Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине: «ОСиСП»

Тема: ««Средства межпроцессного взаимодействия»»

Выполнил:

Студент 2 курса

Группы ПО-7

Лобан К. Ю.

Проверила:

Давидюк Юлия Ивановна

Брест 2022

Цель работы: Изучить принципы работы со средствами межпроцессного взаимодествия

Вариант 29

Задание : Написать программу, которая порождает дочерний процесс, и общается с ним через средства взаимодействия согласно варианту (табл.А), передавая и получая информацию согласно варианту (табл.Б). Передачу и получение информации каждым из процессов сопровождать выводом на экран информации типа "процесс такой-то передал/получил такую-то информацию". Дочерние процессы начинают операции после получения сигнала SIGUSR1 от родительского процесса.



Код:

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <pthread.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <sys/types.h>

using namespace std;

void handler(int signal);

void parent\_writeIntoCommonFile(int file);

void parent\_readFromCommonFile(int file);

void child\_processCommonFile(int file);

const string COMMON\_FILE = "shared";

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

{

srand(time(NULL));

int fd = open(COMMON\_FILE.c\_str(), O\_CREAT | O\_RDWR | O\_TRUNC);

struct sigaction action;

memset(&action, 0, sizeof(action));

action.sa\_handler = handler;

sigset\_t set;

sigemptyset(&set);

sigaddset(&set, SIGUSR1);

sigaddset(&set, SIGUSR2);

action.sa\_mask = set;

sigaction(SIGUSR1, &action, 0);

sigaction(SIGUSR2, &action, 0);

signal(SIGUSR1, handler);

int signal;

pid\_t pid;

pid = fork();

if (pid > 0)

{

cout << "\_\_\_Parent\_\_\_" << endl;

cout << "Pid: " << getpid() << endl;

cout << "Parent is working" << endl;

// Parent work

parent\_writeIntoCommonFile(fd);

// Send signal to child

kill(pid, SIGUSR1);

// Wait SIGUSR2

sigemptyset(&set);

sigaddset(&set, SIGUSR2);

cout << "Parent wait signal" << endl;

sigwait(&set, &signal);

cout << "\_\_\_\_Parent\_\_\_" << endl;

// Parent work

parent\_readFromCommonFile(fd);

cout << "Parent done work" << endl;

}

else if (pid == 0)

{

// Wait SIGUSR1

sigemptyset(&set);

sigaddset(&set, SIGUSR1);

sigwait(&set, &signal);

cout << "Г" << endl;

cout << "| \_\_\_Child\_\_\_" << endl;

cout << "| Pid: " << getpid() << endl;

cout << "| Child is working" << endl;

// Work

child\_processCommonFile(fd);

// Send SIGUSR2 to parent

kill(getppid(), SIGUSR2);

cout << "| Child done work" << endl;

cout << "L" << endl;

exit(0);

}

close(fd);

return 0;

}

void handler(int signal)

{

if (signal == SIGUSR1)

{

cout << "Got SIGUSR1" << endl;

}

else if (signal == SIGUSR2)

{

cout << "Got SIGUSR2" << endl;

}

else

{

cout << "Other signals" << endl;

}

}

// PROT\_WRITE

void parent\_writeIntoCommonFile(int file)

{

string str =

to\_string(rand()) + " " +

to\_string(rand()) + " " +

to\_string(rand()) + " ";

cout << "Parent is writing string: " << str << endl;

write(file, "", str.length());

char \*ptr = (char \*)mmap(NULL, 4096, PROT\_WRITE, MAP\_SHARED, file, 0);

strcpy(ptr, str.c\_str());

munmap(ptr, 4096);

}

// PROT\_READ

void parent\_readFromCommonFile(int file)

{

int fileNumbers[3];

int size, currPos = 0;

struct stat st;

string str;

fstat(file, &st);

size = st.st\_size;

char \*ptr = (char \*)mmap(NULL, size, PROT\_READ, MAP\_SHARED, file, 0);

for (size\_t i = 0; i < size; i++)

{

str += ptr[i];

}

cout << "Parent got string: " << str << endl;

munmap(ptr, size);

}

// PROT\_WRITE | PROT\_READ

void child\_processCommonFile(int file)

{

int fileNumbers[3];

int size, currPos = 0;

struct stat st;

string str;

fstat(file, &st);

size = st.st\_size;

char \*ptr = (char \*)mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file, 0);

for (size\_t i = 0; i < size; i++)

{

char c = ptr[i];

if (c == ' ')

{

fileNumbers[currPos++] = stoi(str);

str.clear();

continue;

}

str += c;

}

// Bubble sort

for (size\_t i = 0; i < 3; i++)

{

for (size\_t j = 0; j < 3; j++)

{

if (fileNumbers[j] > fileNumbers[i])

{

swap(fileNumbers[i], fileNumbers[j]);

}

}

}

string output = to\_string(fileNumbers[0]) + " " +

to\_string(fileNumbers[1]) + " " +

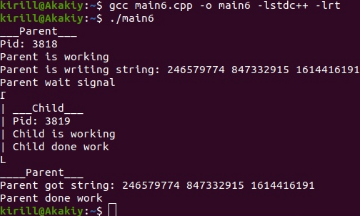
to\_string(fileNumbers[2]) + " ";

strcpy(ptr, output.c\_str());

munmap(ptr, size);

}

Результат:



Вывод: В ходе данной лабораторной работы изучил основы работы со средствами межпроцессного взаимодействия