Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий Фізико-технічний інститут

ОПЕРАЦІЙНІ СИСТЕМИ

Комп'ютерний практикум
Робота №9

Виконав:
студент групи ФІ-12
Завалій Олександр
Перевірив:
Кірієнко О.В.

Київ-2023

Робота №9.
Засоби синхронізації і взаємодії процесів

Мета:

Оволодіння практичними навичками використання засобів міжпроцесової взаємодії в Linux.

Варіант №5

Зміст індивідуального завдання:

Програма моделює роботу примітивної СКБД, що зберігає єдину таблицю в оперативній пам'яті. Виконуючи деякі цикли робіт, K породжених процесів за допомогою черги повідомлень передають батьківському процесові номер рядка і вміст, на який потрібно замінити дані, що у ньому зберігаються. Батьківський процес виконує зазначену операцію і повертає попередній вміст рядка, що був змінений.

```
alex@Oleksandr:~/labs/lab_9$ cat lab9.cpp
#include <iostream>
#include <vector>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <sys/msg.h>

#define MSG_KEY 12345

using namespace std;

struct Message {
    long mtype;
    int row;
    int data;
};

void send_message(int msgqid, int row, int data) {
    Message msg;
    msg.mtype = 1;
    msg.row = row;
    msg.data = data;

    msgsnd(msgqid, &msg, sizeof(msg), 0);
}

void receive_message(int msgqid, vector<int>& table, vector<int>& previousData) {
    Message msg;

    msgrcv(msgqid, &msg, sizeof(msg), 1, 0);

    previousData[msg.row - 1] = table[msg.row - 1];

    table[msg.row - 1] = msg.data;
}

void print_table(const vector<int>& table, const vector<int>& previousData) {
    for (int i = 0; i < table.size(); i++) {
        cout << "PID: " << getpid() << " Row " << i + 1 << ": " << previousData[i] << " -> " << table[i] << endl;
    }
}

int main() {
    int msgqid = msgget(MSG_KEY, IPC_CREAT | 0666);

    vector<int> table = {1, 2, 3, 4, 5};

    vector<int> previousData(5);

    cout << "Before:" << endl;
    for (int i = 0; i < table.size(); i++) {
        cout << "Row " << i + 1 << ": " << table[i] << endl;
    }
```

```
    cout << "\n----------------------------\n";
    for (int i = 0; i < 5; i++) {
        pid_t pid = fork();

        if (pid == 0) {
            send_message(msgqid, i + 1, getpid());

            print_table(table, previousData);

            return 0;
        }
    }

    for (int i = 0; i < 5; i++) {
        receive_message(msgqid, table, previousData);
    }

    for (int i = 0; i < 5; i++) {
        send_message(msgqid, i + 1, previousData[i]);
    }

    for (int i = 0; i < 5; i++) {
        wait(NULL);
    }

    cout << "\nAfter:" << endl;
    for (int i = 0; i < table.size(); i++) {
        cout << "Row " << i + 1 << ": " << table[i] << endl;
    }

    msgctl(msgqid, IPC_RMID, NULL);

    return 0;
alex@Oleksandr:~/labs/lab_9$
```

```
alex@Oleksandr:~/labs/lab_9$ g++ -pthread -o lab9 lab9.cpp
alex@Oleksandr:~/labs/lab_9$ ./lab9
Before:
Row 1: 1
Row 2: 2
Row 3: 3
Row 4: 4
Row 5: 5

----------------------------
PID: 4564 Row 1: 0 -> 1
PID: 4564 Row 2: 0 -> 2
PID: 4564 Row 3: 0 -> 3
PID: 4564 Row 4: 0 -> 4
PID: 4564 Row 5: 0 -> 5
PID: 4566 Row 1: 0 -> 1
PID: 4566 Row 2: 0 -> 2
PID: 4566 Row 3: 0 -> 3
PID: 4566 Row 4: 0 -> 4
PID: 4566 Row 5: 0 -> 5
PID: 4565 Row 1: 0 -> 1
PID: 4565 Row 2: 0 -> 2
PID: 4565 Row 3: 0 -> 3
PID: 4565 Row 4: 0 -> 4
PID: 4565 Row 5: 0 -> 5
PID: 4568 Row 1: 0 -> 1
PID: 4568 Row 2: 0 -> 2
PID: 4568 Row 3: 0 -> 3
PID: 4568 Row 4: 0 -> 4
PID: 4568 Row 5: 0 -> 5
PID: 4567 Row 1: 0 -> 1
PID: 4567 Row 2: 0 -> 2
PID: 4567 Row 3: 0 -> 3
PID: 4567 Row 4: 0 -> 4
PID: 4567 Row 5: 0 -> 5

After:
Row 1: 4564
Row 2: 4565
Row 3: 4566
Row 4: 4567
Row 5: 4568
alex@Oleksandr:~/labs/lab_9$
```

Висновки

Міжпроцесна взаємодія (IPC) є важливою концепцією і має велике значення для розробки програм. У контексті програмування існує кілька методів для здійснення IPC, кожен з яких має свої переваги та обмеження.

Один із способів це використання черг повідомлень (Message Queues). Черги повідомлень дозволяють процесам обмінюватись даними шляхом надсилання повідомлень до черги. З цієї ж черги інші процеси можуть повідомлення отримати. Цей метод забезпечує асинхронну комунікацію і може бути корисним у випадках, коли потрібно передавати невеликі об'єкти даних між процесами.

Ще один метод IPC це спільна пам'ять (Shared Memory). Використовуючи спільну пам'ять, кілька процесів можуть отримувати доступ до одного й того ж блоку пам'яті. Це дозволяє процесам ефективно обмінюватися великими об'єктами даних, не потребуючи копіювання даних між процесами. Однак, спільна пам'ять вимагає синхронізації для уникнення конфліктів доступу до спільних ресурсів.

Також, у Linux існує можливість використання сокетів (Sockets) для забезпечення мережевої взаємодії між процесами, які працюють на різних вузлах мережі. Сокети дозволяють передавати дані через мережу, використовуючи різні протоколи, такі як TCP або UDP.

При розробці програм, які мають взаємодію між процесами, важливо враховувати характеристики і обмеження кожного методу IPC, а також забезпечити відповідну синхронізацію для надійної і безпечної взаємодії між процесами.