МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ “РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ”

Факультет физико-математических и естественных наук

ОТЧЕТ

По лабораторной работе №11 “Программирование в командном процессоре ОС UNIX. Ветвления и циклы.”

Выполнил: Студент группы: НПИбд-01-21 Студенческий билет: №1032211403 ФИО студента: Матюхин Григорий Васильевич Дата выполнения: 02.06.2022

Москва 2022

# 1 Цель работы:

Приобретение практических навыков работы с именованными каналами.

# 2 Выполнение лабораторной работы

Изучите приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишите аналогичные программы, внеся следующие изменения: 1. Работает не 1 клиент, а несколько (например, два).

/\*  
 \* server.c - реализация сервера  
 \*  
 \* чтобы запустить пример, необходимо:  
 \* 1. запустить программу server на одной консоли;  
 \* 2. запустить программу client на другой консоли.  
 \*/  
  
#include "common.h"  
  
int main()  
{  
 int readfd; /\* дескриптор для чтения из FIFO \*/  
 int n;  
 char buff[MAX\_BUFF]; /\* буфер для чтения данных из FIFO \*/  
  
 /\* баннер \*/  
 printf("FIFO Server...\n");  
  
 /\* создаем файл FIFO с открытыми для всех  
 \* правами доступа на чтение и запись  
 \*/  
 if(mknod(FIFO\_NAME, S\_IFIFO | 0666, 0) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-1);  
 }  
  
 for (int i = 0; i < 2; i++)  
 {  
 /\* откроем FIFO на чтение \*/  
 if((readfd = open(FIFO\_NAME, O\_RDONLY)) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-2);  
 }  
   
 /\* читаем данные из FIFO и выводим на экран \*/   
 while((n = read(readfd, buff, MAX\_BUFF)) > 0)  
 {  
 if(write(1, buff, n) != n)  
 {  
 fprintf(stderr, "%s: Ошибка вывода (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-3);  
 }  
 }  
 }  
   
 close(readfd); /\* закроем FIFO \*/  
   
 /\* удалим FIFO из системы \*/  
 if(unlink(FIFO\_NAME) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-4);  
 }  
  
 exit(0);  
}

1. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.

/\*  
 \* client.c - реализация клиента  
 \*  
 \* чтобы запустить пример, необходимо:  
 \* 1. запустить программу server на одной консоли;  
 \* 2. запустить программу client на другой консоли.  
 \*/  
  
#include "common.h"  
#include <time.h>  
  
#define MESSAGE "Hello Server!!!\n"  
  
int main()  
{  
 int writefd; /\* дескриптор для записи в FIFO \*/  
 int msglen;  
  
 /\* баннер \*/  
 printf("FIFO Client...\n");  
  
 /\* получим доступ к FIFO \*/  
 if((writefd = open(FIFO\_NAME, O\_WRONLY)) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-1);  
 }  
   
 for (int i = 0; i < 3; i++)  
 {  
 time\_t mytime = time(NULL);  
 char \* time\_str = ctime(&mytime);  
 time\_str[strlen(time\_str)-1] = '\0';  
 strcat(time\_str, "\n");  
 /\* передадим сообщение серверу \*/  
 msglen = strlen(time\_str);  
 if(write(writefd, time\_str, msglen) != msglen)  
 {  
 fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-2);  
 }  
 sleep(5);  
 }  
  
 /\* закроем доступ к FIFO \*/  
 close(writefd);  
   
 exit(0);  
}

1. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера.

/\*  
 \* server.c - реализация сервера  
 \*  
 \* чтобы запустить пример, необходимо:  
 \* 1. запустить программу server на одной консоли;  
 \* 2. запустить программу client на другой консоли.  
 \*/  
  
#include "common.h"  
#include <time.h>  
  
int main()  
{  
 int readfd; /\* дескриптор для чтения из FIFO \*/  
 int n;  
 char buff[MAX\_BUFF]; /\* буфер для чтения данных из FIFO \*/  
  
 /\* баннер \*/  
 printf("FIFO Server...\n");  
  
 /\* создаем файл FIFO с открытыми для всех  
 \* правами доступа на чтение и запись  
 \*/  
 if(mknod(FIFO\_NAME, S\_IFIFO | 0666, 0) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-1);  
 }  
   
 clock\_t start\_t, end\_t;  
 double total\_t;  
 start\_t = clock();  
 while(1)  
 {  
 /\* откроем FIFO на чтение \*/  
 if((readfd = open(FIFO\_NAME, O\_RDONLY)) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-2);  
 }  
   
 /\* читаем данные из FIFO и выводим на экран \*/  
 while((n = read(readfd, buff, MAX\_BUFF)) > 0)  
 {  
 if(write(1, buff, n) != n)  
 {  
 fprintf(stderr, "%s: Ошибка вывода (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-3);  
 }  
 }  
   
 end\_t = clock();  
 total\_t = (double) (end\_t - start\_t) / CLOCKS\_PER\_SEC;  
 printf("%f\n", total\_t);  
 if (total\_t >= 0.01)  
 {  
 break;  
 }  
 }  
   
 close(readfd); /\* закроем FIFO \*/  
   
 /\* удалим FIFO из системы \*/  
 if(unlink(FIFO\_NAME) < 0)  
 {  
 fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",  
 \_\_FILE\_\_, strerror(errno));  
 exit(-4);  
 }  
  
 exit(0);  
}

# 3 Вывод

В ходе работы я приобрел практические навыки работы с именованными каналами.

# 4 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованный канал привязан к inode (и в итоге к файлу) в файловой системе, и подключение к нему осуществляется с помощью стандартного API работы с файлами. Неименованные каналы, напротив, существуют только в памяти процессов, и для создания необходимо вызвать функцию pipe из стандартной библиотеки unistd – в переданном массиве будут записаны два файловых дескриптора, которые этот процесс может использовать для коммуникации с другими – как правило, с теми, которые были созданы с помощью функции fork.

1. Возможно ли создание неименованного канала из командной строки?

Да, это происходит, когда программы соединяются с помощью оператора | – оболочка создает неименованный канал, затем запускает две программы и соединяет stdout первой с одной половиной канала, а stdin второй – с другой половиной.

1. Возможно ли создание именованного канала из командной строки?

Да, для этого есть команда mkfifo.

1. Опишите функцию языка С, создающую неименованный канал.

#include <unistd.h>  
int pipe(int fildes[2]);

Принимает указатель на массив из двух элементов типа int. Пытается создать именованный канал. Если успешно, возвращает 0. Тогда filedes[0] – дескриптор для чтения, а filedes[1] – для записи в новый канал. Если неуспешно, возвращает -1, задает errno, не создает никаких файловых дескрипторов и не изменяет переданный массив.

1. Опишите функцию языка С, создающую именованный канал.

#include <unistd.h>  
int mkfifo(const char \*pathname, mode\_t mode);

Принимает строку – путь к файлу, и настройку прав доступа. Создает файл. Если файл создан успешно, возвращает 0, иначе возвращает -1 и устанавливает errno.

1. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большего числа байтов?

Если прочитать меньше байтов, чем находится в канале, то оставшиеся байты останутся в канале и будут прочитаны при следующем чтении из этого канала.

Если попробовать прочитать больше байтов, чем находится в канале, то операция чтения будет блокировать, пока в канале не появится достаточное количество байтов. Из-за этого свойства рекомендуется читать по одному байту за раз и собирать их в строку вручную – тогда любое блокирование значит конец сообщения, и это можно обнаружить с использованием select вместо read.

1. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большего числа байтов?

Каналы имеют буфер, размер которого на моей системе равен 65536 байтов. Если из канала не читают, то все записи в канал идут в этот буфер.

Если пространство, которое остается в буфере, больше, чем количество байтов, которое процесс пытается записать, то запись осуществляется моментально и эти данные идут в буфер канала.

Если процесс пытается одномоментно записать в канал больше чем 65536 байтов, или он пытается записать больше байтов, чем осталось места в буфере, то операция записи блокируется. Вызов записи завершится, как только вся исходная строка будет записана в буфер канала.

1. Могут ли два и более процессов читать или записывать в канал?

Да, могут. Если два процесса записывают в один и тот же канал, их записи будут прочитаны в порядке, в котором они были отправлены – если один процесс запишет abcd, а затем другой процесс запишет 1234, то из канала будет прочитано abcd1234.

Одновременное чтение имеет гораздо более сложную структуру. Пусть два процесса оба хотят получить 5 байтов из канала, и первым запросил чтение процесс 1. В канал записывается сначала строка 1234, затем 5, затем 6789, затем 10, затем 1 и 2. Каждый положенный в канал байт оказывается только у одного из читающих процессов. Сначала первый процесс получит 1234, затем второй процесс получит 5, затем первый процесс получит 6789, затем второй процесс получит 10, затем первый процесс получит 10, 1 и 2. В итоге первый процесс прочитает строку 12346, и у него в буфере окажется 789, доступные для дальнейшего прочтения, а второй процесс получит строку 51012 и пустой буфер.

1. Опишите функцию write (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе server.c (строка 42)?

#include <unistd.h>  
ssize\_t write(int fildes, const void \*buf, size\_t nbyte);

Принимает номер файлового дескриптора (где 1 – переданный аргумент – указывает на stdout, а 2 – на stderr), указатель на буфер, который нужно записать в файловый дескриптор, и количество байтов, которые нужно записать. Если сколько-то байтов было успешно записано, то возвращается количество байтов, которые были записаны, и оно никогда не будет больше nbyte. Если произошла ошибка, то возвращается -1 и задается errno.

1. Опишите функцию strerror.

#include <string.h>  
char \*strerror(int errnum);

Принимает номер ошибки (соответствующий тем, которые задаются в errno), и возвращает указатель на строку, которая содержит текстовое описание ошибки. Если это поддерживается системой, то строка переведена в текущую системную локаль согласно LC\_MESSAGES. Если переданный параметр не является кодом ошибки, то возвращается NULL.