Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу «Операционные системы»**

Студент: Чистяков Д. М.

Группа: М8О-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: 04.04.22

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

Цель работы

Приобретение практических навыков в:

* Использовании знаний, полученных в течение курса
* Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Провести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант:

Необходимо написать 3 программы. Далее будем обозначать эти программы A, B и C.

Программа А принимает из стандартного потока ввода строки, а далее их отправляет программе С. Отправка строк должна производиться построчно. Программа С печатает в стандартный вывод строку, полученную от программы А. После получения программа С отправляет программе А сообщение о том, что строка получена. До тех пор, пока программа А не примет “сообщение о получении” от программы С, она не может отправлять следующую строку программе С. Программа В пишет в стандартный поток вывода количество отправленных символов программой А и количество принятых символов программой С. Данную информацию программа В получает от программ А и С соответственно.

Общие сведения о программе

Программа компилируется при помощи make файла в 3 исполняемых файла А.out, В.out и С.out. Для реализации, поставленной задачи, в программе используются следующие системные вызовы:

pipe - создает однонаправленный канал данных, который можно использовать для взаимодействия между процессами.

fork - создает копию текущего процесса, который является дочерним процессом для текущего процесса.

execl - выполняет файл, заменяя текущий образ процесса новым образом процесса.

Так как для межпроцессного взаимодействия выбран системный вызов pipe, программа не нуждается в дополнительной синхронизации параллельной работы процессов, так как при попытке чтения из пустого буфера процесс чтения блокируется до появления данных.

Листинг программы

Makefile:

all: A.out B.out C.out

A.out: A.cpp

g++ A.cpp -o A.out

B.out: B.cpp

g++ B.cpp -o B.out

C.out: C.cpp

g++ C.cpp -o C.out

clean:

rm -rf \*.o A.out B.out C.out

A.cpp:

#include <iostream>

#include <string>

#include <unistd.h>

int main ()

{

    int pipe\_AC[2];

    int pipe\_AB[2];

    int pipe\_CA[2];

    int pipe\_CB[2];

    pipe(pipe\_AC);

    pipe(pipe\_AB);

    pipe(pipe\_CA);

    pipe(pipe\_CB);

    pid\_t id\_C = fork();

    if (id\_C == -1)

    {

        std::cout << "Fork error!" << std::endl;

        return -1;

    }

    else if (id\_C == 0)

    {

        char AC[32];

        char CA[32];

        char CB[32];

        sprintf(AC, "%d", pipe\_AC[0]);

        sprintf(CA, "%d", pipe\_CA[1]);

        sprintf(CB, "%d", pipe\_CB[1]);

        execl("./C.out", AC, CA, CB, (char\*)(NULL));

    }

    else

    {

        pid\_t id\_B = fork();

        if (id\_B == -1)

        {

            std::cout << "Fork error!" << std::endl;

            return -1;

        }

        else if (id\_B == 0)

        {

            char AB[32];

            char CB[32];

            sprintf(AB, "%d", pipe\_AB[0]);

            sprintf(CB, "%d", pipe\_CB[0]);

            execl("./B.out", AB, CB, (char\*)(NULL));

        }

        else

        {

            std::string Str;

            while (true)

            {

                std::cin >> Str;

                if(!std::cin.good()) break;

                size\_t Sended\_char\_count = Str.size();

                uint8\_t confirm;

                write(pipe\_AB[1], &Sended\_char\_count, sizeof(size\_t));

                write(pipe\_AC[1], &Sended\_char\_count, sizeof(size\_t));

                write(pipe\_AC[1], Str.c\_str(), Sended\_char\_count);

                read(pipe\_CA[0], &confirm, sizeof(uint8\_t));

            }

        }

    }

    return 0;

}

B.cpp:

#include <iostream>

#include <unistd.h>

int main (int argc, char\* argv[])

{

    int pipe\_AB = atoi(argv[0]);

    int pipe\_CB = atoi(argv[1]);

    size\_t Sended\_char\_count;

    size\_t Received\_char\_count;

    while (read(pipe\_AB, &Sended\_char\_count, sizeof(size\_t)) > 0)

    {

        std::cout << "B: char count sended from program A = " << Sended\_char\_count << std::endl;

        read(pipe\_CB, &Received\_char\_count, sizeof(size\_t));

        std::cout << "B: char count received by program C = " << Received\_char\_count << std::endl;

        std::cout << std::endl;

    }

}

C.cpp:

#include <iostream>

#include <string>

#include <unistd.h>

int main (int argc, char\* argv[])

{

    int pipe\_AC = atoi(argv[0]);

    int pipe\_CA = atoi(argv[1]);

    int pipe\_CB = atoi(argv[2]);

    size\_t Sended\_char\_count;

    while (read(pipe\_AC, &Sended\_char\_count, sizeof(size\_t)) > 0)

    {

        char char\_str[Sended\_char\_count];

        read(pipe\_AC, char\_str, Sended\_char\_count);

        std::string Str(char\_str, Sended\_char\_count);

        std::cout << "C: string from program A: " << Str << std::endl;

        size\_t Received\_char\_count = Str.size();

        write(pipe\_CB, &Received\_char\_count, sizeof(size\_t));

        uint8\_t confirm = 1;

        write(pipe\_CA, &confirm, sizeof(uint8\_t));

    }

}

**Пример работы**

den@den-VirtualBox:~/Документы/kp$ make

g++ A.cpp -o A.out

g++ B.cpp -o B.out

g++ C.cpp -o C.out

den@den-VirtualBox:~/Документы/kp$ ./A.out

abc

B: char count sended from program A = 3

C: string from program A: abc

B: char count received by program C = 3

12345

C: string from program A: 12345

B: char count sended from program A = 5

B: char count received by program C = 5

HelloWorld

C: string from program A: HelloWorld

B: char count sended from program A = 10

B: char count received by program C = 10

**Вывод**

Использование вызовов pipe при межпроцессном взаимодействии очень удобно в тех случаях, когда процессы пересылают друг другу небольшое

кол-во однотипных данных. Главное преимущество pipe заключается в том, что из-за блокировки процесса чтения при пустом буфере, пропадает необходимость синхронизировать их работу. Однако, при передаче большого количества разнотипных данных следует использовать другие способы взаимодействия, например, mmap. Но в этом случае придется использовать семафор или иные способы синхронизации процессов, так как никакие вызовы уже не будут блокироваться по умолчанию.