## МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

## «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра Вычислительной техники**

## ОТЧЕТ

**по лабораторной работе №7**

## по дисциплине «Организация процессов и программирование в среде Linux»

**Тема: Обмен данными через канал**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Яловега Н.В. |
| Преподаватель |  | Разумовский Г.В. |

Санкт-Петербург

2022

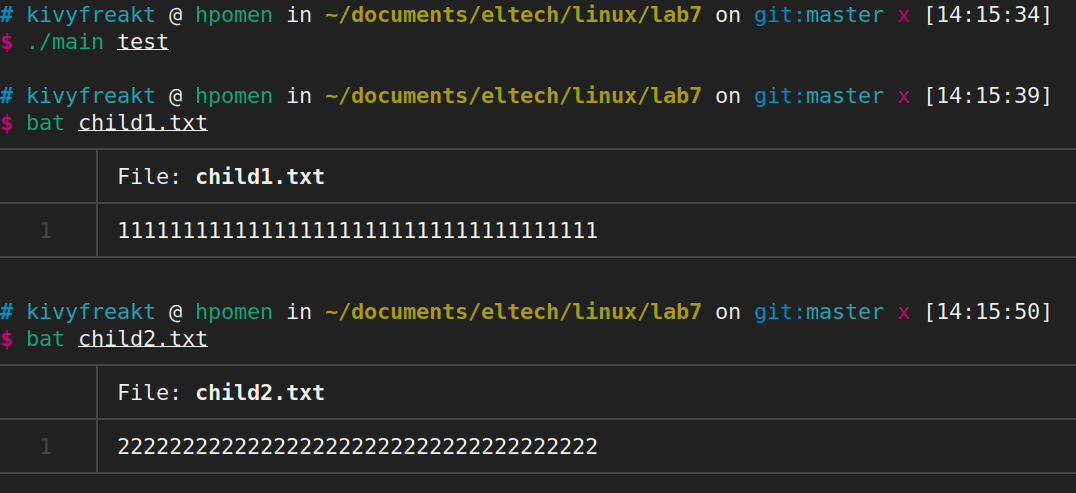
# Цель работы

Знакомство с механизмом обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен.

# Задание

Написать программу, которая обменивается данными через канал с двумя потомками. Программа открывает входной файл, построчно читает из него данные и записывает их в канал. Потомки выполняют свои программы и поочередно читают символы из канала и записывают их в свои выходные файлы: первый потомок – нечетные символы, а второй – четные. Синхронизация работы потомков должна осуществляться напрямую с использованием сигналов SIGUSR1 и SIGUSR2. Об окончании записи файла в канал программа оповещает потомков сигналом SIGQUIT и ожидает завершения работы потомков. Когда они заканчивают работу, программа закрывает канал.

# Примеры выполнения программы



**Рисунок 1.** Результат выполнения программы.

# Исходный код программ

## main.cpp

# #include <iostream>

# #include <fstream>

# #include <sys/signal.h>

# #include <sys/wait.h>

# #include <unistd.h>

# #include <fcntl.h>

# void read\_file\_to\_pipe(std::ifstream& file, int pipe, int first\_child);

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

# {

# signal(SIGUSR1, SIG\_IGN);

# signal(SIGUSR2, SIG\_IGN);

# signal(SIGQUIT, SIG\_IGN);

# if(argc < 2)

# {

# std::cout << "You forgot param: ./lab7 {file to read}" << std::endl;

# return 0;

# }

# char\* sFileName = argv[1];

# int pipes[2]; //0 - read, 1 - write

# int pipe\_status = pipe2(pipes);

# if(pipe\_status == -1)

# {

# perror("Cannot create pipe");

# return errno;

# }

# pid\_t child1 = vfork();

# if(child1 == -1)

# {

# perror("Failed to create child1");

# return errno;

# }

# else if(child1 == 0)

# {

# std::string pipeStr = std::to\_string(pipes[0]);

# execl("child1", "child1", pipeStr.c\_str(), NULL);

# return 0;

# }

# pid\_t child2 = vfork();

# if(child2 == -1)

# {

# perror("Failed to create child2");

# return errno;

# }

# else if(child2 == 0)

# {

# std::string pipeStr = std::to\_string(pipes[0]);

# execl("child2", "child2", pipeStr.c\_str(), NULL);

# return 0;

# }

# std::ifstream file(sFileName);

# read\_file\_to\_pipe(file, pipes[1], child1);

# kill(child1, SIGQUIT);

# kill(child2, SIGQUIT);

# waitpid(child1, NULL, 0);

# waitpid(child2, NULL, 0);

# file.close();

# close(pipes[0]);

# close(pipes[1]);

# return 0;

# }

# void read\_file\_to\_pipe(std::ifstream& file, int pipe, int first\_child)

# {

# sleep(1);

# bool first\_signal;

# if(!file.is\_open())

# {

# perror("Failed to read file");

# exit(errno);

# }

# std::string buff;

# while(getline(file, buff))

# {

# buff += "\n";

# // std::cout << "write to pipe: " << buff << std::endl;

# write(pipe, buff.c\_str(), buff.size());

# if(!first\_signal)

# {

# first\_signal = true;

# kill(first\_child, SIGUSR1);

# }

# }

# if(!first\_signal)

# kill(first\_child, SIGUSR1);

# }

## child1.cpp

#include <iostream>

#include <fstream>

#include <sys/signal.h>

#include <sys/wait.h>

#include <unistd.h>

#include <fcntl.h>

void write\_file\_child(int pipe, int child);

void signal\_handler(int signal);

bool END, READ;

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

{

if(argc < 2)

return 1;

int pipe = atoi(argv[1]);

signal(SIGUSR1, signal\_handler);

signal(SIGQUIT, signal\_handler);

write\_file\_child(pipe, 1);

return 0;

}

void signal\_handler(int iSignal)

{

if(iSignal == SIGQUIT)

END = true;

else

READ = true;

}

void write\_file\_child(int pipe, int child)

{

std::string file\_name = "child" + std::to\_string(child) + ".txt";

std::ofstream out(file\_name.c\_str());

//std::cout << child << ": ready. Pipe is " << pipe << "\n";

int my\_signal = (child == 1 ? SIGUSR1 : SIGUSR2),

other\_signal = (child == 1 ? SIGUSR2 : SIGUSR1);

ssize\_t bytes;

bool bEnd;

char buffer;

while(!bEnd)

{

//std::cout << child << ": wait for " << my\_signal << "\n";

while(!READ)

pause();

READ = false;

//std::cout << child << " got signal. Reading from pipe...\n";

bytes = read(pipe, &buffer, sizeof(buffer));

if(bytes > 0L)

{

//std::cout << child << " got (" << bytes << "): " << buffer << std::endl;

out << buffer;

}

else

{

//perror("Failed to read pipe");

bEnd = END;

}

//std::cout << child << " finished reading from pipe, raise signal...\n";

kill(0, other\_signal);

}

//std::cout << child << ": out\n";

out.close();

}

## child2.cpp

#include <iostream>

#include <fstream>

#include <sys/signal.h>

#include <sys/wait.h>

#include <unistd.h>

#include <fcntl.h>

void write\_file\_child(int pipe, int child);

void signal\_handler(int signal);

bool END, READ;

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

{

if(argc < 2)

return 1;

int pipe = atoi(argv[1]);

signal(SIGUSR1, signal\_handler);

signal(SIGQUIT, signal\_handler);

write\_file\_child(pipe, 1);

return 0;

}

void signal\_handler(int iSignal)

{

if(iSignal == SIGQUIT)

END = true;

else

READ = true;

}

void write\_file\_child(int pipe, int child)

{

std::string file\_name = "child" + std::to\_string(child) + ".txt";

std::ofstream out(file\_name.c\_str());

//std::cout << child << ": ready. Pipe is " << pipe << "\n";

int my\_signal = (child == 1 ? SIGUSR1 : SIGUSR2),

other\_signal = (child == 1 ? SIGUSR2 : SIGUSR1);

ssize\_t bytes;

bool bEnd;

char buffer;

while(!bEnd)

{

//std::cout << child << ": wait for " << my\_signal << "\n";

while(!READ)

pause();

READ = false;

//std::cout << child << " got signal. Reading from pipe...\n";

bytes = read(pipe, &buffer, sizeof(buffer));

if(bytes > 0L)

{

//std::cout << child << " got (" << bytes << "): " << buffer << std::endl;

out << buffer;

}

else

{

//perror("Failed to read pipe");

bEnd = END;

}

//std::cout << child << " finished reading from pipe, raise signal...\n";

kill(0, other\_signal);

}

//std::cout << child << ": out\n";

out.close();

}

# Вывод

В ходе работы были изучены механизмы обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен, в операционной системе Ubuntu.