Лабораторна робота “Семафори”

Мета: Вивчити механізми синхронізації процесів при роботі зі спільною пам’ятью.

Задачи:

1. Ознайомитись з програмними методами сінхронізації процесів. З’ясувати недоліки програмної реалізації.
2. Ознайомитись з механізмом взаємовиключеня при підтримці операційної системи.
3. З’ясувати які можливості надає ОС для реалізації механізму взаємовиключень.
4. Ознайомитись з принципами роботи з семафором. Що таке семафор, як його використання дозволяє синхронізувати роботу процесів.
5. Створити програму відповідно індівідуального завдання.

Номер індивідуального завдання обчислюється за формулою:

```
(< номер завдання >) = (< свій номер у списку > - 1) % 5 + 1
```

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

Завдання:

Згенерувати масив **a** випадкових натуральних чисел та записати їх у файл **array.dat**. Кількість елементів масиву **~10^8**, значення **$0 < a_i < 10^9$**.

Створити програму яка отримає дані з файлу **array.dat**. Обробка даних повинна відбуватися у паралельному режимі двома процесами (батьківському та дочірньому). Результат обробки даних повинен бути записан у файл **result.dat** батьківським процесом. Програма повина гарантувати однакову завантаженність кожного з процесів.

1. Обчислить суму тільки простих значень заданого масиву.
2. Упорядкувати масив по неспадінню. Для упорядкування використовувати Бульбашковий алгорітм, розподіл завдань між процесами – статичний. Після упорядкування частей масиву задіяти алгоритм “Злиття списків”

3. Паралельна заміна елементів масиву: Процес **P1** замінює всі непарні елементи на **0**. Процес **P2** замінює всі парні елементи, значення яких більше за **0** на **1.**

4. Попарна перестановка значень елементів масиву. Процеси переставляють значення двох сусідніх елементів починаючи від початку масиву.

5. Перестановка значень: Спочатку записуються значення усіх не простих елементів, а потім усіх простих. Відносний порядок між не простими елементами та відносний порядок між простими елементами не змінювати.

Семафори

Механізм розподільчої пам’яті дає змогу декільком процесам мати доступ до одного і того ж сегменту пам’яті, що прискорює взаімодію між процесами. Але у разі одночасної обробки одного і того ж сегменту пам’яті виникає проблема порушення цілістності даних, що приводить до отримання невизніченого результату. Для запобігання таких помилок необхідно забеспечити монопольне володіння сегментом пам’яті (або його частини) на час обробки даних одним з процесів, який перший отримав доступ до такого сегменту пам’яті.

Існує декілька способів забеспечити синхронізаціє роботи процесів. Найбільш ефективних з них – це звернутись до операційної системи за допомогою, як до арбітра. Операційна система має можливіть заблокувати процес (змінив стан процесу), який намагається отримати дозвіл на обробку даних у той час, коли іншій процес виконує свою обробку цих даних, а після того, як він завершить цю обробку, чекаючий процес буде разблокован і отримає необхідний дозвіл.

Процес, який намагається зайти до критичного розділу програми повинен перевірити стан цього розділу: вільний чи він вже використовується іншим процесом. У випадку, якщо вільний, то процес виставляє стан “Зайнято” і виконує свою роботу, по завершанню якої повертає стан розділу “Вільно”. У випадку, якщо стан “Зяйнято”, то процес переводиться в стан блокований. Коли стан критичного розділу програми буде змінено, процес буде разблоковано і у нього з’явиться шанс отримати дозвіл на вхід до критичної частини програми.

Нагляд за роботою процесів у критичній частині програми виконує так званий семафор. Семафор – це механізм, який забеспечує сінхронізацію роботи процесів. Щоб створити семафор, процес – сервер повинен викликати функцію ядра операційної системи semget(), ініниціалізувати семафор (надати йому початкове значення), а по завершанню роботи з семафором, видалити його. Процеси – кліенти реєструють у себе цей семафор.

Створення семафору:

```
#include <sys/sem.h>
int sid = semget(key, count, flags);
```

Функція створює **count** семафорів (семафори індексуються починаючи з нуля), ключ беспеки **key** та флаги **flags** (дивись лабораторну роботу “*Спільна пам’ять*”). Функція у разі успіху повертає файловий дескриптор семафору, а у разі невдачи **-1**. Наприклад, створення одного семафору (для сервера):

```
pid_ t sid;
if ((sid = semget(key, 1, IPC_CREAT | 0666)) < 0) {
  printf(“Semget error\n”);
  exit(1);
}
```

Кліент для використання вже існуючего семафору вказує флаг **IPC_EXCL**:

```
pid_ t sid;
if ((sid = semget(key, 1, IPC_EXCL )) < 0) {
  printf(“Semget error\n”);
  exit(1);
}
```

Ініцілазація семафору (тільки для сервера):

```
#include <sys/sem.h>
int semctl(sid, index, SETVAL, val);
```

Наприклад (індекс семафору **index = 0**, ініціалізація значенням **val = 1**):

```
if (semctl(sid, 0, SETVAL, 1) < 0) {
  printf(“Semctl error\n”);
  exit(1);
}
```

Видалення одного семафору. Це робить тільки сервер:

```
int semctl(sid, 1, IPC_RMID /*, NULL */);
```

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

Зауваження: Щоб зменьшити верогідність втручання в данні семафору 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 семафору. Вгадати значення двух параметрів злодію буде набагато складніше, що підвищить надійність роботи семафору.

Робота з семафором

Перед входом до критичного розділу програми необхідно перевірити стан семафору. Це робиться за допомогою системного виклику **semop()**. Якщо критичний розділ вільний, то функція змінить стан семафору і завершить роботу, що дасть процесу продовжити виконання дій у критичній частині програми. По завершанню роботи у критичній частині програми необхідно знов викликати функцію **semop()**, але з іншім набором параметрів для відновлення статусу семафору. Усі дії над семафором функція **semop()** виконує атомарно, що гарантує коректне взамоємовиключення процесів при роботі в критичній частині програми. Якщо критичний розділ вже виконується іншим процесом, то викликав функцію **semop()** процес блокується (**semop()** не завершує роботи, не дозволяя процесу продовдити подальші дії).

Розглянемо роботу функції для випадку бінарного семафору з двома станами: **1** – вільно, **0** – зайнято.

Перед викликом функції необхідно створити структуру **sembuf**, дані якої будуть визначати дії функції:

Структура **sembuf** для **semop()**:

```
struct sembuf {
  unsigned short sem_num,        // індекс семафора
  short sem_op,                  // операції над семафором
  short sem_flg                  // флаги операцій (частіше 0 – без флагів)
};
```

У найпростішому випадку, коли семафор ініціалізован одиницей, операції над семафором зводяться до віднімання одиниці при вході до критичної частини програми, або додавання – у разі вихіду. У цьому випадку зручно оголосити дві структури (для семафору з індексом **0**):

```
struct sembuf lock   = { 0, -1, 0 };
```

```
struct sembuf unlock = { 0, +1, 0 };
```

semop() - керує станом семафору

```
int semop(int semid,              // дескриптор семафору
          struct sembuf* sops,    // операції
          size_t nsops            // кількість операцій (в нашому випадку 1)
);
```

Повертає **0** або **-1** у разі невдачи.

Наприклад:

```
// Вхід до критичної секції
if (semop(semid, &lock, 1) < 0) {
   printf("Error semop\n");
   exit(1);
}
```

// Критичний розділ програми

```
// Вихід з критичної секції
if (semop(semid, &unlock, 1) < 0) {
   printf("Error semop\n");
   exit(1);
}
```

Приклад

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/sem.h>
#include <sys/wait.h>

key_t key;
pid_t sid;

struct sembuf   lock = { 0, -1, 0 };
struct sembuf unlock = { 0, +1, 0 };

void print(const char* s) {
   semop(sid, &lock, 1);
   // Критична секція
   printf("%s 1\n", s);
```

```
  printf("%s 2\n", s);
  semop(sid, &unlock, 1);
}

int main() {
  if ((key = ftok("/tmp", 3128)) < 0) {
    printf("%s\n", "Помилка створення ідентифікаціонного ключа");
    exit(1);
  }
  if ((sid = semget(key, 1, IPC_CREAT | 0666)) < 0) {
    printf("%s\n", "Помилка створення семафора");
    exit(1);
  }
  if (semctl(sid, 0, SETVAL, 1) < 0) {
    printf("%s\n", "Помилка ініціалізації семафора");
    exit(1);
  }
  int i = 0;
  if (fork() == 0) {
  while (i++ < 1000)
    print("CHILD");
    exit(0);
  }
  while (i++ < 1000)
    print("PARENT");
  wait(NULL);
  if (semctl(sid, 1, IPC_RMID, NULL) < 0) {
    printf("%s\n", "Помилка видалення семафора");
    exit(1);
  }
  exit(0);
}
```

Додаток 1

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

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