Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Освоение принципов работы с файловыми системами. Обеспечение обмена данных между процессами посредством технологии «File mapping»**

Группа: М80-210Б-22

Студент: Бонокин Д.С.

Вариант:11

Преподаватель: Соколов А.А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

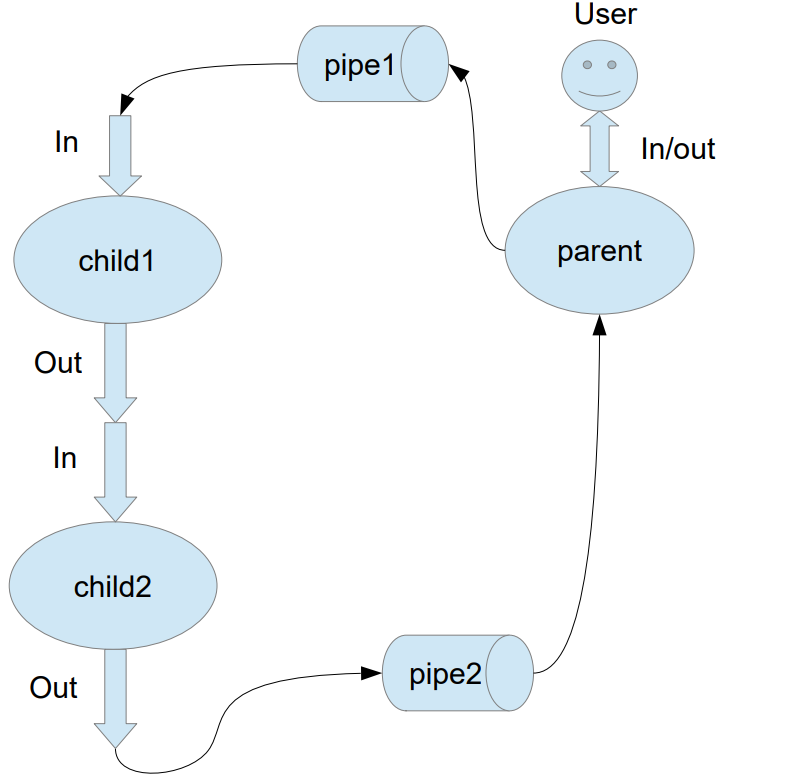
Москва, 2023

Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант № 11



Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «\_».

Листинг программы

**Parent.cpp**

#include <iostream>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/wait.h>

int MAX\_LENGHT = 1024;

void new\_program(const char \*name,const char\* argv){

if(execl(name, name, argv, NULL) == -1){

perror("execl error!\n");

exit(-1);

}

}

int create\_process() {

pid\_t pid = fork();

if (pid == -1) {

perror("Fork error!\n");

exit(-1);

}

return pid;

}

int main(int argc, char\*\* argv) {

if (argc != 2) {

perror("Too few arguments. Usage: ./lab03 NAME\_OF\_FILE");

exit(1);

}

std::string mm\_name(argv[1]);

int fd = shm\_open(mm\_name.c\_str(), O\_CREAT | O\_RDWR, S\_IREAD | S\_IWRITE);

if (fd == -1) {

perror("shm\_open\n");

exit(1);

}

if (ftruncate(fd, sizeof(char) \* MAX\_LENGHT) == -1) {

perror("ftruncate\n");

exit(1);

}

char\* data = (char\*) mmap(NULL, (sizeof(char) \* MAX\_LENGHT), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

char c = getchar();

int i = 0;

while (c != EOF && i < MAX\_LENGHT && c != '\n') {

data[i] = c;

i++;

c = getchar();

}

data[i] = '\n';

pid\_t pid = create\_process();

if (pid == 0) { // child 1

new\_program("../build/child1", mm\_name.c\_str());

} else { // parent

wait(0);

for (int i = 0; data[i] != '\n'; ++i) {

putchar(data[i]);

}

putchar('\n');

munmap(data, (sizeof(char) \* MAX\_LENGHT));

int err = shm\_unlink(mm\_name.c\_str());

if(err == -1){

perror("shm\_unlink");

exit(-1);

}

}

return 0;

}

**Child1.cpp**

#include <iostream>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/wait.h>

int MAX\_LENGHT = 1024;

void new\_program(const char \*name,const char\* argv){

if(execl(name, name, argv, NULL) == -1){

perror("execl error!\n");

exit(-1);

}

}

int create\_process() {

pid\_t pid = fork();

if (pid == -1) {

perror("Fork error!\n");

exit(-1);

}

return pid;

}

int main(int argc, char\*\* argv) {

std::string mm\_name(argv[1]);

int fd = shm\_open(mm\_name.c\_str(), O\_CREAT | O\_RDWR, S\_IREAD | S\_IWRITE);

if (fd == -1) {

perror("shm\_open\n");

exit(-1);

}

if (ftruncate(fd, sizeof(char) \* MAX\_LENGHT) == -1) {

perror("ftruncate\n");

exit(-1);

}

char\* data = (char\*) mmap(NULL, (sizeof(char) \* MAX\_LENGHT), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

for (int i = 0; data[i] != '\n'; i++) {

data[i] = toupper(data[i]);

}

int pid = create\_process();

if (pid == 0) { // child 2

new\_program("../build/child2", argv[1]);

} else { // child 1

wait(0);

}

munmap(data, (sizeof(char) \* MAX\_LENGHT));

return 0;

}

**Child2.cpp**

#include <iostream>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/wait.h>

int MAX\_LENGHT = 1024;

int main(int argc, char\*\* argv) {

std::string mm\_name(argv[1]);

int fd = shm\_open(mm\_name.c\_str(), O\_CREAT | O\_RDWR, S\_IREAD | S\_IWRITE);

if (fd == -1) {

perror("shm\_open\n");

exit(-1);

}

if (ftruncate(fd, sizeof(char) \* MAX\_LENGHT) == -1) {

perror("ftruncate\n");

exit(-1);

}

char\* data = (char\*) mmap(NULL, (sizeof(char) \* MAX\_LENGHT), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

for (int i = 1; data[i] != '\n'; ++i) {

if (data[i] == ' ') {

data[i] = '\_';

}

}

munmap(data, (sizeof(char) \* MAX\_LENGHT));

return 0;

}

Примеры работы

danil@danil-HYM-WXX:~/Desktop/lab\_os/lab3/build$ ./main 1.txt

aaa aaaaaaa ttt fff ff f

AAA\_AAAAAAA\_TTT\_\_\_\_\_\_FFF\_FF\_F\_\_\_\_\_\_

danil@danil-HYM-WXX:~/Desktop/lab\_os/lab3/build$ ./main 1.txt

qqqq qqqqqqqqqqqqqq

QQQQ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_QQQQQQQQQQQQQQ

danil@danil-HYM-WXX:~/Desktop/lab\_os/lab3/build$

Вывод

В ходе выполнения лабораторной работы я познакомился с технологией "File mapping",а также с инструментами, которые операционная система предоставляет для релизации этой технологии. Самое интересное в этой лабораторной работе было организовать синхронизацию между процессами - для этого мне пришлось изучить то, какие бывают средства синхронизации и как их использовать.