Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Курсовой проект

«межпроцессное взаимодействие»

Студент: Марочкин И.А.
Группа: М8О-206Б-19
Преподаватель: Соколов А.А.
Дата: 24.04.2021
Оценка:

Москва, 2020

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 A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производиться построчно. Программа C печатает в стандартный вывод строку, полученную 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 получении” от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный поток вывода количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно.

Общие сведения о программе

Программа компилируется при помощи Makefile в 3-х исполняемых файла `A`, `B` и `C`. 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.

Из-за того, что для межпроцессного взаимодействия были выбраны 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

program_A.cpp

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

using std::string;
using std::cin;
using std::cout;

int main (){
    int pipe_A2C[2];
    int pipe_A2B[2];
    int pipe_C2A[2];
    int pipe_C2B[2];

    pipe(pipe_A2C);
    pipe(pipe_A2B);
    pipe(pipe_C2A);
    pipe(pipe_C2B);

    pid_t id_C = fork();
    if (id_C == -1){
        throw std::runtime_error("ERROR: fork error");
    }
    else if (id_C == 0){ // program_C
            char A2C[32];
            char C2A[32];
            char C2B[32];

            sprintf(A2C, "%d", pipe_A2C[0]);
            sprintf(C2A, "%d", pipe_C2A[1]);
            sprintf(C2B, "%d", pipe_C2B[1]);

            execl("./C", A2C, C2A, C2B, (char*)(NULL));
    } // program_C end
    else {
        pid_t id_B = fork();
        if (id_B == -1){
            throw std::runtime_error("ERROR: fork error");
        }
        else if (id_B == 0){ // program_B
            char A2B[32];
            char C2B[32];
            sprintf(A2B, "%d", pipe_A2B[0]);
            sprintf(C2B, "%d", pipe_C2B[0]);
            execl("./B", A2B, C2B, (char*)(NULL));
        } // program_B end
        else { // program_A
            string Str;
            while (cin >> Str){
                size_t Sended_char_count = Str.size();
                uint8_t confirm;

                write(pipe_A2B[1], &Sended_char_count, sizeof(size_t));
                write(pipe_A2C[1], &Sended_char_count, sizeof(size_t));
```

```
                write(pipe_A2C[1], Str.c_str(), Sended_char_count);

                read(pipe_C2A[0], &confirm, sizeof(uint8_t));
            }
        } // program_A end
    }
    return 0;
}
```

program_B.cpp

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

int main (int argc, char* argv[]){
    int pipe_A2B = atoi(argv[0]);
    int pipe_C2B = atoi(argv[1]);

    size_t Sended_char_count;
    size_t Received_char_count;

    while (read(pipe_A2B, &Sended_char_count, sizeof(size_t)) > 0){
        std::cout << "B: char count sended from program A = " << Sended_char_count <<
std::endl;
        read(pipe_C2B, &Received_char_count, sizeof(size_t));
        std::cout << "B: char count received by program C = " << Received_char_count <<
std::endl;
        std::cout << std::endl;
    }
}
```

program_C.cpp

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

int main (int argc, char* argv[]){
    int pipe_A2C = atoi(argv[0]);
    int pipe_C2A = atoi(argv[1]);
    int pipe_C2B = atoi(argv[2]);

    size_t Sended_char_count;

    while (read(pipe_A2C, &Sended_char_count, sizeof(size_t)) > 0){
        char char_str[Sended_char_count];
        read(pipe_A2C, 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_C2B, &Received_char_count, sizeof(size_t));

        uint8_t confirm = 1;
        write(pipe_C2A, &confirm, sizeof(uint8_t));
    }
}
```

Пример работы

```
Vanya:Src ivan$ ./A
jvnajkfdnvkajdf
B: char count sended from program A = 15
C: string from program A: jvnajkfdnvkajdf
B: char count received by program C = 15

123456789
B: char count sended from program A = 9
C: string from program A: 123456789
B: char count received by program C = 9

Hello!
B: char count sended from program A = 6
C: string from program A: Hello!
B: char count received by program C = 6
```

Вывод

Использование pipe’ов при межпроцессном взаимодействии очень удобно в тех случаях, когда процессы “перекидывают” друг другу небольшое кол-во однотипных данных. Главное преимущество pipe’ов 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.