

Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Д.А. Савельев

**ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ
«ОПЕРАЦИОННЫЕ СИСТЕМЫ»**

Самара 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.
КОРОЛЕВА»

Д.А. Савельев

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве методических указаний по курсу «Операционные системы».

Самара
Издательство Самарского университета
2022

Оглавление

Введение	5
1 Знакомство с операционной системой Linux на примере дистрибутива Ubuntu	9
1.1 Знакомство с Linux и Ubuntu	9
1.2 Виртуальная машина VMware Workstation	10
1.3 Использование командной строки	14
1.4 Задание на лабораторную работу № 1	19
1.5 Контрольные вопросы	22
2 Операционная система Ubuntu: системные вызовы и работа с файлами	23
2.1 Особенности использования системных вызовов	23
2.2 Права доступа к файлам в Ubuntu	24
2.3 Задание на лабораторную работу № 2	27
2.4 Контрольные вопросы	29
3 Межпроцессное взаимодействие в операционной системе Ubuntu	30
3.1 Особенности межпроцессного взаимодействия	30
3.2 Задание на лабораторную работу № 3	32
3.3 Контрольные вопросы	36
4 Межпроцессное взаимодействие на основе каналов и сокетов в операционной системе Windows	37
4.1 Сравнение семейств операционных систем Windows и Linux	37
4.2 Межпроцессное взаимодействие на примере каналов	39
4.3 Межпроцессное взаимодействие на примере сокетов	45
4.3 Задание на лабораторную работу № 4	51

4.4 Контрольные вопросы.....	54
5 Использование семафоров и мьютексов для клиент-серверного взаимодействия.....	55
5.2 Синхронизация процессов и потоков	55
5.2 Задание на лабораторную работу № 5.....	62
5.3 Контрольные вопросы.....	72
6 Применение GUI (graphical user interface) для клиент-серверного взаимодействия.....	73
6.3 Графический интерфейс пользователя.....	73
6.2 Задание на лабораторную работу № 6.....	74
6.3 Контрольные вопросы.....	74
Заключение	75
Список литературы	76

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 прикладным программистам (и прикладным программам, естественно) 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]. И на ее основе была предложена разработка универсальной общей компьютерной системы, которая должна была поддерживать одновременную работу сотен пользователей в режиме разделения времени. За основу была взята система распределения электроэнергии. Когда вам нужна электроэнергия, вы просто вставляете штепсель в розетку и получаете столько энергии, сколько вам нужно. Проектировщики этой системы, известной как MULTICS (MULTiplexed Information and Computing Service – мультиплексная информационная и вычислительная служба), представляли себе одну огромную вычислительную машину, воспользоваться услугами которой мог любой проживающий в окрестностях Бостона человек. Коммерчески была не очень удачна, но было реализовано много идей, легших в основу Unix [3, 5]

Чтобы появилась возможность писать программы, работающие в любой UNIX-системе, Институт инженеров по электротехнике и электронике (IEEE) разработал стандарт системы UNIX, названный POSIX, который в настоящее время поддерживается большинством версий UNIX. Стандарт POSIX 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. Перед каждой лабораторной работой приведены некоторые сопутствующие теоретические сведения, список заданий, примеры реализации, а также контрольные вопросы. Уровень сложности лабораторных работ возрастает с увеличением их номера от элементарных операций и до межпроцессного взаимодействия и GUI.

Требования к лабораторным работам

Лабораторная работа – один из видов практических занятий, целью которых является углубление и закрепление теоретических знаний, а также развитие навыков программирования.

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. Успешная сдача всех лабораторных работ является необходимым условием для допуска к экзамену. В курсе приняты следующие критерии оценки заданий к лабораторным работам:

5 баллов («отлично») – обучающийся показывает прочные знания основных процессов изучаемой предметной области; отличается глубиной и полнотой раскрытия темы, владением терминологического аппарата, умением объяснять сущность явлений, процессов, событий; способен самостоятельно делать выводы и обобщения, давать аргументированные ответы, приводить примеры; наблюдается логичность и последовательность в ответах; обучающийся дает качественные и полные ответы на вопросы.

4 балла («хорошо») – обучающийся показывает прочные знания основных процессов изучаемой предметной области; отличается глубиной и полнотой раскрытия темы, владением терминологического аппарата, умением объяснять сущность явлений, процессов, событий; способен самостоятельно делать выводы и обобщения, давать аргументированные ответы, приводить примеры; наблюдается логичность и последовательность в ответах. Однако допускается одна – две неточности в ответе.

3 балла («удовлетворительно») – обучающийся показывает основные знания процессов изучаемой предметной области, его ответ отличается недостаточной глубиной и полнотой раскрытия темы, слабо сформированы навыки анализа явлений, процессов; недостаточное умение давать аргументированные ответы и приводить примеры; слабо наблюдается логичность и последовательность в ответах. Допускается несколько ошибок в содержании ответа.

2 балла («неудовлетворительно») – обучающийся демонстрирует незнание процессов изучаемой предметной области, отличается неглубоким раскрытием темы, незнанием основных вопросов теории, несформированными навыками анализа явлений, процессов, неумением давать аргументированные ответы, отсутствием логичности и последовательности. Допускаются серьезные ошибки в содержании ответа.

1 Знакомство с операционной системой Linux на примере дистрибутива Ubuntu

1.1 Знакомство с Linux и Ubuntu

Linux – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU [7].

История Linux началась в 1991 году и тесно связана с именем финского программиста Линуса Торвальдса. Именно тогда он стал разрабатывать ядро операционной системы и выложил свои разработки на сервере в открытый доступ. Далее другие разработчики поддержали его проект - общими усилиями на свет появилась полноценная операционная система [8].

На Linux значительно повлияла система Unix, что видно и по названию в том числе. Изначально проект назывался Freax: от слов “free” (бесплатный) и “freak” (странный). В дальнейшем название было изменено на гибрид имени создателя (Линус) и Unix.

Эмблема Linux – пингвин (Tux), нарисованный в 1996 году программистом и дизайнером Ларри Юингом. Теперь эта эмблема является символом не только операционных систем семейства Linux, но и свободного программного обеспечения в целом. Первая официальная версия Linux 1.0 вышла в 1994 году.

С самого начала операционные системы семейства Linux распространялись как свободное программное обеспечение с лицензией GPL. Данная лицензия означает возможность для любого пользователя увидеть и при необходимости доработать исходный код операционной системы. Единственное условие - измененный, модифицированный код должен быть так же доступен всем и распространяться по лицензии GPL, что дает возможность использовать код и в то же время не иметь проблем из-за авторских прав.

Сейчас благодаря своей гибкости Linux используется на множестве разных устройств, начиная от компьютеров и заканчивая серверами и мобильными устройствами.

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 Виртуальная машина VMware Workstation

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 виртуальных машин.

Можно сказать, что виртуальная машина (virtual machine) является эмуляцией компьютерной системы. Это компьютерная программа, которая представляет имитацию оборудования для операционной системы, работающей как внутрисистемный процесс. Фактически, можно говорить о том, что есть программная и/или аппаратная система, которая эмулирует аппаратное обеспечение некоторой платформы (гостевая (target) платформа) и исполняет программы для такой гостевой платформы на платформе-хозяине (host-платформе) или виртуализирует некоторую платформу и создает на ней среды, изолирующие друг от друга программы и операционные системы [9].

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 дистрибутива Ubuntu. Для запуска нужно найти на рабочем столе иконку с надписью VMware Player (рисунок 1.1)

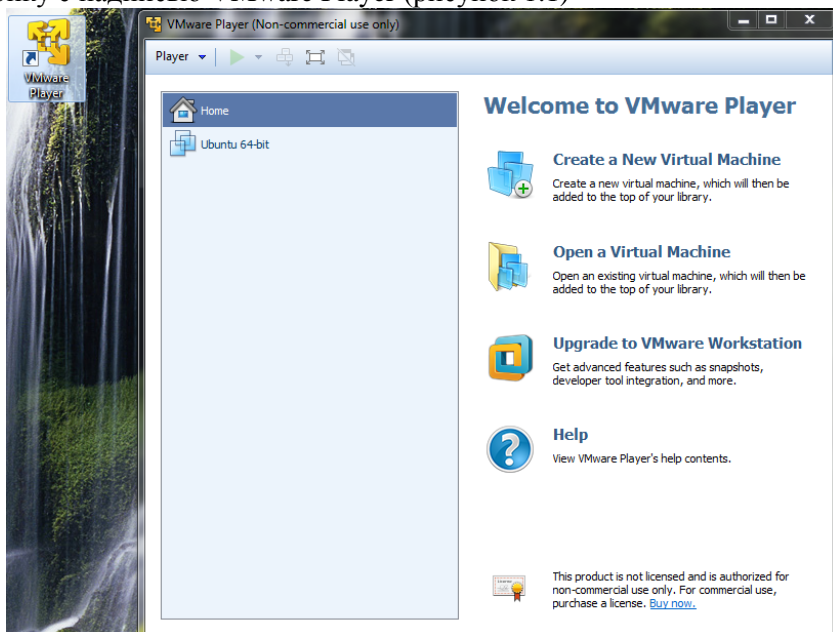


Рисунок 1.1 – Запуск VMware Player

Далее выбрать дистрибутив Ubuntu и запустить, нажав Play virtual machine (рисунок 1.2).

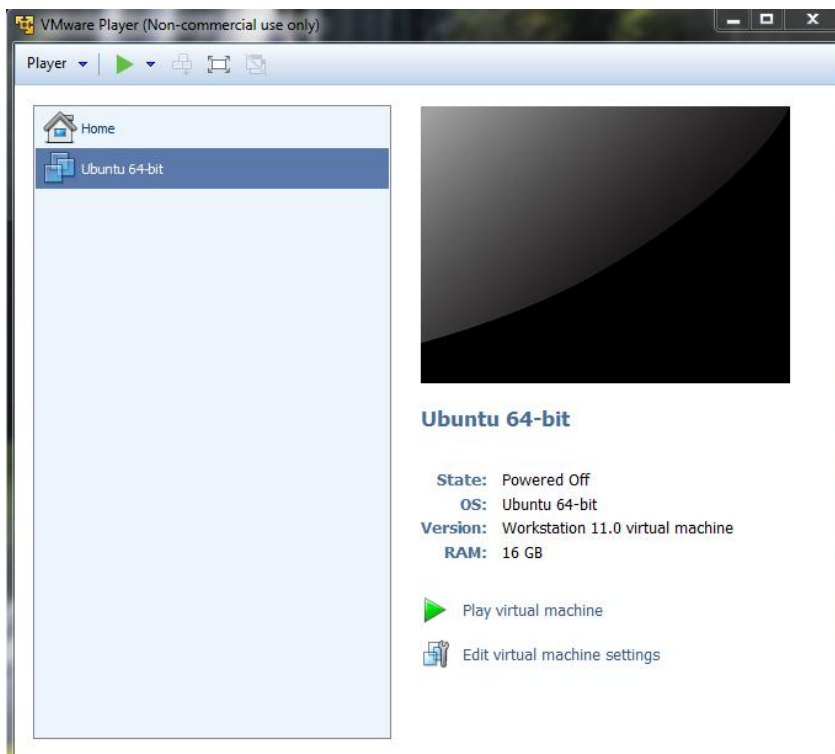


Рисунок 1.2 – Запуск дистрибутива Ubuntu

Далее нужно ввести пароль (рисунок 1.3). После чего загрузится операционная система Ubuntu (рисунок 1.4)

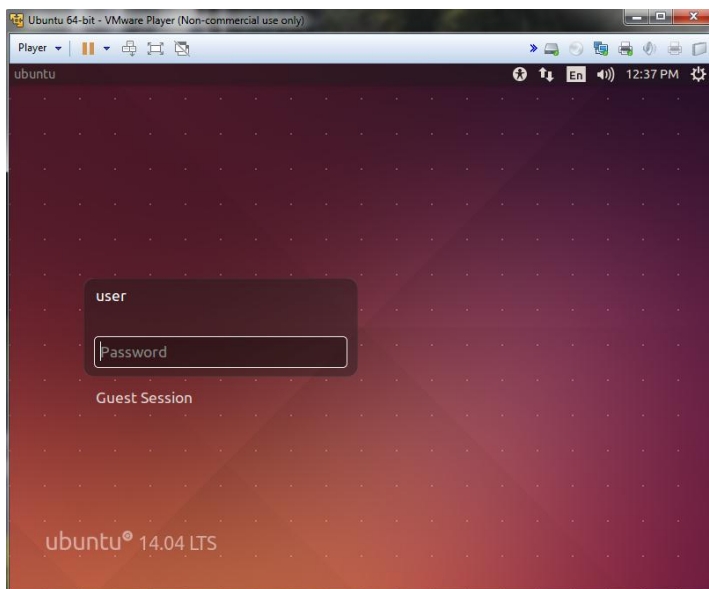


Рисунок 1.3 – Ввод пароля

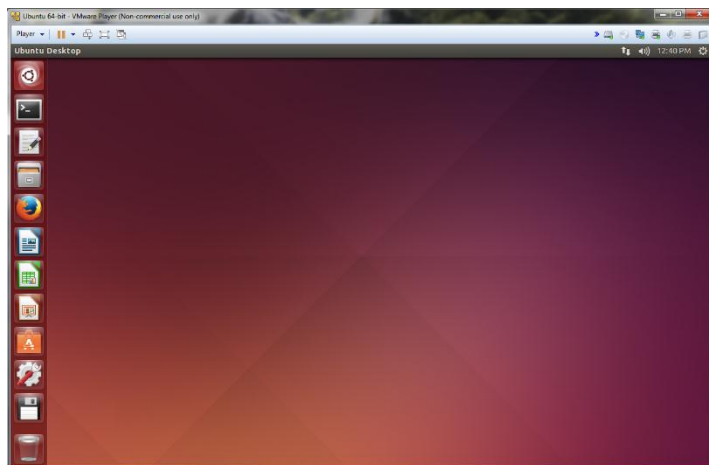


Рисунок 1.4 – Рабочий стол Ubuntu

1.3 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 следующим ссылкам [11].

Интерфейс командной строки - управление программами с помощью команд. Команды состоят из букв, цифр, 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 (рисунок 1.5).

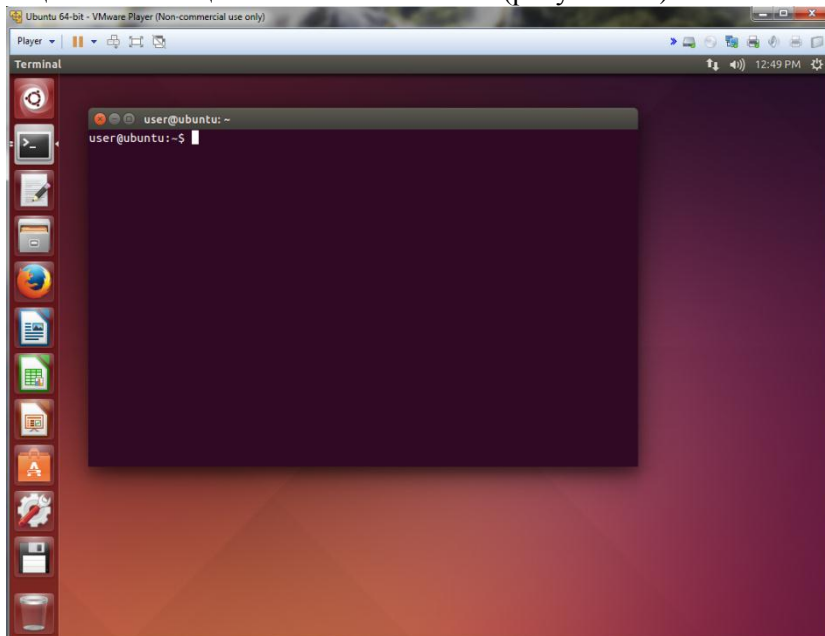


Рисунок 1.5 – Запуск терминала

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

Таблица 1.1 – Файловые 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 директорию
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 директорию
ln file1 lnk1	создать «жесткую» (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 будет C/C++. Использование дополнительных языков программирования обговаривается отдельно с преподавателем.

Существует много компиляторов языка C, совместимые с различными стандартами. Одним из наиболее применимым 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
```

При сборке программы из нескольких модулей компилятору можно подавать на вход несколько исходных файлов или файлов объектного кода, например,

```
gcc -o program main.c module1.o module2.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

отсутствует 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 (рисунок 1.6).

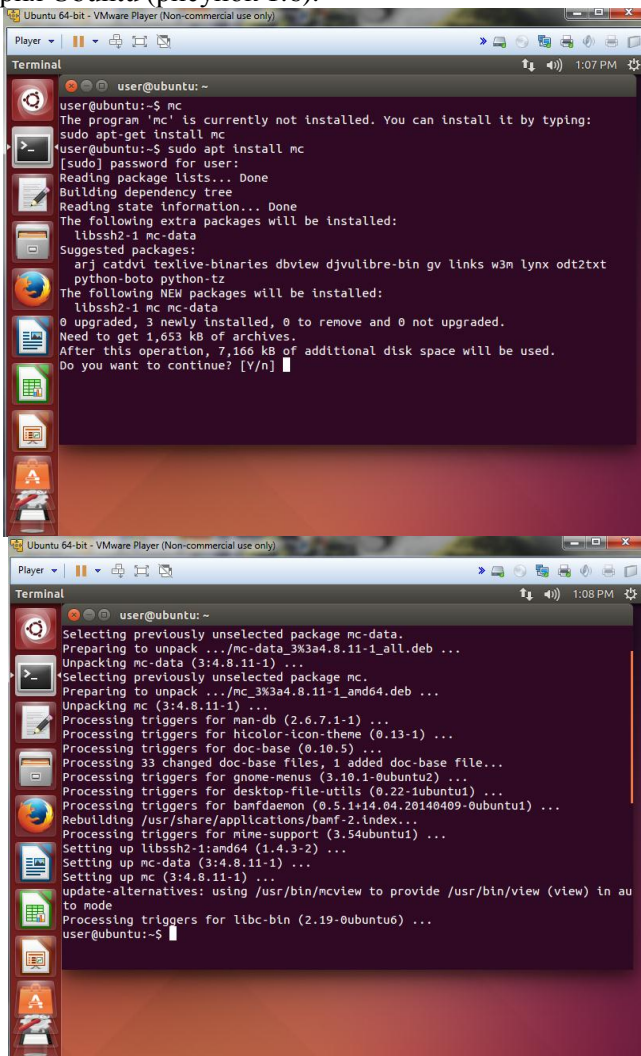


Рисунок 1.6 – Установка Midnight Commander

Вызвать Midnight Commander можно, набрав в терминале команду `mc`. Внешний вид показан на рисунке 1.7.

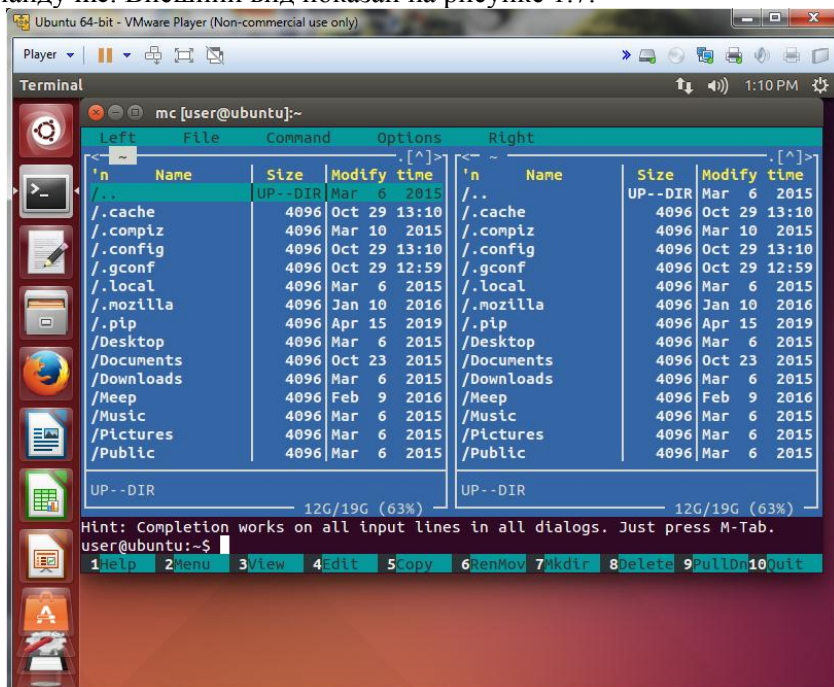


Рисунок 1.7 – Внешний вид Midnight Commander

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
```

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]

1.4 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 1

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

Вариант 2

Перевод из десятичной системы счисления в двоичную систему счисления.

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, введенных пользователем.

Вариант 22

Написать программу суммирования только нецелых чисел из набора 10 чисел, введенных 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.

1.5 Контрольные вопросы

1. Приведите определение свойства системы и примеры свойств.
2. Дайте определение термина «Операционная система».
3. Какие функции операционных систем вы можете привести?
4. Что такое мультиплексирование?
5. Дайте определение Linux.
6. Что такое виртуальная машина?
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?
10. Как запустить в командной строке исполняемый файл?

2 Операционная система Ubuntu: системные вызовы и работа с файлами

2.1 Особенности использования системных вызовов

Под системными вызовами мы будем понимать 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,
- `ls -l 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.
- и т.д.

Некоторые команды для работы с файлами нами разбирались в разделе 1.3. 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.

При использовании команда `pwd` («print working directory») можно узнать, в какой директории вы находитесь в данный момент.

2.2 Права доступа к файлам в 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.

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

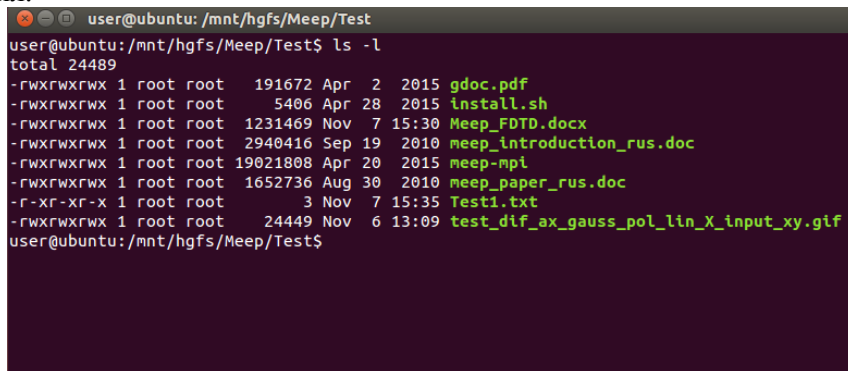
Под группой понимается любая группа пользователей, существующая в системе и привязанная к файлу. Обычно это группа владельца, хотя для файла можно назначить и другую группу.

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 2.1):

```
ls -l
```


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



```
user@ubuntu: /mnt/hgfs/Meep/Test
user@ubuntu:/mnt/hgfs/Meep/Test$ ls -l
total 24489
-rwxrwxrwx 1 root root 191672 Apr  2  2015 gdoc.pdf
-rwxrwxrwx 1 root root 5406 Apr 28  2015 install.sh
-rwxrwxrwx 1 root root 1231469 Nov  7 15:30 Meep_FDTD.docx
-rwxrwxrwx 1 root root 2940416 Sep 19  2010 meep_introduction_rus.doc
-rwxrwxrwx 1 root root 19021808 Apr 20  2015 meep-mpi
-rwxrwxrwx 1 root root 1652736 Aug 30  2010 meep_paper_rus.doc
-r-xr-xr-x 1 root root 3 Nov  7 15:35 Test1.txt
-rwxrwxrwx 1 root root 24449 Nov  6 13:09 test_dif_ax_gauss_pol_lin_X_input_xy.gif
user@ubuntu:/mnt/hgfs/Meep/Test$
```

Рисунок 2.1 – Пример использования команды `ls -l`

Как можно заметить из рисунка 8, 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.

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 – доменное гнездо (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).

```

user@ubuntu: /mnt/hgfs/Meep/Test
user@ubuntu: /mnt/hgfs/Meep/Test$ chmod --help
Usage: chmod [OPTION]... MODE[,MODE]... FILE...
       or: chmod [OPTION]... OCTAL-MODE FILE...
       or: chmod [OPTION]... --reference=RFILE FILE...
Change the mode of each FILE to MODE.
With --reference, change the mode of each FILE to that of RFILE.

  -c, --changes           like verbose but report only when a change is made
  -f, --silent, --quiet   suppress most error messages
  -V, --verbose           output a diagnostic for every file processed
  --no-preserve-root       do not treat '/' specially (the default)
  --preserve-root         fail to operate recursively on '/'
  --reference=RFILE       use RFILE's mode instead of MODE values
  -R, --recursive        change files and directories recursively
  --help                 display this help and exit
  --version               output version information and exit

Each MODE is of the form '[ugoa]*([+=[rwxXst]*|[ugo]))+|[-+=[0-7]+'.

Report chmod bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'chmod invocation'
user@ubuntu: /mnt/hgfs/Meep/Test$

```

Рисунок 2.2 – Синтаксис команды chmod

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

```

user@ubuntu: /mnt/hgfs/Meep/Test
user@ubuntu: /mnt/hgfs/Meep/Test$ chmod ugo+rwX Test1.txt
user@ubuntu: /mnt/hgfs/Meep/Test$ ls -l
total 24489
-rwxrwxrwx 1 root root 191672 Apr 2 2015 gdoc.pdf
-rwxrwxrwx 1 root root 5406 Apr 28 2015 install.sh
-rwxrwxrwx 1 root root 1231469 Nov 7 15:30 Meep_F0TD.docx
-rwxrwxrwx 1 root root 2940416 Sep 19 2010 meep_introduction_rus.doc
-rwxrwxrwx 1 root root 19021808 Apr 20 2015 meep-npi
-rwxrwxrwx 1 root root 1652736 Aug 30 2010 meep_paper_rus.doc
-rwxrwxrwx 1 root root 3 Nov 7 15:35 Test1.txt
-rwxrwxrwx 1 root root 24449 Nov 6 13:09 test_dif_ax_gauss_pol_lin_X_input_xy.gif
user@ubuntu: /mnt/hgfs/Meep/Test$

```

Рисунок 2.3 – Применение команды chmod

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

Также существует цифровой формат записи [17, 21], значения приведены в таблице 2.1

Таблица 2.1 – 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

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

```
chmod 777 Test1.txt
```

2.3 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);`

2.4 Контрольные вопросы

1. По каким категориям можно сгруппировать системные вызовы?
2. Назовите 5 любых команд для управления процессами.
3. Назовите 5 любых команд для управления файлами.
4. Продемонстрируйте работу в терминале 5 любых команд для управления файлами.
5. Какие параметры доступа к файлам вам известны?
6. Перечислите категории пользователей, для которых можно устанавливать различные сочетания прав доступа.
7. Какой командой можно посмотреть права доступа к файлам в Linux?
8. Как обозначаются разные типы файлов в Linux?
9. Какой командой можно воспользоваться для изменения прав доступа к файлу?
10. Расскажите о цифровом формате записи прав для команды `chmod`.

3 Межпроцессное взаимодействие в операционной системе Ubuntu

3.1 Особенности межпроцессного взаимодействия

Процесс - совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие.

Фактически, **программа** – совокупность инструкций, **процесс** – непосредственное выполнение этих инструкций [3].

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.

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**.

Общая схема создания процесса [3,7]:

1. Инициализация системы.
2. Выполнение работающим процессом системного вызова, предназначенного для создания процесса.
3. Запрос пользователя на создание нового процесса.
4. Инициация пакетного задания.

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]. Обычно после

этого дочерний процесс изменяет образ памяти и запускает новую программу, выполняя системный вызов *execve()* или ему подобный.

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 (например: когда они борются за один ресурс).
- Согласование действий процессов (например: когда один процесс предоставляет данные, а другой их выводит на печать. Если согласованности не будет, то второй процесс может начать печать раньше, чем поступят 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].

3.2 Задание на лабораторную работу № 3

Необходимо написать программу, которая разделяется на две части. Первая часть – клиентская – взаимодействует с пользователем, 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 .crrp 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. В идеале, также входные значения должны загружаться из файлов, результат записываться в файл, но это для дополнительного бонуса.

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

Решение дифференциального уравнения методом Рунге-Кутты.

Вариант 11

Вычисление полиномов Эрмита.

Вариант 12

Вычисление многочленов Лагеррра (первые 6).

Вариант 13

Вычисление интегралов Френеля.

Вариант 14

Метод Гаусса для решения систем уравнений.

Вариант 15

Матричный метод решения систем линейных алгебраических уравнений.

Вариант 16

Метод Крамера решения систем линейных алгебраических уравнений.

Вариант 17

Метод прогонки решения систем линейных алгебраических уравнений.

Вариант 18

Реализация метода левых прямоугольников для функции одной переменной

Вариант 19

Реализация метода прямоугольников (средних) для функции одной переменной

Вариант 20

Реализация метода трапеций для функции одной переменной.

Вариант 21

Написать функцию вычисления приближенного значения:

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

Вариант 22

Написать функцию вычисления приближенного значения:

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

Вариант 23

Написать функцию вычисления приближенного значения:

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

Вариант 24

Написать функцию вычисления приближенного значения:

$$f(x) = \sum_{n=0}^{1000} \frac{\sin x + \cos n}{x + n!}$$

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:

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

Вариант 28

Написать функцию вычисления приближенного значения:

$$f(x, y) = \sum_{n=0}^{\infty} \frac{x^n + y^n}{n + \lg xy}$$

3.3 Контрольные вопросы

1. Дайте определение процесса.
2. Что содержит адресное пространство?
3. Какие категории процессов вам известны?
4. Опишите общую схему создания процесса.
5. Какая функция используется для создания копии вызывающего процесса?
6. В каких случаях процессам требуется взаимодействовать друг с другом.
7. Дайте определение канала.
8. 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 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 правильнее 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 [<http://gs.statcounter.com>] Windows занимает более 70% 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 3.1.

Таблица 3.1 – Соответствие консольных команд Windows и Linux для работы с файлами и каталогами

Описание	Команда CMD 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
Вывод на экран содержимого файла	TYPE, MORE	cat, less, more
Сравнение содержимого	COMP, FC	cmp, diff, diff3,

двух файлов		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.

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>. 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 4.1. Некоторые флаги параметра dwPipeMode приведены в таблице 4.2.

С остальными можно ознакомиться в [28].

Таблица 4.1 – 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 4.2 – Флаги параметра dwPipeMode

Имя флага	Значение
PIPE_TYPE_BYTE	Режим работы канала, ориентированный на передачу байт
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 показан именованных каналов показан в листинге 1.

Листинг 1. 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_PARAMETER.

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 [29] функцию ConnectNamedPipe () (таблица 4.3).

Таблица 4.3 – 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`) [30], 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`) [31], 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` [32]. В таблице 4.4 приведены ее параметры.

Таблица 4.4 – Параметры функции `CreateFile`

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

hTemplateFile	Дополнительные атрибуты, обычно задается равным NULL
---------------	--

Для передачи используются функции WriteFile () [33] и ReadFile () [34] с атрибутами, описанными в таблицах 4.5-4.6. Соответственно, если функции завершаются успешно, возвращаемое значение отлично от нуля (True).

Таблица 4.5 – 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 4.6 – Параметры функции ReadFile

Имя параметра	Значение
hFile	Дескриптор канала
lpBuffer	Указатель на буфер, который получает данные
nNumberOfBytesToRead	Размер данных в байтах, которые будут прочитаны
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 2 приведен пример использования вышеописанных функций.

Листинг 2. 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)** [35].

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 сокет.
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) [36], 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 3 приведен пример использования данной функции.

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 выставить значение по умолчанию (0 или NULL), чтобы операционная система выбрала протокол. При неудаче функция вернет нулевое значение.

Листинг 3. Пример использования функции WSASocket (WORD wVersionRequired, LPWSADATA lpWSADATA)

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

Далее нам необходимо установить соединение с удалённым узлом с помощью функции [38]:

```
int WSASocket(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 [39]:

```
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 [40]:

```
int WSAAPI listen (SOCKET s, int backlog),
```

где первый аргумент – этот тот сокет, который нам нужно перевести в режим ожидания, второй аргумент – максимальное количество сообщений в очереди. При удачном завершении функция возвращает нулевое значение.

В листингах 4-5 показан пример использования вышеописанных функций.

Листинг 4. 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 5. Пример использования функции connect на стороне клиента

```
//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 [41]:

`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 [42, 43]:

`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 4, продемонстрируем использование функций `accept`, `send` и `recv` в листингах 6 – 7.

Далее, когда вся информация передана и работа завершена требуется закрыть сокет и разорвать соединение. Для этого используется следующая функция [44]:

`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 8.

Листинг 6. 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 4
    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 7. Пример использования функции `recv` на стороне клиента (продолжение листинга 5)

```
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);
    }
}
```

Листинг 8. 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;
}
```

4.3 Задание на лабораторную работу № 4

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

Вариант 1-2.

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.

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. Также 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 эхо-печати (т.е. сразу выводится на экран) получать все введенные данные с клиента. Предусмотреть также случай стирания текста через `backspace`. Клиент же должен отправлять данные о каждом введенном символе, причём эхопечать должна поддерживать русский язык. Если не поддерживает русский язык, задание может быть зачтено, но по минимальному баллу. Минимум 1 клиент. Если студент хочет сделать версию 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 запускаться 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 одного клиента, но желательно хотя бы два.

Вариант 17-18.

Существует 4 типа скобок: {}, (), <>, []. С клиента на сервер передается скобочная последовательность, а на сервере производится 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. От одного клиента, если клиентов несколько – требуется также пересылать идентификатор клиента, чтобы на сервере выводилось, от какого клиента получено сообщение.

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.

4.4 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. Дайте определение канала.
6. Какой функцией можно создать именованный канал?
7. Какими функциями пользуются для передачи информации при использовании именованных каналов?
8. Что такое сокеты?
9. Опишите схему взаимодействия двух процессов (условного сервера и условного клиента) на основе сокетов.
10. Какой командой используются для проверки подключения к сокету?
11. какие команды используются для передачи данных в сокетах?
12. Что необходимо сделать, когда сокет уже не нужен и программа завершается?

5 Использование семафоров и мьютексов для клиент-серверного взаимодействия

5.2 Синхронизация процессов и потоков

В 1965 году Эдсгер Дейкстра (автор работ в области применения математической логики при разработке компьютерных программ, один из авторов концепции структурного программирования, создатель алгоритма Дейкстры) предложил использовать целочисленную переменную для подсчета количества активизаций, отложенных на будущее [45]. Этот новый вид переменной, он предложил назвать **семафор** (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):

- up увеличит значение семафора на 1 или разблокирует процесс, находящийся в ожидании.
- down уменьшает значение семафора на 1 или блокирует процесс, если семафор = 0.

Проверка значения, его изменение, и, при необходимости, приостановка процесса осуществляются как единое и неделимое **атомарное** действие. Тем самым гарантируется, что с началом семафорной операции никакой другой процесс не может получить доступ к семафору до тех пор, пока операция не будет завершена или заблокирована. Необходимость этого очевидна. **Атомарность** — необходимое условие для решения проблем синхронизации и исключения состязательных ситуаций.

Достаточно интересно принцип работы семафоров показан в публикации в англоязычном оригинале [46] и русскоязычном переводе [47].

Возможны ситуации, когда требуется всего два состояния рассматриваемой выше переменной — заблокированном и незаблокированном. Такой аналог одноместного семафора называется **мьютекс**.

Мьютекс (mutex, от mutual exclusion – взаимное исключение) – это упрощенная версия семафора, совместно используемая переменная, которая может находиться в одном из двух состояний: заблокированном или незаблокированном [48]

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 [48].

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 [49].

Следующий пример показывает создание семафора с максимальным числом три [50]. Для блокировки семафора используются пять потоков (листинг 9).

Вообще говоря, каждому процессу соответствует адресное пространство и одиночный **поток** исполняемых команд. По сути, если бы у каждого потока были собственные адресное пространство, открытые файлы, необработанные аварийные сигналы и т. д., то он был бы отдельным процессом. Соответственно, у одного процесса может быть много потоков и потоки процесса разделяют общие ресурсы процесса.

Преимущества потоков перед процессами:

- Упрощение программы в некоторых случаях, за счет использования общего адресного пространства.
- Быстрота создания потока, по сравнению с процессом, до 100 раз.
- Повышение производительности самой программы, т.к. есть возможность одновременно выполнять вычисления на процессоре и операцию ввода/вывода.

Листинг 9. 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());
}
```

Для общезыковой среды выполнения (CLR) для использования в качестве семафора для событий ожидания можно создать объект `IHostSemaphore` с использованием функции `CreateSemaphore` [51], или использовать функцию `Win32` с тем же именем. Рассмотрим функцию `CreateSemaphoreA` из `Win32 API (Win32)` [52]:

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

Здесь `lpSemaphoreAttributes` – это атрибуты защиты, `InitialCount` – 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 10.

Листинг 10. 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 по следующей ссылке [53].

Далее рассмотрим уже знакомые нам мьютексы (`Mutex`) в операционной системе `Windows` [54]. В листинге 11 показан пример использования `Mutex` для синхронизации доступа к защищенному ресурсу [55].

Листинг 11. 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 **события** (event). Объекты-события используются для уведомления ожидающих потоков о наступлении какого-либо события (операции), бывают двух типов: со сбросом вручную (manual - reset events) и с автосбросом (auto - reset events) [27, 56].

Event в Win32 создается с помощью функции CreateEventA [57]:

```
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 вручную(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 [58] и кратко в листинге 12.

Листинг 12. 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);
    }
}
}

```

5.2 Задание на лабораторную работу № 5

Использовать семафоры (мьютексы) для синхронизации работы процессов (поток) 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. Также желательно предусмотреть для всех заданий событие, при котором соответствующий процесс завершает свою работу, если это возможно.

Список вариантов

Вариант 1

Холодная война. Шпионские игры. 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

Шпионские игры-2. 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 сообщение.

Передача остальных сообщений от других процессов должна блокироваться.

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 данных 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 №Х 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 - сложение матриц. Матрицы вводятся на клиентах – по одной на каждый клиент. На сервере происходит вычисление матриц. Одновременно можно складывать только по две матрицы, то есть работа начинается, если есть четное число процессов клиентов.

Вариант 8

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 9

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 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 многопроцессное 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 X. «Здравствуй, господин цирюльник» ... «Всего доброго, приходите к нам еще!» При желании, 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, поочередно отнимают от числа от 1 до 3. Проигрывает тот, кто первый обнулит число или сделает его отрицательным. Ave, Caesar!

Вариант 13

Задача о Винни-Пухе и Пятачке. В одном лесу живут очень трудолюбивые, а главное, доброжелательные пчелы, которые уважают медведей. Пусть их будет М. В этом же лесу живут Винни-Пух и его друг Пятачок, у которых всего 1 горшок для меда. Вместимость этого

горшка они считают в глотках, у Винни-Пуха глоток в два раз больше, чем у Пятачка. Сначала горшок пустой. Пока горшок не наполнится, все, кроме пчел спят. Как только горшок заполняется, Винни-Пух и Пятачок просыпаются и по очереди начинают пить мед. Пятачок наедается за 3 глотка, а Винни-Пух за 5. Как только они наелись – они засыпают, причем сон у них разной продолжительности (Винни-Пух спит в 3 раза дольше). Каждая пчела многократно собирает по одному глотку меда и кладет его в горшок. Пчела, которая приносит последнюю порцию меда, будит Винни-Пуха или Пятачка. В первый раз пробуждение у них двоих сразу. Смоделировать поведение пчел, медведя и его друга. Допустимо использовать в рамках 1 процесса пчелы ряд потоков, моделирующих отдельную пчелу, также допустимо, но не обязательно использовать в рамках одного процесса Винни-Пуха и Пятачка как отдельные потоки. Горшок с медом – отдельный сервер, есть из горшка может только 1 клиент.

Вариант 14

Задача о рыцарях-философах. Пять рыцарей короля Артура сидят возле круглого стола. Они проводят жизнь, чередуя приемы пищи и размышления о спасении страждущих. В центре стола находится большое блюдо еды (какая именно еда – несущественно, у рыцарей подготовленный желудок). По взятому на себя обету, чтобы съесть порцию, рыцари должны использовать две вилки. К несчастью, после предыдущего нападения на замок короля, осталось только пять вилок. Между каждой парой рыцарей-философов лежит одна вилка, поэтому эти высококультурные люди договорились, что каждый будет пользоваться только теми вилками, которые лежат рядом с ним (слева и справа). Требуется промоделировать день из жизни рыцарей. Программа должна избегать фатальной ситуации, в которой все рыцари голодны, но ни один из них не может взять обе вилки (например, каждый из них держит по одной вилке, задумался и не хочет отдавать ее). Решение должно быть симметричным, то есть все процессы-рыцари должны выполнять один и тот же код. В качестве сервера выступает стол, к которому обращаются процессы-клиенты.

Вариант 15

Задача о курильщиках. Есть три типа процессов-курильщиков и один процесс-посредник. Курильщик скручивает сигары и курит их так часто, записано у него в ежедневнике. Чтобы скрутить сигару и

закурить, нужны табак, бумага и спички. У одного процесса-курильщика есть табак, у второго - бумага, а у третьего - спички. Это – ограниченный ресурс. У посредника есть все. Посредник кладет на стол по два разных случайных компонента. Тот процесс-курильщик, у которого есть третий компонент, забирает компоненты со стола и курит. Если компоненты никто не забрал за секунду, он забирает их. Посредник дожидается, пока курильщик закончит, затем процесс повторяется. Провести моделирование поведение курильщиков и посредника. Процессов-курильщиков может быть больше 3, но доступ к столу имеют только трое. После курения, процесс-курильщик «засыпает» на некоторое время, заданное пользователем. Частота курения также может задаваться пользователем.

Вариант 16

Обед племени. Племя из n первобытных людей ест вместе из большого горшка, который вмещает v кусков тушеного носорога. Когда процесс-человек хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе он будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает (считаем, что он напробовався во время приготовления еды, и есть не хочет). Считаем, что на сервере есть два потока, моделирующих работу повара и котла, а процессы-клиенты подключаются к нему во время еды. Одновременно есть может только один человек.

Вариант 17

В магазине комиксов работают два отдела, каждый отдел обслуживает один продавец. Покупатель, зайдя в магазин, делает покупки в произвольных отделах (база данных комиксов хранится на сервере, при постановке в очередь на клиента отсылается то, что он будет читать в очереди или купит). Если в выбранном отделе продавец не свободен, покупатель становится в очередь и погружается в чтение, пока продавец не освободится. Если продавец не освобождается за определенное время, покупатель берет другой комикс и снова становится в очередь и читает. Если и в этом случае не удалось произвести покупку – покупатель уходит. Смоделировать работу магазина. В качестве сервера – магазин с отдельными потоками-продавцами. Процессы-клиенты – это покупатели. На сервере требуется выводить информацию о том, где находится соответствующий покупатель, в каком отделе, что читает. На клиенте

– что именно было куплено. После завершения рабочего дня нужно подвести итог на сервере – сколько было клиентов и сколько и что они купили. Соответствующим образом требуется изменить базу данных.

Вариант 18

Задача о супермаркете ПяПерМаг. В супермаркете работают X кассиров, покупатели заходят в супермаркет, делают покупки и становятся в очередь к случайному кассиру. Изначально он один. Кассиру 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 X. 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 врач 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. 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 изготавливают 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 Монголии. В отеле 2 номера с ценой 1000 тугриков за ночь, 4 номеров с ценой 1600 тугриков за ночь и 1 номер с ценой 2500 тугриков за ночь. Клиент, который решил приехать на родину Чингисхана имеет определенные финансовые ресурсы и может снять номер, если у него хватает денег. Если номеров свободных нет, но он идет ночевать в степь. 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

Викторианский отель. Середина 19 века. Окрестности Лондона. В викторианском отеле существуют строгие правила – дам можно селить только с дамами, а джентльменов – только с джентльменами. В отеле 6 номеров, которые рассчитаны на одного человека и 5 номеров рассчитаны на двух человек. В отель приходят дамы и джентльмены. Если для очередного клиента не находится подходящего номера, он уходит искать ночлег в другое место. Считаем, что если в двухместном номере есть дама или джентльмен, они готовы потесниться и переночевать с представителем своего пола. Создать приложение, моделирующее работу викторианского 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.

5.3 Контрольные вопросы

1. Дайте определение семафора.
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 GUI (graphical user interface) для клиент-серверного взаимодействия

6.3 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 [59].

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

GUI был изобретен Дугласом Энгельбартом (Douglas Engelbart) и его исследовательской группой из Стэнфордского исследовательского института [3].

Если сосредоточиться на персональных компьютерах – в GUI есть четыре важных элемента, обозначаемых символами **WIMP**. Эти буквы означают соответственно: окна — Windows, значки — Icons, меню — Menus и указывающие устройства — Pointing device.

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. Для ввода в системах GUI обычно используются клавиатура и мышь, а вот вывод практически всегда направляется на специальное устройство, называемое **графическим адаптером**. Этот адаптер состоит из специального блока памяти, называемого видеопамью, в которой хранятся изображения, появляющиеся на экране.

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.

Существует огромное число примеров и видео для разработки приложений, использующих GUI [60, 61]. 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 некоторых 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 сред 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.

Список литературы

1. Большой Российский энциклопедический словарь. — М.: БРЭ. — 2006, 1887 с.
2. Корилов А.М., Павлов С.Н. Теория систем и системный анализ: учеб. пособие. — Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2008. — 264 с.
3. Таненбаум Э. С., Херберт Б. Современные операционные системы. 4-е изд. — Питер СПб, 2019. — 1120 с.
4. Corbató F. J., Merwin-Daggett M., Daley R. C. An experimental time-sharing system //Proceedings of the May 1-3, 1962, spring joint computer conference. — 1962. — P. 335-344.
5. Дейтел Х. М., Дейтел П. Д., Чофнес Д. Р. Операционные системы. Основы и принципы: Третье издание // М.: ООО» Издательство Бинум. — 2011. — 1024 с.
6. Operating System Market Share Worldwide [Electronic resource] // StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share – StatCounter. 2020. URL: <https://gs.statcounter.com> (дата обращения: 05.02.2022).
7. Столлингс, В. Операционные системы: внутренняя структура и принципы проектирования, 9-е изд. : Пер. с англ. - СПб.: ООО "Диалектика", 2020. — 1264 с.
8. Краткая история Linux [Электронный ресурс] // Хостинговое сообщество «Timeweb Community». 2018. URL: <https://timeweb.com/ru/community/articles/kratkaya-istoriya-linux-1/> (дата обращения: 05.02.2022)
9. Virtual machine [Electronic resource] // Wikipedia. 2020. URL: https://en.wikipedia.org/wiki/Virtual_machine (дата обращения: 07.02.2021)
10. VMware Workstation Player [Электронный ресурс] // VMware Россия — для облака, мобильности, сети, безопасности. 2020. URL: <https://www.vmware.com/ru/products/workstation-player.html> (дата обращения: 06.02.2021)

11. Командная строка [Электронный ресурс] // Русскоязычная документация по Ubuntu. 2016. URL: https://help.ubuntu.ru/wiki/командная_строка (дата обращения: 03.02.2022)
12. Знакомство с компилятором GCC [Электронный ресурс] // Файловый архив для студентов. StudFiles. URL: <https://studfile.net/preview/4153566/> (дата обращения: 05.02.2022)
13. Мэтью, Н. Основы программирования в Linux: Пер. с англ. / Н. Мэтью, Р. Стоунс. 4-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 896 с.
14. Как установить Midnight Commander (mc) в Ubuntu 18.04 [Электронный ресурс] // Losst - Linux Open Source Software Technologies. 2018. URL: <https://losst.ru/kak-ustanovit-mc-v-ubuntu-18-04> (дата обращения: 05.02.2022)
15. Уорд Б. Внутреннее устройство Linux //СПб.: Питер. – 2016.
16. Роберт Лав – Linux. Системное программирование, 2016.
17. Михаэль Кофлер – Linux. Установка, настройка, администрирование, СПб.: Питер, 2014. — 768 с.
18. Системный вызов [Электронный ресурс] // Национальная библиотека им. Н. Э. Баумана. 2017. URL: https://ru.bmstu.wiki/Системный_вызов (дата обращения: 05.02.2022)
19. Права доступа к файлам в Linux [Электронный ресурс] // Linux Open Source Software Technologies. 2016. URL: <https://losst.ru/prava-dostupa-k-fajlam-v-linux> (дата обращения: 05.02.2022)
20. Терминал Linux. Права доступа к каталогам и файлам в Linux, команды chmod и chown. [Электронный ресурс] // Статьи о Linux / Ubuntu. LinuxRussia.com. URL: <https://linuxrussia.com/terminal-chmod-chown.html> (дата обращения: 05.02.2022)
21. Команда chmod в Linux [Электронный ресурс] // Пингвинус - Linux в деталях. 2018. URL: <https://pingvinus.ru/note/chmod> (дата обращения: 05.02.2022)
22. Соответствие консольных команд Windows и Linux [Электронный ресурс] // Системное администрирование для начинающих. 2019. URL: <https://white55.ru/cmd-sh.html> (дата обращения: 05.02.2022)
23. Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб.: Питер, 2003. – 461 с.

24. Шилдт, Г. С++: базовый курс / Г. Шилдт. – М.: Вильямс, 2012. – 624 с.
25. Абрамовиц М., Стиган И. Справочник по специальным функциям с формулами, графиками и таблицами // 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, О.Н. Поддубная, Е.И. Шилкина, – Минск : БГЭУ, 2014. – 299 с.
27. Гулько А. Программирование (в среде Windows). – Litres, 2019. – 155 с.
28. CreateNamedPipeA function (winbase.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createnamedpipea> (дата обращения: 05.02.2022)
29. ConnectNamedPipe function (namedpipeapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-connectnamedpipe> (дата обращения: 05.02.2022)
30. DisconnectNamedPipe function (namedpipeapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-disconnectnamedpipe> (дата обращения: 05.02.2022)
31. WaitNamedPipeA function (winbase.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-waitnamedpipea> (дата обращения: 05.02.2022)
32. CreateFileA function (fileapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-createfilea> (дата обращения: 05.02.2022)
33. WriteFile function (fileapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile> (дата обращения: 05.02.2022)
34. ReadFile function (fileapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile>

us/windows/win32/api/fileapi/nf-fileapi-readfile (дата обращения: 05.02.2022)

35. Сокеты Windows [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/cpp/mfc/windows-sockets?view=msvc-160> (дата обращения: 05.02.2022)

36. WSASStartup function (winsock.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-wsastartup> (дата обращения: 05.02.2022)

37. Socket function (winsock2.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket> (дата обращения: 05.02.2022)

38. Connect function (winsock2.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-connect> (дата обращения: 05.02.2022)

39. Bind function (winsock.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind> (дата обращения: 05.02.2022)

40. Listen function (winsock2.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-listen> (дата обращения: 05.02.2021)

41. Accept function (winsock2.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-accept> (дата обращения: 05.02.2022)

42. Send function (winsock2.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-send> (дата обращения: 05.02.2022)

43. Recv function (winsock.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-recv> (дата обращения: 05.02.2022)

44. Closesocket function (winsock.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-closesocket> (дата обращения: 05.02.2022)

45. Семафор (Операционные Системы) [Электронный ресурс] // Национальная библиотека им. Н. Э. Баумана. 2016. URL: [https://ru.bmstu.wiki/Семафор_\(Операционные_Системы\)](https://ru.bmstu.wiki/Семафор_(Операционные_Системы)) (дата обращения: 05.02.2022)

46. Semaphores are Surprisingly Versatile [Electronic resource] // Preshing on Programming. 2015. URL: <https://preshing.com/20150316/semaphores-are-surprisingly-versatile/> (дата обращения: 05.02.2022)

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

48. Мьютекс [Электронный ресурс] // Национальная библиотека им. Н. Э. Баумана. 2016. URL: <https://ru.bmstu.wiki/Мьютекс> (дата обращения: 21.01.2022)

49. Классы Semaphore и SemaphoreSlim [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/dotnet/standard/threading/semaphore-and-semaphoreslim> (дата обращения: 12.01.2022)

50. Semaphore Класс (System.Threading) [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.semaphore?view=netframework-4.8> (дата обращения: 12.01.2022)

51. Метод IHostSyncManager::CreateSemaphore [Электронный ресурс] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/dotnet/framework/unmanaged-api/hosting/ihostsynchronizer-createsemaphore-method> (дата обращения: 14.01.2022)

52. CreateSemaphoreA function (winbase.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createsemaphorea> (дата обращения: 14.01.2022)

53. Using Semaphore Objects - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/windows/win32/sync/using-semaphore-objects> (дата обращения: 15.01.2022)

54. Mutexes [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/dotnet/standard/threading/mutexes> (дата обращения: 16.01.2022)

55. Mutex Конструктор (System.Threading) [Электронный ресурс] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.mutex.-ctor?view=netframework-4.8> (дата обращения: 18.01.2022)

56. Синхронизация процессов и потоков [Электронный ресурс] // CodeNet - все для программиста. 2016. URL: <http://www.codenet.ru/progr/cpp/process-threads-sync.php> (дата обращения: 20.01.2022)

57. CreateEventA function (synchapi.h) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createeventa> (дата обращения: 22.01.2022)

58. Using Event Objects (Synchronization) - Win32 apps [Electronic resource] // Microsoft Docs. URL: <https://docs.microsoft.com/ru-ru/windows/win32/sync/using-event-objects> (дата обращения: 22.01.2022)

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

60. Visual C++ Tutorial 1 -Windows Forms Application [Electronic resource] // YouTube. 2013. URL: <https://www.youtube.com/watch?v=YR6fxe1wa8g&list=PLS1QulWo1RIZz6uDid--I09EOImRmPHS0> (дата обращения: 23.01.2022)

61. Qt Tutorials For Beginners 1 - Introduction [Electronic resource] // YouTube. 2020. URL: <https://www.youtube.com/watch?v=EkjaiDsiMQ&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA> (дата обращения: 23.01.2022)

Учебное издание

ЛАБОРАТОРНЫЕ РАБОТЫ ПО КУРСУ
«ОПЕРАЦИОННЫЕ СИСТЕМЫ»

Методические указания

Составители: Дмитрий Андреевич Савельев

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34

Изд-во Самарского университета.
443086, Самара, Московское шоссе, 34.