Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

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

Студент: Маринин Иван Сергеевич

Группа: М8О-208Б-20

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

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 Makefile в 3-х исполняемых файла А, В и С. 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

***A.cpp***

#include <iostream>

#include <string>

#include <unistd.h>

using namespace std;

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){

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

return -1;

}

else if (id\_C == 0){ // program\_C

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", AC, CA, CB, (char\*)(NULL));

} // program\_C end

else {

pid\_t id\_B = fork();

if (id\_B == -1){

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

return -1;

}

else if (id\_B == 0){ // program\_B

char AB[32];

char CB[32];

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

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

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

} // program\_B end

else { // program\_A

string Str;

while (true){

cin >> Str;

if(!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));

}

} // program\_A end

}

return 0;

}

***B.cpp***

#include <iostream>

#include <unistd.h>

using namespace std;

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){

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

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

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

cout << endl;

}

}

***C.cpp***

#include <iostream>

#include <string>

#include <unistd.h>

using namespace std;

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);

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

cout << "C: string from program A: " << Str << 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

ivanmarinin@MacBook-Air-Ivan src % ./A

hello

B: char count sended from program A = 5

C: string from program A: hello

B: char count received by program C = 5

123456789

C: string from program A: 123456789

B: char count sended from program A = 9

B: char count received by program C = 9

afbirbvlkvijjdd

C: string from program A: afbirbvlkvijjdd

B: char count sended from program A = 15

B: char count received by program C = 15

**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.