Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Шатунова Юлия Викторовна
Группа: М8О-208Б-20
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/s0bakkaa/OS/tree/main/CP

Постановка задачи

Необходимо написать 3 программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор, пока программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессорного взаимодействия выбирает студент.

Общие сведения о программе: программа состоит из четырёх файлов: A.cpp, B.cpp, C.cpp и main.cpp, который объединяет в себе три предыдущих файла.

Общий метод и алгоритм решения: В начале работы в main.cpp создаются два дочерних процесса для B и C, а родительский процесс замещается программой A с помощью execl, сначала A с помощью getline считывает строку, передаёт в B количество считанных символов, а в C — количество считанных символов и саму строку посимвольно, затем B выводит количество введённых символов, C выводит строку и передаёт B количество выведенных символов, после чего B выводит количество выведенных символов и цикл начинается заново. Межпроцессорное взаимодействие основано на семафорах и pipe.

Исходный код:

main.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>
#include <stdarg.h>

int human_get(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void human_set(sem_t *semaphore, int n)
{
    while (human_get(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (human_get(semaphore) > n)
```

```
    {
        sem_wait(semaphore);
    }
}

int main()
{
    int fdAC[2];
    int fdAB[2];
    int fdBC[2];
    pipe(fdAC);
    pipe(fdAB);
    pipe(fdBC);
    sem_unlink("_semA");
    sem_unlink("_semB");
    sem_unlink("_semC");
    sem_t* semA = sem_open("_semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0777, 0);
    if ((semA == SEM_FAILED)||(semB == SEM_FAILED)||(semC == SEM_FAILED))
    {
        perror("sem_open");
        return -1;
    }
    std::cout << "Enter some strings:\n";
    pid_t C = fork();
    if (C == -1)
    {
        perror("fork");
        return -1;
    }
    if (C == 0)
    {
        pid_t B = fork();
        if (B == -1)
        {
            perror("fork");
            return -1;
        }
        if (B == 0)
        {
            execl("B",    std::to_string(fdAB[0]).c_str(),    std::to_string(fdAB[1]).c_str(),
std::to_string(fdBC[0]).c_str(), std::to_string(fdBC[1]).c_str(), NULL);
        }
        else
        {
            execl("C",    std::to_string(fdAC[0]).c_str(),    std::to_string(fdAC[1]).c_str(),
std::to_string(fdBC[0]).c_str(), std::to_string(fdBC[1]).c_str(), NULL);
        }
    }
    else
    {
        execl("A",      std::to_string(fdAC[0]).c_str(),      std::to_string(fdAC[1]).c_str(),
std::to_string(fdAB[0]).c_str(), std::to_string(fdAB[1]).c_str(), NULL);
    }
    return 0;
}
```

A.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
```

```
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdarg.h>
#include <signal.h>

int human_get(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void human_set(sem_t *semaphore, int n)
{
    while (human_get(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (human_get(semaphore) > n)
    {
        sem_wait(semaphore);
    }
}

int main(int args, char* argv[])
{
    int fdAC[2];
    fdAC[0] = atoi(argv[0]);
    fdAC[1] = atoi(argv[1]);
    int fdAB[2];
    fdAB[0] = atoi(argv[2]);
    fdAB[1] = atoi(argv[3]);
    sem_t* semA = sem_open("_semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0777, 0);
    while(1)
    {
        std::string str;
        getline(std::cin, str);
        if (str == "END")
        {
            human_set(semA, 2);
            human_set(semB, 2);
            human_set(semC, 2);
            break;
        }
        int size = str.length();
        write(fdAC[1], &size, sizeof(int));
        write(fdAB[1], &size, sizeof(int));
        for (int i = 0; i < size; ++i)
        {
            write(fdAC[1], &str[i], sizeof(char));
        }
        human_set(semB, 1);
        human_set(semA, 0);
        while (human_get(semA) == 0)
        {
            continue;
        }
    }
    sem_close(semA);
    sem_destroy(semA);
    sem_close(semB);
```

```
    sem_destroy(semB);
    sem_close(semC);
    sem_destroy(semC);
    close(fdAC[0]);
    close(fdAC[1]);
    close(fdAB[0]);
    close(fdAB[1]);
    return 0;
}
```

B.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdarg.h>
#include <signal.h>

int human_get(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void human_set(sem_t *semaphore, int n)
{
    while (human_get(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (human_get(semaphore) > n)
    {
        sem_wait(semaphore);
    }
}

int main(int args, char* argv[])
{
    int fdAB[2];
    fdAB[0] = atoi(argv[0]);
    fdAB[1] = atoi(argv[1]);
    int fdBC[2];
    fdBC[0] = atoi(argv[2]);
    fdBC[1] = atoi(argv[3]);
    sem_t* semA = sem_open("_semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0777, 0);
    while (1)
    {
        while(human_get(semB) == 0)
        {
            continue;
        }
        if (human_get(semB) == 2)
        {
            break;
        }
        int size;
        read(fdAB[0], &size, sizeof(int));
```

```
        std::cout << "Number of input symbols is " << size << std::endl;
        human_set(semC, 1);
        human_set(semB, 0);
        while (human_get(semB) == 0)
        {
            continue;
        }
        if (human_get(semB) == 2)
        {
            break;
        }
        read(fdBC[0], &size, sizeof(int));
        std::cout << "Number of output symbols is " << size << std::endl;
        human_set(semA, 1);
        human_set(semB, 0);
        while(human_get(semB) == 0)
        {
            continue;
        }
        if (human_get(semB) == 2)
        {
            break;
        }
    }
    sem_close(semA);
    sem_close(semB);
    sem_close(semC);
    close(fdAB[0]);
    close(fdAB[1]);
    close(fdBC[0]);
    close(fdBC[1]);
    return 0;
}
```

C.cpp

```
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdarg.h>
#include <signal.h>

int human_get(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void human_set(sem_t *semaphore, int n)
{
    while (human_get(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (human_get(semaphore) > n)
    {
        sem_wait(semaphore);
    }
}
```

```
int main(int args, char* argv[])
{
    int fdAC[2];
    fdAC[0] = atoi(argv[0]);
    fdAC[1] = atoi(argv[1]);
    int fdBC[2];
    fdBC[0] = atoi(argv[2]);
    fdBC[1] = atoi(argv[3]);
    sem_t* semA = sem_open("_semA", O_CREAT, 0777, 1);
    sem_t* semB = sem_open("_semB", O_CREAT, 0777, 0);
    sem_t* semC = sem_open("_semC", O_CREAT, 0777, 0);
    while(1)
    {
        while(human_get(semC) == 0)
        {
            continue;
        }
        if (human_get(semC) == 2)
        {
            break;
        }
        int size;
        std::string str;
        read(fdAC[0], &size, sizeof(int));
        int t = 0;
        for (int i = 0; i < size; ++i)
        {
            char c;
            read(fdAC[0], &c, sizeof(char));
            str.push_back(c);
            t = i;
        }
        ++t;
        std::cout << str << std::endl;
        write(fdBC[1], &t, sizeof(int));
        human_set(semB, 1);
        human_set(semC, 0);
    }
    sem_close(semA);
    sem_close(semB);
    sem_close(semC);
    close(fdAC[0]);
    close(fdAC[1]);
    close(fdBC[0]);
    close(fdBC[1]);
    return 0;
}
```

Makefile

```
files: main A B C

main: main.cpp
	g++ -pthread main.cpp -o main

A: A.cpp
	g++ -pthread A.cpp -o A

B: B.cpp
	g++ -pthread B.cpp -o B

C: C.cpp
```

```
g++ -pthread C.cpp -o C
```

Демонстрация работы программы

```
[yulia@andromeda src]$ ./main
Enter some strings:
abc
Number of input symbols is 3
abc
Number of output symbols is 3
abc def
Number of input symbols is 7
abc def
Number of output symbols is 7
*&*(^869yhih
Number of input symbols is 12
*&*(^869yhih
Number of output symbols is 12
```

Выводы

При написании курсового проекта я закрепила знания и навыки, полученные мной во время прохождения курса операционных систем.