МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчёт по лабораторной работе № 31(10)**

по дисциплине: ”Системное программирование”

на тему: ***”МЕЖПРОЦЕССНОЕ***

***ВЗАИМОДЕЙСТВИЕ***

Вариант -

Выполнил**:** студент группы 10701222 Зухта К. М

Медведский Е.В.

Принял**:** пр. Давыденко Н.В.

# Лабораторная работа № 31.

## Задание 1.

Задание 1

Передайте через неименованный и именованный каналы следующую инфор-

мацию связанную с периодами выполнения задания:

1-4 неделя семестра системное время и дату.

5-7 неделя семестра получить календарь, системное время и дату.

8-10 неделя семестра получить имя ядра и название операционной системы, а

также системную дату и время.

11-16 неделя семестра получить имя операционной системы, версии и реали-

зацию ядра, системную дату и время, а также календарь.

$ Command –help

Бонус. Для автоматизации процесса напишите соответствующие скрипты.

Листинг кода

#### Скрипт для записи (writer.sh)

#!/bin/bash

# Создание именованного канала

FIFO="/tmp/my\_fifo"

if [[ ! -p $FIFO ]]; then

mkfifo $FIFO

fi

# Функция для получения системного времени

get\_system\_time() {

date +"%Y-%m-%d %H:%M:%S"

}

# Основной цикл по неделям семестра

for week in {1..16}; do

case $week in

1|2|3|4)

echo "1-4 неделя: $(get\_system\_time)" > $FIFO

;;

5|6|7)

echo "5-7 неделя: Календарь:\n$(cal)\n$(get\_system\_time)" > $FIFO

;;

8|9|10)

echo "8-10 неделя: Ядро: $(uname -s), ОС: $(uname -o), $(get\_system\_time)" > $FIFO

;;

11|12|13|14|15|16)

echo -e "11-16 неделя: ОС: $(uname -o), Версия ядра: $(uname -r), Реализация ядра: $(uname -v), $(get\_system\_time)\nКалендарь:\n$(cal)" > $FIFO

;;

esac

sleep 1 # Задержка для удобства

done

#### Скрипт для чтения (reader.sh)

#!/bin/bash

FIFO="/tmp/my\_fifo"

# Чтение сообщений из именованного канала

while true; do

if read line < $FIFO; then

echo -e "$line"

fi

done

Скрипт run.sh для запуска предыдущих скриптов одновременно

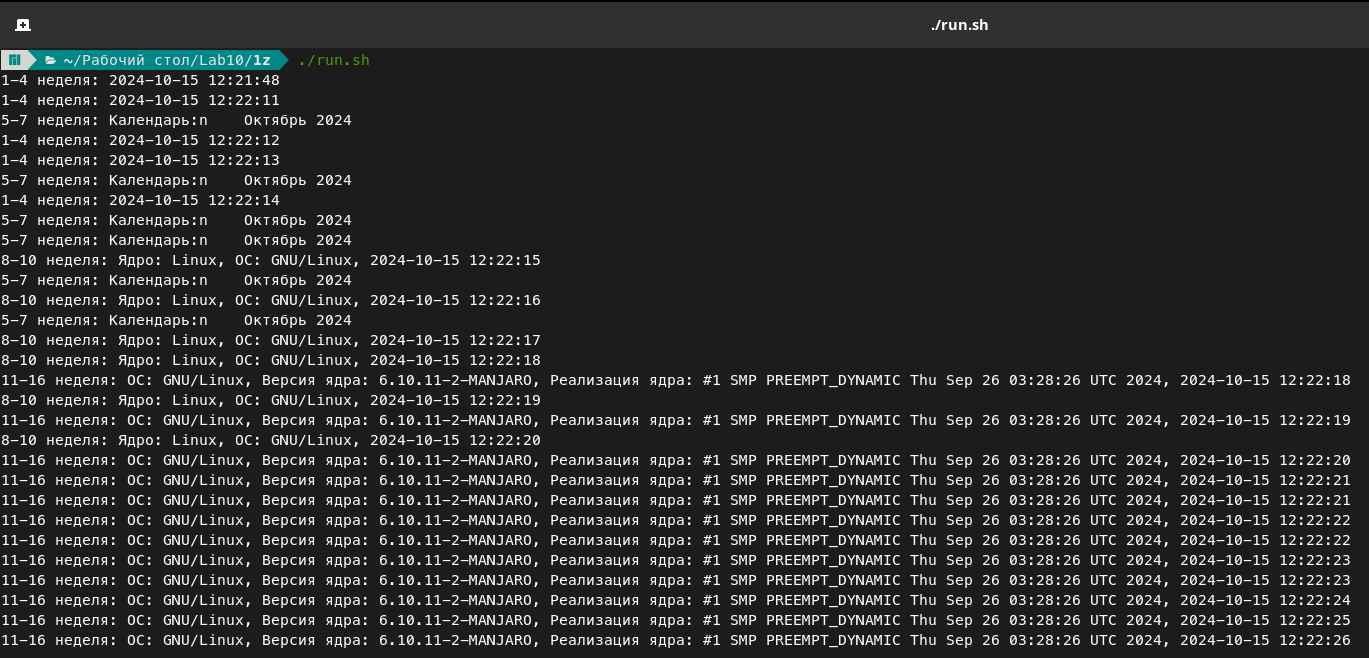
#!/bin/bash

# Запуск скриптов

bash writer.sh & # Запуск записи в фоновом режиме

bash reader.sh # Запуск чтения

Скриншоты



### Задание 2

Задание 2

Реализуйте взаимодействия двух процессов-братьев через канал, созданный в

родительском процессе, с использованием системных вызовов (см. упражне-

ние 2, пример №1).

Разработайте универсальную программу перенаправления стандартного

ввода-вывода (см. упражнение 2, пример №2).

Выполните программу, указанную в упражнении 2, пример №3 с выводом со-

держимого каталогов на выбор: proc, lib, mnt, usr, etc, home или dev.

Выполните программу, указанную в упражнении 2, пример №4 с выводом с

выводом данных, указанных в задании 1.

\* Во всех программах отобразить данные, указанные в задании 1, а также сведения,

касающиеся разработчиков, выполнивших данное задание

### Взаимодействие двух процессов через канал

#### pipe\_example.c

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#define STR\_SIZE 32

int main(void) {

int pf[2];

int pid1, pid2;

char spf[2][STR\_SIZE];

// Создание канала

if (pipe(pf) == -1) {

fprintf(stderr, "pipe() error\n");

return 1;

}

// Запись в строку

sprintf(spf[0], "%d", pf[0]);

sprintf(spf[1], "%d", pf[1]);

// Первый процесс

if ((pid1 = fork()) == 0) {

close(pf[0]); // Закрываем неиспользуемый конец

execl("./pipe1-src", "pipe1-src", spf[1], NULL);

fprintf(stderr, "exec() [src] error\n");

return 1;

}

// Второй процесс

if ((pid2 = fork()) == 0) {

close(pf[1]); // Закрываем неиспользуемый конец

execl("./pipe1-dst", "pipe1-dst", spf[0], NULL);

fprintf(stderr, "exec() [dst] error\n");

return 1;

}

// Ожидание завершения процессов

waitpid(pid1, NULL, 0);

close(pf[0]);

close(pf[1]);

waitpid(pid2, NULL, 0);

return 0;

}

#### pipe1-src.c

#include <stdio.h>

#include <unistd.h>

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

int write\_fd = atoi(argv[1]);

const char \*message = "Hello from src";

write(write\_fd, message, sizeof(message));

close(write\_fd);

return 0;

}

#### pipe1-dst.c

#include <stdio.h>

#include <unistd.h>

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

int read\_fd = atoi(argv[1]);

char buffer[32];

read(read\_fd, buffer, sizeof(buffer));

printf("Received: %s\n", buffer);

close(read\_fd);

return 0;

}

### Перенаправление ввода-вывода

#### redirect\_output.c

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

#define FILENAME "myfile"

int main(void) {

char \*message = "Hello World!\n";

int fd = open(FILENAME, O\_WRONLY | O\_CREAT | O\_TRUNC, 0640);

if (fd == -1) {

fprintf(stderr, "open() error\n");

return 1;

}

if (dup2(fd, 1) == -1) { // Перенаправление stdout

fprintf(stderr, "dup2() error\n");

return 1;

}

printf("%s", message); // Печатаем в файл

close(fd);

return 0;

}

### Выполнение команды с передачей аргументов

#### pipe\_ls\_grep.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

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

int pf[2];

int pid1, pid2;

if (argc < 3) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

if (pipe(pf) == -1) {

fprintf(stderr, "pipe() error\n");

return 1;

}

// Первый процесс: ls

if ((pid1 = fork()) == 0) {

dup2(pf[1], 1); // Перенаправление stdout

close(pf[0]);

execlp("ls", "ls", argv[1], NULL);

fprintf(stderr, "exec() [1] error\n");

return 1;

}

// Второй процесс: grep

if ((pid2 = fork()) == 0) {

dup2(pf[0], 0); // Перенаправление stdin

close(pf[1]);

execlp("grep", "grep", "-i", argv[2], NULL);

fprintf(stderr, "exec() [2] error\n");

return 1;

}

close(pf[1]);

waitpid(pid1, NULL, 0);

close(pf[0]);

waitpid(pid2, NULL, 0);

return 0;

}

### Работа с именованными каналами (FIFO)

#### fifo\_server.c

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#define FIFO\_NAME "myfifo"

#define BUF\_SIZE 512

int main(void) {

FILE \*fifo;

char \*buf;

if (mkfifo(FIFO\_NAME, 0640) == -1) {

fprintf(stderr, "Can't create fifo\n");

return 1;

}

fifo = fopen(FIFO\_NAME, "r");

if (fifo == NULL) {

fprintf(stderr, "Cannot open fifo\n");

return 1;

}

buf = (char \*)malloc(BUF\_SIZE);

if (buf == NULL) {

fprintf(stderr, "malloc() error\n");

return 1;

}

fscanf(fifo, "%s", buf);

printf("Received: %s\n", buf);

fclose(fifo);

free(buf);

unlink(FIFO\_NAME);

return 0;

}

#### fifo\_client.c

#include <stdio.h>

#include <fcntl.h>

#include <string.h>

#include <stdlib.h>

#define FIFO\_NAME "myfifo"

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

int fifo;

if (argc < 2) {

fprintf(stderr, "Too few arguments\n");

return 1;

}

fifo = open(FIFO\_NAME, O\_WRONLY);

if (fifo == -1) {

fprintf(stderr, "Cannot open fifo\n");

return 1;

}

if (write(fifo, argv[1], strlen(argv[1])) == -1) {

fprintf(stderr, "write() error\n");

return 1;

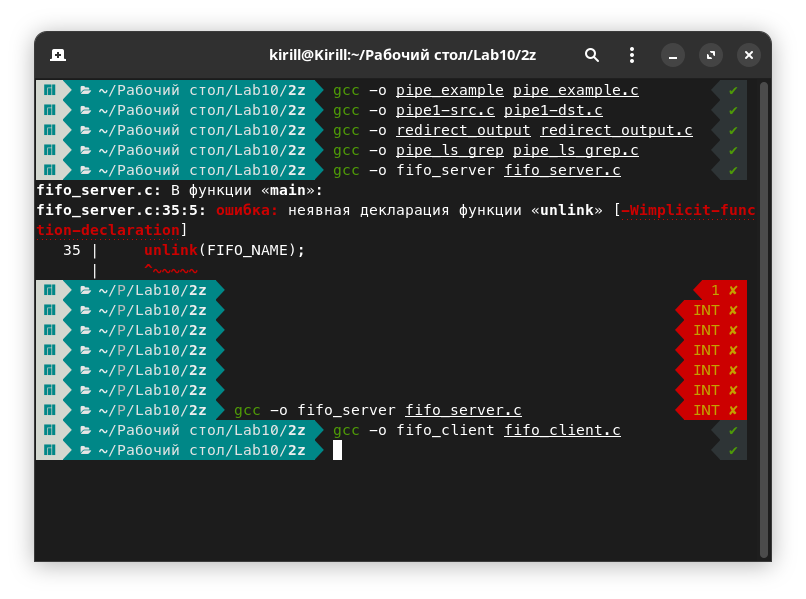
}

close(fifo);

return 0;

}

Скриншоты

Компиляция

**взаимодействия процессов через канал**:



**написанное процессом** pipe1-src**, было успешно передано через канал и получено процессом** pipe1-dst**.**

**перенаправления ввода-вывода:**

**выполнения команды с передачей аргументов**:



**работы с именованными каналами**:  
В одном терминале:



В другом терминале:



Задание 3

Разработайте универсальную программу передачи данных с помощью имено-

ванного канала с выводом данных о разработчике программы и параметров,

указанных в задании №1.

\* при работе с именованными каналами посмотрите файловый дескриптор канала в

системе (бонус).

Листинг кода

#### Программа-отправитель (named\_sender.c)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <time.h>

#define FIFO\_NAME "my\_fifo"

void print\_time() {

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

printf("Системное время: %02d:%02d:%02d, Дата: %04d-%02d-%02d\n",

tm.tm\_hour, tm.tm\_min, tm.tm\_sec, tm.tm\_year + 1900,

tm.tm\_mon + 1, tm.tm\_mday);

}

int main() {

mkfifo(FIFO\_NAME, 0666);

for (int week = 1; week <= 16; week++) {

int fd = open(FIFO\_NAME, O\_WRONLY);

char message[256];

if (week <= 4) {

snprintf(message, sizeof(message), "Неделя %d: ", week);

print\_time();

} else if (week <= 7) {

snprintf(message, sizeof(message), "Неделя %d: Получение календаря. ", week);

print\_time();

} else if (week <= 10) {

snprintf(message, sizeof(message), "Неделя %d: Имя ядра и ОС. ", week);

print\_time();

} else {

snprintf(message, sizeof(message), "Неделя %d: Информация о ОС и версии. ", week);

print\_time();

}

write(fd, message, sizeof(message));

close(fd);

sleep(1); // Задержка для синхронизации

}

return 0;

}

#### Программа-получатель (named\_receiver.c)

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#define FIFO\_NAME "my\_fifo"

int main() {

char buffer[256];

while (1) {

int fd = open(FIFO\_NAME, O\_RDONLY);

read(fd, buffer, sizeof(buffer));

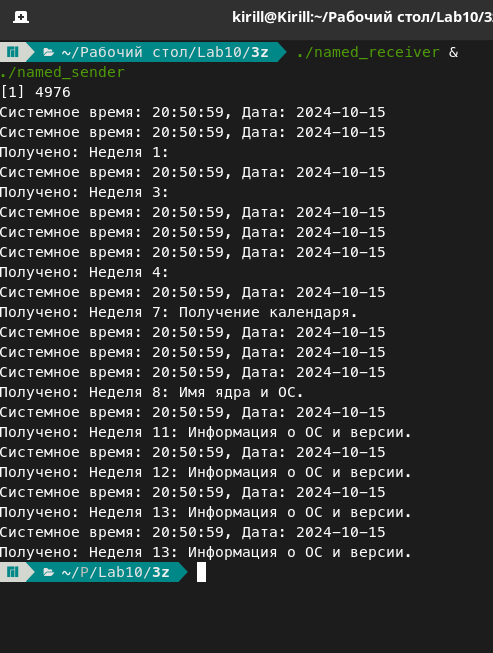
printf("Получено: %s\n", buffer);

close(fd);

}

return 0;

}



Вопросы

Зухта

Команды, системные вызовы и библиотечные функции, относящиеся к

каналам, и библиотеки их подключения.

### Именованные каналы (FIFO)

#### Системные вызовы и функции

* mkfifo(const char \*pathname, mode\_t mode)
  + Создает именованный канал (FIFO).
  + pathname — имя FIFO.
  + mode — права доступа к FIFO.
* open(const char \*pathname, int flags)
  + Открывает FIFO для чтения или записи.
* read(int fd, void \*buf, size\_t count)
  + Читает данные из FIFO.
* write(int fd, const void \*buf, size\_t count)
  + Записывает данные в FIFO.
* unlink(const char \*pathname)
  + Удаляет именованный канал.
* close(int fd)
  + Закрывает файловый дескриптор.

### Неименованные каналы (Pipe)

#### Системные вызовы и функции

* pipe(int pipefd[2])
  + Создает неименованный канал.
  + pipefd[0] — файловый дескриптор для чтения.
  + pipefd[1] — файловый дескриптор для записи.
* fork(void)
  + Создает новый процесс, который является копией родительского процесса.
* read(int fd, void \*buf, size\_t count)
  + Читает данные из канала.
  + fd — файловый дескриптор канала.
  + buf — буфер для хранения прочитанных данных.
  + count — количество байтов для чтения.
* write(int fd, const void \*buf, size\_t count)
  + Записывает данные в канал.
  + fd — файловый дескриптор канала.
  + buf — буфер с данными для записи.
  + count — количество байтов для записи.
* close(int fd)
  + Закрывает файловый дескриптор.

Медведский

Библиотечные функции popen() и pclose() в C предназначены для запуска внешних программ из вашего кода и получения от них данных.

popen()

• Запуск процесса: Функция popen() запускает программу, указанную в первом аргументе, и возвращает указатель на поток (FILE\*), который может использоваться для чтения или записи данных.

• Тип доступа: Второй аргумент popen() определяет, как будет использоваться поток:

\* "r" — для чтения данных из процесса

\* "w" — для записи данных в процесс

pclose()

• Закрытие потока: pclose() закрывает поток, полученный с помощью popen().

• Ожидание завершения: Функция pclose() ожидает, пока запущенная программа завершится, и возвращает статус выхода процесса.