МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

САВЕЛЬЕВ ДМИТРИЙ АНДРЕЕВИЧ

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ ОПЕРАЦИОННЫЕ СИСТЕМЫ

учебно-методическое пособие

САМАРА
Издательство Самарского университета
2024

УДК 004.4
ББК
Г

Рецензенты:
кандидат технических наук, доцент, заведующий кафедрой суперкомпьютеров и общей информатики Самарского университета Гошин Е.В.

Савельев, Дмитрий Андреевич
Лабораторные работы по курсу Операционные системы: учебное пособие / *Д.А. Савельев* – Самара: Изд-во Самарского университета, 2024. – 129 с.: ил.

ISBN

Учебное пособие организовано в виде набора глав, посвящённых определённым лабораторным работам курса «Операционные системы». Главы сопровождаются разобранными примерами кода, теоретическими пояснениями и контрольными вопросами, направленными на повышение качества усвоения материала.

Дисциплина рекомендована для специальностей 01.03.02 «Прикладная математика и информатика», 03.03.01 «Прикладные математика и физика», 12.03.03 «Фотоника и оптоинформатика», 09.03.01 «Информатика и вычислительная техника».

Подготовлено на кафедре технической кибернетики.

УДК 004.4
ББК
Г

ISBN

Оглавление

Введение

Существуют различные определения термина «Система». В частности, система – множество элементов, находящихся в отношениях и связях друг с другом, которое образует определённую целостность, единство [1].

Состоянием системы называется совокупность существенных свойств, которыми система обладает в каждый момент времени. Свойство системы – сторона объекта, обуславливающая его отличие от других объектов или сходство с ними и проявляющаяся при взаимодействии с другими объектами [2]. Свойствами систем, в частности, являются целостность, сложность, связность, организованность и т.д.

Перейдем к определению собственно, операционной системы. Одно из общих определений звучит следующим образом:

Операционные системы – это совокупность программ, которые предназначены для управления компьютером и вычислительными процессами, а также для организации взаимодействия пользователя с аппаратурой [3].

Дать точное определение операционной системы бывает сложно в связи с тем, что операционные системы осуществляют две значительно отличающиеся друг от друга функции: предоставляют разработчикам и прикладным программам вполне понятный абстрактный набор ресурсов взамен неупорядоченного набора аппаратного обеспечения и управляют этими ресурсами. Таким образом, можно рассматривать следующие функции операционных систем [3]:

- операционная система как расширенная машина, которая берет на себя низкоуровневые процессы;

- операционная система в качестве менеджера ресурсов, мультиплексирование (распределение) ресурсов двумя различными способами: во времени и в пространстве.

История развития операционных систем тесно связана с историей развития цифровых компьютеров. В целом, об операционных системах можно начинать говорить с появлением транзисторов и систем пакетной обработки (1955–1965). Типичными операционными системами тогда были FMS (Fortran Monitor System) и IBSYS (операционная система, созданная корпорацией IBM для компьютера IBM 7094) [3].

В следующем поколении операционных систем были введен прообраз свойства многозадачности (свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких процессов): желание сократить время ожидания ответа привело к разработке режима разделения времени – варианту многозадачности, при котором у каждого пользователя есть свой диалоговый терминал [3]

Первая универсальная система с режимом разделения времени CTSS (Compatible Time Sharing System) была разработана в Массачусетском технологическом институте (M.I.T.) на специально переделанном компьютере IBM 7094 [4]. Задача была поддерживать одновременную работу сотен пользователей в реждиме разделения времени. В качестве основы для создания такой системы была взята система распределения электроэнергии. То есть проектировалась вычислительная машина достаточно значительной мощности, вычислительными услугами которой мог пользоваться любой проживающий в окрестностях Бостона человек. Эта система была известна как MULTICS (MULTiplexed Information and Computing Service – мультиплексная информационная и вычислительная служба). Несмотря на то, что коммерчески была не очень удачна, в

рамках этой системы было реализовано много идей, легших в основу Unix [3, 5]

Вполне очевидно, что если системы проектируются в соответствие с определенными стандартами, то появляется возможность упрощения написания программ с учетом этих стандартов. Институтом инженеров по электротехнике и электронике (IEEE) был разработан стандарт системы UNIX, названный POSIX, который в настоящее время поддерживается большинством версий UNIX. Данный стандарт определяет минимальный интерфейс системных вызовов, который должны поддерживать совместимые с ним системы UNIX [3, 5].

Дальнейшее развитие и появление персональных компьютеров вызвало появление операционных систем, которые микрокомпьютеров полностью основывались на командах, вводимых пользователем с клавиатуры, в качестве примера можно привести CP/M (Control Programs for Microcomputers) и MS-DOS (MicroSoft Disk Operating System).

Графический интерфейс пользователя (GUI, Graphical User Interface) как и появление мобильных компьютеров также внесли коррективы в разработку операционных систем. На текущий момент, по статистике использования самыми популярными с долей более 90% являются следующие семейства операционных систем: Windows, macOS, Android, IOS [6].

В данном пособии предлагается ряд заданий для лабораторных работ, включающих межпроцессное взаимодействие, а также знакомство с операционной системой Ubuntu. При написании этого издания ставилась цель заинтересовать читателя задачами, возникающими при разработке программного обеспечения в различных операционных системах, и мотивировать его к дальнейшему самостоятельному решению подобного рода задач.

Каждая лабораторная работа соответствует одной главе издания. Для каждой лабораторной работой приведены некоторые сопутствующие теоретические сведения, список заданий, контрольные вопросы. Для первых пяти лабораторных работ также приводятся примеры реализации. Уровень сложности лабораторных работ возрастает с увеличением их номера от элементарных операций и до межпроцессного взаимодействия и GUI.

Проведение лабораторных работ в рамках данной дисциплины включает следующие этапы:

1) ознакомление с заданием: студент должен внимательно прочитать указания для лабораторных работ, при возникновении вопросов задать их преподавателю;

2) выполнение задания и описание его результатов: студент должен выполнить задание (на указанном языке программирования, операционной системе и т.п.) и ответить на вопросы преподавателя, затрагивающие ход работы, используемые приемы и интерпретацию полученных результатов.

Лабораторные работы выполняются индивидуально. Вариант задания назначается случайным образом.

1 Знакомство с операционной системой Linux на примере дистрибутива Ubuntu

1.1 Знакомство с Linux и Ubuntu

Среди семейства Unix-подобных операционных систем особое место занимает Linux – операционная система (ОС), включающая кроме ядра определенный набор утилит и программ проекта GNU (GNU's Not Unix, GNU - не Unix) [7].

История Linux началась в конце прошлого века, в 1991 году. Если говорить о Linux, то нельзя не упомянуть финского программиста Линуса Торвальдса. Именно тогда он стал разрабатывать ядро операционной системы и выложил свои разработки на сервере в открытый доступ. А уже в дальнейшем его вклад был поддержан другими разрабочиками, и в итоге была создана полноценная операционная система [8].

Безусловно, на Linux значительно повлияла система Unix. В самом начале проект назывался Freax (от слов «free» - бесплатный, и «freak» – странный). Далее название изменилось на гибрид имени создателя (Линус) и Unix. Первая официальная версия Linux 1.0 вышла в 1994 году.

У каждой операционной системе есть логотип, эмблема, которая отличает ее от конкурентов. Эмблемой Linux является пингвин, который был нарисован в 1996 году программистом и дизайнером Ларри Юингом. В дальнейшем по мере развития проекта данная эмблема стала символизировать не только дистрибутивы из семейства операционных систем Linux, но и в целом, свободного программного обеспечения (ПО) в целом.

Главная особенность операционных систем семейства Linux, которая проявилась с самой первой версии – распространение как свободное ПО с лицензией GPL, т.е. для любого пользователя есть возможность доработки исходного кода ОС. Но существует условие: модифицированный код должен быть так же доступен всем и

распространяться по лицензии GPL. Соответственно, это дает возможность использовать код без возникновения проблем из-за авторских прав.

Благодаря гибкости Linux используется на множестве разных устройств: мобильных устройствак, персональных компьютерах, высокопроизводительных серверах.

Залог успеха данной операционной системы – бесплатное распространение при поддержке Фонда бесплатно распространяемых программ (Free Software Foundation - FSF) [7]. Давайте чуть подробнее коснемся устройства данной операционной системы.

Особенность ОС Linux в модульной структуре: система организована в виде модулей, которые принято назвать загружаемыми модулями. Загружаемые модули имеют две важные характеристики [7]: динамическое подключение (т.е. модуль можно загрузить, когда ядро уже в памяти, а также отключить от ядра и удалить в любое время) и стековую конструкцию (иерархическая структура организации модулей, когда отдельные модули играют роль библиотек).

Загрузить отдельный модуль можно с помощью команд modprobe, insmod, а удалить – modprobe (с опцией remove), rmmod. Загружать и выгружать модули можем не только мы, но и ядро Linux, если это потребуется для работы.

На рис. 1.1 показаны типичные компоненты ядра Linux [7].

Мы видим структуру, состоящую из набора компонент, которые определенным образом взаимодействуют друг с другом. К основным компонентам ядра можно отнести [7]:

- Сигналы, которые используются для обращения к процессу. Например, для уведомления процесса об ошибках.

- Системные вызовы, с помощью которых процесс запрашивает определенную службу ядра. Их существуют сотни, и для удоб-

ства их часто группируют в категории: файловая система, процессы, межпроцессное взаимодействие и т.д.

- Процессы и планировщик, в данной компоненте ядра происходит создание, управление и планировка процессов, о которых мы подробнее поговорим на лекциях.

- Виртуальная память: модуль для выделения виртуальной памяти для процессов и управления этой памятью.

- Файловая система: в данном модуле предоставляется глобальное иерархическое пространство имен для файлов, каталогов и других объектов, связанных с файлами и функциями файловой системы.

- Сетевые протоколы, где происходит поддержка пользовательского интерфейса сокетов для набора протоколов TCP/IP.

- Драйверы символьных устройств: управление устройствами, которые требуют от ядра отправки или получения данных посимвольно, например, принтеры.

- Драйверы блочных устройств: управление устройствами, которые читают и записывают данные блоками, как, например, различные виды вторичной памяти.

- Драйверы сетевых устройств: управление картами сетевых интерфейсов и коммуникационными портами, через которые происходит подключение к сетевым устройствам, таким как роутеры.

- Ловушки и отказы: обработка генерируемых процессором прерываний, например, при сбое памяти.

- Физическая память: управление пулом страниц и выделение страницы для виртуальной памяти.

- Прерывания: обработка прерываний от периферийных устройств.

Подробнее с компонентами ядра Linux можно ознакомиться в работе [7].

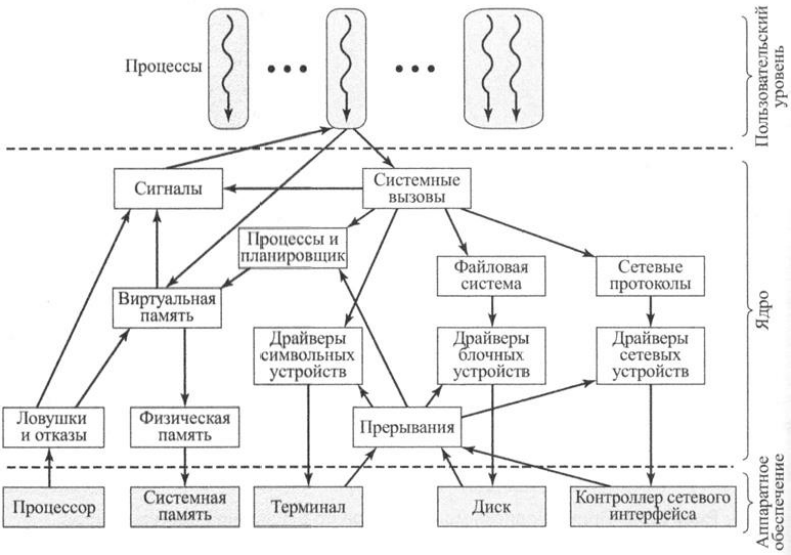

Рис. 1.1. Типичные компоненты ядра Linux [7]

Ubuntu – один из самых распространенных дистрибутивов Linux. В рамках данного курса мы коснемся некоторых аспектов работы в данном дистрибутиве и проведем сравнение с операционной системой Windows.

1.2 Виртуализация и виртуальные машины

Понятно, что на один компьютер можно установить несколько операционных систем и просто выбирать при запуске, что загрузить. Но более простым способом поставить (и при необходимости удалить) несколько операционных систем является использование виртуализации, в частности, виртуальных машин.

О виртуализации, виртуальных машинах и контейнерах мы подробнее будем говорить в рамках лекционного курса, в данном разделе произведем краткий экскурс в тему.

Можно сказать, что виртуальная машина (virtual machine) является эмуляцией компьютерной системы. Это компьютерная про-

грамма, которая представляет имитацию оборудования для операционной системы, работающей как внутрисистемный процесс. Тогда можно говорить о том, что есть программная и/или аппаратная система, которая эмулирует аппаратное обеспечение некоторой платформы (гостевая платформа) и исполняет программы для такой гостевой платформы на платформе-хозяине (host-платформе) либо виртуализирует некоторую платформу и создает на ней среды, изолирующие друг от друга программы и операционные системы [9].

Одно из решений, которое обеспечивает возможность виртуализации – монитор виртуальных машин (virtual machine monitor - VMM) [7]. Также он зазывается гипервизор.

Что же такое гипервизор? Это программное обеспечение, которое служит посредником в предоставлении ресурсов между аппаратными средствами и виртуальными машинами. Благодаря гипервизору на одном сервере могут быть несколько виртуальных машин, соответственно, можно совместно пользоваться ресурсами этого сервера.

Над аппаратной платформой находится программное обеспечение виртуализации, которое выступает посредником для гостевых операционных систем и предоставляет абстракцию всех физических ресурсов, в том числе процессора, оперативной памяти, сети и запоминающих устройств, а, следовательно, допускает наличие многих вычислительных стеков, именуемых виртуальными машинами, на одном физическом узле [7].

Принято выделять два типа гипервизоров, основное отличие которых в наличии или отсутствии операционной системы между гипервизором и платформой-хозяином (рис. 1.2).

Гипервизор первого типа загружается как программный уровень непосредственно над физическим уровнем, аналогично тому,

как загружается операционная система, и может непосредственно управлять физическими ресурсами узла.

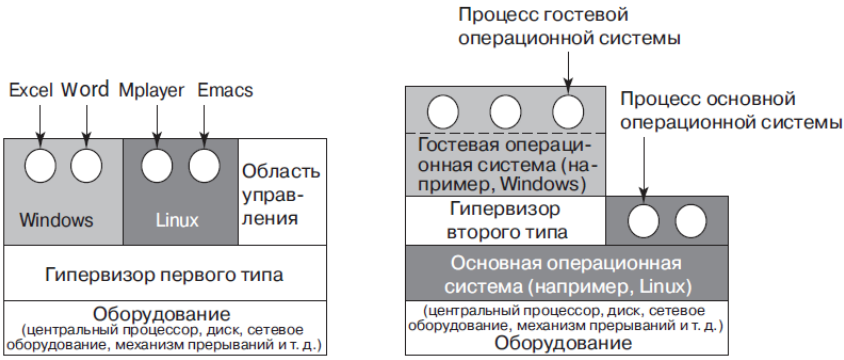

Рис. 1.2. Гипервизоры первого (слева) и второго (справа) типов [3]

В гипервизоре второго типа используются ресурсы и функции операционной системы-хозяина, а действует он как программный модуль над операционной системой. Соответственно, для организации взаимодействия с оборудованием от имени самого гипервизора, он опирается на операционную систему-хозяина [7].

1.3 Виртуальная машина VMware Workstation

В рамках данного курса базовой будет виртуальная машина VMware Workstation [10], но сдавать лабораторные работы можно и при использовании других виртуальных машин, например, Oracle Virtualbox.

VMware Workstation Player (ранее VMware Player) — бесплатный для некоммерческого использования программный продукт, на основе виртуальной машины VMware Workstation, предназначенный для запуска образов виртуальных машин, созданных в других продуктах VMware, а также в Microsoft VirtualPC и Symantec LiveState Recovery [10].

На рабочих компьютерах в аудитории данный программный продукт уже установлен, на нем также установлен образ дистрибу-

тива Ubuntu. Для запуска нужно найти на рабочем столе иконку с надписью VMware Player (рис. 1.3). Далее выбрать дистрибутив Ubuntu и запустить, нажав Play virtual machine (рис. 1.4).

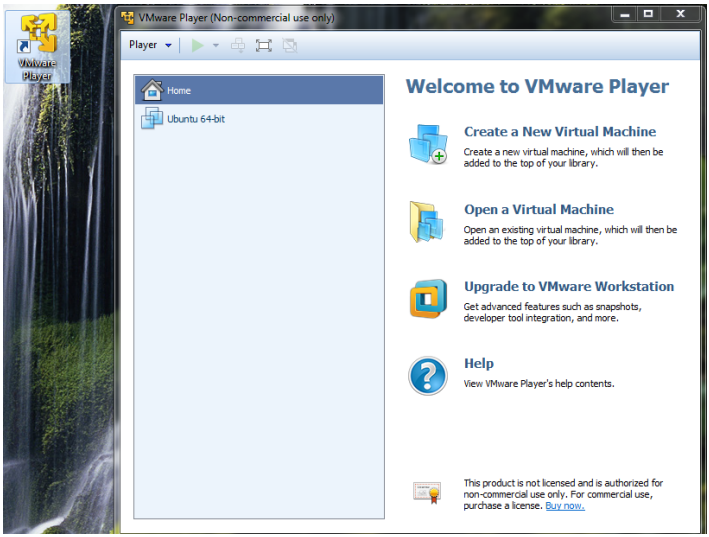

Рис. 1.3. Запуск VMware Player

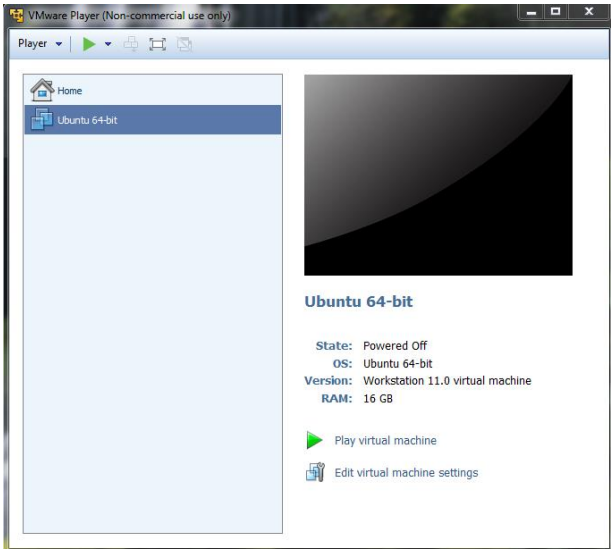

Рис. 1.4. Запуск дистрибутива Ubuntu

Далее нужно ввести пароль (рис. 1.5). После чего загрузится операционная система Ubuntu (рис. 1.6)

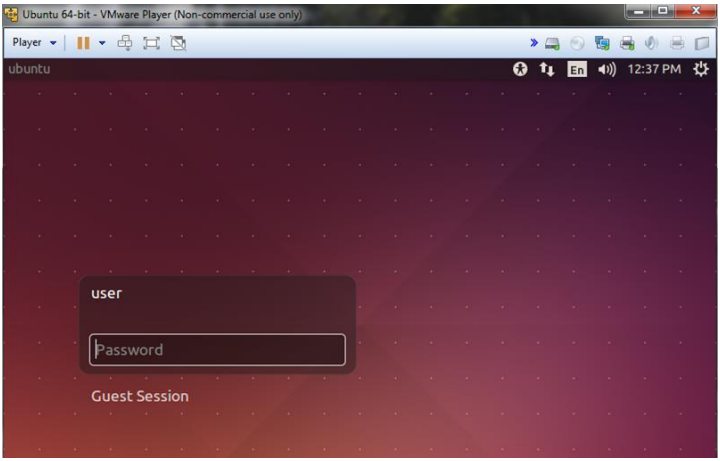

Рис. 1.5. Ввод пароля

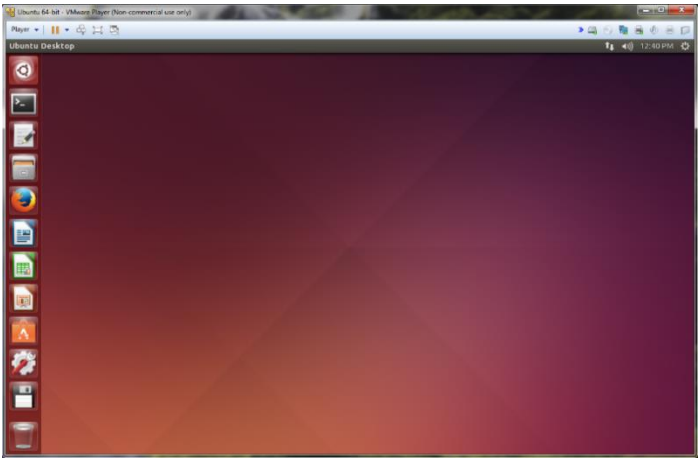

Рис. 1.6. Рабочий стол Ubuntu

1.4 Использование командной строки

В рамках знакомства с дистрибутивом Ubuntu потребуется изучить основные команды работы с командной строкой, а также скомпилировать простейшую программу.

Ознакомимся с рядом команд, которые потребуются для сдачи лабораторной работы. С остальными можно ознакомиться по следующей ссылке [11].

Интерфейс командной строки – управление программами с помощью команд. Команды могут состоять из цифр, букв, символов. Набираются построчно, выполняются после нажатия клавиши Enter. Данный интерфейс встроен в ядро системы, он будет доступен, даже если графический интерфейс не запустится [11].

Работать с командной строкой мы будем с использованием специальной графической программы, эмулирующей консоль – терминала. Запуск терминала осуществляется в зависимости от системы.

В нашем случае его можно запустить, в частности, с помощью комбинации клавиш: Ctrl+Alt+T (рис. 1.7).

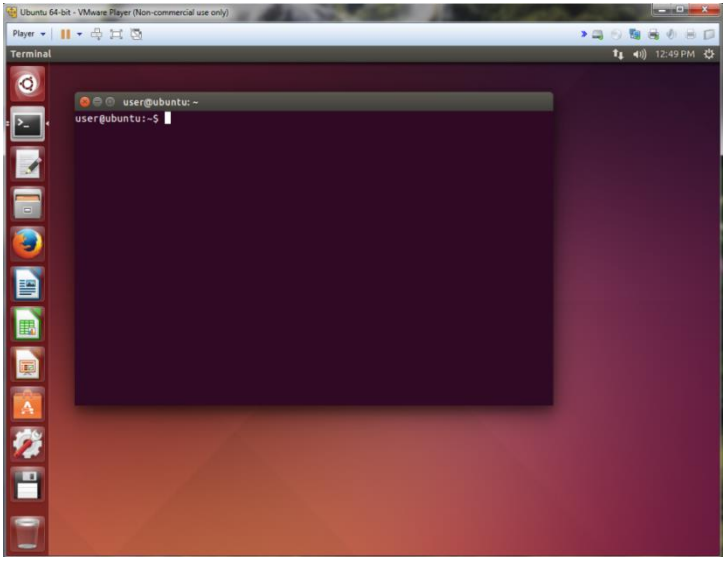

Рис. 1.7. Запуск терминала

В табл. 1.1 указаны ряд команд, которые могут пригодиться для работы [11]. Другие команды можно найти по вышеуказанной ссылке.

Таблица 1. Файловые команды в Ubuntu [11]

Команда	Описание команды
cd ../..	перейти в директорию двумя уровнями выше
cd	перейти в домашнюю директорию
cd ~user	перейти в домашнюю директорию пользователя user
pwd	показать текущую директорию
mkdir dir	создать каталог dir
mkdir dir1 dir2	создать две директории одновременно
mkdir -p /tmp/dir1/dir2	создать дерево директорий
rm file	удалить file
rm -r dir	удалить каталог dir
rmdir dir1	удалить директорию с именем 'dir1'
cp file1 file2	скопировать file1 в file2
cp -r dir1 dir2	скопировать dir1 в dir2; создаст каталог dir2, если он не существует
cp dir/	копировать все файлы директории dir в текущую директорию
cp -a /tmp/dir1	копировать директорию dir1 со всем содержимым в текущую директорию
cp -a dir1 dir2	копировать директорию dir1 в директорию dir2
mv dir1 new_dir	переименовать или переместить файл или директо-рию
mv file1 file2	переименовать или переместить file1 в file2. если file2 существующий каталог - переместить file1 в каталог file2
ln -s file1 lnk1	создать символическую ссылку на файл или директо-рию
ln file1 lnk1	создать «жёсткую» (физическую) ссылку на файл или директорию
touch file	создать file
cat > file	направить стандартный ввод в file
more file	вывести содержимое file
head file	вывести первые 10 строк file
tail file	вывести последние 10 строк file

Базовым языком программирования для выполнения лабораторных работ будет C/C++. Использование дополнительных языков программирования обговаривается отдельно с преподавателем.

Существует много компиляторов языка C, совместимые с различными стандартами. Одним из наиболее применимых компиляторов в среде GNU/Linux остаётся компилятор C, входящий в комплект GNU Compilers Collection (GCC) [12, 13].

Компилятор GCC запускается из командной оболочки командой вида:

gcc [OPTIONS] student.c

где student.c — имя входного файла. По умолчанию, выходной файл называется a.out. Если мы хотим, чтобы выходной файл назывался, например, program, это можно сделать с помощью опции компилятора -o:

gcc -o program student.c

Программу можно собрать из нескольких модулей. Тогда компилятору можно подавать на вход несколько исходных файлов или файлов объектного кода, например,

gcc -o program main.c m1.o m2.o, m3.o …

Бывает так, что не все версии Linux после установки имеют установленный компилятор, для его установки можно выполнить команды:

apt-get install gcc (компилятор C)

apt-get install g++ (компилятор C++)

Если используется дистрибутив семейства Debian с отключенной учетной записью root будет необходимо писать sudo перед административными командами. Например, следующая команда позволит установить Midnight Commander [14]:

sudo apt install mc

Midnight Commander – это файловый менеджер, состоящий из двух панелей и используемый в том числе на серверах, где отсут-

ствует графическое окружение. Это позволяет удобно управлять файлами прямо в терминале. После ввода вышеописанной команды и нажатия Enter потребуется ввести пароль. Программа будет установлена из стандартного репозитория Ubuntu (рис. 1.8).

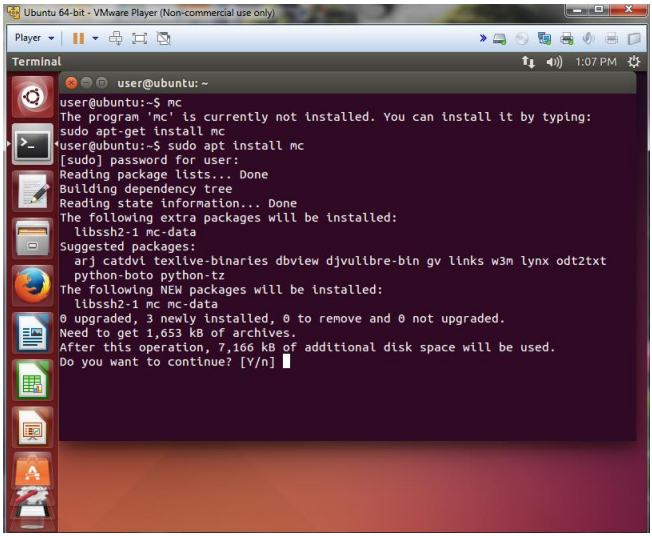

Рис. 1.8. Установка Midnight Commander

Вызвать Midnight Commander можно, набрав в терминале команду mc. Внешний вид показан на рис. 1.9.

Будем считать, что у нас стоит компилятор gcc. Пусть файл student.c содержит простейшую программу вида «Hello, world!»:

```
#include <stdio.h>
int main(int argc, char* argv[]) {
printf("Hello world!\n");
return 0;
}
```

Тогда для компиляции из директории, в которой расположен этот файл, требуется (как упоминалось ранее) дать команду вида:

```
gcc -o program student.c
```

В результате в той же директории появится исполняемый файл с именем program. Запустить его можно командой:

./program

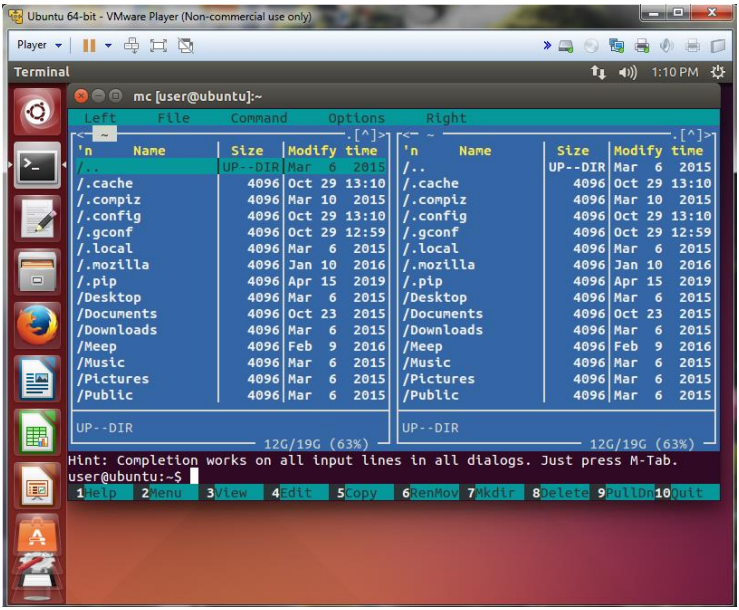

Рис. 1.9. Внешний вид Midnight Commander

В целом, этого вполне достаточно, чтобы скомпилировать простейшую программу. Для более подробного ознакомления с основами программирования в операционной среде Linux существует большое число материалов [13, 15-17].

1.5 Задание на лабораторную работу № 1

1) Требуется продемонстрировать понимание основ навигации при использовании командной строки в операционной системе Linux и умение использовать простейшие команды, такие как: создание каталога, файла, копирование данных из одного файла в другой, перемещение файла в другой каталог и т.п.

2) Выполнить один из вариантов, приведенных ниже. Вариант 0 разобран в следующем разделе в качестве примера.

Вариант 0.

Перевод из десятичной системы счисления в двоичную систему счисления.

Вариант 1.

Написать простейший калькулятор – результат элементарных операций с числами (+, -, *, /).

Вариант 2.

Вводится положительное число из нескольких цифр. Требуется найти наибольшую и наименьшую цифры в этом числе

Вариант 3.

Перевод из шестнадцатеричной системы счисления в десятичную систему счисления.

Вариант 4.

Перевод из двоичной системы счисления в восьмеричную систему счисления.

Вариант 5.

Перевод из семеричной системы счисления в десятичную систему счисления.

Вариант 6.

Выполнить проверку введенного пользователем числа: является ли данное число целой положительной степенью двойки.

Вариант 7.

Найти наибольший общий делитель двух целых чисел.

Вариант 8.

Обговаривается тип геометрической прогрессии. Пользователь вводит начальный элемент и число элементов. Требуется вычислить сумму заданного количества начальных элементов этой геометрической прогрессии

Вариант 9.

Сформировать выражение, определяющее максимальное из двух значений аргументов.

Вариант 10.

Написать функцию вычисления суммы элементов заданного одномерного массива.

Вариант 11.

Написать функцию вычисления среднего значения элементов заданного одномерного массива.

Вариант 12.

Пользователь вводит 2 числа: сумма вклада и годовой процент. Требуется рассчитать, какая сумма окажется через год с учетом капитализации процентов.

Вариант 13.

Пользователь вводит несколько элементов числового массива. Требуется удалить повторяющиеся числа.

Вариант 14.

Требуется найти число, которое встречается во введенной пользователем последовательности наибольшее количество раз.

Вариант 15.

Требуется найти слово, которое встречается во введенной пользователем строке наибольшее количество раз. В данном контексте слово – это последовательность символов латинского алфавита, цифр или иных символов. Разделителем считается пробел.

Вариант 16.

Пользователь вводит строку символов. Требуется написать функцию, которая подсчитывает частоту появления во введенной строке каждого имеющегося символа.

Вариант 17.

Пользователь вводит строку символов, включая цифры. Требуется найти все числа, встречающиеся в текстовом файле и вывести в отдельной строке. Цифры могут быть частью слова.

Вариант 18.

Написать функцию, реализующую сортировку пузырьком одномерного массива.

Вариант 19.

Переписать введенную пользователем строку из нескольких любых слов, изменив порядок следования слов на обратный (Было: «я сдал лабораторные вовремя», стало: «вовремя лабораторные сдал я»).

Вариант 20.

Написать функцию, реализующую слияние двух отсортированных массивов в один отсортированный массив.

Вариант 21.

Написать программу суммирования только положительных чисел из набора 10 чисел, введённых пользователем.

Вариант 22.

Написать программу суммирования только нецелых чисел из набора 10 чисел, введённых пользователем.

Вариант 23.

Переписать введенную пользователем строку, заменив пробелы, на запятые.

Вариант 24.

Написать функцию поиска максимального и минимального значения в массиве за один проход.

Вариант 25.

Написать функцию, которая считает, сколько раз в переданном ей массиве встречается указанное пользователем значение.

Вариант 26.

Написать функцию копирования одной строки, введенной пользователем в другую.

Вариант 27.

Написать функцию, выполняющую переворот введенной пользователем строки (последний символ первой строки становится первым символом второй строки).

Вариант 28.

Написать функцию сравнения двух строк на равенство.

1.6 Пример реализации лабораторной работы № 1

Для разработки можно использовать как любой текстовый редактор, в том числе расширенный (например, Vim или Emacs) с последующей компиляцией кода. Либо воспользоваться одной из интегрированных сред разработки (IDE), например, QtCreator, NetBeans, Eclipse, Anjuta.

Далее приведен пример реализации 0 варианта лабораторной работы (листинг 1) на языке C++.

Листинг 1. Пример реализации 0 варианта лабораторной работы № 1

```
#include<iostream>
#include<list>
#include<limits>

using namespace std;

void Program() {
            cout << "Hello!\n" << "This program converts a number from decimal to binary." << endl;
}
// функция, которая не возвращает никаких значений
void uncorrect() {
    // Возврат оператора ввода в состояние без ошибок
  cin.clear();
  // очистка буфера по максимальному значению типа int (streamsize) до того, пока не найдет enter
  cin.ignore(numeric_limits<streamsize>::max(), '\n');
  cout << "I don't understand you, sorry. Please, try again.\n";
}
char valid_continue() {
```

```
cout << "Do you want to continue? (y/n) >> ";
  char answer;
  cin >> answer;
  // С помощью библиотеки locale и функции tolower понижаем регистр
  answer = tolower(answer);
  while (answer != 'y' && answer != 'n' || cin.peek() != '\n') {
    uncorrect();
    cout << "Do you want to continue? (y/n) >> ";
    cin >> answer;
  }
  return answer;
}
int main() {
  char answer;
  Program();
  do {
    long long decNum;
    list<int> binNum;
    cout << "Please, enter a non-negative number >> ";
    cin >> decNum;
    while (cin.fail() || decNum < 0 || cin.peek() != '\n') {
      uncorrect();
      cout << "Enter a non-negative number >> ";
      cin >> decNum;
    }
    while (decNum) {
      binNum.push_front(decNum % 2);
      decNum /= 2;
    }
    if (binNum.empty())
    cout << "Your number in binary notation >> 0";
    else {
      cout << "Your number in binary notation >> ";
      for (int x : binNum)
      cout << x;
    }
    cout << endl;
```

Окончание лист. 1

```
  answer = valid_continue();
  } while (answer == 'y');
  cout << "Thanks for using this program.\n" << "Goodbye!";
  return 0;
}
```

1.7 Контрольные вопросы

1. Дайте определение термина «Операционная система».

2. Какие функции операционных систем вы можете привести?

3. Что такое мультиплексирование?

4. Какие компоненты ядра Linux вы можете назвать (не менее 5)?

5. Что такое гипервизор?

6. Какие типы гипервизоров существует и в чем между ними отличие?

7. Как запустить терминал в операционной системе Ubuntu?

8. Опишите синтаксис команды копирования файлов. Чем он отличается от копирования каталогов?

9. Как скомпилировать в командной строке программу, написанную на языке программирования C/C++?

10. Как запустить в командной строке исполняемый файл?

11. Продемонстрируйте работу в терминале 5 любых команд для управления файлами.

2 Операционная система Linux: файлы и системные вызовы

2.1 Файловая система Linux

Безусловно, файловая система в любой операционной системе является одной из самых важных ее частей. Файлы нужны для ввода и вывода данных, практически вся информация хранится в файлах той или иной структуры. Соответственно, в практически любой операционной системе существует подсисьтема управления файлами. Для файлов можно выделить следующие свойства, в рамках которых нееобходимо продумать программную реализацию так, чтобы они выполнялись [7]:

- Долгосрочное существование: как упоминалось, в файлах хранятся информция на диске или в другой вторичной памяти и было бы странно, если бы эта информация исчезала при выходе пользователя из системы.

- Совместное использование процессами: файлы могут иметь связанные с ними права доступа, которые обеспечивают управляемый совместный доступ, чтобы несколько процессов, например, могли одновременно читать инфомрацию из файла открытого на чтение.

- Структура: файл может иметь внутреннюю структуру, удобную для определенных приложений. Причем данная структура может находиться в зависимости от файловой системы конкретнной ОС.

Структурирование информации в виде файла также подразумевает и возможность выполнения операций над файлами. Типичные операции над файлами: создание, удаление, открытие, закрытие, чтение и запись информации.

О системах и архитектурах файловой системы мо поговорим в рамках соответствующей лекции, далее приведено введение в файловую систему Linux.

Как упоминалось в 1 разделе, Linux относится к типу Unix-подобных операционных систем. Файловая система UNIX различает шесть типов файлов [7]: обычные файлы, каталоги, специальные файлы (для доступа к периферийным устройствам, например, к принтеру), именованные каналы, ссылки, символические ссылки. В Linux обычно выделяют обычные файлы, символьные ссылки, блочные устройства, символьные устройства, каталоги, каналы и сокеты.

Важной особенностью ОС Linux для управления файлами является использование виртуальной файловой системы (virtual file system - VFS), которая предоставляет единый, унифицированный интерфейс файловой системы для пользовательских процессов [7]. В концепции VFS файлы являются объектами в запоминающем устройстве персонального компьютера (ПК), имеющие общие свойства независимо от целевой файловой системы или аппаратного обеспечения. В рамках конретной файловой системы нужно преобразовать характеристики реальной файловой системы в характеристики, ожидаемые виртуальной файловой системой, чем занимается специальный модуль преобразования.

Пользовательский процесс выдает системный вызов (например, чтения), используя схему VFS. VFS преобразует его во внутренний (для ядра) вызов файловой системы, который передается в функцию преобразования для конкретной файловой системы. В большинстве случаев функция преобразования представляет собой простое отображение одной схемы функциональных вызовов файловой системы на другую. В некоторых случаях функция преобразования оказывается более сложной [7].

2.2 Права доступа к файлам в Linux (Ubuntu)

Для успешного выполнения лабораторной работы нам также требуется ознакомиться с правами доступа к файлам в Linux [19].

Каждый файл имеет три параметра доступа:

- чтение;
- запись;
- выполнение.

Рассмотрим подробнее. Параметр чтение для файла позволяет получить содержимое файла, параметр чтение для каталога позволяет получить список файлов и каталогов, расположенных в нем.

Параметр запись позволяет создавать и изменять файлы и каталоги, записывать новые данные в файл.

Параметр выполнение устанавливается для всех программ и скриптов, то есть именно с помощью флага выполнение система может понять, что этот файл нужно запускать как программу

Также каждый файл имеет три категории пользователей, для которых можно устанавливать различные сочетания прав доступа: владелец, группа, остальные [17, 19].

Под владельцем понимается набор прав для владельца файла, то есть пользователя, который его создал или сейчас установлен его владельцем.

Под группой понимается любая группа пользователей, существующая в системе и привязанная к файлу. Обычно это группа владельца, хотя для файла можно назначить и другую группу.

И, наконец, под остальными подразумеваются все пользователи, кроме владельца и пользователей, входящих в группу файла.

Чтобы посмотреть права доступа к файлам в Linux с подробной информации обо всех флагах, в том числе специальных, нужно использовать следующую команду (рисунок 2.2):

ls -l

Данную команду следует выполнить из той папки, где находится файл.

Как можно заметить из рисунка 2.1, существуют ряд условных обозначений флагов прав:

- «-» – нет прав,

- «r» – разрешено только на чтение,

- «w» – разрешена запись и изменение файла,

- «x» – есть права на выполнение файла, как программы.

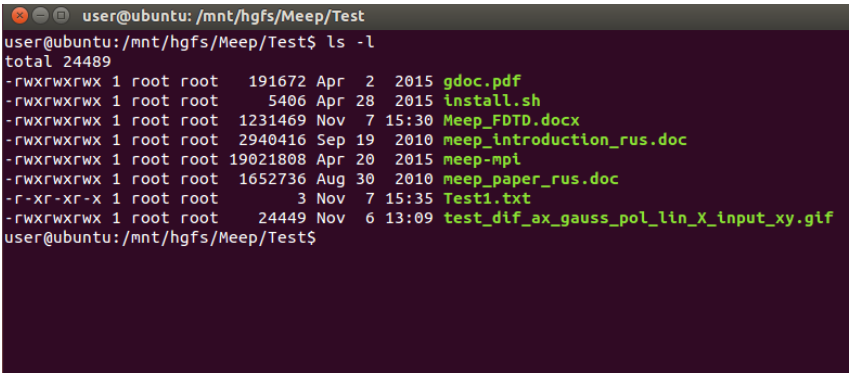

Рис. 2.1. Пример использования команды ls –l

Права сгруппированы поочередно сначала для владельца, затем для группы и для всех остальных. Всего девять значений на права и одно на тип в самом начале. Типы файлов могут быть следующими [17, 20]:

- «b» – файл блочного устройства,

- «c» – файл символьного устройства,

- «d» – каталог,

- «l» – символическая ссылка (link),

- «p» – именованный канал (pipe),

- «s» – сокет (socket),

- «-» – обычный файл.

Права данные изначально можно менять. Для изменения прав доступа к файлу существует команда chmod.

Если запустить ее с ключом –help, то можно ознакомиться с синтаксисом данной команды (рисунок 2.2).

Как можно заметить, у файла Test1.txt (рисунок 2.1) существует запрет на редактирование, причем для всех. С помощью команды chmod разрешим запись и изменение файла (рисунок 2.3):

chmod ugo+rwx Test1.txt

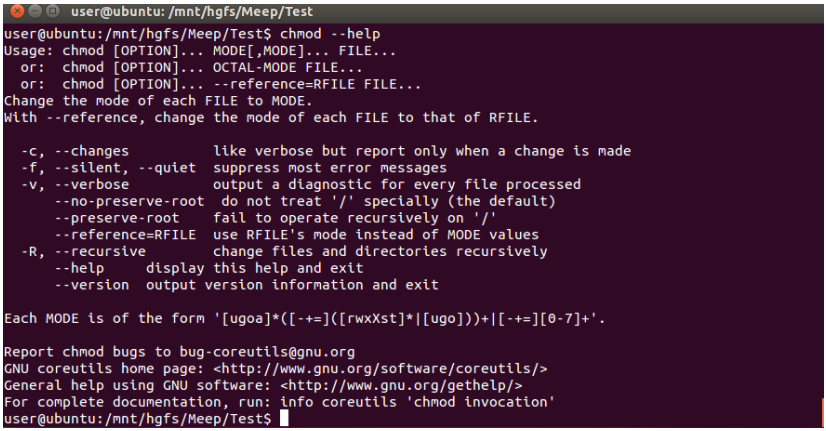

Рис. 2.2. Синтаксис команды chmod

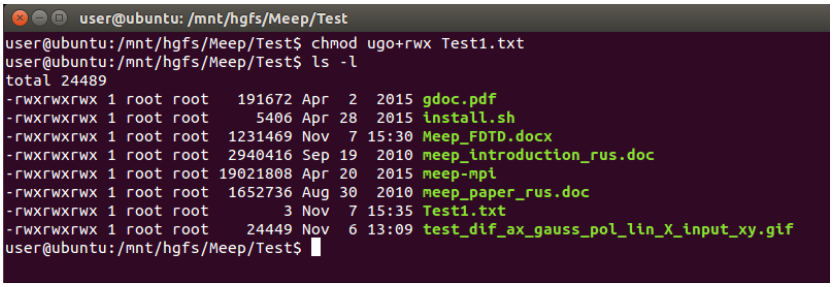

Рис. 2.3. Применение команды chmod

Параметры ugo означают выставление прав для u (владельца объекта), g (группы), o (всех остальных), + означает добавить указанные права («-» – убрать права, «=» – заменить права объекта на указанные); «rwx» – знакомые нам флаги прав.

Также существует цифровой формат записи [17, 21], значения приведены в таблице 2.

Таким образом, аналогичный вызов команды chmod будет выглядеть следующим образом:

chmod 777 Test1.txt

Таблица 2. Цифровой формат записи прав для команды chmod

Число	Двоичное представление	Символьное обозначение	Права
0	000	---	Нет прав
1	001	--x	Права на выполнение (запуск) файла
2	010	-w-	Права на запись и изменение файла
3	011	-wx	Права на запуск и изменение файла
4	100	r--	Права на чтение файла
5	101	r-x	Права на чтение и запуск файла
6	110	rw-	Права на чтение и изменение файла
7	111	rwx	Права на чтение, изменение и запуск файла

2.3 Особенности использования системных вызовов

Под системными вызовами обычно понимается обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции, то есть требует некоторой службы реализуемой ядром и вызывает специальную функцию. Системные вызовы можно сгруппировать в несколько категорий [18]:

- управление процессами,
- работа с файлами,
- управление устройствами,
- работа с информацией,
- связь, коммуникация.

Для управления процессами существуют, в том числе такие команды как [11]:

- top - показать все запущенные процессы,
- ps - вывести ваши текущие активные процессы,
- pstree - отобразить дерево процессов,
- kill -TERM 889 - корректно завершить процесс с PID 889,

- lsof -p 889 - отобразить список файлов, открытых процессом с PID 889,

- last user1 - отобразить историю регистрации пользователя user1 в системе и время его нахождения в ней,

- free -m - показать состояние оперативной памяти в мегабайтах.

Некоторые команды для работы с файлами нами разбирались в разделе 1.4. С другими командами управления устройствами, работы с информацией, связью можно ознакомиться по ссылке [11].

Вспомним, что команда ls показывает все файлы в текущей директории, команда cp копирует файл, команда mv перемещает файл в другое место (или переименовывает файл в зависимости от вызова), команда rm удаляет файл. Узнать в какой директории вы находитесь в данный момент можно с помощью команды pwd (Print Working Directory).

2.4 Задание на лабораторную работу № 2

Необходимо написать программу для работы с файлами, получающую информацию из командной строки или из консоли ввода.

Программа должна корректно обрабатывать ключи и аргументы в случае ввода из командной строки, либо программа должна показывать список действий в случае ввода из консоли ввода.

Программа должна иметь возможность осуществлять:

- копирование файлов,

- перемещение файлов,

- получение информации о файле (права, размер, время изменения),

- изменение прав на выбранный файл.

Просто вызвать функцию для копирования файла нельзя. Также программа должна иметь help для работы с ней, он должен вызываться при запуске программы с ключом --help.

Копирование файла должно производиться при помощи команд блочного чтения и записи в файл. Размер буфера для чтения и записи должен быть больше единицы. Не допускается так же производить копирование файла при помощи однократной команды чтения и записи, так как при таком подходе предполагается, что оперативной памяти достаточно, чтобы прочитать весь файл одной командой. Это неверно, так как в общем случае размер файла может существенно превышать оперативную память и файл подкачки.

Основные модули для работы с файлами:

- #include <fcntl.h>
- #include <sys/stat.h>
- #include <unistd.h>

Основные процедуры для работы с файлами (помощь по ним вызывается из командной строки командой $ man <имя_команды>) :

- Изменение текущей маски прав для программы: int umask(int newmask).

- Открытие файла: int open(char* pathname, int flags, mode_t mode).

- Закрытие файла: int close(int fd).

- Чтение, запись: size_t read(int fd, void * buf, size_t length), size_t read(int fd, const void * buf, size_t length), size_t write(int fd, void * buf, size_t length), size_t write(int fd, const void * buf, size_t length).

- Установка указателя чтения / записи: int lseek(int fd, off_t offset, int whence); где whence: SEEKSET (начало файла), SEEK_CUR (текущая позиция в файле), SEEKEND (конец файла).

- Сокращение файла: int truncate(const char *pathname, size_t length), int ftruncate(int fd, size_t length).

- Синхронизация файлов: int fsync(int fd), int fdatasync(int fd).

- Запрос информации о файле:

#include <sys/stat.h> ;

int stat(const char *pathname, struct stat *statbuf);

int lstat (const char *pathname, struct stat *statbuf);

int fstat(int fd, struct stat *statbuf).

- Изменение прав доступа:

int chmod(const char *pathname, mode_t mode);

int fchmod(int fd, mode_t mode).

- Создание жестких ссылок:

int link(const char *origpath, const char *newpath).

- Создание символических ссылок:

int symlink(const char *origpath, const char *newpath).

- Удаление файла:

int unlink(char *pathname).

- Переименование файлов:

int rename(const char *oldpath, const char *newpath).

2.5 Пример реализации функции копирования файлов для лабораторной работы № 2

В данном разделе приведен пример реализации функции копирования (листинг 2). Считается, что подключены все нужные библиотеки. Функция с помощью блочного чтения с определенным размером буфера копирует содержимое файла, название которого вводит пользователь в другой файл, имя которого также вводит пользователь.

Листинг 2. Пример реализации функции копирования файлов для лабораторной работы №2

```
#include<iostream>
#include <string>
```

```
#include <fstream>
…
using namespace std;
…
/*
Функция копирования файлов
- название копируемого файла
- название копии файла
- копирует содержимое
*/
void cp()
{
   cout << "Введите название копируемого файла: \n > ";
   string filename;
   cin >> filename;
   cout << "Введите название копии файла: \n > ";
   string newFilename;
   cin >> newFilename;
   //ошибка
   if(filename == newFilename){
      defects();
      return;
   }
   ifstream fin;
   size_t bufsize = 4;
   char* buf = new char[bufsize];
   fin.open(filename, ios::binary);
   if (fin.is_open()) {
      ofstream fout;
      fout.open(newFilename, ios::binary);
      while(!fin.eof()) {
         fin.read(buf, bufsize);
       if (fin.gcount()) fout.write(buf, fin.gcount());
      }
      cout <<"  > Копирование прошло успешно!" << endl;
      delete[] buf;
      fin.close();
```

```
        fout.close();
    }
    else {
        defects();
        return;
    }
}
```

2.6 Контрольные вопросы

1. Назовите свойства файлов, которые нужно учитывать при разработке файловой системы.

2. Какие операции над файлами вам известны?

3. Какие типы файлов обычно выделяют в Unix-подобных операционных системах?

4. Зачем нужна виртуальная файловая система в операционной системе Linux?

5. По каким категориям можно сгруппировать системные вызовы?

6. Назовите 5 любых команд для управления процессами.

7. Какие параметры доступа к файлам вам известны?

8. Перечислите категории пользователей, для которых можно устанавливать различные сочетания прав доступа.

9. Какой командой можно посмотреть права доступа к файлам в Linux?

10. Как обозначаются разные типы файлов в Linux?

11. Какой командой можно воспользоваться для изменения прав доступа к файлу?

12. Расскажите о цифровом формате записи прав для команды chmod.

3 Межпроцессное взаимодействие в операционной системе Ubuntu

3.1 Процессы и потоки

Операционную систему можно рассматривать как программное обеспечение, предоставляющее единое абстрактное представление ресурсов, которые могут быть запрошены приложениями и к которым они могут получить доступ [7]. В качестве таких ресурсов могут выступать основная память, файловые системы, сетевые интерфейсы, и т.д. Соответственно, если у нас есть такие абстракции ресурсов, которыми могут пользоваться приложения, то нужна также система управления использованием таких ресурсов.

Для эффективного распределения имеющихся ресурсов в современных операционных системах принята модель, в которой выполнение приложения соответствует существованию одного или нескольких процессов.

Что такое процесс? Общее определение, процесс - совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие. Тогда общее определение программы – совокупность инструкций, процесс – непосредственное выполнение этих инструкций [3].

Существуют другие определения, в частности, можно говорить, что процесс – это [7]:

- выполняемая программа,
- экземпляр программы, выполняющейся на компьютере,
- сущность, которая может быть назначена процессору и выполнена на нем,
- единица активности, характеризуемая выполнением последовательности команд, текущим состоянием и связанным с ней множеством системных ресурсов.

Важными элементами процесса являются программный код (который может совместно использоваться другими процессами,

которые выполняют ту же самую программу) и набор данных, связанный с этим кодом [7]. Также следует отметить, что любой процесс имеет ряд связанных с ним элементов: идентификатор, приоритет по сравнению с другими процессами, состояние (например, выполнение или блокировка), указатели памяти, информация о состоянии ввода-вывода и другие.

С каждым процессом связывается его адресное пространство, из которого он может читать и в которое он может писать данные.

Адресное пространство или образ памяти содержит:

- саму программу;
- данные к программе;
- стек программы.

Таким образом, процесс – это контейнер, в котором содержится вся информация, необходимая для работы программы.

Процессы можно условно разбить на три категории:

- системные;
- фоновые (демоны);
- прикладные (пользовательские).

Фоновые процессы, предназначенные для обработки какой-либо активной деятельности, связанной, например, с электронной почтой, веб-страницами, и т. д., называются демонами.

Часто выделяют пять основных состояний процесса: выполнения (действия), готовности к выполнению (готовности), заблокированный (также иногда используется термин ожидающий), новый, завершающийся. С сотояниями новый и завершающийся, в общем-то, все понятно. А переходы между остальными тремя схематично показаны на рисунке 3.1, где цифрами показаны [3].

1. Процесс заблокирован в ожидании ввода.
2. Диспетчер выбрал другой процесс.
3. Диспетчер выбрал данный процесс.
4. Входные данные стали доступны.

Рис. 3.1. Модель переходов между состояниями процесса [3]

В целом, концепцию процесса можно охарактеризовать свойством владения ресурсами, а также свойством планирования/выполнения [7]. Соответственно, по отношению к процессам операционная система выполняет планирование и диспетчеризацию. По сути, данные свойства можно рассматривать независимо. Тогда единицу диспетчеризации можно назвать потоком (thread) или облегченным процессом (lightweight process), а единицу владения ресурсами - процессом (process) или заданием (task). Хотя данная терминология несколько условна, потому что для ряда операционных систем она не выдерживается. Тем не менее, для общего понимания мы будем ей пользоваться с такой оговоркой.

Тогда многопоточностью (multithreading) будем называть способность операционной системы поддерживать в рамках одного процесса несколько параллельных путей выполнения [7].

Если рассматривать конкретную операционную систему, такую как Linux, то там процесс (задание), представляется структурой данных task_struct. В этой структуре данных информация разбивается на следующие категории [7]: состояние, информация по планированию, идентификаторы, обмен информации между процессами, связи, файловая система, адресное пространство и другие.

Подробнее о процессах и потоках мы поговорим в рамках лекций по курсу, а также можно посмотреть в данном учебнике [7].

3.2 Особенности межпроцессного взаимодействия

В Unix-подобных системах используется системный вызов fork(), который создает точную копию вызывающего процесса. После выполнения системного вызова fork() два процесса, родительский и дочерний, имеют единый образ памяти, единые строки описания конфигурации и одни и те же открытые файлы [3]. После этого дочерний процесс обычно изменяет образ памяти и запускает новую программу, выполняя системный вызов execve() или ему подобный [3].

Для всех процессов существует ряд ситуаций, когда им приходится взаимодействовать. В частности, это может быть:

- Передача информации от одного процесса другому.

- Контроль над деятельностью процессов (в качестве примера можно привести ситуацию борьбы процессов за один ресурс).

- Согласование действий процессов.

Классический пример последней ситуации – согласование действий процессов во время печати документа, когда один процесс поставляет данные, а другой их выводит на печать. Если будет наблюдаться рассогласованность, томожет возникнуть ситация, когда печать может начаться раньше, чем поступят актуальные данные.

Одним из средств передачи информации между процессами является использование каналов. Канал – это разновидность «псевдофайла», используемый для соединения двух процессов [3]. С одного конца записывается информация, а с другого конца – считывается.

Существует два типа каналов: именованные и неименованные. Совместно использовать неименованные каналы могут только связанные друг с другом процессы; не связанные друг с другом процессы могут совместно использовать только именованные каналы.

В данной лабораторной работе мы будем работать с неименованными каналами. Программа должна разделяться на две части при помощи системного вызова fork(). Создание неименованных каналов командой int pipe(int fds[2]) должно происходить до вызова fork().

Чтение и запись данных из канала осуществляются командами read() и write(). Для этого достаточно указать командам чтения и записи из канала адрес переменной, которую необходимо передать. Например если есть переменная:

double X;

то для ее записи в канал достаточно выполнить:

write(channel_variable[1], &X, sizeof(double)),

где channel_variable – канал для записи на сервер, обозначенный ранее массивом:

int chanel_variable[2].

Чтение производится аналогично. Если необходимо передавать несколько переменных, то их можно сгруппировать в один блок данных с помощью структуры и передавать ее целиком.

Если Вы выбрали переопределение стандартных потоков, то каналами следует пользоваться как стандартным вводом и выводом (printf, cout, и т.д.).

Также для выполнения лабораторной работы нужно будет вспомнить особенности работы с массивами в C++ [23, 24]. А еще потребуется вспомнить, ознакомиться или изучить некоторые математические методы для работы с матрицами, вычислением полиномов и многочленов, решения систем линейных алгебраических уравнений и т.д. [25, 26].

3.3 Задание на лабораторную работу № 3

Необходимо написать программу, которая разделяется на две части. Первая часть – клиентская – взаимодействует с пользовате-

лем, содержит интерфейс ввода вывода чисел из консольного ввода. Желательно обойтись без передачи параметров расчета через параметры программы (argv). Справка так же выводится по ключу –help, как и ранее. Вторая часть (серверная) ничего не выводит на экран, только принимает информацию от клиентской части, производит вычисления в зависимости от варианта ниже и отправляет клиентской части результат.

Программа должна разделяться на две части при помощи системного вызова fork(). Создание неименованных каналов должно происходить до вызова fork().

Исходный код и клиентской, и серверной частей программы можно разместить в одном файле внутри одной функции main. При отладке программы в отладчике gdb необходимо помнить о том, что клиентская и серверная части работают одновременно и представляют собой разные процессы.

Другой вариант написания лабораторной работы – подмена стандартных ввода и вывода (функция int dup2(int oldfd, int newfd)), и запуск расчетной и клиентской частей через функцию семейства exec. При этом oldfd будет указывать на ваш созданный канал, newfd – на файл, соответствующий стандартному потоку ввода вывода – 0 или 1. В этом случае лабораторная работа будет состоять из трех .cpp файлов, и трех запускных, соответственно.

Для корректной работы необходимо создать два канала, каждый из которых описывается массивом двух чисел. Один канал будет использоваться для передачи данных и команд серверу от клиента, второй – для передачи результата от клиента к серверу.

Таким образом, стандартный набор действий следующий:

1. Проверка командной строки на ключ --help. При наличии такового должен быть выведен help и программа должна завершить свою работу.

2. Создание каналов.

3. Разделение на клиентскую и серверную части.

4. Работа

5. После обмена (достаточно единственного) и сервер, и клиент могут завершить свою работу, команд для завершения работы сервера от клиента предусматривать не требуется.

Не забывайте закрывать каналы после использования.

Работа клиентской части:

1. Предложение ввода параметров, ожидание ввода с клавиатуры.

2. Передача данных в канал на сервер

3. Чтение результатов из канала от сервера

4. Вывод результатов расчета на экран

5. Выход.

Работа серверной части:

1. Чтение данных из канала от клиента.

2. Расчет.

3. Запись данных в канал клиенту.

4. Выход.

К сдаче лабораторной работы требуется подготовить файл с описанием рассматриваемого метода. Для дополнительного бонуса допускается также реализация работы с файлами, когда входные значения должны загружаться из файла, а результат записываться в файл. В следующем разделе разбирается реализация лабораторной работы для варианта 0.

Вариант 0.

Реализация метода прямоугольников (средних) для функции одной переменной

Вариант 1.

Нахождение корней некоторого полинома.

Вариант 2.

Сложение матриц.

Вариант 3.

Транспонирование матрицы.

Вариант 4.

Умножение двух матриц.

Вариант 5.

Перевод из произвольной (выбирает пользователь) системы счисления в 10-чную и обратно.

Вариант 6.

Вычислить определитель заданной пользователем матрицы.

Вариант 7.

Нахождение обратной матрицы.

Вариант 8.

Нахождение ранга матрицы.

Вариант 9.

Решение дифференциального уравнения методом Эйлера.

Вариант 10.

Вычисление полиномов Эрмита.

Вариант 11.

Вычисление многочленов Лагерра (первые 6).

Вариант 12.

Метод Гаусса для решения систем уравнений.

Вариант 13.

Матричный метод решения систем линейных алгебраических уравнений.

Вариант 14.

Метод Крамера решения систем линейных алгебраических уравнений.

Вариант 15.

Реализация метода левых прямоугольников для функции одной переменной.

Вариант 16.

Реализация метода трапеций для функции одной переменной.

Вариант 17.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{N}\frac{\sin y+x^n}{(xy+1)!}.$$

Вариант 18.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{100}\frac{x^n+tgy+2xy}{(n+1)!}.$$

Вариант 19.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{150}\frac{2x}{x!+y!}+\sin x.$$

Вариант 20.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{N}\frac{x^n+2xy+y^{n+1}}{n!}.$$

Вариант 21.

Написать функцию вычисления приближенного значения:

$$\exp(x)=\sum_{n=0}^{\infty}\frac{x^n}{n!}.$$

Вариант 22.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{N}\frac{y^n-\sin x}{xy-n!}.$$

Вариант 23.

Написать функцию вычисления приближенного значения:

$$f(x)=\sum_{n=0}^{N}\frac{\sin x^n}{(n+1)!}.$$

Вариант 24.

Написать функцию вычисления приближенного значения:

$$f(x)=\sum_{n=0}^{100}\frac{\sin x+\cos n}{x+n!}.$$

Вариант 25.

Написать функцию вычисления приближенного значения:

$$f(x)=\sum_{n=0}^{\infty}\frac{2x^n-x^2}{x!}.$$

Вариант 26.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{500}\frac{\sin x^n+tgy}{(n+x)!}.$$

Вариант 27.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{500}\frac{\cos y}{(\sin x+n)y}.$$

Вариант 28.

Написать функцию вычисления приближенного значения:

$$f(x,y)=\sum_{n=0}^{N}\frac{x^n+y^n}{n+tgxy}.$$

3.4 Пример реализации лабораторной работы № 3

В данном разделе приведен пример реализации 0 варианта лабораторной работы №3. То есть требуется реализовать метод прямоугольников (средних) для функции одной переменной с учетом заявленных в задании требований.

Один из вариантов реализации приведен в листингах 3-5.

На листинге 3 показано начало программы, а также общие функции и структуры: структура интеграла, которая содержит левую границу, правую границу и количество прямоугольников,

объявляются неименованные каналы связи. Также на листинге 3 приводится реализация вспомогательной функции help, функций чтения чисел, в том числе с плавающей точкой (в которых также проводятся проверки ввода), функция вычисления интеграла методом прямоугольников.

На листинге 4 приведена условно «клиентская» часть, где принимаются данные (левую и правые границы, число прямоугольников), производится запись и чтение из неименованных каналов, происходит вывод результата вычислений с серверной части. Также на данном листинге показана условно «серверная» часть, где производятся вычисления по данным, переданным клиентской частью по неименованному каналу и записывается результат вычисления в неименованный канал.

На листинге 5 приводится функция main.

Листинг 3. Пример реализации 0 варианта лабораторной работы №3, функции help, read_double, read_int, function, calculation

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <unistd.h>
#include <cstring>
#include <cmath>
using namespace std;
/*
//Структура интеграла
//Содержит левую границу, правую границу и количество прямоугольников
*/
struct IntegralData{
   double a;
   double b;
   int n;
};
//Неименованные каналы связи in и out
int pipe_in[2];
int pipe_out[2];
```

```
pid_t pid;
void help(){
   cout << "Справка:\n"
   <<"Чтобы посчитать интеграл методом прямоугольников, запустите программу
без ключей\n"
   <<"Для смены функции вычисления измените function() в исходном коде про-
граммы\n>_<\n";
}
/*
//Функция чтения чисел с плавающей точкой
//Производит проверку ввода, с случае ошибки ввод повторяется
//Возвращает правильное число, введенное пользователем
*/
double read_double(const string& msg)
{
   double result;
   bool flag = true;
   while (flag)
   {
      cout << msg;
      cin >> result;
      if((cin.fail() || (cin.peek() != '\n'))){
         cin.clear();
         cin.ignore(1000, '\n');
         cout << " > Ошибка ввода, попробуйте еще раз\n";
      }else{
         flag = false;
      }
   }
   return result;
}
/*
//Функция чтения чисел
//Производит проверку ввода, с случае ошибки ввод повторяется
//Возвращает правильное число, введенное пользователем
*/
int read_int(const string& msg)
```

```
{
    int result;
    string input;
    bool flag = true;

    while (flag)
    {
        cout << msg;
        cin >> result;
        if((result <=0) || (cin.fail() || (cin.peek() != '\n'))){
            cin.clear();
            cin.ignore(1000, '\n');
            cout << " > Ошибка ввода, попробуйте еще раз\n";
        }else{
            flag = false;
        }
    }
    return result;
}
/*
//Функция для вычисления заданной функции зависящей от параметра
//Принимает значение x
//Возвращает результат вычисления
*/
double function(double x){
    return 5 * x * x * x + 2 * x * x  + 10 * x;
}
/*
//Функция вычисления интеграла от функции одной переменной методом прямо-
угольников
//Принимает левую границу, правую границу и число разбиений
//Возвращает результат вычисления
*/
double calculation(double a, double b, int n)
{
    double h = (b - a) / n;
    double result = 0;
```

```
    for(int i = 0; i < n; i++)
    {
        result += function(a + (h / 2) + (i * h) );
    }
    result *= h;
    return result;
}
```

Листинг 4. Пример реализации 0 варианта лабораторной работы №3, «клиентская» и «серверная» части программы

```
/*
//Клиентская часть программы
//Принимает данные: левую границу, правую границу и количество прямоугольников
//Записывает и читает в/из неименованных каналов
//Выводит результат вычислений
*/
void frontend()
{
    IntegralData data;
    data.a = read_double("Введите начало сегмента интегрирования: ");
    data.b = read_double("Введите конец сегмента интегрирования: ");
    data.n = read_int("Введите количество разбиений: ");
    write(pipe_in[1], &data, sizeof(IntegralData));
    double result;
    read(pipe_out[0], &result, sizeof(double));
    cout << "Результат: " << result << endl << ">_<\n";
    exit(0);
}
/*
//Серверная часть программы
//Производит вычисления по данным, переданным клиентской частью по неименованному каналу
//Записывает результат вычисления в неименованный канал
*/
void backend()
{
    IntegralData data;
```

```
    read(pipe_in[0], &data, sizeof(IntegralData));
    double result = calculation(data.a, data.b, data.n);
    write(pipe_out[1], &result, sizeof(double));
}
```

Листинг 5. Пример реализации 0 варианта лабораторной работы №3, функция main

```
int main(int argc, char const *argv[]) {
    setlocale(LC_ALL, 0);
    if (argc == 2 && !strcmp(argv[1], "--help"))
    {
        help();
        exit(0);
    }
    else if (argc != 1)
    {
        cout << "Запустите программу с ключом --help для получения справки" << endl;
        exit(1);
    }
    else
    {
        cout << "Программа для нахождения интеграла функции одной переменной методом прямоугольников\n";
        cout << "Для функции y(x) = 5x^3+2x^2+10x\n";

        pipe(pipe_in);
        pipe(pipe_out);
        pid = fork();
        if (pid < 0)
        {
            cerr << "Критическая ошибка! Форк не создан" << endl;
            exit(1);
        }
        else if (pid > 0)
        {
            frontend();
        }
```

```
        else
        {
            backend();
        }
        for (int i = 0; i < 2; ++i)
        {
            close(pipe_in[i]);
            close(pipe_out[i]);
        }
    }
    return 0;
}
```

3.5 Контрольные вопросы

1. Дайте определение процесса.
2. Какие элементы присущи каждому процессу?
3. Что содержит адресное пространство?
4. Какие категории процессов вам известны?
5. Какие основные состояния процесса вам известны?
6. Что такое поток?
7. Что такое многопоточность?
8. Какая функция используется для создания копии вызывающего процесса?
9. В каких случаях процессам требуется взаимодействовать друг с другом?
10. Дайте определение канала.
11. Какие типы каналов вам известны?

4 Межпроцессное взаимодействие на основе каналов и сокетов в операционной системе Windows

4.1 Сравнение семейств операционных систем Windows и Linux

Откровенно говоря, данные семейства операционных систем достаточно сложно сравнивать. Собственно говоря, главная причина – из-за открытого кода и GNU существуют сотни дистрибутивов Linux, и в каждый из этих дистрибутивов могут быть десятки разновидностей. Соответственно, существует много конфигураций, широта поддержки может различаться у разных поставщиков. Также стоит отметить, что изначально под Linux подразумевалось ядро операционной системы. Операционные системы на основе ядра Linux, утилит проекта GNU корректнее называть GNU/Linux, но часто упрощают до Linux.

Тем не менее, сравнение можно проводить по стоимости владения, популярности на стационарных компьютерах, безопасности, объему операционной системы и т.д. В частности, следует отметить, что в общем случае последние версии Windows более требовательны к ресурсам, чем последние версии семейства Linux. Если говорить о стоимости, то Linux полностью бесплатная система, в отличие от операционных систем семейства Windows.

Важной составляющей сравнения также может быть поддерживаемое программное обеспечение, то есть число сторонних программ и утилит, которые способны запускаться и функционировать в среде рассматриваемых семейств операционных систем. В этом контексте, учитывая, что Microsoft Windows – самая распространенная система для домашних и офисных компьютеров, то производители программного обеспечения чаще разрабатывают свои программы для Windows, чем для Linux. Тем не менее, самые необходимые утилиты обычно включаются в установщик выбранного дистрибутива Linux и уже доступны для использования. В

рамках безопасности, ОС Linux (в дальнейшем иногда для сокращения будем употреблять сокращение ОС вместо термина «операционная система») гораздо лучше обеспечивает приватность в сравнении с ОС Windows. Но существует еще один критерий, по которому Windows значительно опережает Linux: совместимость с выпускаемыми играми. Учитывая, что [6] Windows занимает более 60% рынка ОС для персональных компьютеров (Desktop Operating System), то у разработчиков гораздо больше мотивации создавать игры именно для Windows.

Подытоживая проведенное небольшое сравнение, отметим, что ОС семейства Windows являются лидером в домашнем сегменте, для простых пользователей, большим достоинством для которых является простота использования и огромное количество поддерживаемого программного обеспечения включая игры. ОС семейства Linux обладают гибкостью в настройке, большей защищенностью и бесплатностью и используются не только простыми пользователями, но и разработчиками программного обеспечения.

Учитывая, что в рамках данной лабораторной работы потребуется запустить написанную программу в ОС Windows, рассмотрим консольные команды операционных систем Windows и Linux для работы с файлами и каталогами [22]. Некоторые из них приведены в таблице 3.

Таблица 3. Цифровой формат записи прав для команды chmod [22]

Описание	**Команда Windows**	**Команда Linux**
Отображение списка файлов и каталогов	DIR	dir, ls
Создание каталога	MKDIR	mkdir
Удаление каталога	RMDIR	rmdir
Удаление файла	DEL, ERASE	rm
Переход в другой каталог	CD	cd
Копирование файлов или каталогов	COPY, XCOPY	cp
Переименование файла	REN, RENAME	mv
Перемещение файлов	MOVE	mv

Окончание табл. 3

Описание	Команда Windows	Команда Linux
Вывод на экран содержимого файла	TYPE, MORE	cat, less, more
Сравнение содержимого двух файлов	COMP, FC	cmp, diff, diff3, sdiff
Сортировка строк в текстовом файле	SORT	sort
Изменение атрибутов файла	ATTRIB	chmod

Далее рассматривать межпроцессное взаимодействие на основе именованных каналов и сокетов мы будем с использованием операционной системы Windows.

В данной операционной системе каждый процесс предоставляет ресурсы, которые нужны для выполнения программы, имеет виртуальное адресное пространство, уникальный идентификатор, выполнимый код, контекст безопасности, переменные среды, как минимум один поток выполнения и т.п. [7].

Среди характеристик процессов Windows следует отметить следующие [7]:

- Процессы Windows реализованы как объекты.

- Процесс может быть создан как новый процесс или как копия существующего процесса.

- Выполнимый процесс может содержать один или несколько потоков.

- Как объекты процессов, так и объекты потоков имеют встроенные возможности синхронизации.

Таким образом, можно определить процесс как объект, соответствующий заданию или приложению пользователя, который владеет собственными ресурсами, такими как память и открытые файлы. Тогда поток - это диспетчеризуемая единица работы, которая выполняется последовательно и является прерываемой, что позволяет процессору переключиться на выполнение другого потока [7].

Операционная система Windows поддерживает многопоточность, потоки одного процесса могут обмениваться между собой информацией с помощью общего адресного пространства и, соответственно, имеют доступ к совместным ресурсам процесса. Многопоточный процесс является эффективным инструментом реализации серверных приложений: каждый новых запрос может обрабатываться в рамках нового созданного потока.

Поток в Windows может находиться в следующих состояниях [7]: готовый к выполнению (догично, из названия – может быть направлен на выполнение), резервный (будет запущен следующим на данном процессоре), выполняющийся, ожидающий, переходный (готов к выполнению, но ресурсы недоступны), завершенный.

4.2 Межпроцессное взаимодействие на примере каналов

Каналы – средство межпроцессного взаимодействия, можно сказать, что это особый тип файлов, которые, как и обычные можно читать и/или модифицировать, в зависимости от указанных на это прав процесса, который подключается к каналу. Каналы в Windows разделяются на 2 типа: анонимные (Anonymous Pipes) и именованные (Named Pipes) [27]. Нас будут интересовать только именованные каналы, т.к. через анонимные можно передавать данные в одном направлении (полудуплексный режим работы канала) и в основном они используются для передачи данных от родительского процесса к дочернему. Именованные каналы являются компонентами WinAPI и, следовательно, для их использования необходимо включение библиотеки <Windows.h>.

В дальнейшем будут приведены примеры с использованием библиотеки Microsoft Developer Network (MSDN), в которой содержится документация на продукты Microsoft, включая примеры кода [28]. Для более подробного изучения нужно зайти на сайт [28] и в строке поиска указать интеерсующую функцию.

Для того, чтобы создать именованный канал, необходимо выполнить функцию CreateNamedPipe (CreateNamedPipeA) [28].

Параметры функции:

LPCSTR lpName, // имя канала

DWORD dwOpenMode, // атрибуты канала

DWORD dwPipeMode, // режим передачи данных

DWORD nMaxInstances, // максимальное количество экземпляров канала

DWORD nOutBufferSize, // размер выходного буфера

DWORD nInBufferSize, // размер входного буфера

DWORD nDefaultTimeOut, // время ожидания связи с клиентом

LPSECURITY_ATTRIBUTES lpSecurityAttributes // атрибуты защиты

Рассмотрим некоторые параметры чуть подробнее. Параметр dwOpenMode должен быть равен 0 либо одному из ряда флагов, часть из которых приведена в таблице 4. Некоторые флаги параметра dwPipeMode приведены в таблице 5.

С остальными можно ознакомиться в [28].

Таблица 4. Флаги параметра dwOpenMode

Имя флага	**Значение**
PIPE_ACCESS_DUPLEX	Использование канала для чтения и записи
PIPE_ACCESS_INBOUND	Использование канала только для чтения
PIPE_ACCESS_OUTBOUND	Использование канала только для записи
FILE_FLAG_FIRST_PIPE_ INSTANCE	При попытке создания нескольких экземпляров канала с этим флагом, создание первого экземпляра выполнится успешно, создание следующего экземпляра завершится неудачно с ERROR_ACCESS_DENIED.

Таблица 5. Флаги параметра dwPipeMode

Имя флага	**Значение**
PIPE_TYPE_BYTE	Режим работы канала, ориентированный на передачу байт.

Окончание табл. 5

Имя флага	Значение
PIPE_TYPE_MESSAGE	Данные записываются в канал как поток сообщений. Канал обрабатывает байты, записанные во время каждой операции записи, как единицу сообщения.
PIPE_READMODE_ BYTE	Данные читаются из канала как поток байтов. Этот режим может использоваться с PIPE_TYPE_MESSAGE или PIPE_TYPE_BYTE.
PIPE_READMODE_ MESSAGE	Данные читаются из канала как поток сообщений. Этот режим можно использовать только в том случае, если также указан параметр PIPE_TYPE_MESSAGE.
PIPE_WAIT	Режим блокировки включен. Когда дескриптор канала указан в функциях ReadFile, WriteFile или ConnectNamedPipe, операции не завершаются до тех пор, пока нет данных для чтения, все данные записаны или клиент не подключен. Использование этого режима может означать, что в некоторых ситуациях клиентский процесс ожидает выполнения определенного действия.

Пример использования именованных каналов показан в листинге 6.

Листинг 6. Пример использования именованного канала

```
…
wchar_t wbuf[1000];
HANDLE pipes[100];
…
//при создании массива каналов
pipes[i] = CreateNamedPipe (wbuf, PIPE_ACCESS_DUPLEX,
PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT,
PIPE_UNLIMITED_INSTANCES, 1, 1, 0, NULL);
…
CloseHandle( pipes[i] );
```

При успешном создании канала функция возвращает дескриптор канала, с помощью которого можно к нему обратиться. Иначе

возвращается значение INVALID_HANDLE_VALUE или ERROR_INVALID_PARAMETR.

После того как канал будет создан (например, на сервере), необходимо дождаться ответа о соединении с каналом со стороны подключающегося процесса (клиента). Для этого нужно использовать [28] функцию ConnectNamedPipe (), параметры которой приведены в таблице 6.

Таблица 6. Параметры функции ConnectNamedPipe

Имя параметра	Значение
hNamedPipe	Дескриптор серверной части экземпляра именованного канала. Этот дескриптор возвращается функцией CreateNamedPipe.
lpOverlapped	Указатель на структуру OVERLAPPED

Функция ConnectNamedPipe возвращает True, если сервер дождался ответа от клиента за время, указанное при создании экземпляра канала и False, если это не произошло.

Для отсоединения процесса нужно использовать функцию DisconnectNamedPipe (HANDLE hNamedPipe) [28], в качестве параметра передавая дескриптор отключаемого канала. Функция возвращает True при удачном отключении и False иначе.

Со стороны процесса-клиента перед подсоединением к каналу логично проверить, занят канал или свободен. Это можно сделать с помощью функции WaitNamedPipeA (LPCSTR lpNamedPipeName, DWORD nTimeOut) [28], где первый параметр является указателем на имя канала, второй параметр – интервал ожидания. Функция возвращает True, если канал не занят и False, если за интервал ожидания не было получено ответа.

Именованный канал – это в какой-то мере своеобразный файл. Для подключения к каналу нужно использовать функцию CreateFileA [28]. В таблице 7 приведены ее параметры.

Для передачи используются функции WriteFile () [33] и ReadFile () [28] с атрибутами, описанными в таблицах 8-9. Соот-

ветственно, если функции завершаются успешно, возвращаемое значение отлично от нуля (True).

Таблица 7. Параметры функции CreateFile

Имя параметра	Значение
lpFileName	Указатель на имя канала
dwDesiredAccess	Чтение/запись в канал. Наиболее часто используемые значения - GENERIC_READ, GENERIC_WRITE или (GENERIC_READ | GENERIC_WRITE).
dwShareMode	Режим совместного использования в том числе на чтение, запись, удаление, их комбинаций (FILE_SHARE_READ, FILE_SHARE_WRITE, FILE_SHARE_DELETE)
lpSecurityAttributes	Атрибуты защиты. Может принимать значение NULL.
dwCreationDisposition	Флаг открытия канала, для работы с существующими каналами должен быть равен OPEN_EXISTING
dwFlagsAndAttributes	Другие флаги и атрибуты. Если задан 0 , то тогда флаги и атрибуты будут определены по умолчанию
hTemplateFile	Дополнительные атрибуты, обычно задается равным NULL

Таблица 8. Параметры функции WriteFile

Имя параметра	Значение
hFile	Дескриптор канала
lpBuffer	Указатель на буфер, содержащий данные для записи
nNumberOfBytesToWrite	Размер данных в байтах, которые будут записаны
lpNumberOfBytesWritten	Указатель на переменную, которая получает количество записанных байтов
lpOverlapped	Указатель на структуру OVERLAPPED требуется, если параметр hFile был открыт с помощью FILE_FLAG_OVERLAPPED, в противном случае этот параметр может иметь значение NULL.

Таблица 9. Параметры функции ReadFile

Имя параметра	Значение
hFile	Дескриптор канала
lpBuffer	Указатель на буфер, который получает данные
nNumberOfBytesToRead	Размер данных в байтах, которые будут прочитаны

Окончание табл. 9

Имя параметра	Значение
lpNumberOfBytesRead	Указатель на переменную, которая получает количество прочитанных байтов
lpOverlapped	Указатель на структуру OVERLAPPED требуется, если параметр hFile был открыт с помощью FILE_FLAG_OVERLAPPED, в противном случае этот параметр может иметь значение NULL.

В листинге 7 приведен пример использования вышеописанных функций.

Листинг 7. Пример использования именованного канала со стороны процесса-клиента

```
char buf[10000];
int i, j, p, l;
DWORD rd;
HANDLE pipe;
//Открываем канал
pipe = CreateFile( argv[1], GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL );
if (pipe == INVALID_HANDLE_VALUE) // если
{
  wprintf( L"Error opening pipe %s. Error: %x", argv[1], GetLastError() );
  _getch ();
  return 1;
}
while (true) // читаем, пока не кончится
{
  memset( buf, ' ', sizeof(buf) ); // очищаем буфер, если вдруг был заполнен
  buf[sizeof(buf)-1] = 0;
  l = 0;
  while (!ReadFile( pipe, &l, 1, &rd, NULL )) //Читаем сообщение в 1 байт
  { // Если файл пустой, то ф-ия успешно возвращает 0 прочитанных байт.
    //Поэтому мы читаем до тех пор, пока не прочитаем этот байт.
  Sleep(1);
  continue;
  }
  for( i = 0; i <= l; ) // то же самое с остальной строкой
```

```
  {
    if (ReadFile( pipe, &buf[i], l-i+1, &rd, NULL ) && (rd > 0) )
    {
            buf[i+rd] = 0;
            i += rd;
     }
     else
          Sleep(1);
    }
   buf[l] = 0;
   if (buf[0] == 0) break;
   printf( "Word: %s\n", buf );
}
CloseHandle( pipe );
…
```

4.3 Межпроцессное взаимодействие на примере сокетов

Сокеты – поддерживаемый ядром механизм, скрывающий особенности среды и позволяющий единообразно взаимодействовать процессам, как на одном компьютере, так и в сети. То есть они необходимы для передачи информации от одного процесса другому [3].

Интерфейс программного программирования, созданный для реализации приложений в сети на основе протокола TCP/IP для операционной системы Windows, называется Windows socket (WinSock) [28].

Разберем взаимодействие двух процессов (условного сервера и условного клиента) на основе сокетов в операционной системе Windows.

Схематично, нам нужно реализовать следующее:

1. Подключить библиотеку winsock2.h, в которой реализованы сокеты.

2. Проверить, можно ли пользоваться сокетами с помощью функции WSAStartup

3. Создать сокет

4. Установить соединение.

5. Извлечь запросы на подключение к сокету

6. Читать и передавать сообщения

7. При завершении работы – закрыть сокет и разорвать соединение.

Разберем подробнее. Функция WSAStartup (WORD wVersionRequired, LPWSADATA lpWSAData) [28], где в качестве первого аргумента следует указать версию интерфейса Windows Sockets, в качестве второго аргумента – указатель на структуру данных WSADATA, которая должна получать сведения о реализации Windows Sockets. При неудаче функция возвращает ненулевое значение.

В листинге 8 приведен пример использования данной функции.

Листинг 8. Пример использования функции WSAStartup (WORD wVersionRequired, LPWSADATA lpWSAData)

```
//загружаем WSAStartup
WSAData wsaData; //создаем структуру для загрузки
WORD DLLVersion = MAKEWORD(2, 1);
if (WSAStartup(DLLVersion, &wsaData) != 0) { //проверка на подключение библиотеки
  std::cout << "Error" << std::endl;
  exit(1);
}
```

Далее мы должны создать сокет, для чего существует соответствующая функция socket [37]:

SOCKET WSAAPI socket (int af, int type, int protocol),

где первый аргумент указывает на семейство протоколов. Например, при передаче по сети обычно используется IPv4 или

IPv6, соответственно аргументов в этом случае будет AF_INET или AF_INET6, соответственно.

Второй аргумент функции означает тип создаваемого сокета. Например, аргумент SOCK_STREAM означает, что будут использоваться потоковые сокеты.

И, наконец, третий аргумент функции означает тип протокола. Рекомендуется выставлять значение по умолчанию (0 или NULL), чтобы операционная система выбрала протокол. При неудаче функция вернет нулевое значение.

Далее нам необходимо установить соединение с удалённым узлом с помощью функции [28]:

int WSAAPI connect(SOCKET s, const sockaddr *name, int namelen),

где первый аргумент – это подключаемый сокет, второй аргумент – структура, содержащая номер порта, адрес, и семейство протоколов; третий аргумент функции – это длина структуры в байтах. При ошибках функция вернет ненулевое значение.

Для использования сокета на стороне сервера, его необходимо связать с локальным адресом. Для этого воспользуемся следующей функцией [28]:

int bind(SOCKET s, const sockaddr *addr, int namelen),

где первый аргумент функции – это связываемый сокет, второй аргумент – структура, содержащая номер порта, адрес, и семейство протоколов; третий аргумент – длина структуры в байтах. Если нет ошибок, вернется 0.

Далее требуется перевести сокет в режим ожидания. Сделать это можно данной функцией [28]:

int WSAAPI listen (SOCKET s, int backlog),

где первый аргумент – этот тот сокет, который нам нужно перевести в режим ожидания, второй аргумент – максимальное ко-

личество сообщений в очереди. При удачном завершении функция возвращает нулевое значение.

В листингах 9-10 показан пример использования вышеописанных функций.

Листинг 9. Пример использования функций socket, bind, listen

```
SOCKADDR_IN addr;
int sizeofaddr = sizeof(addr); //размер
addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //адрес
addr.sin_port = htons(1111); //порт
addr.sin_family = AF_INET; //семейство протоколов
//сокет для прослушивания порта
SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL);
//привязка адреса сокету
bind(sListen, (SOCKADDR*)&addr, sizeof(addr));
// прослушивание, сколько запросов ожидается
listen(sListen, 2);
…
```

Листинг 10. Пример использования функций socket, bind, listen

```
//информация об адресе сокета
SOCKADDR_IN addr;
int sizeofaddr = sizeof(addr);
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
addr.sin_port = htons(1111);
addr.sin_family = AF_INET;
//сокет для соединения с сервером
Connection = socket(AF_INET, SOCK_STREAM, NULL);
//проверка на подключение к серверу
if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr)) != 0)
{
  std::cout << "Error: failed connect to server.\n";
  return 1;
}
std::cout << "Connected!\n"; //подключился
…
```

Далее, нам необходимо извлечь запросы на подключение к сокету. Это можно сделать с помощью следующей функции [28]:

SOCKET WSAAPI accept (SOCKET s, sockaddr *addr, int *addrlen),

которая разрешает попытку входящего соединения через сокет, возвращая значение типа SOCKET. Первый аргумент функции – это дескриптор, который идентифицирует сокет, который был переведен в состояние прослушивания с помощью функции прослушивания. Второй аргумент – необязательный указатель на буфер, который получает адрес подключающегося объекта. И, наконец, третий аргумент функции – это необязательный указатель на целое число, которое содержит длину структуры, на которую указывает параметр addr.

Следующее, что нужно сделать после благополучного завершения всех предыдущих пунктов – можно передавать данные. Это можно делать с помощью функций [28]:

int WSAAPI send (SOCKET s, const char *buf, int len, int flags);

int recv (SOCKET s, char *buf, int len, int flags);

где с помощью первой можно отправлять сообщения, а с помощью второй – принимать. Параметры у них схожи: s – дескриптор, идентифицирующий подключенный сокет; buf – указатель на буфер, содержащий данные для передачи/получения; len – длина буфера в байтах, на который указывает параметр buf, flags – набор флагов, которые определяют способ выполнения вызова для отправления или приема сообщений.

Продолжая код листинга 9, продемонстрируем использование функций accept, send и recv в листингах 11 – 12.

Листинг 11. Пример использования функций accept и send

```
int mass = 3;
int ind = 1;
SOCKET newConnection; //сокет для соединения с клиентом
for (int i = 0; (ind <= mass); i++)
{
  // новое соединение, sListen и прочее – определены в листинге 9
```

```
  newConnection = accept (sListen, (SOCKADDR*)&addr, &sizeofaddr);
  if (newConnection == 0)
  {
    std::cout << "Error in connect! \n";
  }
  else
  {
    std::cout << "Client Connected!\n"; //отправили сообщение клиенту
    Connections[i] = newConnection; //сохранили соединение
    Counter++; //прибавили к общему числу соединений
    if (ind <= mass)
    {
      char msg[256];
      sprintf(msg, "%d", ind);
      send(Connections[i], msg, sizeof(msg), NULL);
      ind++;
    }
  }
}
```

Листинг 12. Пример использования функции recv на стороне клиента (продолжение листинга 10)

```
char msg[256];
while (true)
{
  recv(Connection, msg, sizeof(msg), NULL); //получение сообщения
  if (msg[0] == 'e')
  {
    std::cout << "Exit ...";
    Sleep(3000);
    return 0;
  }
  else {
    std::cout << msg << std::endl; //вывод сообщения
    Sleep(100);
   send(Connection, msg, sizeof(msg), NULL);
  }
}
```

Далее, когда вся информация передана и работа завершена требуется закрыть сокет и разорвать соединение. Для этого используется следующая функция [28]:

int closesocket(IN SOCKET s);

где s – дескриптор, идентифицирующий сокет для закрытия. Если ошибки не возникает, closesocket возвращает ноль. В противном случае возвращается значение SOCKET_ERROR, и конкретный код ошибки можно получить, вызвав WSAGetLastError.

Для отключения библиотеки WinSock2 и освобождения ресурсов перед закрытием программы нужно вызвать функцию int WSACleanup().

Возвращаемое значение равно нулю, если операция прошла успешно. В противном случае возвращается значение SOCKET_ERROR.

Пример использования данных функций показан в листинге 13.

Листинг 13. Пример использования функций closesocket и WSACleanup

```
#define MY_PORT 555
//связывание  сокета с локальным адресом
SOCKET mysocket;
sockaddr_in local_addr;
local_addr.sin_family = AF_INET;
local_addr.sin_port = htons(MY_PORT);
local_addr.sin_addr.s_addr = 0;
//-----------------------------------------------------
//вызываем bind для связи сокета с адресом
if ( bind(mysocket, (sockaddr*)&local_addr, sizeof(sockaddr)))
{
  printf("Error bind %d\n", WSAGetLastError());
  closesocket(mysocket); // Закрытие созданного сокета
  WSACleanup();          // Деинициализация библиотеки Winsock
  return -1;
}
```

4.4 Задание на лабораторную работу № 4

Задания выполняются на C++ в операционной системе Windows. В качестве дополнительного задания можно избрать любой другой язык программирования и другую операционную систему. Вариант 0 разбирается в следжующем разделе. Следует отметить, что число клиентов зависит от задания. Задания выполняются индивидуально. Нечетный вариант (1, 3, 5…) – реализация через именованные каналы, четный вариант (2, 4, 6…) – реализация через сокеты. Варианты заданий:

Вариант 0.

Сервер должен на каждый из запущенных процессов клиентов записать числа от одного до n, причём процессы клиентов могут запускаться произвольно пользователем. В случае разрыва связи с клиентом, который на данный момент получает значения, сервер должен моментально «ловить» следующий клиент, который ждёт подключения, и заполнять уже его. Минимум 2 клиента. Реализация через каналы и через сокеты.

Вариант 1-2.

С процесса-сервера запускается n процессов клиентов (не менее двух). На каждом из клиентов вводится матрица, которая затем отсылается на сервер. Далее, на сервере происходит суммирование этих матриц, и результат рассылается по клиентам.

Вариант 3-4.

На сервере запускается n клиентов. После запуска каждый клиент передает сообщение о подключении на сервер. Если подключиться не удалось – нужно передать ошибку и ее код. Сообщения, полученные от разных клиентов нужно различать. Т.е. клиенты должны быть пронумерованы. После создания клиентов: сервер должен «мониторить» в режиме постоянного опроса, запущены клиенты на данный момент или нет. При отключении очередного

клиента требуется выдавать информативное сообщение. Не менее 3 клиентов.

Вариант 5-6.

Сервер должен получать от одного единственного клиента сообщения в формате Арифметическая Операция Целое или Дробное Число (например, +322, -55, /400.54, *322 и т.д.). После каждого такого ввода на сторону клиента должно быть передано сообщение со всей формулой на данный момент. Полученные от клиента части выражения нужно приписывать справа от уже имеющегося. Следует предусмотреть набор служебных команд для проверки выражения на корректность (в частности, деление на 0), очистка последовательности, расчет выражения. Следует также учесть при расчете приоритетность арифметических операций.

Так же должны быть предусмотрены такие вещи как набор служебных команд для очищения последовательности вычисления, расчета выражения, проверка корректности выражения (отсутствие в нём /0). Выражение должно считаться по правилам арифметики с соответствующим приоритетом операций.

Вариант 7-8.

Задача сервера заключается в создании n процессов-клиентов. Далее с помощью командной строки и соответствующего набора команд требуется изменять цвет фона консоли процесса-клиента на один из заранее предусмотренных цветов. В данном случае синтаксис не принципиален, главное, чтобы работало в соответствии с заданием. Требуется предусмотреть команду, с помощью которой можно восстановить оригинальный цвет консоли. От клиента требуется получать сообщение и действовать согласно заданию и команде (т.е. изменить свой цвет фона). Достаточно 1 клиента.

Вариант 9-10.

Требуется создать и запустить с сервера n процессов, открыть файл (имя файла вводится через консоль), и раздать все слова из

открытого файла между всеми клиентами по следующему правилу: 1-й клиент должен получить 1-е слово, 2-й – второе и т.д. вплоть до n процесса, после чего процесс должен повторяться вплоть до конца считываемого файла. Не менее 3 клиентов.

Вариант 11-12.

Сервер должен в режиме эхопечати (т.е. сразу выводится на экран) получать все введенные данные с клиента. Предусмотреть случай стирания текста через backspace. Клиент должен отправлять данные о каждом введенном символе, эхопечать должна поддерживать русский язык (в случае отсутствия поддержки задание засчитывается по минимальному баллу). Минимум 1 клиент. Если выбрана реализация с несколькими клиентами, то каждый из них должен выводить свои символы на сервере своим уникальным для него цветом текста.

Вариант 13-14.

Требуется запустить несколько процессов-клиентов. При запуске сервера, он должен закрыть все открытые таким образом процессы-клиенты и не давать запускаться новым клиентам. Также он не должен давать запуститься запустить более чем одной копии себя самого. Клиенты до запуска сервера должны считать каждую секунду числа от одного до 1.000.000. От 3 до 5 клиентов достаточно.

Вариант 15-16.

На сервере хранится база данных (достаточно в виде набора записей из нескольких полей, информация хранится в виде текстового файла, откуда берется при запуске сервера). При обращении к серверу от клиента, должна выводиться вся информация по передаваемому запросу. Не менее двух клиентов.

Вариант 17-18.

Существует 4 типа скобок: {}, (), <>, []. С клиента на сервер передается скобочная последовательность, а на сервере произво-

дится анализ, корректна ли переданная последовательность. Примеры последовательностей: (){[]} – верно, [}q() – не верно. Ограничений на длину последовательности нет.

Вариант 19-20.

Криптографическая задача – использование шифра Цезаря. На клиенте вводится текстовая последовательность, шифруется и отсылается в зашифрованном виде на сервер. Сервер принимает сообщение, расшифровывает и выводит текст. Минимальный вариант реализации – один клиент. Если клиентов несколько, тогда требуется также пересылать идентификатор клиента, чтобы на сервере выводилось, от какого клиента получено сообщение. Усложненный вариант – для каждого клиента свой вариант шифра.

Вариант 21-22.

Через сервер требуется создать 2 процесса клиента и случайно сгенерированное число от 28 до 111. Клиенты играют через сервер в игру: каждый из клиентов поочередно отнимает числа от 1 до 3 от сгенерированного числа. Тот, кто сделает число отрицательным или нулевым – проиграл.

Вариант 23-24.

С процесса-сервера запускается n процессов клиентов. Для каждого из созданных клиентов указывается время жизни (в секундах). Клиент запускается, существует заданное время и завершает работу. Также следует предусмотреть значение для бесконечного времени. Требуется не менее трех одновременно запускаемых процессов-клиентов.

Вариант 25-26.

С процесса-сервера запускается n процессов клиентов. На клиенте вводится текст, который затем отсылается на сервер. На сервере происходит анализ: в полученном сообщении подсчитывается частота появления каждого символа. Далее информация отсылается на клиент.

Вариант 27-28.

С процесса-сервера запускается n процессов клиентов. На клиенте пользователь вводит строку символов, которая отсылается на сервер. На сервере происходит анализ: подсчитывается частота появления во введенной строке гласных букв и цифр. Далее информация отсылается на клиент.

4.5 Пример реализации лабораторной работы № 4

В данном разделе приведен пример реализации 0 варианта лабораторной работы №4. На листингах 14-15 показана реализация задания с помощью именованных каналов, на листингах 16-17 аналог, но с помощью сокетов. Необходимые комментарии по коду добавлены в текст листингов.

Листинг 14. Реализация 0 варианта лабораторной работы № 4 с помощью именованных каналов, серверная часть

```
#include <windows.h>
#include <iostream>
#include <thread>
#define CONNECTING_STATE 0
#define READING_STATE 1
#define WRITING_STATE 2
#define INSTANCES 1
#define PIPE_TIMEOUT 5000
#define BUFSIZE 512
 using namespace std;
typedef struct
{
  OVERLAPPED oOverlap;  //Структура, позволяющая производить асинхронные операции
  HANDLE hPipeInst;  //Дескриптор канала
  unsigned int cbRead; //Хранит кол-во прочитанных байт
  unsigned int size;
  unsigned int dwState;  //Состояние работы канала: запись, чтение или подключение
  bool fPendingIO;  //Состояние, отображающее, подключен ли к каналу клиент
```

```
} PIPEINST;
void DisconnectAndReconnect(unsigned int);
bool ConnectToNewClient(HANDLE, LPOVERLAPPED);
PIPEINST Pipe[INSTANCES];
HANDLE hEvents[INSTANCES];
void print_time(){
  SYSTEMTIME lt;
  GetLocalTime(&lt);
  printf("%02d:%02d:%02d\t", lt.wHour, lt.wMinute, lt.wMilliseconds);
}
int main(void)
{
  unsigned int i, dwWait, cbRet, dwErr;
  bool fSuccess;
  bool pSuccess = true;
  //Инициализация всех экземляров канала
  for (i = 0; i < INSTANCES; i++) {
    //Событие для экземпляра
    hEvents[i] = CreateEvent(
      NULL,    //Атрибут защиты
      true,    //Ручное управление
      true,    //Начальное состояние - активно
      NULL);   //Имя
    if (hEvents[i] == NULL) {
      print_time();
      cout << "[ERROR] CreateEvent failed with " << GetLastError() << '\n';
      return 0;
    }
    //Присваиваем структуре OVERLAP событие
    Pipe[i].oOverlap.hEvent = hEvents[i];
    //Инициализируем сам канал
    Pipe[i].hPipeInst = CreateNamedPipe(
      "\\\\.\\pipe\\mynamedpipe",            //Имя канала
      PIPE_ACCESS_DUPLEX |     //Чтение и запись в канал
      FILE_FLAG_OVERLAPPED,    //Включен перекрывающийся ввод/вывод
      PIPE_TYPE_MESSAGE |   // Канал передает сообщения, а не поток байт
      PIPE_READMODE_MESSAGE |  // Канал считывает сообщения
```

```
      PIPE_WAIT,          // Клиент будет ожидать, когда канал станет доступен
      INSTANCES,          // Максимальное количество экземпляров канала
      BUFSIZE*4,   //Выходной размер буфера
      BUFSIZE*4,   //Входной размер буфера
      PIPE_TIMEOUT,           // Время ожидания клиента
      NULL);                  //Атрибут защиты
    print_time();
    cout << "[MESSAGE] Initializing Pipe[" << i << "]\n";
    if (Pipe[i].hPipeInst == INVALID_HANDLE_VALUE) {
      print_time();
      cout << "[ERROR] CreateNamedPipe failed with " << GetLastError() << '\n';
      return 0;
    }
    //Подключаем канал к клиенту
    Pipe[i].fPendingIO=ConnectToNewClient(Pipe[i].hPipeInst, &Pipe[i].oOverlap);
        //Смотрим, произошло ли подключение к клиенту
    Pipe[i].dwState = Pipe[i].fPendingIO ?
      CONNECTING_STATE : // Нет
      READING_STATE;     //Да
  }
   while (1) {
    int j = 1;
    //Ждем, пока один из каналов закончит одну из операций
    dwWait = WaitForMultipleObjects(
      INSTANCES,    // Количество событий
      hEvents,      // События
      false,        // Не ждать всех
      INFINITE);    // Ждать бесконечно
     //dwWait указывает, какой канал завершил операцию. WAIT_OBJECT_0 -
первый элемент массива каналов
    i = dwWait - WAIT_OBJECT_0;   //Указывает какой канал в массиве завершил
операцию
    if (i < 0 || i > (INSTANCES - 1)) {
      print_time();
      cout << "[ERROR] Index out of range.\n";
      return 0;
    }
```

```
if (Pipe[i].fPendingIO) {
  fSuccess = GetOverlappedResult(
    Pipe[i].hPipeInst, //Экземпляр канала
    &Pipe[i].oOverlap, //Структура OVERLAPPED соответствующего канала
    &cbRet,          //Количество переданных байтов
    false);          //Не ждать
   switch (Pipe[i].dwState) {
    //Ждет начала операции чтения
    case CONNECTING_STATE:
      print_time();
      cout << "[MESSAGE] Connected\n";
      if (! fSuccess) {
        print_time();
        cout << "[ERROR] Error code" << GetLastError() << '\n';
        return 0;
      }
      Pipe[i].dwState = READING_STATE;
      break;
     //Ждет начала операции записи
    case READING_STATE:
      if (! fSuccess || cbRet == 0) {
        DisconnectAndReconnect(i);
        continue;
      }
      Pipe[i].cbRead = cbRet;
      Pipe[i].dwState = WRITING_STATE;
      break;
     //Ждет начала операции чтения
    case WRITING_STATE:
      if (!fSuccess || cbRet != 4) {
        DisconnectAndReconnect(i);
        continue;
      }
      Pipe[i].dwState = READING_STATE;
      break;
    default: {
      print_time();
```

```
            cout << "[ERROR] Invalid pipe state.\n";
            return 0;
          }
        }
      }
      //Выполняем текущее действие
      switch (Pipe[i].dwState) {
        //Читаем данные из канала
        case READING_STATE:
          print_time();
          cout << "[MESSAGE] [" << i << "]:Read\n";
          fSuccess = ReadFile(
            Pipe[i].hPipeInst,
            &Pipe[i].size,
            4,
            &Pipe[i].cbRead,
            &Pipe[i].oOverlap);
          //При успешном завершении чтения
          if (fSuccess && Pipe[i].cbRead != 0) {
            Pipe[i].fPendingIO = false;
            Pipe[i].dwState = WRITING_STATE;
            continue;
          }
          //Если операция чтения еще не закончилась, то пропускаем
          dwErr = GetLastError();
          if (! fSuccess && (dwErr == ERROR_IO_PENDING)) {
            Pipe[i].fPendingIO = true;
            continue;
          }
          //Иначе если произошла ошибка, отключаемся от клиента
          DisconnectAndReconnect(i);
          break;

        //Запись данных в канал
        case WRITING_STATE: {
          if(Pipe[i].size > BUFSIZE) {
            print_time();
```

```
            cout << "[ERROR] The numbers size exceeds the size of the buffer\n";
            dwErr = -1;
            fSuccess = WriteFile(Pipe[i].hPipeInst, &dwErr, 4, &cbRet,
&Pipe[i].oOverlap);
            break;
          }
          print_time();
          cout << "[MESSAGE] [" << i << "]:Write\n";
          for(; j <= Pipe[i].size; j++) {
            Sleep(500);
            fSuccess = WriteFile(Pipe[i].hPipeInst, &j, 4, &cbRet, &Pipe[i].oOverlap);
            if(!fSuccess && pSuccess) {
              DisconnectAndReconnect(i);
              pSuccess = fSuccess;
            }
          }
          int b = -1;
          fSuccess = WriteFile(Pipe[i].hPipeInst, &b, 4, &cbRet, &Pipe[i].oOverlap);
          //При успешном завершении записи
         if (fSuccess && cbRet == 4) {
            Pipe[i].fPendingIO = false;
            Pipe[i].dwState = READING_STATE;
            pSuccess = true;
            j = 1;
            continue;
          }
          //Если запись еще не завершилась, то пропускаем
          dwErr = GetLastError();
          if (! fSuccess && (dwErr == ERROR_IO_PENDING)) {
            Pipe[i].fPendingIO = true;
            continue;
          }
          //Иначе если произошла ошибка, отключаемся от клиента
          DisconnectAndReconnect(i);
          break;
        }
        default: {
```

```
          print_time();
          cout << "[ERROR] Invalid pipe state.\n";
          return 0;
        }
      }
  }
  return 0;
}
//Отключаемся от текущего клиента и ждем нового
void DisconnectAndReconnect(unsigned int i)
{
  //Отключаемся от текущего
  //print_time();
  //cout << "[MESSAGE] [" << i << "]:Disconnecting\n";
  if (! DisconnectNamedPipe(Pipe[i].hPipeInst) ) {
    print_time();
    cout << "[ERROR] DisconnectNamedPipe failed with " << GetLastError() << '\n';
  }
  //Подключаемся к новому клиенту
  Pipe[i].fPendingIO = ConnectToNewClient(Pipe[i].hPipeInst, &Pipe[i].oOverlap);
  //Проверяем состояние
  Pipe[i].dwState = Pipe[i].fPendingIO ?
    CONNECTING_STATE : //Ждем подключения
    READING_STATE;     //Подключен
}
//Если клиент подключился, то fPendingIO = false
bool ConnectToNewClient(HANDLE hPipe, LPOVERLAPPED lpo)
{
  BOOL fConnected, fPendingIO = false;
  //Пытаемся подключиться к каналу
  fConnected = ConnectNamedPipe(hPipe, lpo);
  if (fConnected) {
    print_time();
    cout << "[ERROR] ConnectNamedPipe failed with " << GetLastError() << '\n';
    return 0;
  }
  switch (GetLastError())
```

```
  {
    //Подключение в процессе. Функцию выполнили, и если подключения не произошло, выбрасывается данная ошибка
    case ERROR_IO_PENDING:
      //print_time();
      //cout << "[MESSAGE] Waiting\n";
      fPendingIO = true;
      break;
    //Если клиент уже ожидал подключения. Просто обновляем событие у канала
    case ERROR_PIPE_CONNECTED:
      print_time();
      cout << "[MESSAGE] Connected\n";
      if (SetEvent(lpo->hEvent))
        break;
    //Ошибка
    default: {
      print_time();
      cout << "[ERROR] ConnectNamedPipe failed with " << GetLastError() << '\n';
      return 0;
    }
  }
  return fPendingIO;
}
```

Листинг 15. Реализация 0 варианта лабораторной работы № 4 с помощью именованных каналов, клиентская часть

```
#include <windows.h>
#include <iostream>
#define BUFSIZE 512
using namespace std;
void print_time(){
  SYSTEMTIME lt;
  GetLocalTime(&lt);
  printf("%02d:%02d:%02d\t", lt.wHour, lt.wMinute, lt.wMilliseconds);
}
int main(int argc, char *argv[])
{
  HANDLE hPipe;                 // Канал
```

```
bool fSuccess = false; // Переменная для проверки корректности операций
unsigned int  cbIO;            //Количество считанных/записанных байт
int number_count;              //Количество чисел, которые надо передать
//Пытаемся открыть канал и ждем его, если недоступен
while (1)
{
  hPipe = CreateFile( "\\\\.\\pipe\\mynamedpipe",   //Имя канала
    GENERIC_READ |             //Чтение и запись из канала
    GENERIC_WRITE,
    0,                         //Уровень общего доступа
    NULL,                      //Атрибуты безопасности
    OPEN_EXISTING,             //Открыть существующий
    0,                         //Атрибуты файла
    NULL);                     //Файл шаблона
  if (hPipe != INVALID_HANDLE_VALUE)
    break;
  //Если не удалось открыть канал
  if (GetLastError() != ERROR_PIPE_BUSY) {
    print_time();
    cout << "[ERROR] Could not open pipe. GLE=" << GetLastError() << '\n';
    return -1;
  }
  //Если все экземпляры канала заняты, ждем 20 секунд
  if (!WaitNamedPipe("\\\\.\\pipe\\mynamedpipe", 20000)) {
    print_time();
    cout << "[ERROR] Could not open pipe: 20 second wait timed out.";
    return -1;
  }
}
number_count = 20;
//Отправляем серверу, какое кол-во значений надо получить клиенту
//print_time();
//cout << "[MESSAGE] Write\n";
fSuccess = WriteFile(
  hPipe,               //Канал
  &number_count,       //Данные для передачи
  4,                   //Количество переданных байт
```

```
    &cbIO,          //Сколько байт передалось на самом деле
    NULL);          //Не асинхронный вывод
  if (!fSuccess) {
    print_time();
    cout << "[ERROR] WriteFile to pipe failed. GLE=" << GetLastError() << '\n';
      return -1;
  }
  print_time();
  cout << "[MESSAGE] Message sent to server, receiving reply as follows:\n";
  //Читаем из канала необходимые значения
  print_time();
  cout << "[MESSAGE] Read\n";
  int* x = new int[number_count];
  do {
    for(int i = 0; i < number_count; i++){
      fSuccess = ReadFile(
        hPipe,          // Канал
        &(*(x+i)),      //Куда записываем данные
        4,              //Какое кол-во байт записываем
        &cbIO,          //Сколько записали байт
        NULL);          //Не асинхронное чтение
      if((*(x + i) == 1 && i != 0) || *(x + i) == -1) break;
      cout << x[i] << '\n';
    }
    if (!fSuccess && GetLastError() != ERROR_MORE_DATA)
      break;
  } while (!fSuccess);
  if (!fSuccess) {
    print_time();
    cout << "[ERROR] ReadFile from pipe failed. GLE=" <<  GetLastError() << '\n';
    return -1;
  }
  print_time();
  cout << "[MESSAGE] Closing pipe and exit\n";
  CloseHandle(hPipe);
  return 0;
}
```

Листинг 16. Реализация 0 варианта лабораторной работы № 4 с помощью сокетов, серверная часть

```
#pragma comment(lib, "ws2_32.lib")
#include <iostream>
#include<winsock2.h>
#pragma warning(disable: 4996)//для inet_addr
using namespace std;
//проверка на то, отправилось и считалось ли число
bool printCausedBy(int Result, const char* nameOfOper) {
   if (!Result) {
      cerr << "Connection closed.\n";
      return false;
   }
   else if (Result < 0) {
      cout << nameOfOper;
      cerr << " failed: ", WSAGetLastError();
      return false;
   }
   return true;
}
int main() {
   //Загрузка библиотеки
   WSAData wsaData; //создаем структуру для загрузки
   WORD DLLVersion = MAKEWORD(2, 1); // Версия библиотеки winsock
   if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка подключения
      cerr << "Error: failed to link library.\n";
      return 1;
   }
   //Заполняем информацию об адресе сокета
   SOCKADDR_IN addr;
   int sizeOfAddr = sizeof(addr);
   addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   addr.sin_port = htons(1111);
   addr.sin_family = AF_INET;
   SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL); //сокет для
прослушивания порта
   if (bind(sListen, (SOCKADDR*)&addr, sizeOfAddr) == SOCKET_ERROR) {
//привязка адреса сокету
```

```
        printf("Error bind %d\n", WSAGetLastError());
        closesocket(sListen);
        WSACleanup();
        return 1;
    } //подключение прослушивания с максимальной очередью
    if (listen(sListen, SOMAXCONN) == SOCKET_ERROR) {
        cout << "Listen failed;\n";
        closesocket(sListen);
        WSACleanup();
        return 1;
    }
    int max = 0, current = 1; //переменные для максимального числа и текущего
    char buffer[256] = "1";
    while (true) {
        SOCKET newConnection = accept(sListen, (SOCKADDR*)&addr,
&sizeOfAddr); //сокет для соединения с клиентом
        if (!newConnection) { //проверяем, произошло ли соединение с клиентом
            cout << "Error in connect!\n";
            if (closesocket(sListen) == SOCKET_ERROR)
                cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastEr-
ror();
            WSACleanup();
            return 1;
        }
        else {
            cout << "New client connected\n";
            if (!max) {
                cout << "Input max number: ";
                cin >> max;
                while (cin.fail() || max <= 0 || cin.peek() != '\n') {
                    cin.clear();
                    cin.ignore(32768, '\n');
                    cout << "I don't understand you, sorry. Please, try again.\n";
                    cout << "Input max number >> ";
                    cin >> max;
                }
                while (printCausedBy(send(newConnection, buffer, 256, NULL), "Send")
```

```
&& current < max) {
                itoa(++current, buffer, 10);
                Sleep(1000);
            }
            cout << ("Client disconnect or current number greater than max number\n\n");
            if (closesocket(newConnection) == SOCKET_ERROR)
                cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
        }
        else {
            if (current > max) {
                if (closesocket(newConnection) == SOCKET_ERROR)
                    cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
                break;
            }

            while (printCausedBy(send(newConnection, buffer, 256, NULL), "Send") && current < max) {
                itoa(++current, buffer, 10);
                Sleep(1000);
            }
            cout << ("Client disconnect or current number greater than max number\n\n");
            if (closesocket(newConnection) == SOCKET_ERROR)
                cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
        }
    }
  }
if (closesocket(sListen) == SOCKET_ERROR)
    cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
  WSACleanup();
  return 0;
}
```

Листинг 17. Реализация 0 варианта лабораторной работы № 4 с помощью сокетов, клиентская часть

```
#pragma comment(lib, "ws2_32.lib")
#include <iostream>
#include<winsock2.h>
#pragma warning(disable: 4996)//для inet_addr
using namespace std;
//проверка на то, отправилось и считалось ли число
bool printCausedBy(int Result, const char* nameOfOper) {
   if (!Result) {
      cout << "Connection closed.\n";
      return false;
   }
   else if (Result < 0) {
      cout << nameOfOper;
      cerr << " failed:\n", WSAGetLastError();
      return false;
   }
   return true;
}
int main() {
   //Загрузка библиотеки
   WSAData wsaData; //создаем структуру для загрузки
   WORD DLLVersion = MAKEWORD(2, 1); //запрашиваемая версия библиотеки winsock
   if (WSAStartup(DLLVersion, &wsaData) != 0) { //проверка подключения библиотеки
      cerr << "Error: failed to link library.\n";
      return 1;
   }
   //Заполняем информацию об адресе сокета
   SOCKADDR_IN addr;
   addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //адрес
   addr.sin_port = htons(1111); //порт
   addr.sin_family = AF_INET; //семейство протоколов
   //сокет для прослушивания порта
   SOCKET Connection = socket(AF_INET, SOCK_STREAM, NULL);
   //проверка на подключение к серверу
```

```
    if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr))) {
        cerr << "Error: failed connect to the server.\n";
        return 1;
    }
    cout << "Server connection established.\n";
    char buffer[256];
    buffer[0] = 'g';
    //считываем из буфера с сервера текущее число и выводим
    while (printCausedBy(recv(Connection, buffer, 256, NULL), "Recv")) {
        cout << buffer << ' ';
        Sleep(1000);
    }
    cout << "Server has been stopped\n";
    if (closesocket(Connection) == SOCKET_ERROR)
        cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
    WSACleanup();
    cout << "Correct exit done\n";
    return 0;
}
```

4.6 Контрольные вопросы

1. Почему сложно сравнивать семейства операционных систем Windows и Linux?

2. По каким критериям можно проводить сравнение семейств операционных систем Windows и Linux?

3. Какими преимуществами обладают операционные системы семейства Windows?

4. Какими преимуществами обладают операционные системы семейства Linux?

5. Какие особенности процессов в операционной системе Windows вам известны?

6. В каких состояниях может находиться поток в операционной системе Windows?

7. Дайте определение канала.

8. Какой функцией можно создать именованный канал?

9. Какими функциями пользуются для передачи информации при использовании именованных каналов?

10. Что такое сокеты?

11. Опишите схему взаимодействия двух процессов (условного сервера и условного клиента) на основе сокетов.

12. Какой командой используются для проверки подключения к сокету?

13. Какие команды используются для передачи данных в сокетах?

14. Что необходимо сделать, когда сокет уже не нужен, и программа завершается?

5 Использование семафоров и мьютексов для клиент-серверного взаимодействия в Windows

5.1 Модель «клиент/сервер»

Модель «клиент/сервер» известна своим использованием для распределенных вычислений [7]. В целом, она нам уже знакома, кратко каждый серверный процесс ожидает от клиента запрос к предоставляемой службе. Клиенты (могут быть прикладной программой или другой серверной программой) запрашивают предоставляемую службу (например, при использовании знакомых нам сокетов). Сервер выполняет требуемые операции и возвращает информацию обратно клиенту.

К явным преимуществам модели «клиент/сервер» можно отнести упрощение исполнительной системы, повышение надежности, а также то, что данная модель является базой для распределенных вычислений [7]. Следует подчеркнуть, что кроме собственно серверов и клиентов, важной составляющей данной модели является сеть. И данная модель все же обычно является распределенной.

Подробное рассмотрение компьютерных сетей и протоколов выходит за рамки данного курса, тем не менее, ознакомиться с особенностями архитектуры модели «клиент/сервер» можно в [7], глава 18.

5.2 Синхронизация процессов и потоков

Передача информации между процессами предполагает наличие синхронизации: в частности, получатель не в состоянии получить сообщение до тех пор, пока оно не послано другим процессом. Также нужно помнить о том, что между процессами часто происходит конкуренция за ресурсы. О взаимодействии между процессами мы говорили в рамках лекционного курса, вспомним синхронизацию процессов с помощью семафоров и мьютексов.

Немного истории. В 1965 году было предложено использовать целочисленную переменную для подсчета количества активизаций, отложенных на будущее [3]. Человеком, кто предложил данную идею был Эдсгер Дейкстра, один из авторов концепции структурного программирования, создатель алгоритма Дейкстры. Именно он предложил назвать эту новую переменную семафором (semaphore).

Алгоритм использования семафоров следующий: занять ресурс (атомарная операция P), работа с критическим ресурсом, освободить ресурс (атомарная операция V). По Дейкстре операция P блокирует семафор, операция V — деблокирует семафор. Сейчас операции с семафорами в общем виде обычно называют up и down (или signal и wait, соответственно) [3]:

- up увеличит значение семафора на 1 или разблокирует процесс, находящийся в ожидании.

- down уменьшает значение семафора на 1 или блокирует процесс, если семафор равен 0.

Проверка значения, его изменение, и, при необходимости, приостановка процесса осуществляются как единое и неделимое атомарное действие [3]. Таким образом, можно гарантировать, что с началом семафорной операции никакой другой процесс не может получить доступ к семафору до тех пор, пока операция не будет завершена или заблокирована. Атомарность – необходимое условие для решения проблем синхронизации и исключения состязательных ситуаций [3].

Достаточно интересно принцип работы семафоров показан в публикации в англоязычном оригинале [29], также есть русскоязычный перевод данной статьи.

Возможны ситуации, когда требуется всего два состояния рассматриваемой выше переменной – заблокированном и незаблоки-

рованном. Такой аналог одноместного семафора называется мьютекс.

То есть мьютекс – это упрощенная версия семафора, совместно используемая переменная, которая может находиться в одном из двух состояний: заблокированном или незаблокированном [3].

Мьютекс допускает только один поток в контролируемом участке, заставляя другие потоки, которые пытаются получить доступ к этому разделу ждать, пока первый поток не вышел из этого раздела [3].

Семафоры в операционных системах Windows описываются объектами классов Semaphores и SemaphoreSlim. Первый класс является тонкой оболочкой вокруг объекта семафора Win32, которые могут быть использованы для управления доступом к пулу ресурсов. Второй класс представляет упрощенный, быстрый семафор, который можно использовать для ожидания внутри одного процесса, когда предполагается, что время ожидания будут очень коротким [28].

Следующий пример показывает создание семафора с максимальным числом три [28]. Для блокировки семафора используются пять потоков (листинг 18).

Листинг 18. Пример использования объекта класса Semaphore

```
//Семафор, имитирующий ограниченный пул ресурсов.
private static Semaphore _pool
public static void Main()
   {
      /* Создание семафора, контролирующего до 3 обращений
      одновременно. Начальное значение счетчика семафора – 0. */
      _pool = new Semaphore(0, 3);
      // Создание и запуск пять потоков.
      for(int i = 1; i <= 5; i++)
      {
         Thread t = new Thread(new ParameterizedThreadStart(Worker));
         t.Start(i);
```

```
        }
        /* Ждем полсекунды, чтобы все потоки запустились
        и заблокировались на семафоре. */
        Thread.Sleep(500);
        /* Вызов Release (3) возвращает счетчик семафоров к максимальному
        значению и позволяет ожидающим потокам использовать семафор,
       до трех потоков одновременно.       */
        Console.WriteLine("Main thread calls Release(3).");
        _pool.Release(3);
        Console.WriteLine("Main thread exits.");
    }
private static void Worker(object num)
    {
        // Каждый рабочий поток начинается с запроса семафора.
        Console.WriteLine("Thread {0} begins " +
            "and waits for the semaphore.", num);
        _pool.WaitOne();
        // Для упорядочивания вывода, интервал заполнения
        int padding = Interlocked.Add(ref _padding, 100);
        Console.WriteLine("Thread {0} enters the semaphore.", num);
        /* Имитация работы в виде сна длительностью около секунды.
        Каждая нить "работает" немного длиннее для упорядочивания вывода*/
        Console.WriteLine("Thread {0} releases the semaphore.", num);
        Console.WriteLine("Thread {0} previous semaphore count: {1}",
            num, _pool.Release());
}
```

Вообще говоря, каждому процессу соответствует адресное пространство и одиночный поток исполняемых команд [3]. Соответственно, у одного процесса может быть много потоков, и потоки процесса разделяют общие ресурсы процесса.

Преимущества потоков перед процессами [3]:

- Упрощение программы в ряде случаев за счет использования общего адресного пространства.

- Быстрота создания потока в сравнении с процессом до 100 раз.

- Повышение производительности самой программы по причине того, что есть возможность одновременно выполнять, например, вычисления на процессоре и операцию ввода/вывода.

Для общеязыковой среды выполнения (CLR) для использования в качестве семафора для событий ожидания можно создать объект IHostSemaphore с использованием функции CreateSemaphore [28], или использовать функцию Win32 с тем же именем. Рассмотрим функцию CreateSemaphoreA из Win32 API (Win32) [28]:

HANDLE CreateSemaphoreA(LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCSTR lpName);

Здесь lpSemaphoreAttributes – это атрибуты защиты, lInitialCount – начальное значение семафора, lMaximumCount – максимальное значение семафора, lpName – имя семафора. Возвращаемое значение если функция завершается успешно – дескриптор объекта семафора, если функция завершается ошибкой, возвращаемое значение равно NULL. Пример вызова показан в листинге 19.

Листинг 19. Пример использования функции CreateSemaphoreA

```
// создаем семафор
hSemaphore = CreateSemaphoreA(NULL, 0, 10, NULL);
if (hSemaphore == NULL)
  return GetLastError();// Для получения расширенной информации об ошибке
// создаем поток, который готовит элементы массива
hThread = CreateThread(NULL, 0, thread, NULL, 0, &IDThread);
if (hThread == NULL)
  return GetLastError();
…
// закрываем дескриптор объекта
CloseHandle(hSemaphore);
…
```

С более полным примером использования данной функции можно ознакомиться в документации MSDN [28].

Далее рассмотрим уже знакомые нам мьютексы (Mutex) в операционной системе Windows [28]. В листинге 20 показан пример использования Mutex для синхронизации доступа к защищенному ресурсу [28].

Листинг 20. Синхронизация: пример использования Mutex

```
using namespace System;
using namespace System::Threading;
const int numIterations = 1;
const int numThreads = 3;
ref class Test
{
public:
  static Mutex^ mut = gcnew Mutex;   // Создание нового Mutex
  static void MyThreadProc()
  {
    for ( int i = 0; i < numIterations; i++ )
    {
      UseResource();
    }
  }
private:
  /* Моделирование ресурса, который должен быть синхронизирован, чтобы
  за один раз мог войти только один поток*/
  static void UseResource()
  {
    // Ждем пока все будет в порядке.
    mut->WaitOne();
    Console::WriteLine( "{0} has entered protected the area",
Thread::CurrentThread->Name );
    // Имитация некоторой работы
    Thread::Sleep(500);
    Console::WriteLine( "{0} is leaving protected the area\r\n",
Thread::CurrentThread->Name );
    // Освобождение Mutex.
    mut->ReleaseMutex();
```

```
  }
};
int main()
{
  // Создание потоков, которые будут использовать смоделированный ресурс
  for ( int i = 0; i < numThreads; i++ )  {
  Thread^ myThread = gcnew Thread( gcnew ThreadStart(Test::MyThreadProc ) );
   myThread->Name = String::Format( "Thread {0}", i + 1 );
   myThread->Start();
  }
}
```

Основные операции, возникающие при использовании Mutex в Windows: создание mutex, открытие mutex, ожидание и захват mutex, освобождение mutex.

Для решения проблемы синхронизации параллельных потоков также используются такие объекты как события (event), которые используются для уведомления ожидающих потоков о наступлении какого-либо события (либо операции). Выделяют два типа событий: с автосбросом (auto - reset events) и со сбросом вручную (manual - reset events) и [27].

Event в Win32 создается с помощью функции CreateEventA [28]:

```
HANDLE CreateEventA(
  LPSECURITY_ATTRIBUTES lpEventAttributes,
  BOOL bManualReset,
  BOOL bInitialState,
  LPCSTR lpName
);
```

где lpEventAttributes – указатель на структуру SECURITY_ATTRIBUTES. Если этот параметр равен NULL, дескриптор не может наследоваться дочерними процессами. Параметр fManualReset используется для создания события со сбросом вруч-

ную(True) или с автосбросом (False). Параметр fInitialState определяет начальное состояние события – свободное (True) или занятое(False). Параметр lpName – имя объекта события, ограничено символами MAX_PATH. Сравнение имен чувствительно к регистру.

Если функция завершается успешно, возвращаемое значение является дескриптором объекта события. Если функция завершается ошибкой, возвращаемое значение равно NULL.

Применение событий показано в документации [28] и кратко в листинге 21.

Листинг 21. Пример использования событий для синхронизации

```
//создание события Show, будет сообщать о печати
HANDLE hShow = CreateEvent(NULL, FALSE, FALSE, "Show");
HANDLE hPress0 = OpenEvent(EVENT_ALL_ACCESS, FALSE, "Null");
HANDLE hPress1 = OpenEvent(EVENT_ALL_ACCESS, FALSE, "One");
// Если ошибочное состояние
if (hPress0 == NULL || hPress1 == NULL)
{
  cout << "Open event failed." << endl;
  cout << "Input any char to exit." << endl;
  return GetLastError();
}
//массив событий
HANDLE hThread[2];
hThread[0] = hPress0;
hThread[1] = hPress1;
…
for (;;)
{
  /* Ожидание, пока один или все из указанных объектов находятся в
  сигнальном состоянии или пока не истечет интервал времени ожидания*/
  WaitForMultipleObjects(2, hThread, FALSE, INFINITE);
 /* Ожидание, пока объект не окажется в сигнальном состоянии или пока не
  истечет время ожидания. */
  if (WaitForSingleObject(hPress0, 0) == WAIT_OBJECT_0)
  {
```

```
    cout << "0 ";
    ResetEvent(hPress0); // Переводим событие в несигнальное состояние
    SetEvent(hShow); // Устанавливаем событие Show в сигнальное состояние
  }
  else if (WaitForSingleObject(hPress1, 0) == WAIT_OBJECT_0)
  {
    cout << "1 ";
    ResetEvent(hPress1);
    SetEvent(hShow);
  }
}
```

5.3 Задание на лабораторную работу № 5

Использовать семафоры (мьютексы) для синхронизации работы процессов (потоков) в процессе реализации задания на лабораторную работу. При необходимости также можно использовать события. Требуется реализовать консольный вариант. Базовый вариант реализации – Win32 API в Windows. Дополнительные варианты реализации при использовании других операционных систем и языков программирования – на усмотрение студента. Желательно предусмотреть для всех заданий событие, при котором соответствующий процесс завершает свою работу, если это возможно.

Список вариантов:

Вариант 0.

Шпионские игры. Требуется написать клиент-серверное приложение, моделирующее взаимодействие разведывательного управления (сервер) и резидентов (клиенты). Для моделирования передачи сообщений ввести события, которые обозначают «Start», «Message» и «End». Сообщения на клиенте должны шифроваться шифром Цезаря, а на сервере расшифровываться. Принимать сообщение может только от трёх процессов, причем на сервере должно быть ясно, от какого из резидентов было получено сооб-

щение. Передача остальных сообщений от других процессов должна блокироваться.

Вариант 1.

Холодная война. Шпионские игры-2. Требуется написать клиент-серверное приложение, моделирующее взаимодействие разведывательного управления (Алекс-сервер) и разведчиков (Юстасы-клиенты). Для моделирования передачи сообщений ввести события, которые обозначают «точку» и «тире». На выбор студента реализовать перевод в азбуку Морзе сообщения на русском или английском языках, записываемого на клиенте и передачу в таком виде на сервер. На сервере требуется реализовать расшифровку полученного сообщения и вывести на экран. Также следует показывать как отосланное на клиенте, так и присланное на сервер сообщения. Также добавить событие – конец сеанса. Принимать сообщение может только от одного процесса-клиента, передача остальных сообщений от других клиентов должна блокироваться с помощью мьютекса.

Вариант 2.

Эрвин Шредингер известен не только как физик-теоретик, лауреат нобелевской премии, но и как человек, предложивший парадокс кота Шредингера: согласно квантовой механике объект может находиться в двух кардинально противоположных состояниях одновременно, относительно кота в коробе – быть живым и не совсем, что выглядит немного странно. Задача лабораторной работы – разобраться в данном моменте. В качестве сервера выступает Эрвин Шредингер, в качестве клиентов – коты, которые могут находиться в 3 состояниях – «жив», «мертв», «не определен». Из состояния «не определен» случайным образом клиент может перейти в одно из двух оставшихся состояний. По запросу с сервера клиенты должны присылать информацию о своем состоянии. Если клиент находится в неживом состоянии, он может отослать ин-

формацию об этом только 1 раз. Далее он отключается от сервера. В остальных случаях на запрос сервера клиент должен ответить. Не менее 5 клиентов. Клиенты должны быть пронумерованы и на сервере должно указываться, от какого клиента было получено сообщение. Запускать клиентов мы можем сами.

Вариант 3.

Банковские транзакции. Требуется написать клиент-серверное приложение, моделирующее взаимодействие сервера банка и процессов банковских карт (клиенты). Сервер банка должен запрашивает у пользователя количество процессов-клиентов, а также пароль для каждого. Пароль состоит из трех любых цифр. На клиентах будут вводиться суммы трат по карте, но только после введения правильного пароля для соответствующего клиента. Если пароль введен неверно, просьба ввести его снова. Если пароль введен неверно 3 раза, клиент блокируется и вводить сообщения с него уже нельзя. Клиенты на сервере должны различаться. Принимать сообщения о тратах сервер может только от двух процессов, остальные – блокируются с помощью семафора.

Вариант 4.

Автомобильный дилерский центр. Требуется написать клиент-серверное приложение, моделирующее взаимодействие сервера центра и продавцов-клиентов. На сервере хранится база автомобилей с их характеристиками. Требуется передавать часть этой базы по запросу клиента и, при необходимости, вносить в нее изменения. Взаимодействовать с базой данный в 1 момент времени может только 1 клиент. При изменении базы данных все клиенты должны получать сообщение – «Произошло обновления БД!».

Вариант 5.

Требуется написать клиент-серверное приложение - калькулятор. Вычисления происходят на сервере. Одновременно может подключаться к серверу два клиента. На клиенте вводятся числа в

формате «арифметическая операция целое или дробное число пробел целое или дробное число». Достаточно двух чисел для одной операции. На сервере данные от соответствующего клиента отображаются в новой строке, в формате «Получено от клиента №X арифметическая операция «формат операции». Результат - …». Ответ от сервера на клиенте выводится в новой строке.

Вариант 6.

Книжный магазин. Требуется написать клиент-серверное приложение, моделирующее взаимодействие сервера магазина и ПК его отделов (клиенты). На сервере хранится база данных (БД) книг, с указанием отдела магазина, к которому относятся данные книги. На клиентах – локальные БД, относящиеся только к этому отделу. Вносить изменения в БД можно только на сервере. При внесении изменений в конкретном отделе, эти изменения должны отсылаться на конкретный клиент. Одновременно к серверу может быть подключено 3 клиента. При подключении, на клиента отсылается его локальная БД. Указать не менее 4 отделов в БД сервера.

Вариант 7.

Требуется написать клиент-серверное приложение – умножение матриц. Матрицы вводятся на клиентах – по одной на каждый клиент. На сервере происходит вычисление матриц. Одновременно можно умножать только по две матрицы, то есть работа начинается, если есть четное число процессов клиентов.

Вариант 8.

Требуется написать клиент-серверное приложение. Пользователь вводит несколько элементов числового массива (включая дробные) на процессах клиентах. Требуется удалить повторяющиеся числа на каждом процессе-клиенте. Одновременно сервер обрабатывает не более 3 процессов-клиентов. Каждый клиент – рассматривается отдельно, то есть наличие одинаковых чисел на разных клиентах возможно.

Вариант 9.

Требуется написать клиент-серверное приложение - нахождение обратной матрицы. Матрицы вводятся на клиентах – по одной на каждый клиент. На сервере происходит вычисление обратной матрицы. Одновременно можно вычислять 2 матрицы. На сервере должно быть показано, какая матрица получена от клиента и какая отослана на соответствующий клиент.

Вариант 10.

Перевод между системами счисления. Требуется написать клиент-серверное приложение следующего вида: На сервере происходит перевод заданного клиентом числа из десятичной системы счисления в двоичную или шестнадцатеричную и пересылка обратно на текущего клиента. Принимать и работать можно только с 1 клиентом. Требуется предусмотреть событие, завершающее сеанс конкретного клиента (но не отключающее его, с него позже также можно передавать цифры), тогда подключается новый. Начальное число клиентов задается на сервере.

Вариант 11.

Задача о парикмахере. Начало 19 века. В небольшом городе есть только 1 парикмахерская. Там обычно работает один парикмахер. Когда клиентов в очереди становится больше 2, он зовет своего младшего брата-мясника, который ему помогает. После обслуживания 4 клиентов парикмахер отдыхает, как и его брат. Создать много-процессное приложение, моделирующее рабочий день парикмахерской. Допускается использовать в качестве сервера процесс-парикмахерскую, в котором есть два потока – парикмахера и его брата. Клиенты – отдельные процессы. Взаимодействие между ними должно быть на уровне текстового диалога, например, такого: «Добро пожаловать, процесс Х. «Здравствуйте, господин цирюльник» … «Всего доброго, приходите к нам еще!» При жела-

нии, можно добавить диалог между парикмахером и процессом клиентом.

Вариант 12.

Задача о гладиаторах. В Колизее император Нерон снова устраивает гладиаторские игры. Считаем, что в этих играх выступают два типа гладиаторов: самниты и фракийцы. Они выходят на арену Колизея и между ними начинается сражение. Одновременно может сражаться только одна пара гладиаторов, остальные должны ждать своей очереди. Один тип гладиаторов друг с другом сражаться не может. В Колизее ведется учет текущим играм – таблица, в которой подсчитывается учет побед. Смоделировать проведение игр императора Нерона. В качестве процесса – сервера выступает Колизей, к которому подключаются процессы-гладиаторы. В качестве битвы может быть случайное определение победителя и/ или игра: есть случайно сгенерированное число от 20 до 50. Клиенты-гладиаторы, поочерёдно отнимают от числа от 1 до 3. Проиграет тот, кто первый обнулит число или сделает его отрицательным. Ave, Caesar!

Вариант 13.

Робин Гуд и его команда. В Шервудском лесу живет Робин Гуд и его друзья. Они защищают местных жителей от шерифа Ноттингема, за что местные жители Робина очень уважают. Узнав, что Робин любит напиток из ячменя, который сложно приготовить в лесу, они начали делиться с ним излишками (измеряем в бочонках). Пусть у нас есть X местных жителей, уважающих Робина. Это отдельные процессы-клиенты. Каждый житель может принести 1 бочонок к определенному месту под дубом в Шервудском лесу (это сервер) раз в день. Если бочонков становится больше 5, текущий житель с бочонком звонит в колокол на ветке дерева. Услышав звук, Робин идет к дубу (считаем как отдельные потоки на сервере-дубе), благодарит жителей и забирает по 1 бочонку за

раз (предусмотреть время, которое нужно, чтобы отнести бочонок до убежища Робина). Если после первого удара колокола бочонков становится больше 5, то Робин зовет 2 друзей на помощь (запуск 2 отдельных потоков, которым нужно то же время для условной транспортировки бочонка до базы Робина), которые активны, пока не перетаскают все бочонки. Промоделировать рабочую неделю (5 дней) шервудского братства. Сервер условный лес с дубом и отдельными потоками Робина и друзей, клиенты – жители Ноттингема. В конце работы вывести статистику, сколько всего бочек было, и сколько раз Робин звал друзей на помощь

Вариант 14.

Задача о студентах в общежитии. Пять студентов из одного блока общежития №7 приготовили ужин и сели за стол. Этот вечер они будут проводить, чередуя приемы пищи и размышления о сессии. В центре стола находится блюдо еды (какая еда – несущественно, студенты натренированы). Также студенты поспорили, что они будут есть только при использовании двух вилок, общее число которых равно числу студентов. Между кажым из студентов лежит по одной вилке. Чтобы не мешать друг другу студенты договорились, что каждый будет пользоваться только теми вилками, которые лежат рядом с ним (слева и справа). Требуется промоделировать ужин студентов-клиентов. Решение должно быть симметричным, то есть все процессы-студенты должны выполнять один и тот же код. В качестве сервера выступает стол, к которому обращаются процессы-клиенты.

Вариант 15.

Задача о курящих ковбоях. В салун после трудового дня заходят ковбои. Они устали от охоты за бизонами и хотят спокойно провести время, покурив сигары. Для этого процесса им нужны табак, бумага и спички. Но во время скачек за бизонами каждый из ковбоев потерял что-то из ингридиентов, нужных для сигары. Что-

бы взять недостающее, они идут к барной стойке и зовут бармена, у которого есть все (вчера была доставка рейсовым дилижансом). Бармен кладет на стол по два разных случайных компонента из трех. Тот ковбой, у которого есть недостающий ингридиент, забирает компоненты и начинает курить. Если компоненты никто не забрал за секунду, бармен забирает их. Каждый из ковбоев – отдельный процесс. Их может быть 3 типа: у одного процесса-курильщика есть табак, у второго - бумага, а у третьего - спички. Доступ к столу могут иметь одновременно 3 ковбоя, но все они должны иметь разные ингридиентов. Общее число ковбоев в салуне может быть больше 3. После курения, процесс-курильщик «засыпает» на некоторое время, заданное пользователем. Частота курения также может задаваться пользователем. Промоделировать работу салуна.

Вариант 16.

Обед первобытного племени. После удачной охоты на мамонта, в наличии есть еда. Считаем, что племя у нас продвинутое и знает тайну «красного цветка», то есть огня. Соответсвенно, добыча варится в большом котле (мы помним, что племя у нас продвинутое), где ее готовности ждут N человек-клиентов. За готовностью следит повар, который тоже принимал участие в охоте и очень устал. Сварив обед, повар засыпает (считаем, что он напробовался во время приготовления еды, и есть не хочет). Когда процесс-человек хочет есть, он берет из котла 1 кусок (этого достаточно, чтобы он был сыт). Доступ к котлу одновременно может иметь только 1 процесс. Если котел пуст, то это текущий процесс будит повара и ждет, пока тот не приготовит еду. Задача - промоделировать трапезу племени. Считаем, что на сервере есть два потока, моделирующих работу повара и котла, а процессы-клиенты подключаются к нему во время еды. В целом, на сервере каждого

из подключившихся клиентов можно обрабатывать в отдельном потоке.

Вариант 17.

В библиотеке работают три библиотекаря, в разных отделах (научно-популярный, истрический, художественный). Есть каталог книг для каждого отдела. В библиотеку приходят читатели, жаждущие знаний или студены, жаждущие сдать сессию. Нужно промоделировать работу библиотеки. В качестве сервера – библиотека с отдельными потоками-библиотекарями. Процессы-клиенты – это посетители библиотеки. В качестве каталога книг допускается использовать текстовый файл, где в каждой строке – отдельная книга с названием и автором, частота чтения и взятия ее по абонементу на дом. На сервере требуется выводить информацию о том, где находится соответствующий пометитель, в каком отделе, что читает. На клиенте – что именно было прочитано или взято почитать на дом. После завершения рабочего дня нужно подвести итог на сервере – сколько было клиентов, что читали и что взяли на дом и обновить соответствующим образом данные в файлах.

Вариант 18.

Задача о супермаркете, в котором работают N кассиров. Как в любом подобном магазине там очень ждут покупателей, которые делают покупки. После того, как конкретный покупатель завершает свой выбор, что он хочет купить, он становится в очередь к случайному кассиру. Изначально кассир один и ему требуется время, чтобы обслужить покупателя. Если в очереди накапливается больше 3 человек, кассир зовет своего коллегу. Если во всех очередях больше 3 человек – количество кассиров увеличивается, пока не достигнет N. Если покупателей меньше не становится (распродажа мороженого), то требуется звать управляющего. После того, как появляется управляющий, кассиры начинают обслуживать клиентов в два раза быстрее. Смоделировать рабочий день

супермаркета. В качестве сервера – стойка касс с потоками-кассирами. Процессы-клиенты – отдельные покупатели. Подвести итог в конце дня, сколько было клиентов всего и у каждого кассира и сколько денег они оставили в магазине (аналогично, всего и у каждого из кассиров).

Вариант 19.

Паломничество. Средние века. Паломники хотят отправиться по святым для них местам и хотят ускорить свой путь, воспользовавшись кораблями. На корабль вмещается 5 паломников. Во флоте – два одинаковых корабля. Когда корабль заполняется – он отплывает и странствует по дальним морям, через некоторое время возвращаясь. Паломники ждут, пока очередной корабль не прибудет за ними. Если ожидание затягивается (у каждого паломника оно свое), они не ждут места на корабле и идут пешком. Смоделировать паломничество, в качестве сервера – флот кораблей с потоками-кораблями, в качестве процессов-клиентов – сами паломники. После того, как закончатся паломники (это может быть явно заданное событие), подвести итог, сколько паломников было, и как они добирались до важных для них мест.

Вариант 20.

Начало 21 века. Терапевтическое отделение больницы. Заведующий отделением и четыре интерна лечат пациентов (время лечения случайно). Каждый из этой великолепной пятерки может принять только одного пациента за раз, но если он не знает, как лечить (если это интерн), то он должен обратиться к заведующему отделением. Считаем, что заведующий знает все, но после общения с интерном он должен немного отдохнуть, помедитировать и подумать о жизни. Если в это время к нему подходят несколько интернов – они сами образуют очередь. Пациенты покидают отделение уже вылечившись. Создать приложение, моделирующее рабочий день терапевтического отделения. В качестве сервера –

отделение с врачами. Пациенты – это процессы-клиенты. Если врач-интерн, то у него должна быть предусмотрена вероятность того, что он вылечит очередного пациента, а если не получилось, то после этого он идет к заведующему. В конце дня подвести итог – кто и кого смог вылечить, а также сколько времени медитировал заведующий отделением.

Вариант 21.

Древний Египет. Фараон строит себе пирамиду из каменных блоков. Каждый день по Нилу из каменоломен Верхнего Египта идут корабли с каменными блоками (5 блоков в корабле). Разгрузка происходит в порту на Ниле в Нижнем Египте. Обратно корабли загружаются папирусом (2 свитка) и пшеницей (30 бушелей). Одновременно может разгружать только два корабля. Остальные ждут своей очереди на реке, отбиваясь от нильских крокодилов. Команда может отбиться от трех нападений. Если за определенное время корабль так и не смог быть принят в порту, то крокодилы съедают 5 бушелей. Если на корабле не остается пшеницы – злые крокодилы его топят. Корабль с папирусом крокодилы начинают разрушать после того, как не найдут на нем зерна. Смоделировать работу порта в древнем Египте. В качестве сервера – порт. В качестве процессов клиентов – корабли. Где и чем являются крокодилы – на ваше усмотрение.

Вариант 22.

В древней Греции развито было производство керамики. В Афинах в одной мастерской два мастера изготовляют керамику (один – амфоры и второй – кратеры). Для производства высокой амфоры требуется 5 кусков глины, а для производства кратера – 2 куска глины. Их помощники приносят на склад от 2 до 4 кусков глины за 1 раз. После каждого второго рейса помощники отдыхают. Склад вмещает 20 кусков глины. Если он заполнен – помощники ждут. После того, как мастера сделают по 2 предмета, они

отдыхают. Смоделировать работу керамической мастерской. В качестве сервера – сама мастерская со складом и потоками мастерами, в качестве клиентов – процессы-помощники. Предусмотреть определенную продолжительность дня и в конце его подвести итог, сколько времени простаивали помощники и сколько керамики произвели мастера.

Вариант 23.

Разработать чат для обмена сообщениями. Пусть на сервере есть чат, к которому могут одновременно присоединяться только 3 процесса-клиента. Остальные ждут своей очереди. Чат общий для всех, то есть при подключении, отключении клиента и появлении нового сообщения информация об это рассылается по всем подключенным клиентам, но старая история для вновь подключившегося клиента не отсылается.

Вариант 24.

Усовершенствованная версия чата. Разработать чат для обмена сообщениями. Пусть на сервере есть чат, к которому могут одновременно присоединяться только 2 процесса-клиента. Остальные ждут своей очереди. Чат общий для всех, у каждого клиента сообщения пишутся своим цветом. Если клиент только подключился – ему отсылается вся текущая история и задается цвет фона консоли процесса-клиента на один из заранее предусмотренных.

Вариант 25.

Многопользовательская онлайн-игра. На сервере стоит интересная игра, в которую очень хотят поиграть геймеры-клиенты. Но пропускная способность сервера ограничена. Если подключены 3 геймера, все хорошо, если 5 – сервер начинает работать хуже, и начинает «лагать», если 10 – сервер начинает принудительно отключать клиентов, пока их не останется 5. У клиентов есть параметр – терпение. Если терпение 1, клиент при малейших проблемах начинает писать жалобы (проблема уменьшает терпение на 1).

Если 2 – то, соответственно, со 2 раза, и т.д. Клиенты могут писать гневные комментарии в чат на сервере. Если сервер «лагает», то соответствующий клиент с терпением 0 пишет в чат, если же клиента отключили, то он ждет возможности подключиться и пишет сразу две жалобы с тремя восклицательными знаками. Также клиенты с терпением больше 0 могут писать хвалебные комментарии, например: «Ура, ничего не упало!» раз в минуту. Промоделировать работу многопользовательской игры и чата. В конце работы, когда клиентов не остается – вывести статистику по жалобам и хвалебным комментариям. Все клиенты должны отличаться.

Вариант 26.

Семейный отель в Эфиопии. В отеле есть 4 номера с ценой 500 эфиопских быров за ночь, 2 номера с ценой 700 эфиопских быров за ночь и 2 номера с ценой 900 эфиопских быров за ночь. Туристы, решившие посетить город Лалибэлу полюбоваться высеченными в скалах церквями, имеют определенные финансовые ресурсы и могут снять номер, если хватает денег. Если денег не хватает, то, пользуясь хорошей погодой, туристы идут ночевать в саванну. Задача: промоделировать работу семейного отеля. Отель – сервер, клиенты – отдельные процессы-клиенты. В конце работы вывести статистику работы отеля, сколько клиентов было и какой доход получен.

Вариант 27.

Середина 19 века, отель в столице Российсокй империи, Санкт-Петербурге. В этом отеле существуют строгие правила – дам можно селить только с дамами, а джентльменов – только с джентльменами. В отеле есть три типа номеров – на одного (3 номера), двух (4 номера) и трех человек (2 номера). Отель решили почтить своим присутствием несколько дам и джентельменов. Считается, что если номер не одноместный и частично заселен, то дама или джентльмен, которые там уже живут, готовы войти в по-

ложение и переночевать с представителем своего пола. Если для очередного клиента не находится подходящего номера, он уходит искать ночлег в другое место. Создать приложение, моделирующее работу такого отеля. Отель – сервер, дамы или джентльмены – клиенты. В конце работы вывести статистику работы отеля, сколько и каких клиентов было и какой доход получен.

Вариант 28.

Экзамен. Преподаватель-сервер принимает экзамен у студентов-клиентов. Одновременно экзамен могут сдавать 3 студента (получить вопросы), иначе у преподавателя рассеивается внимание. Остальные стоят в очереди. На сервере есть список вопросов, который уникален и раздается случайным образом на клиенты так, чтобы не было повторяющихся вопросов. Каждый клиент получает два вопроса. Студенты готовятся и передают на сервер ответы на вопросы, преподаватель ставит оценку (пользователь сам выставляет оценки на сервере) и эта оценка отсылается на клиент. Отвечать на вопросы преподавателю может одновременно только 1 клиент. Если в очереди на сдачу находится больше 5 клиентов – у преподавателя рассеивается внимание и с некоторой вероятностью он может поставить автомат от 2 до 5 очередному студенту, после получения от него сообщения с ответами.

5.4 Пример реализации лабораторной работы № 5

В данном разделе приведен пример реализации 0 варианта лабораторной работы №5.

Реализация задания производилась с помощью семафоров и событий, на листинге 22 приведен код сервера, на листинге 23 – код клиента. Для передачи информации использовались сокеты.

Необходимые комментарии по коду добавлены в текст листингов.

Листинг 22. Реализация 0 варианта лабораторной работы № 5, серверная часть

```
#include <iostream>
#include <string>
#include <vector>
#include<winsock2.h>
#pragma comment(lib, "ws2_32.lib")
#pragma warning(disable: 4996) // для inet_addr
using namespace std;
void decrypt(string* input) {
	string alphabet = "abcdefghijklmnopqrstuvwxyz";
	for (size_t i = 0; i != (*input).length(); i++) {
		for (size_t j = 0; j != alphabet.length(); j++) {
			if ((*input)[i] == alphabet[j]) {
				(*input)[i] = alphabet[(j - 3) % 26];
				break;
			}
		}
	}
}
//проверка на то, отправилось и считалось ли
bool printCausedBy(int Result, const char* nameOfOper) {
	if (!Result) {
		cout << "Connection closed.\n";
		return false;
	}
	else if (Result < 0) {
		cout << nameOfOper;
		cout << "failed:\n", WSAGetLastError();
		return false;
	}
	return true;
}
int main()
{
	//Загрузка библиотеки
	WSAData wsaData; //создаем структуру для загрузки
	//запрашиваемая версия библиотеки winsock
```

```
	WORD DLLVersion = MAKEWORD(2, 1);
	//проверка подключения библиотеки
	if (WSAStartup(DLLVersion, &wsaData) != 0) {
		cerr << "Error: failed to link library.\n";
		return 1;
	}
	//Заполняем информацию об адресе сокета
	SOCKADDR_IN addr;
	static int sizeOfAddr = sizeof(addr);
	addr.sin_addr.s_addr = inet_addr("127.0.0.1");
	addr.sin_port = htons(1111);
	addr.sin_family = AF_INET;
	//сокет для прослушвания порта
	SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL);
	//привязка адреса сокету
	if (bind(sListen, (SOCKADDR*)&addr, sizeOfAddr) == SOCKET_ERROR)
{
		printf("Error bind %d\n", WSAGetLastError());
		closesocket(sListen);
		WSACleanup();
		return 1;
	}
	//подкючение прослушивания с максимальной очередью
	if (listen(sListen, SOMAXCONN) == SOCKET_ERROR) {
		cerr << "Listen failed.\n";
		closesocket(sListen);
		WSACleanup();
		return 1;
	}
	HANDLE hSemaphore;
	//Создание семафора, контролирующего до 3 обращений одновременно
	hSemaphore = CreateSemaphore(NULL, 3, 3, L"semaphore");
	if (hSemaphore == NULL)
		printf("CreateSemaphore error: %d\n", GetLastError());
	size_t client_number = 0;
	vector <SOCKET> Sockets(10);		//вектор для сокетов
	size_t c_num = 0;
```

```
	int n; //для количества резидентов (клиентов)
	HANDLE pool[3] = {
		CreateEvent(NULL,FALSE,FALSE,L"0"),
		CreateEvent(NULL,FALSE,FALSE,L"1"),
		CreateEvent(NULL,FALSE,FALSE,L"end"),
	};
	for (uint8_t i = 0; i < 2; ++i)
		if (!pool[i])
			return GetLastError();
	cout << "Enter the number of residents from 1 to 10 >> ";
	cin >> n;
	while (cin.fail() || n < 1 || n > 10 || cin.peek() != '\n') {
		cin.clear();
		cin.ignore(32768, '\n');
		cout << "I don't understand you, sorry. Please, try again.\n";
		cout << "Enter the number of residents from 1 to 10 >> ";
		cin >> n;
	}
	//задаем информацию для окна открытия
	STARTUPINFO si;//структура
	PROCESS_INFORMATION pi;// структура с информацией о процессе
	ZeroMemory(&si, sizeof(si));
	si.cb = sizeof(si);// указываем размер
	ZeroMemory(&pi, sizeof(pi));
	//создаём новые окна для клиентов
	for (uint8_t i = 0; i < n; i++)
	{
		if (!CreateProcess(L"C:\\Users\\sereg\\source\\repos\\Client_LB5
_OS\\Release\\Client_LB5_OS.exe",   //  module name
				NULL,        // Command line
				NULL,           // Process handle not inheritable
				NULL,           // Thread handle not inheritable
				FALSE,          // Set handle inheritance to FALSE
				CREATE_NEW_CONSOLE, //creation flags
				NULL,           // Use parent's environment block
				NULL,           // Use parent's starting directory
				&si,           // Pointer to STARTUPINFO structure
```

```
                        &pi)            // Pointer to PROCESS_INFORMATION
structure
                        )
                {
                        printf("CreateProcess failed (%d).\n", GetLastError());
                        return 1;
                }
                Sleep(10);
        }
        for (uint8_t i = 0; i < n; i++) { //сокеты для соединения с клиентом
                Sockets[i] = accept(sListen, (SOCKADDR*)&addr, &sizeOfAddr);
        }
        hile (n) {
                //приостанавливает поток пока любой из объектов не перейдет
                // в сигнальное состояние или не закончится время ожидания
                int ind = WaitForMultipleObjects(3, pool, FALSE, INFINITE);
                if (ind == 0)
                {
                        c_num = client_number;
                        cout << "Intelligence agency is ready to receive a mes-
sage!\n";
                        ++client_number;
                }
                if (ind == 1) {
                        cout << "Resident " << c_num << " is ready to con-
nect.\n";
                        cout << "We receive a message...\n";
                        char buffer[1000];
                        printCausedBy(recv(Sockets[c_num],          buffer,
sizeof(buffer), NULL), "Recv");
                        string str = string(buffer);
                        cout << "Intelligence agency received from resident "
                        << c_num << " this message: " << str << "\n";
                        decrypt(&str);
                        cout << "Resident " << c_num << " said: " << str <<
"\n";
                }
```

```
                    if (ind == 2)
                    {
                            cout << "Resident " << c_num << " was disconnected.\n";
                            ReleaseSemaphore(hSemaphore, 1, NULL);
                           //отключение клиента
                            --n;
                    }
            }
            cout << "Work completed successfully!\n";
            // Close process and thread handles.
            CloseHandle(pi.hProcess);
            CloseHandle(pi.hThread);
            for (uint8_t i = 0; i < 3; ++i)
            {
                    CloseHandle(pool[i]);
            }
            for (uint8_t i = 0; i <= c_num; ++i)
            {
                    closesocket(Sockets[c_num]);
            }
            CloseHandle(hSemaphore);
            if (closesocket(sListen) == SOCKET_ERROR)
                    cerr << "Failed to terminate connection.\n Error code: "
                            << WSAGetLastError();
            WSACleanup();
            return 0;
}
```

Листинг 23. Реализация 0 варианта лабораторной работы № 5, клиентская часть

```
#include<iostream>
#include<string>
#include<vector>
#include<winsock2.h>
#pragma comment(lib, "ws2_32.lib")
#pragma warning(disable: 4996)//для inet_addr
using namespace std;
void encrypt(string* input) {
```

```
		string alphabet = "abcdefghijklmnopqrstuvwxyz";
		for (size_t i = 0; i != (*input).length(); ++i) {
			for (size_t j = 0; j != alphabet.length(); ++j) {
				if ((*input)[i] == alphabet[j]) {
					(*input)[i] = alphabet[(j + 3) % 26];
					break;
				}
			}
		}
}
//проверка на то, отправилось и считалось ли
bool printCausedBy(int Result, const char* nameOfOper) {
		if (!Result) {
			cout << "Connection closed.\n";
			return false;
		}
		else if (Result < 0) {
			cout << nameOfOper;
			cerr << "failed:\n", WSAGetLastError();
			return false;
		}
		return true;
}
int main()
{
		//Загрузка библиотеки
		WSAData wsaData; //создаем структуру для загрузки
		//запрашиваемая версия библиотеки winsock
		WORD DLLVersion = MAKEWORD(2, 1);
		//проверка подключения библиотеки
		if (WSAStartup(DLLVersion, &wsaData) != 0) {
			cerr << "Error: failed to link library.\n";
			return 1;
		}
		//Заполняем информацию об адресе сокета
		SOCKADDR_IN addr;
		addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //адрес
```

```
addr.sin_port = htons(1111); //порт
addr.sin_family = AF_INET; //семейство протоколов
//сокет для прослушвания порта
SOCKET Connection = socket(AF_INET, SOCK_STREAM, NULL);
//проверка на подключение к серверу
if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr))) {
        cout << "Unable to connect to intelligence agency.\n";
        return 1;
}
HANDLE hSemaphore;
//открываем семафор
hSemaphore = OpenSemaphore(SYNCHRONIZE, FALSE, L"semaphore");
if (hSemaphore == NULL) {
        cerr << "Error open semaphore.\n";
        return GetLastError();
}
HANDLE pool[3] = {
        OpenEvent(EVENT_ALL_ACCESS,FALSE,L"0"),
        OpenEvent(EVENT_ALL_ACCESS,FALSE,L"1"),
        OpenEvent(EVENT_ALL_ACCESS,FALSE,L"end"),
};
for (int i = 0; i != 2; ++i)
        if (!pool[i])
                return GetLastError();
 // останавливает выполнение программы пока семафор не окажется
// в сигнальном состоянии
WaitForSingleObject(hSemaphore, INFINITE);
cout << "Connection with intelligence agency established!\n";
cout << "Enter the command <start> in order to begin >> ";
string c;
string str;
cin >> c;
while (cin.fail() || c != "start" || cin.peek() != '\n') {
        cin.clear();
        cin.ignore(32768, '\n');
        cout << "I don't understand you, sorry. Please, try again.\n";
        cout << "Enter the command <start> in order to begin >> ";
```

```
                cin >> c;
        }
        SetEvent(pool[0]); //устанавливаем событие в сигнальное состояние
        cout << "Enter the command <message> in order to enter and send the mes-
sage.\n" << "Enter the command <end> in order to finish\n";
        cin >> c;
while (c != "end")
        {

            while (c != "message" && c != "end" || cin.fail() || cin.peek() != '\n')
                {
                        cin.clear();
                        cin.ignore(32768, '\n');
                        cout << "Input error, please enter message or end >> ";
                        cin >> c;
                }
                if (c == "message")
                {
                        cin.clear();
                        cin.ignore(32768, '\n');
                        SetEvent(pool[1]);
                        cout << "Enter your message >> ";
                        getline(cin, str);
                        encrypt(&str);
                        char buffer[1000];
                        cout << "Encoded message: " << str << endl;
                        strcpy(buffer, str.c_str());
                        printCausedBy(send(Connection,  buffer,  sizeof(buffer),
NULL), "Send");
                        cout << "Enter the next command >> ";
                        cin >> c;
                }
        }
        SetEvent(pool[2]);
        for (int i = 0; i < 3; ++i)
                CloseHandle(pool[i]);
        CloseHandle(hSemaphore);
```

```
        if (closesocket(Connection) == SOCKET_ERROR)
    cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();
        WSACleanup();
}
```

5.5 Контрольные вопросы

1. Что такое модель «клиент/сервер», и какие преимущества модели «клиент/сервер» вам известны?

2. Какие операции с семафорами существуют?

3. Какое действие является атомарным?

4. Что такое мьютекс?

5. В чем разница между процессами и потоками?

6. Назовите преимущества потоков перед процессами.

7. Какой функцией нужно пользоваться для создания семафоров в Win32 API?

8. Что такое события?

9. Какие параметры есть и для чего нужны у функции CreateEventA?

10. Какую функцию нужно использовать для перевода события в несигнальное состояние?

6 Применение графического интерфейса пользователя для клиент-серверного взаимодействия

6.1 Графический интерфейс пользователя

В общем случае интерфейс – это способ взаимодействия некоторой системы с внешним миром (другими системами), совокупность средств и правил, обеспечивающих взаимодействие отдельных систем. В качестве примеров можно вспомнить человеко-машинный интерфейс, интерфейсы программирования приложений (API), шаблоны проектирования [30].

В рамках данной лабораторной работы речь пойдет о графическом интерфейсе пользователя – graphical user interface (GUI).

GUI впервые был предложет исследовательской группой Дугласа Энгельбартом в Стэнфордском исследовательском институте [3]. Важными составляющими графического интерфейса являются окна — Windows, значки — Icons, меню — Menus и указывающие устройства — Pointing device. Все вместе – WIMP.

Для ввода в системах с GUI на персональных компьютерах обычно используются клавиатура и мышь. Вывод практически всегда направляется на специальное устройство, называемое графическим адаптером, состоящим из специального блока памяти (видеопамять), где хранятся изображения, появляющиеся на экране [3].

Программное обеспечение GUI может быть реализовано либо в качестве кода на уровне пользователя, как это сделано в системах UNIX, либо в самой операционной системе, как в случае с системой Windows [3].

Для вывода графики на экран используется пакет, состоящий из сотен процедур, которые собраны воедино в форме интерфейса графических устройств — GDI (Graphics Device Interface). Он может обрабатывать текст и графику, и сконструирован таким образом, чтобы быть независимым от платформ и устройств [3].

Базовым языком программирования в данном курсе является C++. Если при написании программ на этом языке используется интегрированная среда разработки (IDE) MS Visual Studio и операционная система Windows, то для разработки интерфейса приложений можно использовать Windows Forms (WinForms) и Windows Presentation Foundation (WPF).

Если используется операционная система Linux, то можно использовать, например, IDE Qt Creator. Альтернативой WinForms являются «виджеты» (Qt Widgets), а альтернатива для WPF – Qt Quick.

Далее показан пример GUI для 12 варианта лабораторной работы №5, задачи о гладиаторах, на языке программирования C++ при использовании Windows Forms. Внешний вид окна сервера приведен на рисунке 6.1. Подчеркну, что это лишь пример, ваша реализация может сильно отличаться.

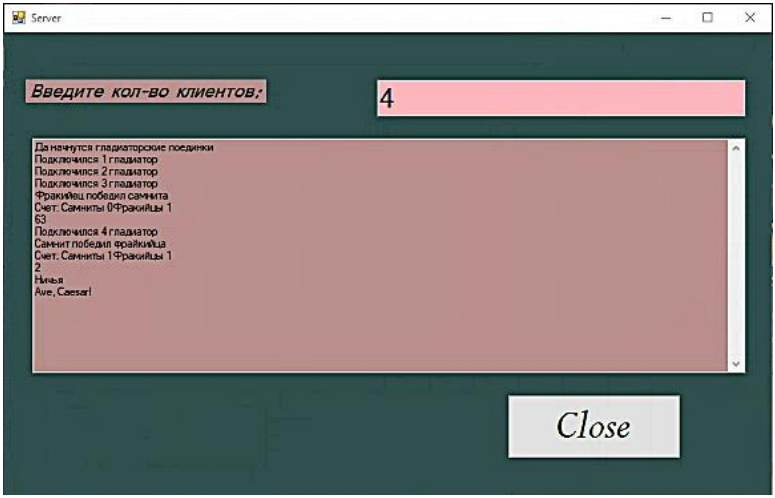

Рис. 6.1. Задача о гладиаторах, 12 вариант, вид сервера

Существует огромное число примеров и видео для разработки приложений, использующих GUI. Чтобы не ограничивать выбор, лабораторную работу № 6 допускается выполнять на любом языке

программирования в любой операционной системе, на усмотрение студента.

6.2 Задание на лабораторную работу № 6

Реализовать графический интерфейс пользователя для функционала межпроцессного взаимодействия ранее сделанной лабораторной работы № 4 или для лабораторной работы № 5. В качестве получения дополнительного бонуса допускается реализация обеих лабораторных работ.

Все происходящие события должны документироваться: должен писаться лог, то есть журнал событий – сколько процессов запущено, когда, что передано каким клиентом/процессом на сервер и т.п. Должно быть предусмотрено сохранение лога в файл, а также возможность просмотра лога предыдущей запущенной сессии (допустимо использование для этого разных файлов).

Ограничений по языку программирования, технологиям, операционной системе нет.

6.3 Контрольные вопросы

1. Дайте определение интерфейса и приведите примеры.
2. Что такое GUI?
3. Как расшифровывается аббревиатура WIMP?
4. Расскажите о выводе графики на экран, что такое GDI?
5. Приведите примеры IDE для разработки программ с GUI для операционных систем семейства Windows и Linux.

Заключение

В настоящем издании предлагается ряд заданий для самостоятельного выполнения лабораторных работ по курсу «Операционные системы». Предлагаемые задания допускают ряд разнообразных решений, отличающихся по сложности реализации, что позволяет выставлять дифференцированную оценку выполненным работам.

Каждая лабораторная работа соответствует одной главе издания, и их уровень сложности возрастает с увеличением номера лабораторной работы. Для лабораторных работ приведены краткие теоретические сведения, описание деталей реализации. Для проверки понимания основных понятий и свойств, для каждой лабораторной работы сформулированы контрольные вопросы.

Основной целью настоящих лабораторных работ является обучение основам разработки программ для операционных систем семейств Windows и Linux.

По завершении курса студенты должны знать характеристики, принципы построения и организации операционных систем, базовые методы и алгоритмы, используемые различными подсистемами ОС, понятия процесса и потока, их свойства и операции над ними, механизмы межпроцессного взаимодействия, а также уметь разрабатывать программы с учетом возможностей и особенностей целевой операционной системы.

Литература

1. Махов, А. Е. Большой Российский энциклопедический словарь/ А. Е. Махов, Л. И. Петровская, В. М. Смолкин – М.: БРЭ, 2006. – 1887 с.

2. Кориков, А. М. Теория систем и системный анализ / А. М. Кориков, С. Н. Павлов // Учебное пособие. Томский государственный университет систем управления и радиоэлектроники, Томск. 2008. – 264 с.

3. Таненбаум, Э. С. Современные операционные системы / Э. С. Таненбаум, Б. Херберт – СПб.: Питер, 2021. – 1120 с.

4. Walden, D. 50th Anniversary of MIT's Compatible Time-Sharing System. / D. Walden // IEEE Annals of the History of Computing. – 2011. – Vol. 33, № 4. P. 84–85.

5. Дейтел, Х. М. Операционные системы. Часть 1. Основы и принципы / Х. М. Дейтел, П. Д. Дейтел, Д. Р. Чофнес – М.: Бином-Пресс, 2011. – 1024 с.

6. Statcounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share [сайт]: URL: https://gs.statcounter.com/, (дата обращение 09.11.2023).

7. Столлингс, В. Операционные системы: внутренняя структура и принципы проектирования / В. Столлингс – СПб.: ООО "Диалектика", 2020. – 1264 с.

8. Краткая история Linux – База знаний Timeweb Community [сайт]: URL: https://timeweb.com/ru/community/articles/kratkaya-istoriya-linux-1/ (дата обращения: 09.11.2023).

9. Kadir, A. A. Virtual machine tools and virtual machining - a technological review / A. A. Kadir, X. Xu, E. Hämmerle // Robotics and computer-integrated manufacturing. – 2011. – Vol. 27, № 3. – P. 494–508.

10. VMware Workstation Player | VMware [сайт]: URL: https://www.vmware.com/ru/products/workstation-player.html (дата обращения: 09.11.2023).

11. Командная строка | Русскоязычная документация по Ubuntu [сайт]: URL: https://help.ubuntu.ru/wiki/командная_строка (дата обращения: 09.11.2023).

12. Знакомство с компилятором GCC [сайт]: URL: https://studfile.net/preview/4153566/ (дата обращения: 09.11.2023).

13. Мэтью, Н. Основы программирования в Linux / Н. Мэтью, Р. Стоунс. – СПб.: БХВ-Петербург, 2009. – 896 с.

14. Как установить Midnight Commander (mc) в Ubuntu 18.04 [сайт]: URL: https://losst.ru/kak-ustanovit-mc-v-ubuntu-18-04 (дата обращения: 09.11.2023).

15. Уорд, Б. Внутреннее устройство Linux / Б. Уорд – СПб.: Питер, 2016. – 384 с.

16. Лав, Р. Linux. Системное программирование / Р. Лав – СПб.: Питер, 2014. – 448 с.

17. Кофлер, М. Linux. Установка, настройка, администрирование / М. Кофлер – СПб.: Питер, 2014. – 768 с.

18. Робачевский, А. Операционная система UNIX / А. Робачевский, С. Немнюгин, О. Стесик – СПб.: БХВ-Петербург, 2015. – 656 с.

19. Права доступа к файлам в Linux [сайт]: URL: https://losst.ru/prava-dostupa-k-fajlam-v-linux (дата обращения: 09.11.2022).

20. Терминал Linux. Права доступа к каталогам и файлам в Linux, команды chmod и chown [сайт]: URL: https://linuxrussia.com/terminal-chmod-chown.html (дата обращения: 09.11.2023).

21. Команда chmod в Linux [сайт]: URL: https://pingvinus.ru/note/chmod (дата обращения: 09.11.2023).

22. Соответствие консольных команд Windows и Linux [сайт]: URL: https://white55.ru/cmd-sh.html (дата обращения: 09.11.2023).

23. Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская – СПб.: Питер, 2003. – 461 с.

24. Шилдт, Г. C++: базовый курс / Г. Шилдт. – М.: Вильямс, 2012. – 624 с.

25. Абрамовиц М., Стиган И. Справочник по специальным функциям с формулами, графиками и таблицами / М. Абрамовиц, И. Стиган. – М.: Наука, 1979. – 832 с.

26. Сборник задач и упражнений по высшей математике: в 2 ч. Ч. 1 / А. В. Конюх, В. В. Косьянчук, С. В. Майоровская [и др] – Минск : БГЭУ, 2014. – 299 с.

27. Гунько, А. Программирование (в среде Windows) / А. В. Гунько // Учебное пособие. Новосибирск: Изд-во НГТУ, 2019 – 155 с.

28. Microsoft Learn: приобретение навыков, которые открывают путь к карьерному росту [сайт]: URL: https://learn.microsoft.com/ru-ru/ (дата обращения: 09.11.2023).

29. Semaphores are Surprisingly Versatile [сайт]: URL: https://preshing.com/20150316/semaphores-are-surprisingly-versatile/ (дата обращения: 09.11.2023).

30. Такие удивительные семафоры [Электронный ресурс] // Хабр. URL: https://habr.com/ru/post/261273/ (дата обращения: 20.01.2022)

31. Интерфейс. Основы проектирования взаимодействия / А. Купер, Р. М. Рейманн, Д. Кронин, К. Носсел; СПб: Питер, 2017. – 720 с.

Учебное издание

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебно-методическое пособие

Составители: Савельев Дмитрий Андреевич

Самарский университет
446086 Самара, Московское шоссе, 34