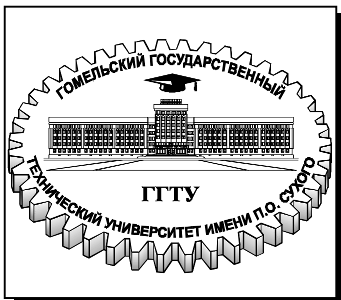

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

ОПЕРАЦИОННЫЕ СИСТЕМЫ И СРЕДЫ

ПРАКТИКУМ
по выполнению лабораторных работ
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Гомель 2023

Библиотека ГГТУ им.П.О.Сухого

УДК 004.42(075.8)
ББК 32.973.22я73
О-60

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 8 от 06.04.2022 г.*)

Составитель *Н. В. Самовендюк*

Рецензент: зав. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *К. С. Курочка*

Операционные системы и среды : практикум по выполнению лаборатор. работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / сост. Н. В. Самовендюк. – Гомель : ГГТУ им. П.О. Сухого, 2023. – 94 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

О-60

Представлены задания и краткие теоретические сведения, позволяющие студентам познакомиться с работой в диалоговой и графической средах Linux, с разработкой и отлаживанием сценариев в Shell, с инструментальными средствами среды программирования Linux, с концепциями многозадачности и межпроцессного взаимодействия.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.42(075.8)
ББК 32.973.22я73

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2023

Библиотека ГГТУ им. П.О. Сухого

Введение

Дисциплина «Операционные системы и среды» является одной из базовых при подготовке студентов по специальности 1-40 04 01 - «Информатика и технологии программирования». Знания и умения, полученные при изучении дисциплины, необходимы для освоения последующих специальных дисциплин и позволят программистам рационально использовать все ресурсы операционной системы, а также проектировать эффективные программы.

Практикум по выполнению лабораторных работ ориентирован на формирование у студентов теоретических основ построения, структуры и функционирования операционных систем, получение знаний об архитектурных н функциональных особенностях операционных систем семейства Unix/Linux, овладение концепциями многозадачности и многопоточности, принципами и механизмами управления процессами и ресурсами.

В практикуме представлены краткие теоретические сведения и задания к лабораторным работам. Рассматриваются вопросы по работе в диалоговой и графической средах Linux, разработке и отлаживанию сценариев в Shell. Отдельно представлены задания по использованию инструментальных средств среды программирования Linux, многозадачности, межпроцессному взаимодействию и управлению ресурсами в Linux.

Лабораторный практикум полностью соответствует учебной программе по дисциплине «Операционные системы и среды».

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 1
Работа с интерфейсом командной строкой в ОС Linux

Цель: изучить интерфейс командной строки ОС Linux, приобрести основные навыки по работе с терминалом командной строки оболочки bash.

Теоретические сведения

Командная оболочка — это программа, взаимодействующая с пользователем с помощью текстового интерфейса. Он называется интерфейсом командной строки (CLI). Оболочка позволяет пользователю запускать программы и выполнять команды операционной системы.

Оболочка интерпретирует введенные пользователем команды, преобразуя их в инструкции операционной системы.

Встроенные системные команды Linux-систем состоят из процедур оболочки и исполняемых файлов. Встроенные команды в разных оболочках могут выполняться по-разному, однако они выполняются быстрее, чем системные. Для многих встроенных команд есть системные аналоги.

Для работы с CLI используется терминал. Для запуска терминала в графическом режиме используется сочетание клавиш Ctrl+Alt+T. Закрыть терминал можно введя команду exit или сочетанием клавиш Ctrl+D.

Переключиться в виртуальный терминал можно нажав комбинацию клавиш Ctrl+Alt+F1, выйти из виртуального терминала: Ctrl+Alt+F7.

Другие клавиатурные сочетания оболочки bash:

- <Ctrl> + <B> курсор влево;
- <Ctrl> + <F> курсор вправо;
- <Alt> + <B> курсор на слово влево;
- <Alt> + <F> курсор на слово вправо;
- <Ctrl> + <A> курсор в начало строки;

Библиотека ГГТУ им. П.О. Сухого

- <Ctrl> + <E> курсор в конец строки;
- <Ctrl> + <H> удаление символа перед курсором;
- <Ctrl> + <D> удаление символа в позиции курсора;
- <Alt> + <D> удаление слова;
- <Ctrl> + <L> очистка экрана;
- <Alt> + <T> перемена мест аргументов;
- <Alt> + <U> перевод слова в верхний регистр;
- <Alt> + <L> перевод слова в нижний регистр;
- <Ctrl> + <C> остановка выполнения команды;
- <Ctrl> + <Z> приостановка выполнения задания (bg, kill);
- <Ctrl> + <R> поиск команды в истории.

Выполнение множества команд доступны только **root** пользователям. Root - это специальный аккаунт в UNIX-подобных системах с UID (User IDentifier) 0, владелец, которого имеет право на выполнение всех без исключения операций. Для того, что бы войти под root сначала его необходимо активировать, а затем в командной строке ввести команду su (sudo), после прохождения авторизации ваша работа не будет ограничивается в правах доступа.

В основном, команды, запускаемые из командной строки, имеют следующий формат:

command -options <filename> .

где options и <filename> являются необязательными параметрами.

Существуют команды, не требующие ввода ни одного из них, и команды, требующие ввода нескольких опций и имен файлов. Если используются несколько опций одновременно, их можно сгруппировать. Например, для просмотра подробного списка (-l) всех файлов текущего каталога, включая скрытые файлы (-a), воспользуйтесь командой: ls –al.

Для ввода длинной команды используется \ (перевод строки). Для ввода нескольких команд в одной строки используется «;». При использовании синтаксиса Команда1 && Команда2 — Команда2 выполняется только в случае удачного выполнения

Библиотека ГГТУ им.П.О.Сухого

Команды1. Команда1 || Команда2 — Команда2 выполняется только в случае неудачного выполнения Команды1.

Получить помощь по командам в Unix-системе можно несколькими способами.

1. help – встроенная помощь оболочки. Большинство команд Linux могут быть запущены с параметром --help . Например, эта команда даст Вам краткую помощь по команде cp (копирование): cp --help | less.

2. man — система помощи в любой Unix системе. Более расширенная информация доступна из командной строки с использованием так называемых страниц руководства (manual pages). Например: man cp выведет на экран страницу руководства для команды cp (копирование).

Страницы man состоят из стандартных разделов:

- NAME – информация, которая будет использована при поиске по ключевому слову;
- SYNOPSIS – формат вызова, опции и аргументы;
- DESCRIPTION – описание объекта (программы, файла, библиотеки);
- OPTIONS – подробное описание опций;
- FILES – файлы, связанные с командой;
- AUTHOR – имя автора с указанием электронной почты;
- SEEALSO– указатели на другие страницы man;
- COPYRIGHT – права собственности, политика распространения.

3. info— гипертекстовая иерархическая система Gnu TexInfo.

Например: info cp выведет Вам информацию о команде "cp" (копирование).

Часто info содержит информацию схожую с man, но более свежую.

Команды: <n> следующий узел, <p> предыдущий узел, <u> родительский узел, <l> предыдущая страница, <s> поиск строки на странице, <q> выход.

4. /usr/share/doc — документация программ.

Библиотека ГГТУ им. П.О.Сухого

Страницы руководства man в Linux делятся на следующие секции:

1. Команды пользовательского уровня и приложения;
2. Системные вызовы и коды ошибок ядра;
3. Библиотечные функции;
4. Информация о файлах устройств и других специальных файлах;
5. Форматы конфигурационных файлов;
6. Помощь по играм;
7. Макросы, кодировки, интерфейсы;
8. Команды системного администрирования;
9. Внутренние интерфейсы и спецификации ядра.

В Linux формирование файловой системы и каталогов осуществляется по таким правилам:

- логически файловая структура организована в виде иерархии;
- каждый каталог может иметь множество подкаталогов, но у каждого подкаталога имеется только один родительский каталог;
- имя корневого каталога - / (он сам для себя является родительским);
- прописные и строчные буквы в именах файлов различаются: TheFile и thefile — разные файлы;
- абсолютные имена файлов показывают путь к файлу от корневого каталога (имена начинаются с /) /home/user1/homework/lab1.html;
- относительные имена показывают путь к файлу от текущего каталога;
- имена файлов могут содержать точки: archive.tar.gz;
- имя файла не может содержать символов / и \0 (null).

Типы файлов в Linux:

- - - обычные файлы;

Библиотека ГГТУ им.П.О.Сухого

- d – каталоги;
- l – символические ссылки (указатель на другой файл);
- b – блочные устройства (специальные файлы для обращения к устройствам, например, жесткому диску);
- c – символьные устройства (специальные файлы, предназначенные для ввода/вывода с таких устройств, как терминал или мышь);
- p – именованный канал (один из вариантов организации взаимодействия между процессами);
- s – сокеты (предназначены для организации сетевого межпроцессного взаимодействия).

Понятие параметра в оболочке bash подобно понятию переменной в обычных языках программирования. Именем (или идентификатором) параметра может быть слово, состоящее из алфавитных символов, цифр и знаков подчеркивания (только первый символ этого слова не может быть цифрой), а также число или один из следующих специальных символов: *, @, #, ?, - (дефис), $, !, 0, _ (подчеркивание). Чтобы отличать команды от переменных, переменные лучше обозначать большими буквами (пример - HOSTNAME).

Таблица 1 - Переменные оболочки и окружения

Экранирование строки	VAR1='Bolshoy Privet!'
	VAR1='Vam vsem '$VAR1
Изменение значения	VAR1=${VAR1}ZZ
Список всех переменных оболочки	set
Уничтожить переменную	unset имя
Список переменных окружения	env

Говорят, что параметр задан или установлен, если ему присвоено значение. Значением может быть и пустая строка. Чтобы вывести значение параметра, используют символ $ перед его именем. Так, команда «[user]$ echo name» выдаст на экран слово name, а команда «[user]$ echo $name» выдаст значение переменной name (если таковое, конечно, задано).

Библиотека ГГТУ им. П.О. Сухого

Переменные оболочки доступны только в той оболочке, в которой были описаны.

В ОС Linux важнейшими переменными окружения являются:

- HOME — путь к домашнему каталогу;
- LOGNAME и USER — имя пользователя;
- MAIL — путь к почтовому ящику;
- PATH — путь поиска исполняемых файлов;
- PS1 — вид приглашения оболочки;
- PWD — имя текущего каталога;
- OLDPWD — имя предыдущего каталога;
- SHELL — имя исполняемого файла оболочки;
- TERM — тип терминала;
- HOSTNAME — имя хоста;
- SHLVL — номер загруженной оболочки.

Таблица 2 - Базовые команды bash

Описание	**Команда**	**Пример**
очистить экран	clear	
вывести имя текущего каталога	pwd	
вывести содержимое текущего каталога	ls	
вывести содержимое произвольного каталога	ls <каталог>	
вывести подробную информацию	ls –l	
переместиться в домашний каталог	cd	
переместиться в заданный каталог	cd <каталог>	
cd переместиться в предыдущий каталог	cd-	
простейший способ создать	<filename>	

Библиотека ГГТУ им.П.О.Сухого

файл		
создание файла или изменение даты модификации файла	touch <filename(s)>	date > f1; cat f1; ls –l f*; sleep 60; touch f1 f2; date; cat f1; ls –l f*
удаление файла(ов) и каталогов. Опции: f– не спрашивать подтверждения; i– спрашивать подтверждения; r– рекурсивно удалить каталог и его содержимое	rm <filename(s)>	
Проверка имен файлов перед удалением по шаблону	touch f{1,2,3} ls f*[1,2] rm f*[1,2]	
создание каталога	mkdir <dir>	
создание дерева	mkdir –p dir1/dir2/dir3	
удаление пустого каталога	rmdir <dir>	
удаление дерева пустых каталогов	rmdir –p	
Копирование	cp <source> <destination>	
Рекурсивное копирование	cp R <dir1> <dir2>	
Перемещение (переименование)	mv <source> <destination>	
Поиск файлов. Критерии: name (по имени); iname (по имени игнорируя регистр); type (по типу); size (по размеру); empty (пустые); mtime (по дате модификации); perm (по правам доступа); user (по принадлежности); group (по принадлежности).	Find <места_поиска> <критерии>	Если используются шаблоны подстановки, то необходимы кавычки: find ~ -name "dom*". Два критерия, объединенных условием ИЛИ: find ~ -name "dom*" -o –empty. Подставляются имена файлов: find ~ -name "*core*" –exec rm –f {} \;
Поиск файлов в базе данных командой locate	locate <подстрока имени файла>	
Поиск файла в каталогах, входящих в переменную PATH (строку поиска)	which <что-ищем>	
Поиск файла в системных каталогах (не смотря на строку поиска)	whereis <что-ищем>	
Определение типа файлов	file /etc/passwd file /bin/ls	
Свободное место	df –h	
Сколько занимает папка	du –sh <имя-папки>	

Библиотека ГГТУ им. П.О. Сухого

Практическая часть:

1. Перейти в корневую директорию (папку). Проверьте, в какой директории находитесь.

2. Вывести пронумерованный список директорий далее работать с директорией # (ваш –номер по списку);

3. Вывести содержимое директории:

- в формате по умолчанию;
- в обратном порядке;
- содержимое поддиректорий;
- вывести все файлы включая скрытые;
- вывести файлы с указанием их размера в КБ/МБ/ГБ;
- вывести файлы отсортированные по размеру, с указанием размера в КБ/МБ/ГБ;
- только имена вложенных директорий, расположенных в текущей директории;
- отсортированное по дате создания файла;
- отсортированное по дате обращения к файлу;
- только файлы, вторая буква имени которых – гласная англ. алфавита
- записать список файлов и папок в текущей директории (с полной информацией о них) в файл dirlist.txt , который лежит в домашней директории.

4. Перейти в домашнюю директорию с помощью короткой команды;

5. Вернуться в предыдущую директорию;

6. Вернуться обратно в домашнюю;

7. Перейти на уровень выше;

8. Перейти в каталог /tmp. С помощью одной команды перейти в подкаталог local/bin каталога /usr.

9. Вывести содержимое файла dirlist.txt:

- просто;
- в обратном порядке;
- с нумерацией непустых строк;

Библиотека ГГТУ им. П.О. Сухого

- с нумерацией всех строк;
- схлопывая подряд идущие пустые строки в одну;

14. Создать в домашней директории папку linux_lab1
15. Войти в директорию linux_lab1
16. Скопировать в нее файл dirlist.txt из домашней директории
17. Удалить файл dirlist.txt из домашней директории
18. Создать директорию manyfiles
19. Создать в ней 100 файлов с именами a1, a2, a3, …. a100.
20. Удалить только файлы с четными номерами.
21. Вывести строки файла dirlist.txt, содержащие файлы с определенным месяцем (в зависимости от номер варианта 1- январь,..12 – декабрь, 13 – опять январь) и записать их в файл grep_month_name.txt.
22. Записать строки, не содержащие этот месяц, в файл grep_other_monthes.txt.
23. Создать папку grep, переместить в нее файлы, созданные в пунктах 21 и 22.
24. Находясь в папке linux_lab1 найти все файлы в этой директории и ее поддиректориях в которых встречается подстрока root, вывести строки с указанием их номеров.
25. Найти все файлы в системе, содержащие в имени bash;
26. Найти файлы, измененные за последний час.
27. Найти символические ссылки в каталоге / (но не глубже чем на 2), вывести, на что они указывают.
28. Просмотреть, какие переменные окружения заданы в вашей системе;
29. Поменять приглашение командной строки, добавить текущее время.
30. Удалить весь каталог manyfiles со всеми файлами;
31. Создайте текстовый файл следующего содержания:

    1+2
    6*4
    97%12
    43215/43*100

Библиотека ГГТУ им.П.О.Сухого

Посчитайте все примеры из файла с помощью одной команды.

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:

1. Титульный лист.
2. Цель работы.
3. Задания.
5. Скриншот запуска команды и результаты её выполнения.
6. Выводы.

Контрольные вопросы для защиты:

1. Как запустить терминал в графическом режиме?
2. Как перейти в домашний каатлог?
3. Как просмотреть содержимое каталога?
4. Какие типы файлов в Linux?
5. Как определить объем каталога?
6. Как узнать переменные окружения?
7. Как найти файл/каталог?
8. Что такое канал и как он используется?
9. Как применить механизм регулярных выражений для фильтрации результатов?
10. Какие команды предназначены для создания элементов файловой системы?
11. Как скопировать данные?
12. Как переместить данные?
13. Как удалить данные?
14. Какие виды ссылок используются в Linux?
15. Как смонтировать внешнюю память?

Библиотека ГГТУ им. П.О.Сухого

Лабораторная работа № 2
Администрирование в Linux. Управление пользователями

Цель: Познакомиться с принципами аутентификации, форматами файлов для хранения учетных записей и изучить команды для управления учетными записями.

Теоретические сведения

Linux — многопользовательская система, учётная запись — это информация о пользователе а системная политика представляет собой правила работы в системе.

Учётная запись пользователя – это необходимая для операционной системы информация о пользователе, хранящаяся в специальных файлах. Информация используется Linux для аутентификации пользователя и назначения ему прав доступа.

Аутентификация – системная процедура, позволяющая операционной системе определить, какой именно пользователь осуществляет вход. Вся информация о пользователе обычно хранится в файлах /etc/passwd и /etc/group. В файле /etc/passwd содержится информация о пользователях. Запись для каждого пользователя занимает одну строку.

Рисунок 1 - Запись информации о пользователе в файл/etc/passwd

На рисунке 1 показана структура записи о пользователя, в которую входит следующая информация:

- а – имя пользователя;
- б – шифрованный пароль;
- в – число дней с последнего изменения пароля, начиная с 1 января 1970 года;
- г – число дней, перед тем как пароль может быть изменен;

Библиотека ГГТУ им. П.О. Сухого

- д – число дней, после которых пароль должен быть изменен;
- е – число дней, за сколько пользователя начнут предупреждать, что пароль устаревает;
- ж – число дней, после устаревания пароля для блокировки учетной записи;
- з –дней, отсчитывая с 1 января 1970 года, когда учетная запись будет заблокирована;
- и – зарезервированное поле;

При добавлении нового пользователя происходит новая запись в /etc/passwd и /etc/shadow, создаётся домашний каталог. Каталог /etc/skel содержит файлы, которые копируются в создаваемый домашний каталог пользователя.

Добавляя нового пользователя чаще всего используются опции команды useradd такие как:

- s — файл оболочки по умолчанию;
- d — путь к домашнему каталогу;
- m — необходимо создавать домашний каталог;
- M — не создавать домашний каталог;
- k — путь к альтернативному каталогу скелета;
- u — назначить UID;
- g — назначить GID (первичную группу);
- G — список групп пользователя;
- e — дата блокировки учетной записи;
- f — срок после устаревания пароля до блокировки учетной записи.

Управление паролями в Linux - часть системной политики. Управляя паролями требуется определить категорий пользователей, которые могут сами выбирать пароль, правила выбора паролей и требования к их уровню сложности, сроки устаревания паролей, длительность периода запрета на изменение пароля. Это является типичными правилами при управлении паролями.

Опции при управлении паролями:

Библиотека ГГТУ им. П.О.Сухого

- -l — блокировка;
- -u — разблокирование;
- -S — текущее состояние пароля;
- -d — удаление пароля;
- -n — период запрета смены пароля;
- -x — максимальный срок использования пароля;
- -w — период предупреждений;
- -i — период после устаревания пароля до блокировки.

Группы пользователей очень удобный механизм администрирования пользователей в ОС Linux. Информация о группах пользователей хранится в файле /etc/group. Запись для каждой группы пользователей так же занимает одну строку как и в /etc/passwd. Строка в /etc/group имеет следующую структуру:

<имя группы>:<пароль группы>:<GID группы>:<список пользователей>.

В файлах профиля пользователя устанавливаются переменные окружения. PATH- путь поиска исполняемых файлов, USER- имя пользователя, HOME- путь к домашнему каталогу. Так же в профиле пользователя хранятся настройки пользователя, файл сценария оболочки, который выполняется автоматически при входе в сеанс и umask. Umask - это права на доступ к файлам и каталогам по умолчанию.

Файл ~/.bashrc — это ресурс оболочки (для пользователя), выполняется только при запуске оболочки из командной строки (в отличие от файлов профиля), содержит дополнительные настройки оболочки /etc/bashrc — общесистемный файл ресурсов оболочки (удобно задавать псевдонимы для команд).

При входе в сеанс bash исполняются:

1. /etc/profile;
2. ~/.bash_profile;
3. ~/.bashrc;
4. /etc/bashrc.

Если выполняется запуск оболочки из командной строки выполняются пункты 3) и 4).

Библиотека ГГТУ им. П.О. Сухого

Отчёты об активности пользователей хранятся в файлах как и прочая информация в ОС Linux. В файле /var/run/utmp хранится список пользователей, находящихся в сеансе, в файле /var/log/wtmp хранится информация об открытых и законченных сеансах пользователей, файл /var/log/lastlog хранит информацию о последних входах в сеанс.

Опции команды who:

- -b — время последней загрузки системы;
- -H — печать заголовка;
- -login — информация о системных процессах, контролирующих вход в сеанс;
- -q — имена всех пользователей в системе и их количество;
- -u — подробная информация о сеансах;
- -a — полная информация о статусе процессов, контролирующих вход в сеанс.

Опции команды lastlog:

- -b — входы в сеанс, ранее указанного количества дней;
- -t — входы в сеанс, за указанный период;
- -u — входы в сеанс пользователя.

Таблица 1 - Команды для администрирования пользователей и учетных пользователей

Описание	Команда
регистрация нового пользователя	useradd <username>
получить информацию о пользователе	id <username>
выдать настройки команды по умолчанию	useradd –D
изменение учетной записи	usermod <options> <username>
удаление учетной записи	userdel <username>
управление паролями	passwd <options> <username>
создание группы	groupadd <groupname>
удаление группы	groupdel <groupname>
назначение администратора	gpasswd –A <username>

Библиотека ГГТУ им.П.О.Сухого

группы	<groupname>
добавление пользователя к группе, удаление пользователя из группы (администратором группы)	gpasswd –a <username> <groupname>, gpasswd –d <username> <groupname>
задать пароль на вход в группу для не членов группы	gpasswd <password>
изменить текущую группу	newgrp <groupname>
выполнить файл профиля	source <profile_file>
список пользователей, находящихся в сеансе	who
информация об открытых и законченных сеансах	last
информация о последних входах в сеанс	lastlog

Практическая часть:

1. Ознакомиться с содержимым файлов:
   - /etc/passwd;
   - /etc/shadow;
   - /etc/group.
2. Создать следующие группы:
   - Workers;
   - Teachers;
   - Students.

3. Создать пользователей user_[номер варианта]_ N, где N =1, 2, .., 5, uid учетной записи должен быть равен 1000+N. Пользователей с N, равным 1 и 2, добавить в группу workers вручную внеся изменения в конфигурационный файл. После добавления пользователей осуществить проверку файла /etc/group на ошибки. Пользователей с N, равным 3, 4 и 5, добавить в группу students при

Библиотека ГГТУ им. П.О. Сухого

помощи команд администрирования. Проверьте результат, выполнив действия п.1.

4. Создать пользователя teacher_[номер варианта]. В комментарии к учетной записи должны быть Ваше имя и фамилия. uid учетной записи должен быть равен 3000. Пользователя добавить в группу teachers.

5. Для всех пользователей задайте пароли, используя команду passwd.

6. Создать директорию labs в корневом каталоге. В нем создать каталоги library и tests.

7. Создать файлы book_[фамилия студента]_N и поместить их в library.

8. Создать текстовый файл test_[имя студента], и поместить в tests. Файлы должны содержать скрипт на создание пользователя user[номер варианта] и задание ему пароля pass[номер варианта]. Сделайте эти файлы исполняемыми для пользователей группы students.

9. В директории labs создать файл list, который должен содержать список файлов директории /etc.

10. Дать право на изменение файла только пользователю teacher_[номер варианта], а на чтение пользователям группы workers.

11. Настроить права доступа к каталогу library и tests, таким образом, чтобы пользователи группы teachers могли изменять и создавать там файлы, а пользователи группы students имели доступ на чтение.

12. Просмотрите файл /etc/shadow (с правами root). У всех ли пользователей содержимое второго поля выглядит приблизительно одинаково? Какие символы могут содержаться в шифрованной строке пароля в /etc/shadow?

13. Зарегистрируйте пользователя test1, для которого запрещен вход в сеанс, имеющего домашний каталог /home/nouser и являющегося членом групп user и mail. Пользователь должен иметь UID равный 2000.

Библиотека ГГТУ им.П.О.Сухого

14. Создайте учетную запись для пользователя test2 с настройками по умолчанию. Проверьте, создался ли домашний каталог пользователя, наполнен ли он файлами и какому пользователю он принадлежит.

15. Измените имя пользователя test2 на test3.

16. Удалите пользователя test3.

17. Помимо файла /etc/default/useradd имеется еще один конфигурационный файл, влияющий на поведение команды useradd. Найдите его и изучите его содержание. Какая настройка позволяет изменять минимальный UID для новых пользователей?

18. Зарегистрируйте пользователя test4 с настройками по умолчанию и установите для него пароль. Изучите содержимое соответствующей записи в /etc/shadow.

19. Установите дату устаревания пароля для пользователя на 31 декабря текущего года. Проверьте, что изменилось в /etc/shadow.

20. Удалите пароль пользователя и проверьте изменения в /etc/shadow.

21. Заблокируйте учётную запись test4.

22. Создайте группу пользователей xusers с GID, равным 1010.

23. Зарегистрируйте себя в качестве участника группы xusers. Проверьте результат выполненного действия.

24. Измените имя группы на yusers.

25. Сделайте так, чтобы при запуске оболочки из командной строки выдавалось приветствие.

26. Определите, когда последний раз была загружена система.

27. Кто входил в сеанс за последние 2 недели?

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:

4. Титульный лист.
5. Цель работы.
6. Задания.
7. Скриншот запуска команды и результаты её выполнения.

Библиотека ГГТУ им. П.О. Сухого

8. Выводы.

Контрольные вопросы для защиты:

1. Какова структура записи пользователя в файле /etc/passwd?
2. Какие опции у команды useradd?
3. Как получить отчет об активности пользователя?
4. Как создать группу пользователей?
5. Как изменить права доступа к файлу?
6. Как сменить владельца группы?
7. Как настроить политику безопасности для входа в систему?
8. Как изменить учетную запись?
9. Как узнать информацию об открытых и законченных сеансах?
10. Как получить информацию о пользователе?

Библиотека ГГТУ им. П.О.Сухого

Лабораторная работа № 3
Программирование в Shell

Цель: изучить основы программирования на языке Shell, приобрести практические навыки по созданию пакетных исполняемых файлов.

Теоретические сведения

Язык **shell** по своим возможностям приближается к высокоуровневым алгоритмическим языкам программирования. Операторы языка **shell** позволяют создавать собственные программы, не требуют компиляции, построения объектного файла и последующей компоновки, так как shell, обрабатывающий их, является транслятором интерпретирующего, а не компилирующего типа.

Текст процедуры набирается как обычный текстовый файл. Проверенный и отлаженный shell-файл может быть вызван на исполнение, например, следующим способом:

```
$ chmod u+x shfil
$ shfil
$
```

Такая форма предполагает, что файл процедуры новый и его надо сначала сделать выполняемым. Можно использовать также и следующий способ:

$ sh -c "shfil" или $ sh shfil

Процедуре при ее запуске могут быть переданы аргументы. В общем случае командная строка вызова процедуры имеет следующий вид:

Библиотека ГГТУ им. П.О.Сухого

$ имя_процедуры $1 $2 ...$9

Каждому из девяти первых аргументов командной строки в тексте процедуры соответствует один из позиционных параметров: $1, $2, ..., $9 соответственно. Параметр $0 соответствует имени самой процедуры, т.е. первому полю командной строки. К каждому из 10 первых аргументов можно обратиться из процедуры, указав номер его позиции. Количество аргументов присваивается другой переменной: **$#**(диез).

Сам интерпретатор shell автоматически присваивает значения следующим переменным (параметрам):

? значение, возвращенное последней командой;

$ номер процесса;

! номер фонового процесса;

число позиционных параметров, передаваемых в shell;

* перечень параметров, как одна строка;

@ перечень параметров, как совокупность слов;

- флаги, передаваемые в shell.

При обращении к этим переменным (т.е при использовании их в командном файле - shell-программе) следует впереди ставить "$".

При написании сценариев shell используют некоторые вспомогательные операторы:

echo - вывод сообщений из текста процедуры на экран;

$ echo "начало строки

> продолжение строки";

- для обозначения строки комментария в процедуре (строка не будет обрабатываться shell-ом)**;**

banner - вывод сообщения на экран заглавными буквами (например для идентификации следующих за ним сообщений).

set - присвоить значения позиционным параметрам, запуск set без параметров выводит список установленных системных переменных;

shift - сдвинуть позиционные параметры влево на одну позицию.

Библиотека ГГТУ им. П.О.Сухого

В UNIX при написании операторов важное значение отводится кавычкам (апострофам):

'...' - для блокирования специальных символов, которые могут быть интерпретированы как управляющие;

"..." - блокирование наиболее полного набора управляющих символов или указания того, что здесь будет обрабатываться не сам аргумент, а его значение;

`...` - (обратные кавычки или знак ударения) для указания того, что они обрамляют команду и здесь будет обрабатываться результат работы этой команды (подстановка результатов работы указанной команды).

Для ввода строки текста со стандартного устройства ввода используется оператор:

read имя1 [имя2 имя3] - чтение строки слов со стандартного ввода

Команда вводит строку, состоящую из нескольких полей (слов), со стандартного ввода, заводит переменную для каждого поля и присваивает первой переменной имя1, второй переменной - имя2, и т.д. Если имен больше, чем полей в строке, то оставшиеся переменные будут инициализированы пустым значением. Если полей больше, чем имен переменных, то последней переменной будет присвоена подстрока введенной строки, содержащая все оставшиеся поля, включая разделители между ними. В частности, если имя указано только одно, то соответствующей ему переменной присваивается значение всей строки целиком.

В отличии от рассмотренных в начале системных переменных среды, переменные языка shell называются **локальными переменными** и используются в теле процедур для решения обычных задач. Локальные переменные могут иметь имя, состоящее из одного или нескольких символов. Присваивание значений переменным осуществляется с помощью известного оператора:

Библиотека ГГТУ им. П.О.Сухого

"=" - присвоить (установить) значение переменной.

При этом если переменная существовала, то новое значение замещает старое. Если переменная не существовала, то она строится автоматически shell.

Процедура подстановки выполняется shell-ом каждый раз когда надо обработать значение переменной если используется следующая конструкция:

$имя_переменной - на место этой конструкции будет подставлено значение переменной.

$ a = 10

Переменной можно присвоить результат работы команды

$ files=`ls`

С переменными можно выполнять арифметические действия как и с обычными числами с использованием специального оператора:

expr - вычисление выражений.

Для **арифметических операций**, выполнимых командой **expr**, используются операторы:

+ сложение; - вычитание; * умножение (обратная косая черта \ используется для отмены действия управляющих символов, здесь *); / деление нацело; % остаток от деления.

Для **логических операций** арифметического сравнения чисел командой **expr** используются следующие обозначения: = равно; != не равно; \< меньше; \<= меньше или равно; \> больше; \>= больше или равно.

Все операнды и операторы являются самостоятельными аргументами команды **expr** и поэтому должны отделяться друг от друга и от имени команды **expr** пробелами.

Команда **expr** выводит результат вычисления на экран. Поэтому, если он не присвоен никакой переменной, то не может быть использован в программе.

При решении логических задач, связанных с **обработкой символьных строк (текстов)** команда **expr** может быть

Библиотека ГГТУ им.П.О.Сухого

использована, например, как средство для подсчета символов в строках или для вычленения из строки цепочки символов. Операция обработки строк символов задается **кодом операции ":"** и шаблонами. В частности:

'.*' - шаблон для подсчета числа символов в строке,

'...\(.*\)....' - шаблон для выделения подстроки удалением символов строки, соответствующих точкам в шаблоне.

Выводимая информация - количество символов или подстрока - может быть присвоена некоторой переменной и использована в дальнейших вычислениях.

Синтаксис **$(командная_строка)** также подставляет результаты выполнения команды перед запуском на исполнение командной строки.

В данном случае в качестве подставляемой команды может быть использована любая имеющая смысл sh-процедура. Shell просматривает командную строку и выполняет все команды между открывающей и закрывающей скобками.

Для удаления переменных используется команда **unset.**

Проверка условий и ветвление вычислительных процессов

Все команды UNIX вырабатывают код завершения (возврата), обычно для того, чтобы в дальнейшем можно было выполнить диагностику посредством проверки значения кода завершения и определить: нормально завершилось выполнение команды (=0 или true) или не нормально (# 0 или false). Например, если (=1), то ошибка синтаксическая.

Код завершения после выполнения каждой команды помещается автоматически в специальную **системную переменную ?**. Ее значение можно вывести на экран:

echo $?

Код завершения используется для программирования условных переходов в sh-процедурах. Проверка истинности условий для

Библиотека ГГТУ им.П.О.Сухого

последующего ветвления вычислительного процесса процедур может быть выполнена с помощью команды:

test <проверяемое отношение/условие>

Вместо мнемоники команды может использоваться конструкция с квадратными скобками:

[проверяемое отношение/условие] синоним команды test.

Аргументами этой команды могут быть имена файлов, числовые или нечисловые строки (цепочки символов). Командой вырабатывается код завершения (код возврата), соответствующий закодированному в команде test условию. Код завершения проверяется следующей командой. Если закодированное параметрами условие выполняется, то вырабатывается логический результат - true, если нет - false.

Код возврата может обрабатываться как следующей за test командой, так и специальной конструкцией языка: **if-then-else-fi**.

1. **Проверка файлов:**

test -ключ имя_файла

Ключи:

-r файл существует и доступен для чтения;

-w файл существует и доступен для записи;

-x файл существует и доступен для исполнения;

-f файл существует и имеет тип "-", т.е. обычный файл;

-s файл существует, имеет тип "-" и не пуст;

-d файл существует и имеет тип "d", т.е. файл - каталог.

2. Сравнение числовых значений:

test число1 –к число2

Числа могут быть как просто числовыми строками, так и переменными, которым эти строки присвоены в качестве значений. Ключи для анализа числовых значений:

-eq равно; **-ne** не равно;

-lt меньше; **-le** меньше или равно;

-gt больше; **-ge** больше или равно.

3. Сравнение строк:

Библиотека ГГТУ им.П.О.Сухого

test [-n] 'строка' - строка не пуста (n – число проверяемых строк)

test -z 'строка' - строка пуста

test 'строка1' = 'строка2' - строки равны

test 'строка1' != 'строка2' - строки не равны

Необходимо заметить, что все аргументы команды test - строки, имена, числа, ключи и знаки операций являются самостоятельными аргументами и должны разделяться пробелами.

Выражение вида **"$переменная"** лучше заключать в двойные кавычки, что предотвращает в некоторых ситуациях возможную неподходящую замену переменных shell-ом.

Особенности сравнения чисел и строк. Shell трактует все аргументы как числа в случае, если осуществляется сравнение чисел, и все аргументы как строки, если осуществляется сравнение строк.

Ветвление вычислительного процесса в shell-процедурах осуществляется семантической конструкцией:

if список_команд1
then список_команд2
[else список_команд3]
fi

Список_команд - это или одна команда или несколько команд, или фрагмент shell-процедуры. Если команды записаны на одной строке, то они разделяются точкой с запятой. Для задания пустого списка команд следует использовать специальный оператор:

: (двоеточие) -пустой оператор.

Список_команд1 передает оператору if код завершения последней выполненной в нем команды. Если он равен 0, то выполняется список_команд2. Таким образом, код возврата 0 эквивалентен логическому значению "истина". В противном случае он эквивалентен логическому значению "ложь" и выполняется либо список_команд3 после конструкции else, либо - завершение конструкции if словом fi.

В качестве списка_команд1 могут использоваться списки любых команд. Однако, чаще других используется команда test. В операторе if так же допускается две формы записи этой команды:

Библиотека ГГТУ им.П.О.Сухого

if test аргументы

if [аргументы]

Каждый оператор if произвольного уровня вложенности обязательно должен завершаться словом fi.

Использование циклов

Циклы обеспечивают многократное выполнение отдельных участков процедуры до достижения заданных условий.

Цикл типа while (пока true)**:**

while список_команд1

do список_команд2

done

Список_команд1 возвращает код возврата последней выполненной команды. Если условие истинно, выполняется список_-команд2, затем снова список_команд1 с целью проверки условия, а если ложно, выполнение цикла завершается. Таким образом, циклический список_команд2 выполняется до тех пор, пока условие истинно.

Цикл типа until (пока false)**:**

until список_команд1

do список_команд2

done

Логическое условие, определяемое по коду возврата списка_команд1, инвертируется, т.е. цикл выполняется до тех пор, пока условие ложно.

Цикл типа for:

for имя_ переменной [in список_значений]

do список_команд

done

Переменная с указанным в команде именем заводится автоматически. Переменной присваивается значение очередного слова из списка_значений и для этого значения выполняется список_команд. Количество итераций равно количеству значений в

Библиотека ГГТУ им.П.О.Сухого

списке, разделенных пробелами (т.е. циклы выполняются пока список не будет исчерпан).

Ветвление по многим направлениям case

Команда **case** обеспечивает ветвление по многим направлениям в зависимости от значений аргументов команды. Формат:

```
case <string> in
s1) <list1>;;
s2) <list2>;;
.
.
sn) <listn>;;
*) <list>
esac
```

Здесь list1, list2 ... listn - список команд. Производится сравнение шаблона string с шаблонами s1, s2 ... sk ... sn. При совпадении выполняется список команд, стоящий между текущим шаблоном sk и соответствующими знаками ;;.

Некоторые дополнительные команды, которые могут быть использованы в процедурах:

sleep t - приостанавливает выполнение процесса на t секунд

exit [n] - прекращение выполнения процедуры с кодом завершения [n] или с кодом завершения последней выполненной команды.

В качестве “n” можно использовать любое число, например, для идентификации выхода из сложной процедуры, имеющей несколько причин завершения выполнения.

Практическая часть:

1. Создать сценарий реализующий в консольном режиме диалог с пользователем в виде меню. Сценарий должен выполняться циклически пока не выбран пункт «Выход». Первый пункт меню

Библиотека ГГТУ им. П.О. Сухого

должен выводить информацию о создателе (ФИО, группа) и краткое описание выполняемых действий, второй пункт меню должен вычислять математическое выражение 2.1, а остальные пункты реализуют действия указанные в таблице в соответствии с вариантом. Все параметры задаются в результате диалога с пользователем.

A^2+3*B-C/2 (2.1)

Таблица 3.1

Вариант	**Задание**
1	А) Проверить существует ли папка в указанном месте и если нет создать её. Б) Архивации файлов в заданном каталоге, с созданием отдельного архива для каждого файла и удалением заархивированных.
2	А) Копирование файлов с указанного места в заданное. Б) Очистки подкаталога с подтверждением.
3	А) Удаление файлов заданного расширения в заданной папке. Б) Разработать пакетный файл для установки даты и времени (параметры – в командной строке).
4	А) Удаление содержимого заданной папки, с помещением всех удаленных объектов в папку tmp. Б) Проверить существование указанного в параметре файла и выдать сообщение о результате поиска.
5	А)Создание файла со списком папок в указанном месте. Б) Проверить существует ли заданный файл и доступен ли он для записи.
6	А) Копирование файлов заданного расширения с указанного места в папку «BackUp» в каталоге tmp. Б) Запрос и ввод имени пользователя, сравнение с текущим логическим именем пользователя и вывод сообщения: верно/неверно.

Библиотека ГГТУ им. П.О.Сухого

Продолжение табл. 3.1

7	А) Перенос файлов заданного расширения с указанного места в папку «BackUp»в каталоге tmp. Б) Проверить существует ли заданный файл и доступен ли он для исполнения, если нет установить на него атрибуты исполняемого.
8	А) Создание файла со списком папок в указанном месте. Б) Определить является ли заданный каталог пустым , если нет определить количество файлов в нем.
9	А) Создание файла со списком файлов с заданным расширением в указанном месте. Б) Разработать пакетный файл для перехода в заданный, если он существует и его архивирования
10	А) Перенос файлов с указанного места в заданное. Б) Разработать пакетный файл для перехода студента в личный каталог. В специальный файл (logon.data) в домашней папке записывается дата и время входа в систему.
11	А) Проверить существует ли папка в указанном месте и если да переименовать её. Б) Скопировать файлы из заданного каталога в папку tmp/Имя пользователя, с добавив расширение .bak
12	А)Архивирование заданной папки с копированием архива в папку «BackUp» в каталоге tmp Б) В заданной папке вывести все файлы с длиной больше заданного значения
13	А) Архивирование файлов заданного расширения в заданной папке с копированием архива в папку «BackUp» в каталоге tmp Б) В заданной папке определить файл с самым длинным именем

Библиотека ГГТУ им. П.О.Сухого

Окончание табл. 3.1

14	А) Разработать пакетный файл для построения системы студенческих каталогов с запросом на создание каталогов требуемых курсов, групп и запросом максимального числа пользователей в группе Б) Переход в другой каталог, формирование файла с листингом каталога и возвращение в исходный каталог.
15	А) Копирование всех фалов с заданным расширением с указанного места в указанную папку. Б) Сохранение сведений о файлах домашнего каталога в файле «текущая дата.txt», в домашней папке History.

Библиотека ГГТУ им. П.О.Сухого

Лабораторная работа № 4
Инструментальные средства разработки Linux

Цель: изучить инструментальные средства разработки Linux: компиляторы cc/g++/gcc

Теоретические сведения

Средствами, традиционно используемыми для создания программ для открытых операционных систем, являются инструменты разработчика GNU. Целью проекта GNU было создание комплекта программного обеспечения под единой лицензией, которая не допускала бы возможности присваивания кем-то эксклюзивных прав на это ПО. Частью этого комплекта и является набор инструментов для разработчика, которым мы будем пользоваться, и который должен входить во все дистрибутивы Linux.

Одним из этих инструментов является компилятор GCC. Первоначально эта аббревиатура расшифровывалась, как GNU C Compiler. Сейчас она означает – GNU Compiler Collection и поддерживает множество языков программирования (C, C++, Objective C++, Java, Fortran и др.).

Файлы с исходными кодами программ - это обычные текстовые файлы и создавать их можно с помощью любого текстового редактора (например GEdit, KWrite, Kate, а также более традиционные для пользователей Linux – vi и emacs).

Обычно для создания запускаемого файла создается отдельный каталог с исходным кодом, в котором и будет находится исполняемый файл.

Создадим отдельный в домашнем каталоге каталог hello. Это будет каталог нашего первого проекта. Создадим текстовый файл с исходным кодом hello.c со следующим текстом:

```
#include <stdio.h>
int main()
{
```


Библиотека ГГТУ им. П.О. Сухого

```
printf("Hello world!\n");
return 0;
}
```

Теперь в консоли необходимо зайти в каталог проекта и набрать

```
gcc hello.c
```

В каталоге должен появиться новый файл a.out. Это и есть исполняемый файл. Запустим его, набрав в консоли:

```
./a.out
```

Программа должна запуститься, то есть должен появиться текст:

```
Hello world!
```

Компилятор gcc по умолчанию присваивает всем созданным исполняемым файлам имя a.out. Если хотите назвать его по-другому, нужно к команде на компиляцию добавить флаг –o и имя, которым вы хотите его назвать. Наберите такую команду:

```
gcc hello.c -o hello
```

В каталоге должен появиться исполняемый файл с названием hello.

Флаг -o является лишь одним из многочисленных флагов компилятора gcc. Чтобы просмотреть все возможные флаги, можно воспользоваться справочной системой man. Наберите в командной строке:

```
man gcc
```


Библиотека ГГТУ им.П.О.Сухого

Перед вами предстанет справочная система по этой программе. Просмотрите, что означает каждый флаг.

При запуске программ из нашего каталога разработки, мы перед названием файла набираем точку и слэш. Зачем же мы это нужно? Дело в том, что, если мы наберем только название исполняемого файла, операционная система будет искать его в каталогах /usr/bin и /usr/local/bin, и, естественно, не найдет. Каталоги /usr/bin и /usr/local/bin – системные каталоги размещения исполняемых программ. Первый из них предназначен для размещения стабильных версий программ, как правило, входящих в дистрибутив Linux. Второй – для программ, устанавливаемых самим пользователем (за стабильность которых никто не ручается). Такая система нужна, чтобы отделить их друг от друга. По умолчанию при сборке программы устанавливаются в каталог /usr/local/bin. Крайне нежелательно помещать что-либо лишнее в /usr/bin или удалять что-то оттуда вручную, потому что это может привести к краху системы. Там должны размещаться программы, за стабильность которых отвечают разработчики дистрибутива.

Чтобы запустить программу, находящуюся в другом месте, надо прописать полный путь к ней, например так:

/home/nick/projects/hello/hello

Или другой вариант: прописать путь относительно текущего каталога, в котором вы в данной момент находитесь в консоли. При этом одна точка означает текущий каталог, две точки – родительский. Например, команда ./hello запускает программу hello, находящуюся в текущем каталоге, команда ../hello – программу hello, находящуюся в родительском каталоге, команда ./projects/hello/hello – программу во вложенных каталогах, находящихся внутри текущего.

Есть возможность добавлять в список системных путей к программам дополнительные каталоги. Для этого надо добавить новый путь в системную переменную PATH.

Библиотека ГГТУ им.П.О.Сухого

Принцип работы программы gcc

Работа компилятора gcc включает три этапа: обработка препроцессором, компиляция и компоновка (или линковка).

Препроцессор включает в основной файл содержимое всех заголовочных файлов, указанных в директивах #include. В заголовочных файлах обычно находятся объявления функций, используемых в программе, но не определенных в тексте программы. Их определения находятся где-то в другом месте: или в других файлах с исходным кодом или в бинарных библиотеках (ключ -E).

Вторая стадия – компиляция. Она заключается в превращении текста программы на языке C/C++ в набор машинных команд. Результат сохраняется в объектном файле.

Разумеется, на машинах с разной архитектурой процессора двоичные файлы получаются в разных форматах, и на одной машине невозможно запустить бинарник, собранный на другой машине (разве только, если у них одинаковая архитектура процессора и одинаковые операционные системы). Вот почему программы для UNIX-подобных систем распространяются в виде исходных кодов: они должны быть доступны всем пользователям, независимо от того, у кого какой процессор и какая операционная система (ключ -c).

Последняя стадия – компоновка. Она заключается в связывании всех объектных файлов проекта в один, связывании вызовов функций с их определениями, и присоединением библиотечных файлов, содержащих функции, которые вызываются, но не определены в проекте. В результате формируется запускаемый файл – наша конечная цель. Если какая-то функция в программе используется, но компоновщик не найдет место, где эта функция определена, он выдаст сообщение об ошибке, и откажется создавать исполняемый файл (ключ -o).

Если создается объектный файл из исходника, уже обработанного препроцессором (например, такого, какой мы получили выше), то мы должны обязательно указать явно, что компилируемый файл является файлом исходного кода,

Библиотека ГГТУ им.П.О.Сухого

обработанный препроцессором, и имеющий теги препроцессора. В противном случае он будет обрабатываться, как обычный файл С++, без учета тегов препроцессора, а значит связь с объявленными функциями не будет устанавливаться. Для явного указания на язык и формат обрабатываемого файла служит опция -х. Файл С++, обработанный препроцессором обозначается cpp-output

.

```
gcc -x cpp -output -c prog1.cpp
```

Наконец, последний этап – компоновка. Получаем из объектного файла исполняемый.

```
gcc prog1.o -o proga
```

Можно его запускать.

```
./ prog1
```

Как правило исходных файлов несколько, и они объединены в проект. И в некоторых исключительных случаях программу приходится компоновать из нескольких частей, написанных на разных языка. В этом случае приходится запускать компиляторы разных языков, чтобы каждый получил объектный файл из своего исходника, а затем уже эти полученные объектные файлы компоновать в исполняемую программу.

Определения функций могут находиться в библиотеках. Это скомпилированные двоичные файлы, содержащие коллекции однотипных операций, которые часто вызываются из многих программ, а потому нет смысла многократно писать их код в программах. Стандартное расположение файлов библиотек – каталоги /usr/lib и /usr/local/lib (при желании можно добавить путь). Если библиотечный файл имеет расширение .a, то это статическая библиотека, то есть при компоновке весь ее двоичный код включается в исполняемый файл. Если расширение .so, то это динамическая библиотека. Это значит в исполняемый файл программы помещается

Библиотека ГГТУ им.П.О.Сухого

только ссылка на библиотечный файл, а уже из него и запускается функция.

При компоновке любой программы компилятор gcc по умолчанию включает в запускаемый файл библиотеку libc. Это стандартная библиотека языка С. Она содержит функции, необходимые абсолютно во всех программах, написанных на С, в том числе и функцию printf. Поскольку библиотека libc нужна во всех программах, она включается по умолчанию, без необходимости давать отдельное указание на ее включение.

Остальные библиотеки надо включать явно. Например для использования математических функций необходима библиотека libm. Она требует включения в текст программы заголовочного файла <math.h>.

Помимо этого дистрибутивы Linux содержат и другие библиотеки, например:

● libGL Вывод трехмерной графики в стандарте OpenGL. Требуется заголовочный файл <GL/gl.h>.

● libcrypt Криптографические функции. Требуется заголовочный файл <crypt.h>.

● libcurses Псевдографика в символьном режиме. Требуется заголовочный файл <curses.h>.

● libform Создание экранных форм в текстовом режиме. Требуется заголовочный файл <form.h>.

● libgthread Поддержка многопоточного режима. Требуется заголовочный файл <glib.h>.

● libgtk Графическая библиотека в режиме X Window. Требуется заголовочный файл <gtk/gtk.h>.

● libhistory Работы с журналами. Требуется заголовочный файл <readline/readline.h>.

● libjpeg Работа с изображениям в формате JPEG. Требуется заголовочный файл <jpeglib.h>.

● libncurses Работа с псевдографикой в символьном режиме. Требуется заголовочный файл <ncurses.h>.

Библиотека ГГТУ им. П.О.Сухого

● libpng Работа с графикой в формате PNG. Требуется заголовочный файл <png.h>.

● libpthread Многопоточная библиотека POSIX. Стандартная многопоточная библиотека для Linux. Требуется заголовочный файл <pthread.h>.

● libreadline Работа с командной строкой. Требуется заголовочный файл <readline/readline.h>.

● libtiff Работа с графикой в формате TIFF. Требуется заголовочный файл <tiffio.h>.

● libvga Низкоуровневая работа с VGA и SVGA. Требуется заголовочный файл <vga.h>.

Обратите внимание, что названия всех этих библиотек начинаются с буквосочетания lib-. Для их явного включения в исполняемый файл, нужно добавить к команде gcc опцию -l, к которой слитно прибавить название библиотеки без lib-. Например, чтобы включить библиотеку libvga надо указать опцию -lvga.

Практическая часть:

1. В соответствии со своим вариантом разработать программу вычисления значения функции **b=f(x,y,z).** Значения x, y и z должны вводиться пользователем. При выводе информации предусмотреть форматирование документа. Описание функции и ее реализацию представить в отдельных файлах (табл. 4.1).

Таблица 4.1. – Виды функций (по вариантам)

Вариант	Вид функции	Вариант	Вид функции
1	$b = \frac{1+\cos^2(x+z)}{\left\|x^3 - 2y^2\right\|}$	16	$b = x + \frac{\sqrt[3]{zy}}{y + \cos x}$
2	$b = \frac{\ln^2\|z\|}{\sqrt[3]{\|x\| + \|y\|}}$	17	$b = \lg\left(\sqrt{e^{x-y} + x^{\|y\|} + z}\right)$
3	$b = \frac{y^3}{x + y^3\cos^2 z}$	18	$b = 1 + \frac{x^2+1}{3+y^2} + \sin 2z$
4	$b = \sqrt{x + \sqrt[4]{\|y\|}} + \cos^2 z$	19	$b = \left\|\cos x + \cos y\right\| + 2\sin^2 z$
5	$b = \frac{\sqrt[3]{e^{\sin x}} \cdot \cos y}{z^2+1}$	20	$b = \frac{\ln(y^3)(z - x/2)}{2\cos^2 x}$

Библиотека ГГТУ им. П.О. Сухого

Окончание табл. 4.1

Вариант	Вид функции	Вариант	Вид функции
6	$b = z(tg\ y - e^{-(x+3)})$	21	$b = \sqrt{10\left(\sqrt[3]{z} + x^{(y+2)}\right)}$
7	$b = \lvert x - y\rvert\left(\sin^2 z + tg\ z\right)$	22	$b = (\sin z)^2 + \lvert x + y\rvert$
8	$b = \sqrt{y + \sqrt[3]{x}} - 1 + 2z$	23	$b = e^{2z} - \sqrt[3]{y\lvert x\rvert}$
9	$b = x\left(tg\, z + \cos^2 y\right)$	24	$b = e^{(x-1)} + \sin\ y$
10	$b = e^{\lvert x-y\rvert}\left(tg^2 z + 1\right)^x$	25	$b = \sqrt{\lvert z\rvert e^{-(y+x/2)}}$
11	$b = \cos^2 z + tg\ 2x + \lvert y\rvert$	26	$b = \dfrac{4y^2 e^{2x} \sin^2 z}{3z^3 + \ln x}$
12	$b = 5tg\ z - 4y^2 + \lvert xy\rvert$	27	$b = \dfrac{\sqrt{y \ln x} - z x^2}{1 + tg^2 x^2} x$
13	$b = (z - x)\dfrac{y - \ln z}{1 + (y - x)^2}$	28	$b = \dfrac{\lg\left(y + \sqrt{z + x^2}\right)}{y + x^2}$
14	$b = y^z + \sqrt{\lvert x\rvert + \lvert y\rvert}$	29	$b = \dfrac{x^2 + 4}{\sin^2 z^2 + x/2} y$
15	$b = \dfrac{\lg\left(\sqrt{x} + \sqrt{y} + 2\right)}{\lvert 2z\rvert}$	30	$b = \dfrac{\sin x + \sqrt{\lvert z - y\rvert}}{y(x - 2) + x^2}$

2. Разработать программу, в которой используется класс в соответствии с вариантом. Описание и реализация методов класса должны быть в разных файлах.

Вариант 1

Разработайте класс *Окружность*. Свойство: радиус. Методы: длина окружности и площадь круга, ограниченного окружностью.

На основе разработанного класса решите следующую задачу: для заданных радиусов двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:

$C = 2 \cdot \pi \cdot r, \quad S = \pi \cdot r^2,$

где r – радиус окружности;

C – длина окружности;

S – площадь круга.

Библиотека ГГТУ им. П.О. Сухого

Вариант 2

Разработайте класс *Прямоугольный_Треугольник*. Свойства: длины двух катетов. Методы: длина гипотенузы и площадь треугольника.

На основе разработанного класса решите следующую задачу: для заданных длин катетов двух треугольников определите, у какого треугольника большая гипотенуза, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:

$$c = \sqrt{a^2 + b^2}, \quad S = \frac{a \cdot b}{2},$$

где a и b – длины катетов;

c – длина гипотенузы;

S – площадь прямоугольного треугольника.

Вариант 3

Разработайте класс *Куб*. Свойство: длина стороны. Методы: площадь поверхности и объем.

На основе разработанного класса решите следующую задачу: для заданных длин сторон двух кубов определите, у какого куба большая поверхность, насколько объем одного куба больше другого. Ответ выведите на форму.

Формулы для расчета:

$$S = 6 \cdot a^2, \quad V = a^3,$$

где a – длина стороны куба;

S – площадь поверхности куба;

V – объем куба.

Вариант 4

Разработайте класс *Треугольник*. Свойства: длина одной стороны, величины двух прилежащих к заданной стороне углов. Методы: площадь треугольника, величина третьего угла.

На основе разработанного класса решите следующую задачу: для заданных длин сторон и величин углов двух треугольников

Библиотека ГГТУ им.П.О.Сухого

определите, у какого треугольника большая площадь, а у какого самый большой угол. Ответ выведите на форму.

Формулы для расчета:

$$\gamma = 180^\circ - \alpha - \beta, \quad S = \frac{a^2 \cdot \sin\alpha \cdot \sin\beta}{2 \cdot \sin(\alpha + \beta)},$$

где a – длина стороны треугольника;

α и β – углы, прилежащие к стороне a, град.;

γ – третий угол треугольника, град.;

S – площадь треугольника.

Вариант 5

Разработайте класс *Отрезок*. Свойство: координаты концов отрезка. Методы: длина отрезка, проверка параллельности оси ОХ.

На основе разработанного класса решите следующую задачу: для заданных координат концов двух отрезков определите, у какого отрезка большая длина, а какой из отрезков параллелен оси ОХ. Ответ выведите на форму.

Формула для расчета:

$$r = \sqrt{(x1 - x2)^2 + (y1 - y2)^2},$$

где $(x1, y1)$ и $(x2, y2)$ – координаты концов отрезка;

r – длина отрезка.

Условие параллельности отрезка оси ОХ:

$$y1 = y2.$$

Вариант 6

Разработайте класс *Треугольник*. Свойства: координаты вершин треугольника. Методы: площадь треугольника, длина большей стороны.

На основе разработанного класса решите следующую задачу: для заданных координат вершин двух треугольников определите, у какого треугольника большая площадь, а у какого самая большая длина стороны. Ответ выведите на форму.

Формулы для расчета:

Библиотека ГГТУ им. П.О. Сухого

$a=\sqrt{(x1-x2)^2+(y1-y2)^2}$, $b=\sqrt{(x2-x3)^2+(y2-y3)^2}$,
$c=\sqrt{(x1-x3)^2+(y1-y3)^2}$, $p=\frac{a+b+c}{2}$, $S=\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}$, где ($x1$, $y1$), ($x2$, $y2$) и ($x3$, $y3$) – координаты вершин треугольника;

a, b и c – длины сторон треугольника;

p – полупериметр треугольника;

S – площадь треугольника.

Вариант 7

Разработайте класс *Окружность*. Свойство: диаметр. Методы: длина окружности и площадь круга, ограниченного окружностью.

На основе разработанного класса решите следующую задачу: для заданных диаметров двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:

$C=\pi\cdot d$, $S=\pi\cdot\frac{d^2}{4}$,

где d – диаметр окружности;

C – длина окружности;

S – площадь круга.

Вариант 8

Разработайте класс *Прямоугольник*. Свойства: координаты трех вершин прямоугольника. Методы: площадь прямоугольника и длина диагонали.

На основе разработанного класса решите следующую задачу: для заданных координат вершин двух прямоугольников определите, у какого прямоугольника большая диагональ, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:

$a=\sqrt{(x1-x2)^2+(y1-y2)^2}$, $b=\sqrt{(x2-x3)^2+(y2-y3)^2}$,
$d=\sqrt{(x1-x3)^2+(y1-y3)^2}$, $S=a\cdot b$,

где ($x1$, $y1$), ($x2$, $y2$) и ($x3$, $y3$) – координаты вершин прямоугольника;

Библиотека ГГТУ им. П.О. Сухого

a и b – длины сторон прямоугольника;

d – длина диагонали прямоугольника;

S – площадь прямоугольника.

Вариант 9

Разработайте класс *Параллелограмм*. Свойства: длины сторон параллелограмма и острый угол. Методы: площадь и периметр параллелограмма.

На основе разработанного класса решите следующую задачу: для заданных сторон и острого угла двух параллелограммов определите, у какого параллелограмма больший периметр, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:

$S = ab\sin\alpha$, P = 2(a+b)

где a и b – длины сторон параллелограмма;

α – острый угол параллелограмма;

P – периметр параллелограмма;

S – площадь параллелограмма.

Вариант 10

Разработайте класс *Шар*. Свойства: радиус. Методы: площадь поверхности и объём шара.

На основе разработанного класса решите следующую задачу: для заданных радиусов двух шаров определите, у какого шара больший объём, а у какого большая площадь поверхности. Ответ выведите на форму.

Формулы для расчета:

$S = 4\pi R^2, \; V = \frac{4}{3}\pi R^3$

где R – радиус шара;

Вариант 11

Разработайте класс *Окружность*. Свойство: радиус. Методы: длина окружности и площадь круга, ограниченного окружностью.

Библиотека ГГТУ им. П.О. Сухого

На основе разработанного класса решите следующую задачу: для заданных радиусов двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:

$$C = 2 \cdot \pi \cdot r, \quad S = \pi \cdot r^2,$$

где r – радиус окружности;
C – длина окружности;
S – площадь круга.

Вариант 12

Разработайте класс *Прямоугольный_Треугольник*. Свойства: длины двух катетов. Методы: длина гипотенузы и площадь треугольника.

На основе разработанного класса решите следующую задачу: для заданных длин катетов двух треугольников определите, у какого треугольника большая гипотенуза, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:

$$c = \sqrt{a^2 + b^2}, \quad S = \frac{a \cdot b}{2},$$

где a и b – длины катетов;
c – длина гипотенузы;
S – площадь прямоугольного треугольника.

Вариант 13

Разработайте класс *Куб*. Свойство: длина стороны. Методы: площадь поверхности и объем.

На основе разработанного класса решите следующую задачу: для заданных длин сторон двух кубов определите, у какого куба большая поверхность, и насколько объем одного куба больше другого. Ответ выведите на форму.

Формулы для расчета:

$$S = 6 \cdot a^2, \quad V = a^3,$$

где a – длина стороны куба;

Библиотека ГГТУ им.П.О.Сухого

S – площадь поверхности куба;

V – объем куба.

Вариант 14

Разработайте класс *Треугольник*. Свойства: длина одной стороны, величины двух прилежащих к заданной стороне углов. Методы: площадь треугольника, величина третьего угла.

На основе разработанного класса решите следующую задачу: для заданных длин сторон и величин углов двух треугольников определите, у какого треугольника большая площадь, а у какого самый большой угол. Ответ выведите на форму.

Формулы для расчета:

$$\gamma = 180^\circ - \alpha - \beta, \quad S = \frac{a^2 \cdot \sin\alpha \cdot \sin\beta}{2 \cdot \sin(\alpha + \beta)},$$

где a – длина стороны треугольника;

α и β – углы, прилежащие к стороне a, град.;

γ – третий угол треугольника, град.;

S – площадь треугольника.

Вариант 15

Разработайте класс *Отрезок*. Свойство: координаты концов отрезка. Методы: длина отрезка, проверка параллельности оси OX.

На основе разработанного класса решите следующую задачу: для заданных координат концов двух отрезков определите, у какого отрезка большая длина, а какой из отрезков параллелен оси OX. Ответ выведите на форму.

Формула для расчета:

$$r = \sqrt{(x1 - x2)^2 + (y1 - y2)^2},$$

где $(x1, y1)$ и $(x2, y2)$ – координаты концов отрезка;

r – длина отрезка.

Условие параллельности отрезка оси OX:

$$y1 = y2.$$

Библиотека ГГТУ им. П.О. Сухого

Требованияпо содержанию отчета:

В отчете должны быть отображены следующие пункты:

9. Титульный лист.
10. Цель работы.
11. Задания.
12. Исходные тексты программ.
13. Скриншот компиляции программ.
14. Результаты выполнения программ.
15. Выводы.

Контрольные вопросы для защиты:

1. Какие компиляторы используются в Linux?
2. Что такое GNU?
3. Что такое объектный файл и как его создать?
4. Для чего осуществляется компоновка (линковка)?
5. Как подключить дополнительные библиотеки в программу?
6. В чем разница между статической и динамической компоновкой?
7. Как скомпилировать программу из нескольких файлов исходного кода?
8. Как скомпилировать программу из файлов исходных кодов, написанных на разных языках программирования?
9. Как изменить имя исполняемого файла во время компиляции?
10. Укажите стандартное расположение файлов библиотек в Linux?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 5
Знакомство со стандартной утилитой Gnu make для построения проектов в ОС Unix/Linux

Цель: ознакомиться с техникой компиляции программ на языке программирования С (С++) в среде ОС семейства Unix/Linux, а также получить практические навыки использования утилиты GNU make для сборки проекта.

Теоретические сведения

Основы использования утилиты построения проектов make

«Сборка» большинства программ для ОС семейства Unix производится с использованием утилиты **make**. Эта утилита считывает файл (обычно носящий имя «makefile»), в котором содержатся инструкции, и выполняет в соответствии с ними действия, необходимые для сборки программы. Во многих случаях makefile полностью генерируется специальной программой. Например, для разработки процедур сборки используются программы autoconf/automake. Однако в некоторых программах может потребоваться непосредственное создание файла makefile без использования процедур автоматической генерации.

Следует отметить, что существует, как минимум, три различных наиболее распространенных варианта утилиты make: GNU make, System make и Berkeley make

Основными составляющими любого make -файла являются **правила (rules)**. В общем виде правило выглядит так:

<цель_1>…<цель_n><зависимость_1>…<зависимость_n>
<команда_1>
.
.
<команда_n>

Библиотека ГГТУ им.П.О.Сухого

Цель (target) — это некоторый желаемый результат, способ достижения которого описан в правиле. Цель может представлять собой имя файла. В этом случае правило описывает, каким образом можно получить новую версию этого файла.

В следующем примере целью является файл iEdit (исполняемый файл программы некоторого гипотетического проекта текстового редактора с главным файлом проекта main.cpp и дополнительными Editor.cpp, TextLine.cpp). Правило описывает, каким образом можно получить новую версию бинарного файла iEdit (скомпоновать из перечисленных объектных файлов):

```
iEdit: main.o Editor.o TextLine.o
gcc main.o Editor.o TextLine.o -o iEdit
```

Если необходимо скомпилировать проект, написанный на C++, то можно использовать компилятор g++. Следует также отметить, что ключ -o компилятора gcc указывает имя конечного бинарного файла.

Цель также может быть именем некоторого действия. В таком случае правило описывает, каким образом совершается указанное действие.

В следующем примере целью является действие clean (очистка, удаление):

```
clean:
rm *.o iEdit
```

Подобного рода цели называют псевдоцелями (pseudo targets) или абстрактными целями (phony targets).

Зависимость (dependency) — это некие «исходные данные», необходимые для достижения указанной в правиле цели, некоторое «предварительное условие» для достижения цели. Зависимость может представлять собой имя файла. Для того чтобы успешно достичь указанной цели, этот файл должен существовать.

Библиотека ГГТУ им. П.О.Сухого

В предыдущем примере файлы main.o, Editor.o и TextLine.o являются зависимостями. Эти файлы должны существовать для того, чтобы стало возможным достижение цели — построение файла iEdit.

Зависимость также может быть именем некоторого действия. Это действие должно быть предварительно выполнено перед достижением цели, указанной в правиле. В следующем примере зависимость clean_obj является именем действия (удалить объектные файлы программы):

```
clean_all: clean_obj
rm iEdit
clean_obj:
rm *.0
```

Для того, чтобы достичь цели clean_all, необходимо сначала выполнить действие (достигнуть цели) clean_obj.

Команды — это действия, которые необходимо выполнить для обновления либо достижения цели. Утилита make отличает строки, содержащие команды, от прочих строк make-файла по наличию символа табуляции (символа с кодом 9) в начале строки:

```
iEdit: main.o Editor.o TextLine.o
	gcc main.o Editor.o TextLine.o -o iEdit
```

В приведенном выше примере строка gcc main.o Editor.o TextLine.o -o iEdit должна начинаться с символа табуляции.

Общий алгоритм работы make

Типичный make-файл проекта содержит несколько правил. Каждое из правил имеет некоторую цель и некоторые зависимости. Смыслом работы make является достижение цели, которую она выбрала в качестве главной цели (default goal). Если главная цель является именем действия (т. е. абстрактной целью), то смысл работы

Библиотека ГГТУ им. П.О.Сухого

make заключается в выполнении соответствующего действия. Если же главная цель является именем файла, то программа make должна построить самую «свежую» версию данного файла.

Выбор главной цели. Главная цель может быть прямо указана в командной строке при запуске make. В следующем примере make будет стремиться достичь цели iEdit (получить новую версию файла iEdit):

```
make iEdit
```

В этом примере make должна достичь цели clean (очистить директорию от объектных файлов проекта):

```
make clean
```

Если не указывать какой-либо цели в командной строке, то make выбирает в качестве главной первую встреченную в make-файле цель.

Схематично «верхний уровень» алгоритма работы make можно представить так:

```
make()
{
главная_цель = ВыбратьГлавнуюЦель ()
ДостичьЦели (главная_цель)
}
```

Достижение цели. После того как главная цель выбрана, make запускает «стандартную» процедуру достижения цели. Сначала в make-файле выполняется поиск правила, которое описывает способ достижения этой цели (функция «НайтиПравило»). Затем к найденному правилу применяется обычный алгоритм обработки правил (функция «ОбработатьПравило»):

```
ДостичьЦели (Цель)
```


Библиотека ГГТУ им. П.О.Сухого

```
{
правило = НайтиПравило (Цель)
ОбработатьПравило (правило)
}
```

Обработка правил. Обработка правила разделяется на два основных этапа. На первом этапе обрабатываются все зависимости, перечисленные в правиле (функция «ОбработатьЗависимости»). На втором этапе принимается решение о том, нужно ли выполнять указанные в правиле команды (функция «НужноВыполнятьКоманды»). При необходимости перечисленные в правиле команды выполняются (функция «ВыполнитьКоманды»):

```
ОбработатьПравило (Правило)
{
ОбработатьЗависимости (Правило)
если НужноВыполнятьКоманды (Правило)
    {
    ВыполнитьКоманды (Правило)
    }
}
```

Обработка зависимостей. Функция «ОбработатьЗависимости» поочередно проверяет все перечисленные в правиле зависимости. Некоторые из них могут оказаться целями каких-нибудь правил. Для этих зависимостей выполняется обычная процедура достижения цели (функция «ДостичьЦели»). Т.е. зависимости, которые не являются целями, считаются именами файлов. Для таких файлов проверяется факт их наличия. При их отсутствии make аварийно завершает работу с сообщением об ошибке.

```
ОбработатьЗависимости (Правило)
{
цикл от до Правило.число_зависимостей
```


Библиотека ГГТУ им.П.О.Сухого

```
если ЕстьТакаяЦель (Правило.зависимость[ i ])
ДостичьЦели (Правило.зависимость[ i ])
иначе
ПроверитьНаличиеФайла (Правило.зависимость[ i ])
}
```

Обработка команд. На стадии обработки команд решается вопрос о том, следует ли выполнять описанные в правиле команды или нет.

Считается, что нужно выполнять команды в таких случаях, как:

• цель является именем действия (абстрактной целью);

• цель является именем файла и этого файла не существует;

• какая-либо из зависимостей является абстрактной целью;

• цель является именем файла и какая-либо из зависимостей, являющихся именем файла, имеет более позднее время модификации, чем цель.

В противном случае (т. е. ни одно из вышеприведенных условий не выполняется) описанные в правиле команды не выполняются. Алгоритм принятия решения о выполнении команд схематично можно представить так:

```
НужноВыполнятьКоманды (Правило)
{
если Правило.Цель.ЯвляетсяАбстрактной ()
return true
//цель является именем файла
если ФайлНеСуществует (Правило.Цель)
return true
цикл от i=1 до Правило.число_зависимостей
{
если Правило.зависимость[ i ].ЯвляетсяАбстрактной ()
return true
иначе
//зависимость является именем файла
```


Библиотека ГГТУ им. П.О.Сухого

```
{
если ВремяМодификации(Правило.Зависимость[ i ]) >
ВремяМодификации (Правило.Цель)
return true
}
}
return false
}
```

Абстрактные цели и имена файлов. Имена действий от имен файлов утилита make отличает следующим образом: сначала выполняется поиск файла с указанным именем, и если файл найден, то считается что цель или зависимость являются именем файла; в противном случае считается, что данное имя является либо именем несуществующего файла, либо именем действия (различия между этими двумя вариантами не делается, поскольку они обрабатываются одинаково).

Следует отметить, что подобный подход имеет ряд недостатков. Во-первых, утилита make не рационально расходует время, выполняя поиск несуществующих имен файлов, которые на самом деле являются именами действий. Во-вторых, при подобном подходе имена действий не должны совпадать с именами каких-либо файлов или директорий, иначе make-файл будет выполняться ошибочно.

Некоторые версии make предлагают свои варианты решения этой проблемы. Так, например, в утилите GNU make имеется механизм (специальная цель .PHONY), с помощью которого можно указать, что данное имя является именем действия.

Использование переменных

Возможность использования переменных внутри make-файла очень удобное и часто используемое свойство make. В традиционных версиях угилиты, переменные ведут себя подобно макросам языка С.

Библиотека ГГТУ им.П.О.Сухого

Для задания значения переменной используется оператор присваивания. Например, выражение

```
obj_list: = main.o Editor.o TextLine.o
```

присваивает переменной obj_list значение «main.o Editor.o TextLine.o» (без кавычек). Пробелы между символом «=» и началом первого слова игнорируются. Следующие за последним словом пробелы также. Значение переменной можно использовать с помощью конструкции

```
$(имя_переменной)
```

Переменные могут не только содержать текстовые строки, но и «ссылаться» на другие переменные. Например, в результате обработки make-файла

Во многих случаях использование переменных позволяет упростить make-файл и повысить его наглядность. Для того чтобы облегчить модификацию make-файла, можно разместить «ключевые» имена и списки в отдельных переменных и поместить их в начало make-фаqла:

```
program_name := iEdit
obj_list := main.o Editor.o TextLine.o
$(program_name): $(obj_list)
gcc $(obj_list) -o $(program_name)
…
```

Адаптация такого make-файла для сборки другой программы сведется к изменению нескольких начальных строк.

Использование автоматических переменных

Автоматические переменные — это переменные со специальными именами, которые «автоматически» принимают

Библиотека ГГТУ им. П. О. Сухого

определенные значения перед выполнением описанных в правиле команд. Автоматические переменные можно использовать для «упрощения» записи правил. Такое, например, правило

```
iEdit: main.o Editor.o TextLine.o
gcc main.o Editor.o TextLine.o -0 iEdit
```

с использованием автоматических переменных можно записать следующим образом:

```
iEdit: main.o Editor.o TextLine.o
gcc $^ -o $@
```

Здесь $^ и $@ являются автоматическими переменными. Переменная $^ означает «список зависимостей». В данном случае при вызове компилятора gcc она будет ссылаться на строку «main.o Editor.o TextLine.o». Переменная $@ означает «имя цели» и будет в этом примере ссылаться на имя «iEdit». Если бы в примере была использована следующая автоматическая переменная $<, то она указывала бы на первое имя зависимости, т. е. в данном случае на файл main.o.

Иногда использование автоматических переменных совершенно необходимо, например в шаблонных правилах.

Шаблонные правила (implicit rules или pattern rules) — это правила, которые могут быть применены к целой группе файлов. В этом их отличие от обычных правил, описывающих отношения между конкретными файлами. Традиционные реализации make поддерживают так называемую «суффиксную» форму записи шаблонных правил:

```
.<расширение_файлов_зависимостей>.<расширение_файлов_целей>:
<команда_1>
<команда_2>
```


Библиотека ГГТУ им.П.О.Сухого

```
<команда_n>
```

Например, следующее правило гласит, что все файлы с расширением «o» зависят от соответствующих файлов с расширением «cpp»:

```
.cpp.o:
gcc -c $^
```

Для современной реализации make более предпочтительная следующая запись данной цели:

```
%.o: %.cpp
gcc -c $^ -o $@
```

Следует обратить внимание на использование автоматической переменной $^ для передачи компилятору имени файла-зависимости. Поскольку шаблонное правило может применяться к разным файлам, использование автоматических переменных — это единственный способ узнать для каких файлов задействуется правило в данный момент. Шаблонные правила позволяют упростить make-файл и сделать его более универсальным. Рассмотрим простой проектный файл:

```
iEdit: main.o Editor.o TextLine.o
gcc $л -0 $@
main.o: main.cpp
gcc -c y
Editor.o: Editor.cpp
gcc -c y
TextLine.o: TextLine.cpp
gcc -c y
```


Библиотека ГГТУ им. П.О. Сухого

Все исходные тексты программы обрабатываются одинаково: для них вызывается компилятор gcc. С использованием шаблонных правил этот пример можно переписать так:

```
iEdit: main.o Editor.o TextLine.o
gcc $^ -o $@
%.o: %.cpp
gcc -c $^
```

Когда make ищет в файле проекта правило, описывающее способ достижения искомой цели (функция «НайтиПравило»), то в расчет принимаются и шаблонные правила. Для каждого из них проверяется, нельзя ли задействовать это правило для достижения искомой цели.

Практическая часть:

1. Ознакомиться с теоретическим материалом.
2. Воспользоваться утилитой make для автоматизации сборки проектов из лабораторной работы №4 «Инструментальные средства разработки Linux»
3. Создать make-файл с высоким уровнем автоматизированной обработки исходных файлов программы согласно следующим условиям:

• имя скомпилированной программы (выполняемый или бинарный файл), флаги компиляции и имена каталогов с исходными файлами и бинарными файлами (каталоги, src, bin и т. п.) задаются с помощью переменных в makefile;

• зависимости исходных файлов на языке C (C++) и цели в make-файле должны формироваться динамически;

• наличие цели clean, удалающей временные файлы;

• каталог проекта должен быть структурирован следующим образом:

src — каталог с исходными файлами;

Библиотека ГГТУ им.П.О.Сухого

bin — каталог с бинарными файлами (скомпилированными);

makejile.

Все функции размещаются в отдельных файлах.

4. Выполнить программу (скомпилировать, при необходимости отладить) для первого варианта сортировки данных.

5. Изменить тип сортировки и выполнить программу.

6. Показать, что при изменении одного исходного файла и последующем вызове make будут исполнены только необходимые команды компиляции (неизмененные файлы перекомпилированы не будут) и изменены атрибуты и/или размер объектных файлов (файлы с расширением .o).

Библиотека ГГТУ им. П.О.Сухого

Таблица 5.1 Варианты заданий

Варианта	Структура данных (STL)	Функции обработки (таблица 2)	Тип сортировки (таблица 3)
1.	Vector	1	1
2.	Array	2	2
3.	Deque	3	3
4.	List	4	4
5.	Forward_List	5	5
6.	Vector	6	6
7.	Array	7	7
8.	Deque	8	1
9.	List	9	2
10.	Forward_List	10	3
11.	Vector	11	4
12.	Array	12	5
13.	Deque	13	6
14.	List	14	7
15.	Forward_List	15	1
16.	Vector	1	2
17.	Array	2	3
18.	Deque	3	4
19.	List	4	5
20.	Forward_List	5	6
21.	Vector	6	7
22.	Array	7	1
23.	Deque	8	2
24.	List	9	3
25.	Forward_List	10	4

Библиотека ГГТУ им. П.О. Сухого

Таблица 5.2 Функции обработки

N	Функции обработки структурированных данных
1.	Сумма отрицательных элементов
2.	Произведение положительных элементов на четных местах
3.	Количество элементов меньших заданного числа А
4.	Определение минимального элемента
5.	Определение максимального элемента
6.	Сумма элементов попадающих в промежуток [A,B]
7.	Произведение отрицательных элементов на нечетных местах
8.	Количество положительных элементов, на местах кратных 3
9.	Определение минимального элемента среди положительных элементов
10.	Определение максимального среди отрицательных элементов
11.	Сумма элементов на нечетных местах
12.	Произведение элементов, больших заданного D
13.	Количество элементов, попадающих в промежуток [A,B]
14.	Определение минимального элемента на четных местах
15.	Определение максимального элемента на нечетных местах

Таблица 5.3 Типы сортировки

N	Начальная сортировка	Конечная сортировка
1.	Пузырьковая	Вставками
2.	Вставками	Выбором минимального
3.	Обменом	Быстрая
4.	Шелла	Пузырьковая
5.	Пузырьковая	Быстрая
6.	Вставками	Обменом
7.	Обменом	Выбором максимального элемента

Библиотека ГГТУ им. П.О. Сухого

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:
1. Титульный лист.
2 Цель работы.
3. Задания.
4. Исходные тексты программы.
5. Make-файл
6. Результаты выполнения make-файла.
7. Результаты выполнения программы.
8. Выводы.

Контрольные вопросы для защиты:

1. Для чего используется утилита make?
2. Что такое цель?
3. Что такое зависимость?
4. Как описываются правила?
5. Как используются переменные в make-файле?
6. Какие автоматические переменные используются в make-файле?
7.В чем отличие главной цели?
8. Опишите общий алгоритм работы утилиты make?
9. Что такое шаблонные правила?
10. Как сформировать make-файл с высокой степенью автоматизации?

Библиотека ГГТУ им.П.О.Сухого

Лабораторная работа № 6
Планирование процессов

Цель: изучить типовые алгоритмы планирования процессов

Теоретические сведения:

Существует достаточно большой набор разнообразных алгоритмов планирования, которые предназначены для достижения различных целей и эффективны для разных классов задач. Многие из них могут использоваться на нескольких уровнях планирования. Рассмотрим некоторые наиболее употребительные алгоритмы применительно к процессу кратковременного планирования.

First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой FCFS по первым буквам его английского названия – First-Come, First-Served (первым пришел, первым обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он, а точнее, ссылка на его PCB помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование – FIFO, сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс, получивший в свое распоряжение процессор, занимает его до истечения текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди.

Библиотека ГГТУ им. П.О. Сухого

Преимуществом алгоритма FCFS является легкость его реализации, но в то же время он имеет и много недостатков. Например, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут очень долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

Round Robin (RR)

Модификацией алгоритма FCFS является алгоритм, получивший название Round Robin (Round Robin – это вид детской карусели в США) или сокращенно RR. По сути дела, это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых процессов организованным циклически – процессы сидят на карусели. Карусель вращается так, что каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 – 100 миллисекунд (см. рисунок 1). Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.

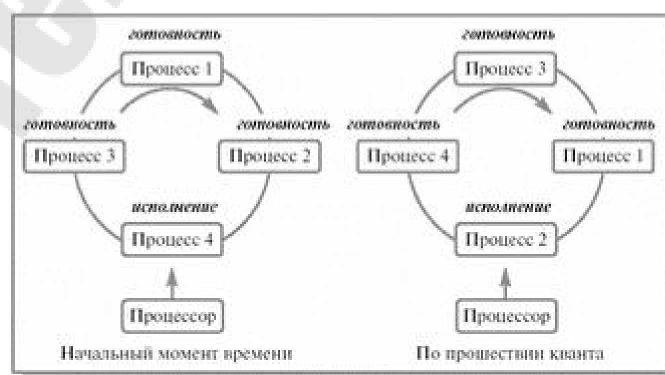

Рисунок 1 - Процессы на карусели

Библиотека ГГТУ им. П.О.Сухого

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта.

Время непрерывного использования процессора, необходимое процессу (остаток текущего CPU burst), меньше или равно продолжительности кванта времени. Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет кванта заново.

Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

На производительность алгоритма RR сильно влияет величина кванта времени. При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой CPU burst до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуальном процессоре с производительностью ~ 1/n от производительности реального процессора. Правда, это справедливо лишь при теоретическом анализе при условии пренебрежения временами переключения контекста процессов. В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

Shortest-Job-First (SJF)

Библиотека ГГТУ им. П.О. Сухого

При рассмотрении алгоритмов FCFS и RR мы видели, насколько существенным для них является порядок расположения процессов в очереди процессов, готовых к исполнению. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если бы мы знали время следующих CPU burst для процессов, находящихся в состоянии готовность, то могли бы выбрать для исполнения не процесс из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный алгоритм получил название "кратчайшая работа первой" или Shortest Job First (SJF).

SJF-алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF-планировании процессор предоставляется избранному процессу на все необходимое ему время, независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF-планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Основную сложность при реализации алгоритма SJF представляет невозможность точного знания продолжительности очередного CPU burst для исполняющихся процессов. В пакетных системах количество процессорного времени, необходимое заданию для выполнения, указывает пользователь при формировании задания.

Приоритетное планирование

Алгоритмы SJF и гарантированного планирования представляют собой частные случаи приоритетного планирования. При

Библиотека ГГТУ им. П.О.Сухого

приоритетном планировании каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс. Для алгоритма гарантированного планирования приоритетом служит вычисленный коэффициент справедливости. Чем он меньше, тем больше у процесса приоритет.

Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов.

Практическая часть:

Задание 1. Не вытесняющие алгоритмы планирования процессов

Выполнить различные алгоритмы планирований – First-Come, First-Served (FCFS) (прямой и обратный), Round Robin (RR), Shortest-Job-First (SJF) (не вытесняющий), Shortest-Job-First (SJF) (не вытесняющий приоритетный) для данных приведенных в таблице 1 в соответствии со своим вариантом (номер по журналу, **время появления в очереди не учитывать!!!**). Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

Задание 2 Вытесняющие алгоритмы планирования процессов

Библиотека ГГТУ им.П.О.Сухого

Выполнить различные алгоритмы планирований – Shortest-Job-First (SJF) (вытесняющий) и Shortest-Job-First (SJF) (приоритетный) для данных приведенных в таблице 1 в соответствии со своим вариантом. Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

Таблица 6.2 Варианты заданий

Вариант	Продолжительности процессов	Время появления в очереди	Приоритеты процессов
1	P0 – 5; P1 – 8; P2 – 2; P3 – 4;	P0 – 0; P1 – 3; P2 – 1; P3 – 4;	P0 – 2; P1 – 1; P2 – 3; P3 – 3;
2	P0 – 10; P1 – 3; P2 – 3; P3 – 7;	P0 – 2; P1 –0 ; P2 – 4; P3 –5;	P0 – 1; P1 – 3; P2 – 3; P3 – 2;
3	P0 – 5; P1 – 8; P2 – 2; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 –0;	P0 – 1; P1 – 2; P2 – 3; P3 – 1;
4	P0 – 2; P1 – 5; P2 – 9; P3 – 3;	P0 – 0; P1 – 4; P2 – 6; P3 – 8;	P0 – 1; P1 – 3; P2 – 3; P3 – 2;
5	P0 – 7; P1 – 8; P2 – 3; P3 – 4;	P0 – 4; P1 – 0; P2 – 3; P3 – 5;	P0 – 1; P1 – 2; P2 – 3; P3 – 4;
6	P0 – 9; P1 – 2; P2 – 2; P3 – 3;	P0 – 3; P1 – 0; P2 – 0; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 – 3;
7	P0 – 10; P1 – 2; P2 – 3; P3 – 1;	P0 – 0; P1 – 4; P2 – 3; P3 – 0;	P0 – 1; P1 – 3; P2 – 2; P3 – 3;
8	P0 – 2; P1 – 3; P2 – 1; P3 – 8;	P0 – 4; P1 – 4; P2 – 0; P3 – 0;	P0 – 3; P1 – 1; P2 – 3; P3 – 4;
9	P0 – 3; P1 – 5; P2 – 2; P3 – 7;	P0 – 3; P1 – 0; P2 – 1; P3 – 7;	P0 – 1; P1 – 3; P2 – 4; P3 – 1;
10	P0 – 7; P1 – 1; P2 – 4; P3 – 6;	P0 – 1; P1 – 2; P2 – 4; P3 – 0;	P0 – 2; P1 – 1; P2 – 3; P3 – 4;
11	P0 – 3; P1 – 1; P2 – 5; P3 – 6;	P0 – 3; P1 – 4; P2 – 4; P3 – 0;	P0 – 1; P1 – 1; P2 – 3; P3 – 3;
12	P0 – 8; P1 – 3; P2 – 10; P3 – 2;	P0 – 8; P1 – 4; P2 – 0; P3 – 3;	P0 – 2; P1 – 2; P2 – 1; P3 – 4;
13	P0 – 3; P1 – 7; P2 – 1; P3 – 3;	P0 – 4; P1 – 0; P2 – 0; P3 – 0;	P0 – 4; P1 – 3; P2 – 3; P3 – 2;
14	P0 – 4; P1 – 7; P2 – 6; P3 – 3;	P0 – 4; P1 – 4; P2 – 0; P3 – 7;	P0 – 1; P1 – 4; P2 – 3; P3 – 3;
15	P0 – 1; P1 – 3; P2 – 2; P3 – 7;	P0 – 0; P1 – 4; P2 – 6; P3 – 4;	P0 – 2; P1 – 1; P2 – 4; P3 – 1;
16	P0 – 5; P1 – 3; P2 – 4; P3 – 2;	P0 – 0; P1 – 4; P2 – 8; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 – 1;
17	P0 – 2; P1 – 4; P2 – 1; P3 – 6;	P0 – 3; P1 – 0; P2 – 6; P3 – 7;	P0 – 3; P1 – 4; P2 – 1; P3 – 3;
18	P0 – 3; P1 – 6; P2 – 3; P3 – 1;	P0 – 4; P1 – 1; P2 – 1; P3 – 0;	P0 – 1; P1 – 3; P2 – 3; P3 – 1;
19	P0 – 10; P1 – 2; P2 – 5; P3 – 3;	P0 – 11; P1 – 7; P2 – 3; P3 – 0;	P0 – 3; P1 – 3; P2 – 1; P3 – 2;
20	P0 – 8; P1 – 3; P2 – 2; P3 –3 ;	P0 – 0; P1 – 4; P2 – 7; P3 – 3;	P0 – 2; P1 – 2; P2 – 3; P3 – 1;
21	P0 – 3; P1 – 6; P2 – 4; P3 – 2;	P0 – 7; P1 – 0; P2 – 5; P3 – 3;	P0 – 4; P1 – 2; P2 – 4; P3 – 1;
22	P0 – 7; P1 – 6; P2 – 5; P3 – 3;	P0 – 0; P1 – 4; P2 – 0; P3 – 4;	P0 – 1; P1 – 3; P2 – 4; P3 – 2;
23	P0 – 3; P1 – 7; P2 – 4; P3 – 2;	P0 – 6; P1 – 0; P2 – 0; P3 – 7;	P0 – 3; P1 – 1; P2 – 2; P3 – 1;
24	P0 – 2; P1 – 5; P2 – 1; P3 – 7;	P0 – 1; P1 – 8; P2 – 0; P3 – ;	P0 – 2; P1 – 1; P2 – 2; P3 – 3;
25	P0 – 3; P1 – 4; P2 – 7; P3 – 1;	P0 – 3; P1 – 4; P2 – 4; P3 – 0;	P0 – 1; P1 – 1; P2 – 2; P3 – 2;

Примечания:

Для алгоритма Round Robin (RR) величина кванта времени 3 для всех вариантов.

Библиотека БГУИР

Для приоритетных алгоритмов меньшее значение соответствует более высокому приоритету.

Задание 3 Разработать программную реализацию алгоритмов задания 1 и 2. Сравнить полученные результаты и сделать выводы.

Требования по содержанию отчета:

В отчете должны быть отображены следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задания.
4. Таблицы, отображающие алгоритмы планирования.
5. Исходные тексты программ, реализующие алгоритмы планирования.
6. Результаты выполнения программ.
7. Выводы.

Контрольные вопросы для защиты:

1. Для чего предназначено планирование процессов?
2. В чем разница вытесняющих и не вытесняющих алгоритмов планирования?
3. Опишите алгоритм FIFS?
4. Опишите алгоритм RR?
5. Опишите алгоритм SJF?
6. Опишите алгоритм приоритетного планирования?
7. Подсчитать среднее время выполнения (ожидания) процессов для алгоритма FIFS?
8. Подсчитать среднее время выполнения (ожидания) процессов для алгоритма RR?
9. Подсчитать среднее время выполнения (ожидания) процессов для алгоритма SJF?
10. Подсчитать среднее время выполнения (ожидания) процессов для алгоритма приоритетного планирования?

Библиотека ГГТУ им.П.О.Сухого

Лабораторная работа № 7
Синхронизация процессов

Цель: изучить типовые механизмы синхронизации процессов

Теоретические сведения:

Синхронизация процессов имеет важное значение, при использовании различными процессами одних и тех же ресурсов.

Задачу упорядоченного доступа к разделяемым данным можно решить, если обеспечить каждому процессу эксклюзивное право доступа к этим данным. Каждый процесс, обращающийся к разделяемым ресурсам, исключает для всех других процессов возможность одновременного общения с этими ресурсами, если это может привести к недетерминированному поведению набора процессов. Такой прием называется **взаимоисключением (mutual exclusion)**. Если очередность доступа к разделяемым ресурсам важна для получения правильных результатов, то одними взаимоисключениями уже не обойтись, нужна **взаимосинхронизация** поведения программ.

Важным понятием при изучении способов синхронизации процессов является понятие **критической секции (critical section)** программы. Критическая секция – это часть программы, исполнение которой может привести к возникновению **race condition** для определенного набора программ. Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо организовать работу так, чтобы в каждый момент времени только один процесс мог находиться в своей критической секции, связанной с этим ресурсом. Иными словами, необходимо обеспечить реализацию взаимоисключения для критических секций программ. Реализация взаимоисключения для критических секций программ с практической точки зрения означает, что по отношению к другим процессам, участвующим во взаимодействии, критическая секция начинает выполняться как атомарная операция.

Библиотека ГГТУ им. П.О. Сухого

Переменная-замок

В качестве объекта синхронизация используется некоторая переменная, доступная всем процессам, с начальным значением равным 0. Процесс может войти в критическую секцию только тогда, когда значение этой переменной-замка равно 0, одновременно изменяя ее значение на 1 – закрывая замок. При выходе из критической секции процесс сбрасывает ее значение в 0 – замок открывается.

```
shared int lock = 0;
/* shared означает, что */
/* переменная является разделяемой */

while (some condition) {
while(lock); lock = 1;
critical section
lock = 0;
remainder section
}
```

Строгое чередование

Данный алгоритм использует общую переменную с начальным значением 0. Только теперь она играть не роль замка для критического участка, а явно указывает, какой процесс может следующим войти в него. Для i-го процесса это выглядит так:

```
shared int turn = 0;
while (some condition) {
while(turn != i);
critical section
turn = 1-i;
```


Библиотека ГГТУ им.П.О.Сухого

```
remainder section
}
```

Флаги готовности

Недостаток предыдущего алгоритма заключается в том, что процессы ничего не знают о состоянии друг друга в текущий момент времени. Давайте попробуем исправить эту ситуацию. Пусть два наших процесса имеют разделяемый массив флагов готовности входа процессов в критический участок

```
shared int ready[2] = {0, 0};
```

Когда i-й процесс готов войти в критическую секцию, он присваивает элементу массива ready[i] значение равное 1. После выхода из критической секции он, естественно, сбрасывает это значение в 0. Процесс не входит в критическую секцию, если другой процесс уже готов к входу в критическую секцию или находится в ней.

```
while (some condition) {
ready[i] = 1;
while(ready[1-i]);
critical section
ready[i] = 0;
remainder section
}
```

Программирование потоков

В Linux каждый поток на самом деле является процессом, и для того, чтобы создать новый поток, нужно создать новый процесс. Однако для создания дополнительных потоков используются процессы особого типа. Эти процессы представляют собой обычные

Библиотека ГГТУ им.П.О.Сухого

дочерние процессы главного процесса, но они разделяют с главным процессом адресное пространство, файловые дескрипторы и обработчики сигналов. Для обозначения процессов этого типа применяется специальный термин — **легкие процессы (lightweight processes)**. Поскольку для потоков не требуется создавать собственную копию адресного пространства (и других ресурсов) своего процесса-родителя, создание нового легкого процесса требует значительно меньших затрат, чем создание полновесного дочернего процесса.

Для работы с потоками используются следующие основные функции:

- **pthread_create** — создание потока;
- **pthread_join** — блокирование работы вызвавшего функцию процесса или потока в ожидании завершения потока;
- **pthread_cancel** — досрочное завершение потока из другого потока или процесса;
- **pthread_exit** — завершает поток, код завершения передается функции pthread_join. Данная функция подобна функции exit, однако вызов exit в «основном» процессе программы приведет к завершению всей программы.

Синхронизация потоков

При выполнении нескольких потоков во многих случаях необходимо синхронизировать их взаимодействие. Существует несколько способов синхронизации потоков:

- взаимные исключения — мьютексы;
- переменные состояния;
- семафоры.

Механизм мьютексов представляет общий метод синхронизации выполнения потоков. Мьютекс можно определить как объект синхронизации, который устанавливается в особое сигнальное

Библиотека ГГТУ им.П.О.Сухого

состояние, когда не занят каким-либо потоком. В любой момент мьютексом может владеть только один поток.

Использование мьютексов гарантирует, что только один поток в некоторый момент времени выполняет критическую секцию кода. Мьютексы можно использовать и в однопоточном коде.

Доступны следующие действия с мьютексом: инициализация, удаление, захват или открытие, попытка захвата.

Для синхронизации потоков с использованием мьютексов используются следующие основные функции:

- **pthread_mutex_init** - инициализирует взаимоисключающую блокировку;
- **pthread_mutex_destroy** - удаляет взаимоисключающую блокировку;
- **pthread_mutex_lock** - устанавливает блокировку. В случае, если блокировка была установлена другим потоком, текущий поток останавливается до снятия блокировки другим процессом;
- **pthread_mutex_unlock**— снимает блокировку.

Семафор предназначен для синхронизации потоков по действиям и данным, и в общем случае способ использования семафора сходен со способом использования мьютексов.

Семафор (S) — это защищенная переменная, значения которой можно опрашивать и менять только при помощи специальных операций P(S) и V(S) и операции инициализации. Семафор может принимать целое неотрицательное значение. При выполнении потоком операции P над семафором S значение семафора уменьшается на 1 при $S > 0$ или поток блокируется, «ожидая на семафоре», при $S = 0$. При выполнении операции V(S) происходит пробуждение одного из потоков, ожидающих на семафоре S, а если таковых нет, то значение семафора увеличивается на 1. В простом случае, когда семафор работает в режиме 2-х состояний ($S > 0$ и $S =$ О), его алгоритм работы полностью совпадает с алгоритмом мьютекса.

Библиотека ГГТУ им.П.О.Сухого

Как следует из вышесказанного, при входе в критическую секцию поток должен выполнять операцию P(S), а при выходе из критической секции операцию V(S).

Прототипы функций для манипуляции с семафорами описываются в файле <semaphore.h>. Ниже приводятся прототипы функций вместе с пояснением их синтаксиса и выполняемых ими действий:

- **int sem_init(sem_t* sem, int pshared, unsigned int value)** — инициализация семафора sem значением value. В качестве pshared всегда необходимо указывать 0;
- **int sem_wait(sem_t* sem)** — «ожидание на семафоре». Выполнение потока блокируется до тех пор, пока значение семафора не станет положительным. При этом значение семафора уменьшается на 1;
- **int sem_post(sem_t* sem)** — увеличивает значение семафора sem на 1;
- **int sem_destroy(sem_t* sem)** — уничтожает семафор sem;
- **int sem_trywait(sem_t* sem)** — неблокирующий вариант функции sem wait. При этом вместо блокировки вызвавшего потока функция возвращает управление с кодом ошибки в качестве результата работы.

Условная переменная позволяет потокам ожидать выполнения некоторого условия (события), связанного с разделяемыми данными. Над условными переменными определены две основные операции: информирование о наступлении события и ожидание события. При выполнении операции «информирование» один из потоков, ожидающих значения условной переменной, возобновляет свою работу.

Условная переменная всегда используется совместно с мьютексом. Перед выполнением операции «ожидание» поток должен заблокировать мьютекс. При выполнении операции «ожидание» указанный мьютекс автоматически разблокируется. Перед

Библиотека ГГТУ им. П.О. Сухого

возобновлением ожидающего потока выполняется автоматическая блокировка мьютекса, позволяющая потоку войти в критическую секцию, после критической секции рекомендуется разблокировать мьютекс. При подаче сигнала другим потокам рекомендуется функцию «сигнализации» так же защитить мьютексом.

Прототипы функций для работы с условными переменными содержатся в файле pthread.h. Ниже приводятся прототипы функций вместе с пояснением их синтаксиса и выполняемых ими действий:

- **pthread_cond_init(pthread_cond_t* cond, const pthread_condattr_t* attr)** - инициализирует условную переменную cond с указанными атрибутами attr или с атрибутами по умолчанию (при указании 0 в качестве attr);
- **int pthread_cond_destroy(pthread_cond_t* cond)** — уничтожает условную переменную cond;
- **int pthread_cond_signal(pthread_cond_t* cond)** — информирование о наступлении события потоков, ожидающих на условной переменной cond;
- **int pthread_cond_broadcast(pthread_cond_t* cond)** — информирование о наступлении события потоков, ожидающих на условной переменной cond. При этом возобновлены будут все ожидающие потоки;
- **int pthread_cond_wait(pthread_cond_t* cond, pthread_mutex_t* mutex)** — ожидание события на условной переменной cond.

Рассмотренных средств достаточно для решения разнообразных задач синхронизации потоков.

Практическая часть

1. Разработать многопоточное приложение с использованием минимум двух потоков и различных средств синхронизации.

Таблица 1 Варианты заданий

Библиотека ГГТУ им. П.О. Сухого

№ варианта	Разделяемый ресурс	Механизм синхронизации
1.	Два потока записывают и читают информацию из одного файла	мьютексы
2.	Два потока записывают и читают информацию из одного файла	семафоры
3.	Два потока записывают и читают информацию из одного файла	условные переменные
4.	Два потока увеличивают значение общей переменной	мьютексы
5.	Два потока увеличивают значение общей переменной	семафоры
6.	Два потока увеличивают значение общей переменной	условные переменные
7.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	мьютексы
8.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	семафоры
9.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	условные переменные

Библиотека ГГТУ им. П.О.Сухого

Продолжение

10.	Два потока записывают и читают информацию из одного файла	мьютексы
11.	Два потока записывают и читают информацию из одного файла	семафоры
12.	Два потока записывают и читают информацию из одного файла	условные переменные
13.	Два потока увеличивают значение общей переменной	мьютексы
14.	Два потока увеличивают значение общей переменной	семафоры
15.	Два потока увеличивают значение общей переменной	условные переменные
16.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	мьютексы
17.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	семафоры
18.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	условные переменные
19.	Два потока записывают и читают информацию из одного файла	мьютексы

Библиотека ГГТУ им. П.О. Сухого

Окончание табл.

20.	Два потока записывают и читают информацию из одного файла	семафоры
21.	Два потока записывают и читают информацию из одного файла	условные переменные
22.	Два потока увеличивают значение общей переменной	мьютексы
23.	Два потока увеличивают значение общей переменной	семафоры
24.	Два потока увеличивают значение общей переменной	условные переменные
25.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	мьютексы
26.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	семафоры
27.	Два потока с различной частотой считывают и записывают данные в общий буфер памяти	условные переменные

2. Время входа в критическую секцию для каждого потока генерировать случайным образом.

3. В процессе работы приложение в консоль должна выводится информация о состоянии потока (работа в некритической секции,

Библиотека ГГТУ им.П.О.Сухого

работа в критической секции, время входа и выхода из критической секции).

4. Убедиться в результативности применения средств синхронизации потоков, сравнив результаты работы программ с использованием и без использования средств синхронизации.

Требование по содержанию отчета:

В отчете должны быть отображены следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задание.
4. Исходные тексты программ.
5. Результаты выполнения.
6. Выводы.

Контрольные вопросы для защиты:

1. Охарактеризовать проблему синхронизации потоков.
2. Определение потока в Linux?
3. Основные алгоритмы синхронизации потоков?
4. Механизмы синхронизации потоков в Linux?
5. Порядок действий при использовании в качестве объекта синхронизации мьютекса?
6. Порядок действий при использовании в качестве объекта синхронизации семафора?
7. Порядок действий при использовании в качестве объекта условных переменных?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 8
Реализация файловой системы

Цель работы: разработать модель файловой системы.

Теоретические сведения:

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы организовать эффективную работу с данными, хранящимися во внешней памяти, и обеспечить пользователю удобный интерфейс при работе с такими данными.

Алгоритмы выделения дискового пространства

Главный вопрос, какой тип структур используется для учета отдельных блоков файла, то есть способ связывания файлов с блоками диска. В ОС используется несколько методов выделения файлу дискового пространства. Для каждого из методов запись в директории, соответствующая символьному имени файла, содержит указатель, следуя которому можно найти все блоки данного файла.

Выделение непрерывной последовательностью блоков

Простейший способ - хранить каждый файл как непрерывную последовательность блоков диска. При непрерывном расположении файл характеризуется адресом и длиной (в блоках). Файл, стартующий с блока b, занимает затем блоки b+1, b+2, ... b+n-1.

Эта схема имеет два преимущества. Во-первых, ее легко реализовать, так как выяснение местонахождения файла сводится к вопросу, где находится первый блок. Во-вторых, она обеспечивает хорошую производительность, так как целый файл может быть считан за одну дисковую операцию.

Связный список

Внешняя фрагментация - основная проблема рассмотренного выше метода - может быть устранена за счет представления файла в виде связного списка блоков диска. Запись в директории содержит указатель на первый и последний блоки файла (иногда в качестве

Библиотека ГГТУ им. П.О. Сухого

варианта используется специальный знак конца файла - EOF). Каждый блок содержит указатель на следующий блок (см. рисунок 1).

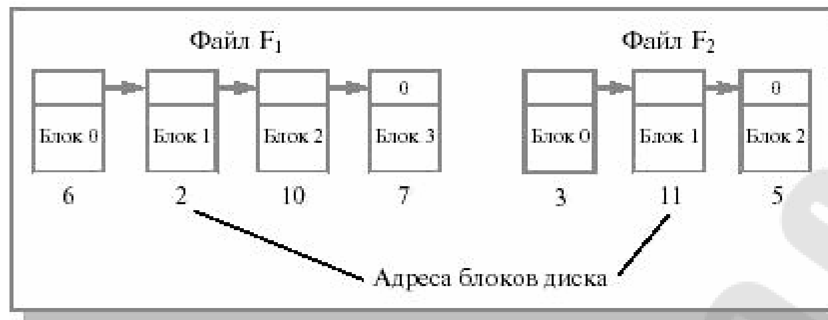

Рисунок 8.1 - Хранение файла в виде связного списка дисковых блоков

Внешняя фрагментация для данного метода отсутствует. Любой свободный блок может быть использован для удовлетворения запроса. Заметим, что нет необходимости декларировать размер файла в момент создания. Файл может расти неограниченно.

Связное выделение имеет, однако, несколько существенных недостатков:

- при прямом доступе к файлу для поиска i-го блока нужно осуществить несколько обращений к диску, последовательно считывая блоки от 1 до i-1, то есть выборка логически смежных записей, которые занимают физически несмежные секторы, может требовать много времени. Здесь мы теряем все преимущества прямого доступа к файлу.
- данный способ не очень надежен. Наличие дефектного блока в списке приводит к потере информации в оставшейся части файла и потенциально к потере дискового пространства, отведенного под этот файл.
- для указателя на следующий блок внутри блока нужно выделить место, что не всегда удобно. Емкость блока, традиционно являющаяся степенью двойки (многие программы читают и пишут блоками по степеням двойки), таким образом, перестает быть степенью двойки, так как указатель отбирает несколько байтов.

Поэтому метод связного списка обычно в чистом виде не используется.

Таблица отображения файлов

Библиотека ГГТУ им.П.О.Сухого

Одним из вариантов предыдущего способа является хранение указателей не в дисковых блоках, а в индексной таблице в памяти, которая называется таблицей отображения файлов (FAT - file allocation table) (см. рисунок 2). Этой схемы придерживаются многие ОС (MS-DOS, OS/2, Windows 9x и др.)

По-прежнему существенно, что запись в директории содержит только ссылку на первый блок. Далее при помощи таблицы FAT можно локализовать блоки файла независимо от его размера. В тех строках таблицы, которые соответствуют последним блокам файлов, обычно записывается некоторое граничное значение, например EOF. Главное достоинство данного подхода состоит в том, что по таблице отображения можно судить о физическом соседстве блоков, располагающихся на диске, и при выделении нового блока можно легко найти свободный блок диска, находящийся поблизости от других блоков данного файла. Минусом данной схемы может быть необходимость хранения в памяти этой довольно большой таблицы.

Номера блоков диска		
1		
2	10	
3	11	Начало файла F_2
4		
5	EOF	
6	2	Начало файла F_1
7	EOF	
8		
9		
10	7	
11	5	

Рисунок 8.2 - Метод связного списка с использованием таблицы в оперативной памяти

Индексные узлы

Наиболее распространенный метод выделения файлу блоков диска - связать с каждым файлом небольшую таблицу, называемую индексным узлом (i-node), которая перечисляет атрибуты и дисковые адреса блоков файла (см. рисунок 3). Запись в директории, относящаяся к файлу, содержит адрес индексного блока. По мере

Библиотека ГГТУ им. П.О. Сухого

заполнения файла указатели на блоки диска в индексном узле принимают осмысленные значения.

Индексирование поддерживает прямой доступ к файлу, без ущерба от внешней фрагментации. Индексированное размещение широко распространено и поддерживает как последовательный, так и прямой доступ к файлу.

Обычно применяется комбинация одноуровневого и многоуровневых индексов. Первые несколько адресов блоков файла хранятся непосредственно в индексном узле, таким образом, для маленьких файлов индексный узел хранит всю необходимую информацию об адресах блоков диска. Для больших файлов один из адресов индексного узла указывает на блок косвенной адресации. Данный блок содержит адреса дополнительных блоков диска. Если этого недостаточно, используется блок двойной косвенной адресации, который содержит адреса блоков косвенной адресации. Если и этого не хватает, используется блок тройной косвенной адресации.

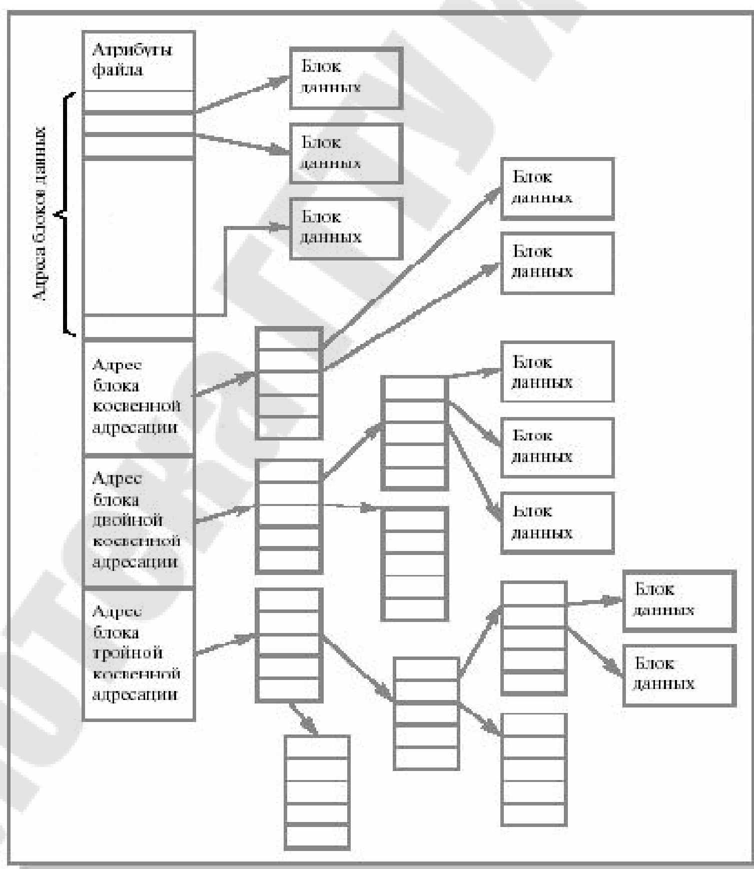

Рисунок 8.3 - Структура индексного узла

Библиотека ГГТУ им. П.О. Сухого

Данную схему используют файловые системы Unix (а также файловые системы HPFS, NTFS и др.). Такой подход позволяет при фиксированном, относительно небольшом размере индексного узла поддерживать работу с файлами, размер которых может меняться от нескольких байтов до нескольких гигабайтов. Существенно, что для маленьких файлов используется только прямая адресация, обеспечивающая максимальную производительность.

Управление свободным и занятым дисковым пространством

Дисковое пространство, не выделенное ни одному файлу, также должно быть управляемым. В современных ОС используется несколько способов учета используемого места на диске. Рассмотрим наиболее распространенные.

Учет при помощи организации битового вектора

Часто список свободных блоков диска реализован в виде битового вектора (bit map или bit vector). Каждый блок представлен одним битом, принимающим значение 0 или 1, в зависимости от того, занят он или свободен.

Например, 00111100111100011000001 ... .

Главное преимущество этого подхода состоит в том, что он относительно прост и эффективен при нахождении первого свободного блока или n последовательных блоков на диске. Многие компьютеры имеют инструкции манипулирования битами, которые могут использоваться для этой цели.

Учет при помощи организации связного списка

Другой подход - связать в список все свободные блоки, размещая указатель на первый свободный блок в специально отведенном месте диска, попутно кэшируя в памяти эту информацию. Подобная схема не всегда эффективна. Для трассирования списка нужно выполнить много обращений к диску. Однако, к счастью, нам необходим, как правило, только первый свободный блок.

Иногда прибегают к модификации подхода связного списка, организуя хранение адресов n свободных блоков в первом свободном

Библиотека ГГТУ им. П.О. Сухого

блоке. Первые n-1 этих блоков действительно используются. Последний блок содержит адреса других n блоков и т. д.

Структура файловой системы на диске

Рассмотрение методов работы с дисковым пространством дает общее представление о совокупности служебных данных, необходимых для описания файловой системы. Структура служебных данных типовой файловой системы, например Unix, на одном из разделов диска, таким образом, может состоять из четырех основных частей (см. рисунок 8.4).

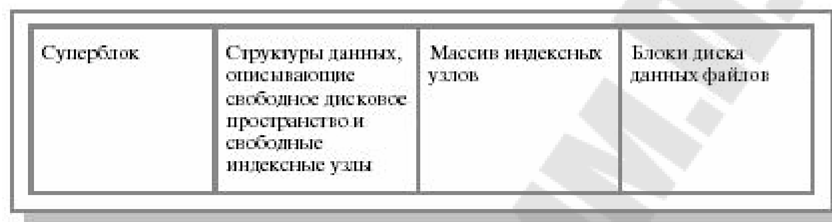

Рисунок 8.4 - Примерная структура файловой системы на диске

В начале раздела находится суперблок, содержащий общее описание файловой системы, например:

- тип файловой системы;
- размер файловой системы в блоках;
- размер массива индексных узлов;
- размер логического блока.

Описанные структуры данных создаются на диске в результате его форматирования (например, утилитами format, makefs и др.). Их наличие позволяет обращаться к данным на диске как к файловой системе, а не как к обычной последовательности блоков.

Поиск и директории

Список файлов в директории обычно не является упорядоченным по именам файлов. Поэтому правильный выбор алгоритма поиска имени файла в директории имеет большое влияние на эффективность и надежность файловых систем.

Библиотека ГГТУ им. П.О.Сухого

Линейный поиск

Существует несколько стратегий просмотра списка символьных имен. Простейшей из них является линейный поиск. Директория просматривается с самого начала, пока не встретится нужное имя файла. Хотя это наименее эффективный способ поиска, оказывается, что в большинстве случаев он работает с приемлемой производительностью. Например, авторы Unix утверждали, что линейного поиска вполне достаточно. По-видимому, это связано с тем, что на фоне относительно медленного доступа к диску некоторые задержки, возникающие в процессе сканирования списка, несущественны.

Хеш-таблица

Хеширование - другой способ, который может использоваться для размещения и последующего поиска имени файла в директории. В данном методе имена файлов также хранятся в каталоге в виде линейного списка, но дополнительно используется хеш-таблица. Хеш-таблица, точнее построенная на ее основе хеш-функция, позволяет по имени файла получить указатель на имя файла в списке. Таким образом, можно существенно уменьшить время поиска.

Практическая часть:

Разработать приложение, создающее виртуальный файл и позволяющее:

- форматировать виртуальный файл с возможностью задания размера кластера;
- создавать каталоги в виртуальном файле;
- производить учёт свободного пространства;
- реализовывать поиск файлов и директорий;
- сохранять в виртуальный файл файлы с жёсткого диска;
- удалять файлы из виртуального файла;
- записывать на жёсткий диск файлы из виртуального файла;
- создавать в виртуальном файле текстовые файлы;
- предоставлять возможность редактировать текстовые файлы внутри виртуального файла.

Библиотека БГУИР

Файловую систему внутри виртуального файла выбрать согласно варианта.

Таблица 8.1 - Варианты заданий

Вариант	Условие задачи	Учёт свободных блоков	Поиск файлов и папок
1	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Best Fit).	Связанный список	Хэш-таблицы
2	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Бинарный поиск
3	Одноуровневые индексные узлы.	Файловый	B-дерево
4	Индексно-последовательная файловая система. Таблица отображения файлов.	Связанный список	Линейный поиск
5	Многоуровневые индексные узлы.	Файловый	Линейный поиск
6	Индексно-последовательная файловая система. Связанный список.	Связанный список	Хэш-таблицы
7	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Worst Fit).	Связанный список	Бинарный поиск
8	Индексно-последовательная файловая система. Таблица отображения файлов.	Файловый	Бинарный поиск
9	Одноуровневые индексные узлы.	Битовый вектор	Бинарный поиск

Библиотека ГГТУ им.П.О.Сухого

Продолжение табл.

10	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Хэш-таблицы
11	Многоуровневые индексные узлы.	Битовый вектор	Линейный поиск
12	Последовательная файловая система. Выделение непрерывной последовательности блоков. (First Fit).	Связанный список	Хэш-таблицы
13	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	B-дерево
14	Одноуровневые индексные узлы.	Связанный список	Хэш-таблицы
15	Индексно-последовательная файловая система. Таблица отображения файлов.	Битовый вектор	Хэш-таблицы
16	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Best Fit).	Связанный список	Хэш-таблицы
17	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Бинарный поиск
18	Одноуровневые индексные узлы.	Файловый	B-дерево
19	Индексно-последовательная файловая система. Таблица отображения файлов.	Связанный список	Линейный поиск
20	Многоуровневые индексные узлы.	Файловый	Линейный поиск
21	Индексно-последовательная файловая система. Связанный список.	Связанный список	Хэш-таблицы

Библиотека ГГТУ им. П.О. Сухого

Окончание табл.

22	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Worst Fit).	Связанный список	Бинарный поиск
23	Индексно-последовательная файловая система. Таблица отображения файлов.	Файловый	Бинарный поиск
24	Одноуровневые индексные узлы.	Битовый вектор	Бинарный поиск
25	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Хэш-таблицы
26	Многоуровневые индексные узлы.	Битовый вектор	Линейный поиск
27	Последовательная файловая система. Выделение непрерывной последовательности блоков. (First Fit).	Связанный список	Хэш-таблицы
28	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	В-дерево
29	Одноуровневые индексные узлы.	Связанный список	Хэш-таблицы
30	Индексно-последовательная файловая система. Таблица отображения файлов.	Битовый вектор	Хэш-таблицы

Требования по содержанию отчета:

В отчете должны быть отображены следующие пункты:
1. Титульный лист.
2. Цель работы.
3. Задание.
4. Исходный код программного кода.

Библиотека ГГТУ им.П.О.Сухого

5. Результаты работы приложения для пазличных операций
6. Выводы.

Контрольные вопросы для защиты:

1. Основные понятия файловый системы?
2. Операции над файлами?
3. Операции над директориями?
4. Способы организации файловой системы?
5. Как учитывается свободное пространство?
6. Как создается структура файловой системы?
7. Описать файловую систему, использующую таблицу отображения файлов (FAT)?
8. Описать файловую систему, использующую индексные узлы?
9. Способы поиска информации?

Библиотека ГГТУ им.П.О.Сухого

Литература

1. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – 3-е изд.. – Санкт-Петербург [и др.] : Питер, 2015. - 1115 с.

2. UNIX: руководство системного администратора / Э. Немет [и др.]. - 3-е изд.. - Санкт-Петербург : Питер, 2008. - 924 с.

3. Шамер, Л. UNIX : рук. пользователя / Л. Шамер; под ред. С. Каратыгина. – М. : БИНОМ, 1996. – 352 с.

4. Донцов, В. П. LINUX на примерах / В. П. Донцов, И. В. Сафин. – СПб. : Наука и техника, 2017. - 346 с.

5. Блум, Р. Командная строка Linux и сценарии оболочки : библия пользователя / Ричард Блум, Кристина Бреснахэн ; пер. с англ. и ред. К. А. Птицина. - 2-е изд.. - Москва [и др.] : Диалектика, 2013. - 784 с.

Библиотека ГГТУ им. П.О. Сухого

СОДЕРЖАНИЕ

Введение 3
Лабораторная работа № 1 Работа с интерфейсом командной строкой в ОС Linux 4
Лабораторная работа № 2 Администрирование в Linux. Управление пользователями 14
Лабораторная работа № 3 Программирование в Shell 22
Лабораторная работа № 4 Инструментальные средства разработки Linux 34
Лабораторная работа № 5 Знакомство со стандартной утилитой Gnu make для построения проектов в ОС Unix/Linux 49
Лабораторная работа № 6 Планирование процессов 64
Лабораторная работа № 7 Синхронизация процессов 71
Лабораторная работа № 8 Реализация файловой системы 82
Литература........................... **Ошибка! Закладка не определена.**

Библиотека ГГТУ им.П.О.Сухого

ОПЕРАЦИОННЫЕ СИСТЕМЫ И СРЕДЫ

Практикум
по выполнению лабораторных работ
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Составитель **Самовендюк** Николай Владимирович

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 11.05.23.
Рег. № 25Е.
http://www.gstu.by

Библиотека ГГТУ им. П.О.Сухого