сторонім програмам треба створити унікальній ключ, який буде відомий тільки легально взаімодіючим процессам. Для цього на базі існуючего файлу або каталогу створюється унікальне ціле число. Для цього слід викликати функцію ядра операційної системи **ftok()**:

```
key_t key = ftok(“/tmp”, code);
```

Де **code** – довільне ціле число, а “/**tmp**” – той файл (каталог), індексний дескриптор якого визначить значення ключа беспеки. Якщо усі процеси для створення ключа викличуть функцію **ftok()** з однаковими значеннями параметрів, то одержать одне и теж значення ключа, що забеспечить підключення до одного і того ж сегменту пам’яті. Вгадати значення двух параметрів злодію буде набагато складніше, що підвищить надійність даних у сегменті пам’яті.

Створення високорівневого інтерфейсу

SHARED_MEMORY

Опис шаблона класу **shared_memory**

Конструктори шаблону класу **shared_memory**:

shared_memory(const size_t n);

Приймає розмір сегменту пам’яті, n – розмір сегменту у елементах

Деструктори класу

~shared_memory();

Від’єднує та видаляє (якщо це сервер) сегмент пам’яті

Методи класу:

int create_shared_memory(const int key, const int flags = 0666);

Створює сегмент пам’яті, повертає дескриптор пам’яті (**key > 0**)

int open_shared_memory(const int key);

Повертає дескриптор пам’яті, яка створена сервером.

T* connect();

Підключює до процесу сегмент памяті та повертає вказівник на цей сегмент пам’яті (**T** – це тип одного елементу пам’яті).

int disconnect();

Від’єднує та видаляє (якщо це сервер) сегмент пам’яті

size_t size() const;

Розмір сегменту пам’яті, у записах (елементах)

size_t search(const T& x) const;

Повертає індекс на перший елемент, який дорівнює x

size_t search(bool (*unare_predicate)(T&)) const;

Повертає індекс на перший елемент, який відповідає умові пошуку

T& at(const int i);

Повертає посилання на i-й елемент сегменту

T& operator [] (const int i) const;

Повертає посилання на i-й елемент сегменту

Поля класу:

pid_t md;

Дескриптор сегменту пам’яті

size_t size;

Розмір сегменту пам’яті, у елементах

int flags;

Флаг безпеки у форматі восімкового числа, наприклад 0666

T* buffer;

Вказівник на сегмент пам’яті

Демонстрація використання класу shared_memory

SERVER...

```
#include <iostream>
#include <unistd.h>
#include "memory.cpp" // Власна “Бібліотека”

#define N 1024

using namespace std;

int main() {
  // Створюємо об’єкт
  shared_memory < int > mem(N);
  // Створюємо спільну пам’ять на N (1024) елемента типу int
  if (mem.create_shared_memory(512, 0666) < 0) {
    cout << "Error" << endl;
    return 1;
  }

  // Підключаємо спільну пам’ять до процесу, отримаємо вказівник
  int* arr = mem.connect();
  arr[0] = 0; // Сигнальна зміна

  // Записуємо у спільну пам’ять данні
  for (size_t i = 1; i < N; ++i) {
    cin >> arr[i];
  }

  // Чекаємо, коли клиент завершить роботу
  while (arr[0] == 0) {
    usleep(100);
  }
  return 0;
}
```

CLIENT...

```
#include <iostream>
#include <unistd.h>
#include "memory.cpp"

#define N 1024

using namespace std;

int main() {
  // Створюємо об’єкт
  shared_memory < int > mem(N);
  // Отримаємо дескриптор на вже створений сервером сегмент пам’яті
  if (mem.open_shared_memory(512) < 0) {
    cout << "Error" << endl;
    return 1;
  }

  // Підключаємо спільну пам’ять до процесу
  mem.connect();
  // Читаємо дані, які записав сервер та виводимо у вихідний потік
  for (int i = 1; i < N; ++i) {
    cout << mem[i] << " ";
  }
  cout << endl;

  // Повідомимо сервер, що кліент работу завершив, разблокуємо сервер
  mem[0] = 1; // Сигнальна зміна
  return 0;
}
```

Додаток 1

Список студентів по курсу «Операційні системи» для визнання номеру індівідуального завдання.

1	Шевченко Богдан Миколайович	ylua5555@gmail.com
2	Адамов Ярослав Родіонович	larchistrike@gmail.com
3	Астапенко Кирило Тарасович	kirill.ast98@gmail.com
4	Башлай Владислав Віталійович	vladbashlay123@gmail.com
5	Бурмагін Нікіта Сергійович	nnburmagin@gmail.com
6	Дереча Олексій Іванович	lollexa215@gmail.com
7	Доценко Денис Олексійович	denis.dotsenko.al@gmail.com
8	Дячко Діана Євгенівна	dianadiachko@gmail.com
9	Євстігнєєв Ростислав Денисович	yevstihnieiev.uni@gmail.com
10	Згурський Данило Олександрович	danilzgurs@gmail.com
11	Здор Віталій Віталійович	azot587@gmail.com
12	Іванченко Софія Олександрівна	sonyaivanchenko1@gmail.com
13	Кабанов Артем Іванович	artemkabanov108@gmail.com
14	Каптюх Костянтин Денисович	ravolut1onr@gmail.com
15	Караваєв Олександр Олександрович	koshak327@gmail.com
16	Колєснік Максим Олександрович	maksimkolesnik99@gmail.com
17	Крутько Дмитро Геннадійович	durkomom735@gmail.com
18	Кузін Іван Олексійович	ivankuzin3049@gmail.com
19	Лукіна Дарія Дмитрівна	lukinadariia@gmail.com
20	Матяш Ярослав Олександрович	matyhya23@gmail.com
21	Мірошниченко Богдан Володимирович	bogdanmirosnicenko@gmail.com
22	Могилін Владислав Олександрович	mogilinvlad@gmail.com
23	Мороко Ярослав Владиславович	legionemptys@gmail.com
24	Петрик Денис Сергійович	denispetrik714@gmail.com
25	Пригарін Богдан Сергійович	pryharin.bohdan@gmail.com
26	Сандак Поліна Андріївна	charasailar@gmail.com
27	Сапа Олександр Олександрович	sahasapa2000@gmail.com
28	Сапа Сергій Олександрович	serchie228@gmail.com
29	Тарасенко Нікіта Сергійович	tarasenkonikita27@gmail.com
30	Темченко Станіслав Андрійович	temchenkosv8@gmail.com
31	Хоренженко Іван Сергійович	ivankhorenzhenko22@gmail.com
32	Черевичний Ярослав Сергійович	yarik16122006@gmail.com
33	Черненко Едуард Кирилович	edikchernenko133@gmail.com
34	Шарій Юрій Андрійович	shariyya777@gmail.com