Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Методические указания к лабораторным работам
и организации самостоятельной работы для студентов
направления подготовки
«Бизнес-информатика»
(уровень бакалавриата)

Томск – 2018

Гриценко Юрий Борисович

Операционные системы: Методические указания к лабораторным работам и организации самостоятельной работы для студентов направления подготовки «Бизнес-информатика» (уровень бакалавриата) / Ю.Б. Гриценко – Томск, 2018. – 88 с.

Оглавление

1 Введение

Целью дисциплины «Операционные системы» является формирование у студента профессиональных знаний по теоретическим основам построения и функционирования компьютеров вычислительных систем, телекоммуникационных вычислительных сетей и коммуникаций, их структурной и функциональной организации, программному обеспечению, эффективности и перспективам развития.

Задачи изучения дисциплины:

1) Изучение принципов построения, функционирования и внутренней архитектуры операционных систем, функциональность всех составных компонентов и механизмы их взаимодействия в одно- и многопроцессорных системах, методы работы с внешними интерфейсами операционных систем.

2) Овладевание способностью работать с компьютером как средством управления информацией, работать с информацией из различных источников, в том числе в глобальных компьютерных сетях.

2 Методические указания по проведению лабораторных работ

2.1 Лабораторная работа «Файлы пакетной обработки в ОС Windows»

Цель работы

Целью данной работы является:

- изучение назначения и основных возможностей командных файлов (Файлов пакетной обработки) операционных систем, построенных на платформе Windows NT;
- знакомство со специальными командами, используемыми в командных файлах;
- исследование стандартных потоков ввода-вывода и их перенаправление

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Язык командных файлов

Командный файл – это текстовый файл (в коде ASCII), состоящий из группы команд. Правила идентификации командных файлов совпадают с общими правилами идентификации файлов. Единственное исключение — командный файл всегда записывается на диск с расширением «.BAT» и/или «.CMD» (для операционных систем Windows на платформе NT).

Обратиться к командному файлу крайне просто. Набирается команда старта – имя файла, и нажимается клавиша Enter. После введения команды файл выбирается из рабочего каталога, указанного или рабочего диска. Если в рабочем каталоге его нет, то поиск файла будет производиться в каталогах, описанных системной переменной %PATH%. При нахождении файла первая из его команд загружается в память, отображается на экране и выполняется. Этот процесс повторяется последовательно для всех команд файла (от первой до последней команды).

Выполнение командного файла можно прервать в любой момент, нажав на клавиши Ctrl-Break (Ctrl-C).

Организация командного файла. Существует несколько способов организации командных файлов. Файл можно создать с помощью любого текстового редактора или введением команд непосредственно с клавиатуры. В этом случае ввод оформляется файлом и записывается на диск.

В сеансе DOS клавиатура называется «CON» (CONsole) Для организации файла используется команда «COPY CON:». Наберите команду и имя создаваемого файла. Например, для создания файла «SAMPLE.BAT» введите:

```
C:\>COPY CON: SAMPLE.BAT
```

После этого введите составляющие файл команды. Набрав последнюю команду, одновременно нажмите клавиши Ctrl-Z (или функциональную клавишу F6) и клавишу Enter.

Стандартные потоки ввода-вывода и перенаправление потоков. Термин CONsole используется для обозначения стандартных потоков ввода-вывода. Когда говорят о вводе с консоли, подразумевается ввод с клавиатуры. Когда говорят о выводе на консоль, подразумевают вывод на экран монитора. Существуют специальные символы для перенаправления стандартных потоков ввода-вывода.

> приемник — перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет создан заново).

>> приемник — перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет сохранен, а информация будет записана в конец файла).

< источник — перенаправить стандартный ввод из источника.

передатчик | приемник — передает вывод одной команды на вход другой.

Замещаемые параметры. Внутри командного файла допускается использование замещаемых параметров. Параметр — это символьная переменная, расположенная в командной строке после имени команды. Он содержит дополнительную информацию, необходимую операционной системе при обработке команды.

Параметром, например, может быть имя файла, к которому относится действие команды. Замещаемый параметр — это специальная переменная, которая в процессе выполнения команды подменяется обычным параметром (например, именем файла). В командном файле замещаемый параметр обозначается знаком процента % и цифрой от 0 до 9. Таким образом, командный файл может включать до десяти замещаемых параметров. Символьные переменные, предназначенные для подмены замещающего параметра, вводятся в командной строке при обращении к командному файлу — набирается команда старта (имя файла) и список параметров в порядке, соответствующем последовательности замещаемых параметров внутри файла.

Параметры заменяются в порядке следования символьных переменных в командной строке. Первая переменная подменяет параметр %1, вторая – параметр %2 и т.д. Вместо замещаемого параметра %0 автоматически подставляется спецификация (имя) командного файла.

При введении замещаемых параметров командный файл становится более гибким. Поясним это на примере. Предположим, что на диске имеется несколько файлов, которые нужно копировать после каждой корректировки. В рассмотренном выше примере командный файл использовался для копирования конкретного файла. Этим же командным файлом можно воспользоваться и для копирования любого файла. В этом случае вместо имени копируемого файла подставляется замещаемый параметр. Имя копируемого файла будет вводиться в командной строке при обращении к командному файлу.

Назовем наш командный файл «COPYALL.BAT». Введем в нем:

```
COPY %1 A:
```

При обращении к файлу набирается его имя и через пробел – имя копируемого файла (в нашем примере «SHOPLIST.DOC»). Введите команду:

```
C:\>COPYALL.BAT SHOPLIST.DOC
```

На экран выводится следующая команда:

```
C:\>COPY SHOPLIST.DOC A:
 1 File(s) copied
```

DOS автоматически подставила имя файла на место замещаемого параметра %1. Усложним пример. Организуем командный файл «DIFNUM.BAT», автоматически копирующий любой указанный файл и присваивающий копии любое указанное имя:

```
COPY %1 A:%2
```

Для обращения к этому файлу наберите его имя, имя копируемого файла, в нашем примере «NEW.DOC», и имя копии «OLD.DOC»:

```
C:\>DIFNUM NEW.DOC OLD.DOC
```

На экране появляется следующая команда файла «DIFNUM.BAT»:

```
C:\>COPY NEW.DOC A:OLD.DOC
 1 File(s) copied
```

Первое имя в командной строке «NEW.DOC» поставлено вместо замещаемого параметра %1. Второе имя «OLD.DOC» – вместо замещаемого параметра %2.

Замещаемые параметры и замещаемые символы. Параметр в командной строке команды старта командного файла может включать замещаемые символы «?» и «*». Если замещаемый символ вводится для обозначения группы параметров, то команда выполняется по количеству параметров в группе (т.е. один раз для каждого параметра). Рассмотрим командный файл:

```
COPY %1 CON:
```

Этот файл копирует на экран (CON) файл, описанный замещаемым параметром %1 (DISPLAY.BAT). Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если указанный файл найден, его содержимое выводится на экран.

Этот файл копирует на экран (con) файл, описанный замещаемым параметром %1. Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если

указанный файл найден, его содержимое выводится на экран. Итак, командный файл «DISPLAY.BAT» записан на диск. Введем команду:

C:\>DISPLAY *.TXT

Все файлы рабочего диска с соответствующей спецификацией будут выведены на экран. Если имя копируемого файла включает обозначение процента, то при введении его в командную строку знак процента набирается два раза подряд. Например, имя «HIHO%.TXT» в командной строке должно быть представлено как «HIHO%%.TXT».

Некоторые команды DOS (Windows)

Для получения полного списка команд DOS, поддерживаемых вашей операционной системой Windows, построенной на платформе NT, необходимо ввести команду[1]:

HELP

Вот ее возможный результат:

Для получения сведений об определенной команде наберите HELP <имя команды>

ASSOC — Вывод либо изменение сопоставлений по расширениям имен файлов.

AT — Выполнение команд и запуск программ по расписанию.

ATTRIB — Отображение и изменение атрибутов файлов.

BREAK — Включение/выключение режима обработки комбинации клавиш CTRL+C.

CACLS — Отображение/редактирование списков управления доступом (ACL) к файлам.

CALL — Вызов одного пакетного файла из другого.

CD — Вывод имени либо смена текущей папки.

CHCP — Вывод либо установка активной кодовой страницы.

CHDIR — Вывод имени либо смена текущей папки.

CHKDSK — Проверка диска и вывод статистики.

[1] Синтаксис представлен для ОС Windows построенной на базе технологии NT, в ОС MS-DOS и Windows 9x количество аргументов и команд несколько меньше.

CHKNTFS— Отображение или изменение выполнения проверки диска во время загрузки.

CLS — Очистка экрана.

CMD — Запуск еще одного интерпретатора командных строк Windows.

COLOR — Установка цвета текста и фона, используемых по умолчанию.

COMP — Сравнение содержимого двух файлов или двух наборов файлов.

COMPACT— Отображение/изменение сжатия файлов в разделах NTFS.

CONVERT— Преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска.

COPY — Копирование одного или нескольких файлов в другое место.

DATE — Вывод либо установка текущей даты.

DEL — Удаление одного или нескольких файлов.

DIR — Вывод списка файлов и подпапок из указанной папки.

DISKCOMP— Сравнение содержимого двух гибких дисков.

DISKCOPY—Копирование содержимого одного гибкого диска на другой.

DOSKEY — Редактирование и повторный вызов командных строк; создание макросов.

ECHO — Вывод сообщений и переключение режима отображения команд на экране.

ENDLOCAL— Конец локальных изменений среды для пакетного файла.

ERASE — Удаление одного или нескольких файлов.

EXIT — Завершение работы программы CMD.EXE (интерпретатора командных строк).

FC — Сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FIND — Поиск текстовой строки в одном или нескольких файлах.

FINDSTR— Поиск строк в файлах.

FOR — Запуск указанной команды для каждого из файлов в наборе.

FORMAT— Форматирование диска для работы с Windows.

FTYPE — Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.

GOTO — Передача управления в отмеченную строку пакетного файла.

GRAFTABL— Позволяет Windows отображать расширенный набор символов в графическом режиме.

HELP — Выводит справочную информацию о командах Windows.

IF — Оператор условного выполнения команд в пакетном файле.

LABEL — Создание, изменение и удаление меток тома для дисков.

MD — Создание папки.

MKDIR — Создание папки.

MODE — Конфигурирование системных устройств.

MORE — Последовательный вывод данных по частям размером в один экран.

MOVE — Перемещение одного или нескольких файлов из одной папки в другую.

PATH — Вывод либо установка пути поиска исполняемых файлов.

PAUSE — Приостановка выполнения пакетного файла и вывод сообщения.

POPD — Восстановление предыдущего значения текущей активной папки, сохраненного с помощью команды PUSHD.

PRINT — Вывод на печать содержимого текстовых файлов.

PROMPT — Изменение приглашения в командной строке Windows.

PUSHD — Сохранение значения текущей активной папки и переход к другой папке.

RD — Удаление папки.

RECOVER — Восстановление читаемой информации с плохого или поврежденного диска.

REM — Помещение комментариев в пакетные файлы и файл CONFIG.SYS.

REN — Переименование файлов и папок.

RENAME— Переименование файлов и папок.

REPLACE— Замещение файлов.

RMDIR — Удаление папки.

SET — Вывод, установка и удаление переменных среды Windows.

SETLOCAL— Начало локальных изменений среды для пакетного файла.

SHIFT — Изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SORT — Сортировка ввода.

START — Запуск программы или команды в отдельном окне.

SUBST — Сопоставляет заданному пути имя диска.

TIME — Вывод и установка системного времени.

TITLE — Назначение заголовка окна для текущего сеанса интерпретатора командных строк CMD.EXE.

TREE — Графическое отображение структуры папок заданного диска или заданной папки.

TYPE — Вывод на экран содержимого текстовых файлов.

VER — Вывод сведений о версии Windows.

VERIFY — Установка режима проверки правильности записи файлов на диск.

VOL — Вывод метки и серийного номера тома для диска.

XCOPY — Копирование файлов и дерева папок.

Чтобы получить информацию о какой-либо команде операционной системы можно также в командной строке набрать имя команды и через пробел указать знак /?. Например,

C:\>PAUSE /?

Далее приводится основной синтаксис некоторых команд, необходимых для выполнения лабораторной работы.

ECHO

ECHO [ON | OFF] — переключение режима отображения команд на экране.

ECHO [сообщение] — вывод сообщений.

Введите ECHO без параметра для определения текущего значения этой команды.

Введите ECHO. (с точкой) для получение пустой строки.

@ — знак экранирования. Отключает вывод на экран текущей строки.

GOTO — передача управления содержащей метку строке пакетного файла.

GOTO метка

метка — строка пакетного файла, оформленная как метка.

Метка должна находиться в отдельной строке и начинаться с двоеточия.

IF — оператор условного выполнения команд в пакетном файле.

IF [NOT] ERRORLEVEL число команда
IF [NOT] строка1==строка2 команда
IF [NOT] EXIST имя_файла команда

NOT — обращает истинность условия: истинное условие становится ложным, а ложное — истинным.

ERRORLEVEL число — условие является истинным, если код возврата последней выполненной программы не меньше указанного числа.

строка1==строка2 — это условие является истинным, если указанные строки совпадают.
IF (%1)==() — проверка на пустой параметр.

EXIST имя_файла — это условие является истинным, если файл с указанным именем существует.
команда — задает команду, выполняемую при истинности условия. За этой командой может следовать ключевое слово ELSE, служащее для указания команды, которая должна выполняться в том случае, если условие ложно.

Предложение ELSE должно располагаться в той же строке, что и команда, следующая за ключевым словом IF. Например:

```
IF EXIST имя_файла. (
del имя_файла.
```

) ELSE (
echo имя_файла. missing.
)

Следующий пример содержит ОШИБКУ, поскольку команда del должна заканчиваться переходом на новую строку:

IF EXIST имя_файла. del имя_файла. ELSE echo имя_файла. missing

Следующий пример также содержит ОШИБКУ, поскольку команда ELSE должна располагаться в той же строке, что и команда, следующая за IF:

IF EXIST имя_файла. del имя_файла.
ELSE echo имя_файла. missing

Вот правильный пример, где все команды расположены в одной строке:

IF EXIST имя_файла. (del имя_файла.) ELSE echo имя_файла. missing

PAUSE — приостановка выполнения пакетного файла и вывод сообщения:
Для продолжения нажмите любую клавишу . . .

DIR — вывод списка файлов и подкаталогов из указанного каталога.

DIR [диск:][путь][имя_файла] [/A[[:]атрибуты]] [/B] [/C] [/D] [/L] [/N] [/O[[:]порядок]] [/P] [/Q] [/S] [/T[[:]время]] [/W] [/X] [/4]

[диск:][путь][имя_файла]
Диск, каталог и/или файлы, которые следует включить в список.

/A Вывод файлов с указанными атрибутами.
атрибуты:
- D Каталоги
- R Доступные только для чтения

- H Скрытые файлы
- A Файлы для архивирования
- S Системные файлы
- Префикс «-» имеет значение НЕ

/B Вывод только имен файлов.

/C Применение разделителя групп разрядов для вывода размеров файлов (по умолчанию). Для отключения этого режима служит ключ /-C.

/D Вывод списка в несколько столбцов с сортировкой по столбцам.

/L Использование нижнего регистра для имен файлов.

/N Отображение имен файлов в крайнем правом столбце.

/O Сортировка списка отображаемых файлов.

порядок:

- N По имени (алфавитная)
- S По размеру (сперва меньшие)
- E По расширению (алфавитная)
- D По дате (сперва более старые)
- G Начать список с каталогов
- Префикс «-» обращает порядок

/P Пауза после заполнения каждого экрана.

/Q Вывод сведений о владельце файла.

/S Вывод списка файлов из указанного каталога и его подкаталогов.

/T Выбор поля времени для отображения и сортировки

время:

- C Создание
- A Последнее использование
- W Последнее изменение

/W Вывод списка в несколько столбцов.

/X Отображение коротких имен для файлов, чьи имена не соответствуют стандарту 8.3. Формат аналогичен выводу с ключом /N, но короткие имена файлов выводятся слева от длинных. Если короткого имени у файла нет, вместо него выводятся пробелы.

/4 Вывод номера года в четырехзначном формате

Стандартный набор ключей можно записать в переменную среды DIRCMD. Для отмены их действия введите в команде те же ключи с префиксом «-», например: /-W.

MD — создание каталога.

MKDIR [диск:]путь
MD [диск:]путь

CD — вывод имени либо смена текущего каталога.

CHDIR [/D] [диск:][путь]
CHDIR [..]
CD [/D] [диск:][путь]
CD [..]

.. обозначает переход в родительский каталог.

Команда CD диск: отображает имя текущего каталога указанного диска.

Команда CD без параметров отображает имена текущих диска и каталога.

Параметр /D используется для одновременной смены текущих диска и каталога.

RD — удаление каталога.

RMDIR [/S] [/Q] [диск:]путь
RD [/S] [/Q] [диск:]путь

/S Удаление дерева каталогов, т. е. не только указанного каталога, но и всех содержащихся в нем файлов и подкаталогов.

/Q Отключение запроса подтверждения при удалении дерева каталогов с помощью ключа /S.

COPY — копирование одного или нескольких файлов в другое место.

COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/A | /B] источник [/A | /B]
[+ источник [/A | /B] [+ ...]] [результат [/A | /B]]

источник Имена одного или нескольких копируемых файлов.

/A Файл является текстовым файлом ASCII.

/B Файл является двоичным файлом.

/D Указывает на возможность создания зашифрованного файла результат Каталог и/или имя для конечных файлов.

/V Проверка правильности копирования файлов.

/N Использование, если возможно, коротких имен при копировании файлов, чьи имена не удовлетворяют стандарту 8.3.

/Y Подавление запроса подтверждения на перезапись существующего конечного файла.

/-Y Обязательный запрос подтверждения на перезапись существующего конечного файла.

/Z Копирование сетевых файлов с возобновлением.

Ключ /Y можно установить через переменную среды COPYCMD.

Ключ /-Y командной строки переопределяет такую установку.

По умолчанию требуется подтверждение, если только команда COPY не выполняется в пакетном файле.

Чтобы объединить файлы, укажите один конечный и несколько исходных файлов, используя подстановочные знаки или формат «файл1+файл2+файл3+...».

REN — переименование одного или нескольких файлов.

RENAME [диск:][путь]имя_файла1 имя_файла2.
REN [диск:][путь]имя_файла1 имя_файла2.

Для конечного файла нельзя указать другой диск или каталог.

DEL — удаление одного или нескольких файлов.

DEL [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена
ERASE [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена

имена — Имена одного или нескольких файлов. Для удаления сразу нескольких файлов используются подстановочные знаки. Если указан каталог, из него будут удалены все файлы.

/P Запрос на подтверждение перед удалением каждого файла.

/F Принудительное удаление файлов, доступных только для чтения.

/S Удаление указанных файлов из всех подкаталогов.

/Q Отключение запроса на подтверждение при удалении файлов.

/A Отбор файлов для удаления по атрибутам.

атрибуты:

- S Системные файлы
- R Доступные только для чтения
- H Скрытые файлы
- A Файлы для архивирования
- Префикс «-» имеет значение НЕ

TYPE — вывод содержимого одного или нескольких текстовых файлов.

TYPE [диск:][путь]имя_файла

FOR — выполнение указанной команды для каждого файла набора.

FOR %переменная IN (набор) DO команда [параметры]

%переменная – подставляемый параметр;

(набор) – набор, состоящий из одного или нескольких файлов. Допускается использование подстановочных знаков;

команда – команда, которую следует выполнить для каждого файла;

параметры – параметры и ключи для указанной команды.

В пакетных файлах для команды FOR используется запись %%переменная вместо %переменная. Имена переменных учитывают регистр букв (%i отличается от %I).

Добавление поддерживаемых вариантов команды FOR при включении расширенной обработки команд:

FOR /D %переменная IN (набор) DO команда [параметры]

Если набор содержит подстановочные знаки, команда выполняется для всех подходящих имен каталогов, а не имен файлов.

FOR /R [[диск:]путь] %переменная IN (набор) DO команда [параметры]

Выполнение команды для каталога [диск:]путь, а также для всех подкаталогов этого пути. Если после ключа /R не указано имя каталога, выполнение команды начинается с текущего каталога.

Если вместо набора указана только точка (.), команда выводит список всех подкаталогов.

FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры]

Набор раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (5 4 3 2 1)

FOR /F [«ключи»] %переменная IN (набор) DO команда [параметры]

FOR /F [«options»] %variable IN («literal string») DO command [command-parameters]

FOR /F [«options»] %variable IN ('command') DO command [command-parameters]

или, если использован параметр usebackq:

FOR /F [«options»] %variable IN (filenameset) DO command [command-parameters]

FOR /F [«options»] %variable IN ('literal string') DO command [command-parameters]

FOR /F [«options»] %variable IN (`command`) DO command [command-parameters]

Набор содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбивки его на отдельные строки текста и выделения из каждой строки заданного числа подстрок (в том числе нуля). Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла. По умолчанию

ключ /F выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательные параметры «ключи» служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую указанные параметры. Ключевые слова:

eol=c — определение символа комментариев в конце строки (допускается задание только одного символа);

skip=n — число пропускаемых при обработке строк в начале файла;

delims=xxx — определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции;

tokens=x,y,m-n — определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла. При использовании этого ключа создаются дополнительные переменные. Формат m-n представляет собой диапазон подстрок с номерами от m по n. Если последний символ в строке tokens= является звездочкой, создается дополнительная переменная, значением которой будет весь оставшийся текст в строке после обработки последней подстроки;

usebackq — применение новой семантики, при которой строки, заключенные в обратные кавычки, выполняются как команды, строки, заключенные в прямые одиночные кавычки, являются строкой литералов команды, а строки, заключенные в двойные кавычки, используются для выделения имен файлов в списках имен файлов.

Поясняющий пример:

FOR /F “eol=; tokens=2,3* delims=,” %%i in (myfile.txt) do @echo %%i %%j %%k

— эта команда обрабатывает файл myfile.txt, пропускает все строки, которые начинаются с символа точки с запятой, и передает вторую и третью подстроки из каждой строки в тело цикла, причем подстроки разделяются запятыми и/или пробелами. В теле цикла переменная %i используется для второй подстроки, %j – для третьей, а %k получает все оставшиеся подстроки после третьей.

Имена файлов, содержащие пробелы, необходимо заключать в двойные кавычки.

Для того чтобы использовать двойные кавычки, необходимо использовать параметр usebackq, иначе двойные кавычки будут восприняты как границы строки для обработки.

Переменная %i явно описана в инструкции for, а переменные %j и %k описываются неявно с помощью ключа tokens=. Ключ tokens= позволяет извлечь из одной строки файла до 26 подстрок, при этом, не допускается использование переменных больших чем буквы 'z' или 'Z'. Следует помнить, что имена переменных FOR являются глобальными, поэтому одновременно не может быть активно более 52 переменных.

Синтаксис команды FOR /F также позволяет обработать отдельную строку, с указанием параметра filenameset, заключенным в одиночные кавычки.

Строка будет обработана как единая строка из входного файла.

Наконец, команда FOR /F позволяет обработать строку вывода другой команды.

Для этого следует ввести строку вызова команды в апострофах вместо набора имен файлов в скобках. Строка передается для выполнения обработчику команд CMD.EXE, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующая команда:

FOR /F “usebackq delims==“ %%i IN (`set`) DO @echo %%i

— выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

SHIFT — изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SHIFT [/n] — команда SHIFT при включении расширенной обработки команд поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть от 0 до 9.

Например, в следующей команде:

SHIFT /2

%3 заменяется на %2, %4 на %3 и т.д., а %0 и %1 остаются без изменений.

CALL — вызов одного пакетного файла из другого.

CALL [диск:][путь]имя_файла [параметры]

параметры – набор параметров командной строки, необходимых пакетному файлу.

CHOICE[2] — ожидает ответа пользователя.

CHOICE [/C[:]варианты] [/N] [/S] [/T[:]c,nn] [текст]

/C[:]варианты — варианты ответа пользователя.
По умолчанию строка включает два варианта: YN
/N Ни сами варианты, ни знак вопроса в строке приглашения не отображаются.
/S Учитывать регистр символов.
/T[:]c,nn Ответ «c» выбирается автоматически после nn секунд ожидания текст Строка приглашения
После выполнения команды переменная ERRORLEVEL приобретает значение, равное номеру выбранного варианта ответа.

FC — сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FC [/A] [/C] [/L] [/LBn] [/N] [/OFF[LINE]] [/T] [/U] [/W]
[/nnnn][диск1:][путь1]имя_файла1 [диск2:][путь2]имя_файла2
FC /B [диск1:][путь1]имя_файла1 [диск2:][путь2]имя_файла2

/A Вывод только первой и последней строк для каждой группы различий.
/B Сравнение двоичных файлов.
/C Сравнение без учета регистра символов.
/L Сравнение файлов в формате ASCII.
/LBn Максимальное число несоответствий для заданного числа строк.
/N Вывод номеров строк при сравнении текстовых файлов ASCII.
/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».

[2] CHOICE — это внешняя команда.

/T Символы табуляции не заменяются эквивалентным числом пробелов.

/U Сравнение файлов в формате UNICODE.

/W Пропуск пробелов и символов табуляции при сравнении.

/nnnn Число последовательных совпадающих строк, которое должно встретиться после группы несовпадающих.

[диск1:][путь1]имя_файла1

Указывает первый файл или набор файлов для сравнения.

[диск2:][путь2]имя_файла2

Указывает второй файл или набор файлов для сравнения.

FIND — поиск текстовой строки в одном или нескольких файлах.

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] «строка» [[диск:][путь]имя_файла[...]]

/V Вывод всех строк, НЕ содержащих заданную строку.

/C Вывод только общего числа строк, содержащих заданную строку.

/N Вывод номеров отображаемых строк.

/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».

/I Поиск без учета регистра символов.

«строка» Искомая строка.

[диск:][путь]имя_файла

Один или несколько файлов, в которых выполняется поиск.

Если путь не задан, поиск выполняется в тексте, введенном с клавиатуры либо переданном по конвейеру другой командой.

SORT — осуществляет сортировку файла.

SORT [/R] [/+n] [/M килобайтов] [/L язык] [/REC символов]

[[диск1:][путь1]имя_файла1] [/T [диск2:][путь2]]

[/O [диск3:][путь3]имя_файла3]

/+n Задает число символов, n, до начала каждого сравнения. /+3 показывает, что каждое сравнение будет начинаться с третьего

символа каждой строки. Строки меньше чем n символов собираются перед всеми остальными строками.

По умолчанию, сравнение начинается с первого символа каждой строки.

/L[OCALE] язык Перекрывает установленные в системе по умолчанию язык и раскладку заданными. Пока существует возможность только одного выбора: ««C»« – наиболее быстрый способ упорядочивания последовательности.

Сортировка всегда идет без учета регистра.

/M[EMORY] килобайтов Задает количество основной памяти, используемой для сортировки, в килобайтах. Размер памяти должен быть не менее 160КБ.

/REC[ORD_MAXIMUM] символов Определяет максимальное число символов в записи (по умолчанию 4096, максимальное 65535).

/R[EVERSE] Обратный порядок сортировки; т.е. сортировка идет от Я до А, и затем от 9 до 0.

[диск1:][путь1]имя_файла1 Определяет имя сортируемого файла. Если оно опущено, то будет использоваться стандартный поток ввода. Явное задание сортируемого файла работает быстрее, чем перенаправление того же файла в качестве стандартного потока ввода.

/T[EMPORARY] [диск2:][путь2] Определяет путь к папке, содержащей рабочие файлы сортировки, в том случае, когда данные не помещаются в основной памяти. По умолчанию используется системная временная папка.

/O[UTPUT] [диск3:][путь3]имя_файла3 Определяет имя файла, в котором сохраняются отсортированные результаты. Если оно опущено данные записываются в стандартный поток вывода. Явное задание файла вывода работает быстрее, чем перенаправление стандартного потока вывода в этот же файл.

План выполнения

1. Согласуйте с преподавателем вариант выполнения задания.
2. Согласно варианту, разработайте программный файл. При разработке учтите возможность неправильного запуска ваших программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки.

Варианты заданий на выполнение

Вариант 1. Разработать командный файл создающий, копирующий или удаляющий файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 2. Разработать командный файл создающий, копирующий или удаляющий каталог, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 3. Разработать командный файл, добавляющий вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) /b /e).

Вариант 4. Разработать командный файл, регистрирующий время своего запуска в файле протокола run.log и автоматически запускающий некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 5. Разработать командный файл, копирующий произвольное число файлов, заданных аргументами из текущего каталога в указываемый каталог.

Вариант 6. Разработать командный файл, который помещает список файлов текущего каталога в текстовый файл и в зависимости от ключа сортирует по какому-либо полю. Реализовать два варианта: с использованием только команды DIR, с использованием команд DIR и SORT.

Вариант 7. Разработать командный файл, который в интерактивном режиме мог бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 8. Разработать командный файл, который дописывал бы имя файла, полученного входным параметром в сам файл N количество раз. N – также задается параметром.

Вариант 9. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает. В работе используйте для сравнения дат команду FC.

Вариант 10. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает. Сравнение дат реализуйте через переменные, а не через файлы.

Вариант 11. Разработать командный файл, который получал в качестве параметра какое-либо имя, и проверял, определена ли токая переменная среды или нет, и выводил соответствующее сообщение.

Вариант 12. Разработать командный файл, который получал в качестве параметра какой-либо символ и в зависимости от второго параметра вырезал или сохранял в заданном файле все строки, начинающиеся на этот символ.

Вариант 13. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос – копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да», то программа копирует файл.

Вариант 14. Разработать командный файл, который бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 15. Разработать командный файл, который бы получал в качестве аргумента имя текстового файла и выводил на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 16. Разработать командный файл (аналог команды tail в Unix). Командный файл печатает конец файла. По умолчанию – 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 17. Разработать командный файл, который бы склеивал текстовые файлы, заданные в качестве аргументов, и сортировал бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 18. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 19. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы измененные).

Вариант 20. Выполняющий в зависимости от ключа один из 3–х вариантов работы:

- с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;
- с ключом /b создает резервные копии указанных файлов;

с ключом /d удаляет указанные файлы после предупреждения.

2.2 Лабораторная работа «Программирование на языке SHELL в ОС Unix»

Цель работы

Изучение языка Shell, использование переменных среды, переменных Shell и предопределенных переменных.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Программирование в языке Shell

Версии Shell

Shell — интерпретатор команд, подаваемых с терминала или из командного файла. Это обычная программа (т.е. не входит в ядро операционной системы UNIX). Ее можно заменить на другую или иметь несколько.

Две наиболее известные версии:

- Shell (версии 7 UNIX) или Bourne Shell (от фамилии автора S.R.Bourne из фирмы Bell Labs);
- C-Shell (версии Berkley UNIX).

Они похожи, но есть и отличия: C-Shell мощнее в диалоговом режиме, а обычный Shell имеет более элегантные управляющие структуры.

Shell — язык программирования, так как имеет:

- переменные;
- управляющие структуры (типа if);
- подпрограммы (в том числе командные файлы);
- передачу параметров;
- обработку прерываний.

Файл начала сеанса (login-файл)

Независимо от версии Shell при входе в систему UNIX ищет файл начала сеанса с предопределенным именем, чтобы выполнить его как командный файл;

- для UNIX версии 7 это: .profile;

- для C-Shell это: .login и/или .cshrc.

В этот файл обычно помещают команды:

- установки характеристик терминала;
- оповещения типа who, date;
- установки каталогов поиска команд (обычно: /bin, /usr/bin);
- смена подсказки с $ на другой символ и т.д.

Процедура языка Shell

Это командный файл. Два способа его вызова на выполнение:

1. $ sh dothat (где dothat — некоторый командный файл);
2. $ chmod 755 dothat (сделать его выполнимым, т.е. -rwxr-xr-x)

$ dothat.

Следует знать порядок поиска каталогов команд (по умолчанию):

- текущий;
- системный /bin;
- системный /usr/bin.

Следовательно, если имя вашего командного файла дублирует имя команды в системных каталогах, последняя станет недоступной (если только не набирать ее полного имени).

Переменные Shell

В языке Shell версии 7 определение переменной содержит имя и значение: var = value.

Доступ к переменной — по имени со знаком $ спереди:

fruit = apple (определение);

echo $fruit (доступ);

apple (результат echo).

Таким образом, переменная — это строка. Возможна конкатенация строк:

```
$ fruit = apple
$ fruit = pine$fruit
$ echo $fruit
pineapple
$ fruite = apple
$ wine = ${fruite}jack
$ echo $wine
applejack
$
```

Другие способы установки значения переменной — ввод из файла или вывод из команды, а также присваивание значений переменной — параметру цикла for из списка значений, заданного явно или по умолчанию.

Переменная может быть:

- Частью полного имени файла: $d/filename, где $d — переменная (например, d = /usr/bin).

- Частью команды:

```
$ S = "sort + 2n + 1 - 2" (наличие пробелов требует кавычек "")
$ $S tennis/lpr
$ $S basketball/lpr
$ $S pingpong/lpr
$
```

Однако внутри значения для команды не могут быть символы |, >, <, & (обозначающие канал, перенаправления и фоновый режим).

Предопределенные переменные Shell

Некоторые из них можно только читать. Наиболее употребительные:

HOME — "домашний" каталог пользователя; служит аргументом по умолчанию для cd;

PATH — множество каталогов, в которых UNIX ищет команды;

PS1 — первичная подсказка (строка) системы (для v.7 - $).

Изменение PS1 (подсказки) обычно делается в login-файле, например:

```
PS1 = ?
или        PS1 = "? " (с пробелом, что удобнее).
```

Изменение PATH:

```
$ echo $PATH                        - посмотреть;
:/bin:/usr/bin                      - значение PATH;
$ cd                                - "домой";
$ mkdir bin                         - новый каталог;
$ echo $HOME                        - посмотреть;
/users/maryann                      - текущий каталог;
$ PATH = :$HOME/bin:$PATH           - изменение PATH;
$ echo $PATH                        - посмотреть;
:/users/maryann/bin:/bin:/usr/bin   - новое значение PATH.
```

Установка переменной Shell выводом из команды

Пример 1:

$ now = `date` (где `` - обратные кавычки)
$ echo $now
Sun Feb 14 12:00:01 PST 1985
$

Пример 2: (получение значения переменной из файла):
$ menu = `cat food`
$ echo $menu
apples cheddar chardonnay (символы возврата каретки заменяются на пробелы).

Переменные Shell — аргументы процедур

Это особый тип переменных, именуемых цифрами.
Пример:
$ dothis grapes apples pears (процедура).

Тогда позиционные параметры (аргументы) этой команды доступны по именам:
$1 = `grapes`
$2 = `apples`
$3 = `pears`
и т.д. до $9. Однако есть команда shift, которая сдвигает имена на остальные аргументы, если их больше 9 (окно шириной 9).

Другой способ получить все аргументы (даже если их больше 9): $*, что эквивалентно $1$2 ...

Количество аргументов присваивается другой переменной: $#(диез).

Наконец, имя процедуры - это $0; переменная $0 не учитывается при подсчете $#.

Структурные операторы Shell

Оператор цикла **for**
Пусть имеется командный файл makelist (процедура)
$ cat makelist
sort +1 -2 people | tr -d -9 | pr -h Distribution | lpr.

Если вместо одного файла people имеется несколько, например: adminpeople, hardpeople, softpeople,..., то необходимо повторить выполнение процедуры с различными файлами. Это возможно с помощью for — оператора. Синтаксис:

for <переменная> in <список значений>
do <список команд>
done

Ключевые слова for, do, done пишутся с начала строки.
Пример (изменим процедуру makelist):
for file in adminpeople, hardpeople, softpeople
do
Sort +1 -2 $file | tr ... | lpr
done.
Можно использовать метасимволы Shell в списке значений.
Пример:
for file in *people (для всех имен, кончающихся на people)
do
...
done.

Если in опущено, то по умолчанию в качестве списка значений берется список аргументов процедуры, в которой содержится цикл, а если цикл не в процедуре, то — список параметров командной строки (то есть в качестве процедуры выступает команда).

Пример:
for file
do
...
done

Для вызова makelist adminpeople hardpeople softpeople будет сделано то же самое.

Условный оператор **if**

Используем имена переменных, представляющие значения параметров процедуры:

sort +1 -2 $1 | tr ... | lpr

Пример неверного вызова:

makelist (без параметров), где $1 неопределен. Исправить ошибку можно, проверяя количество аргументов – значение переменной $# посредством if - оператора.

Пример: (измененной процедуры makelist):

```
if test $# -eq 0
then
echo "You must give a filename"
exit 1
else
sort +1 -2 $1 | tr ... | lpr
fi
```

Здесь test и exit - команды проверки и выхода. Таким образом, синтаксис оператора if:

if <если эта команда выполняется успешно, то>;

then <выполнить все следующие команды до else или, если его нет, до fi>;

[else <иначе выполнить следующие команды до fi>]

Ключевые слова if, then, else и fi пишутся с начала строки.

Успешное выполнение процедуры означает, что она возвращает значение true = 0 (zero) (неуспех - возвращаемое значение не равно 0).

Оператор exit 1 задает возвращаемое значение 1 для неудачного выполнения makelist и завершает процедуру.

Возможны вложенные if. Для else if есть сокращение elif, которое одновременно сокращает fi.

Команда **test**

Не является частью Shell, но применяется внутри Shell-процедур.

Имеется три типа проверок:

- оценка числовых значений;
- оценка типа файла;
- оценка строк.

Для каждого типа свои примитивы (операции op).

1. Для чисел синтаксис такой: N op M, где N, M - числа или числовые переменные;

op принимает значения: -eq, -ne, gt, -lt, -ge, -le.

2. Для файла синтаксис такой: op filename, где op принимает значения:

- -s (файл существует и не пуст);
- -f (файл, а не каталог);
- -d (файл-директория (каталог);
- -w (файл для записи);
- -r (файл для чтения).

Для строк синтаксис такой: S op R, где S, R - строки или строковые переменные или op1 S, где op принимает значения:

- = (эквивалентность);
- != (не эквивалентность);

op1 принимает значения:

- -z (строка нулевой длины);
- -n (не нулевая длина строки).

Наконец, несколько проверок разных типов могут быть объединены логическими операциями -a (AND) и -o (OR).

Примеры:

```
$ if test -w $2 -a -r S1
> then cat $1 >> $2
> else echo "cannot append"
> fi
$
```

В некоторых вариантах ОС UNIX вместо команды test используются квадратные скобки, т.е. if [...] вместо if test ... .

Оператор цикла **while**

Синтаксис:

```
while <команда>
do
<команды>
done
```

Если "команда" выполняется успешно, то выполнить "команды", завершаемые ключевым словом done.

Пример:

```
if test $# -eq 0
then
echo "Usage: $0 file ..." > &2
exit
fi
while test $# -gt 0
  do
if test -s $1
  then
echo "no file $1" > &2
  else
sort + 1 - 2 $1 | tr -d ... (процедуры)
```

```
fi
        shift (* перенумеровать аргументы *)
done
```

Процедуры выполняются над всеми аргументами.

Оператор цикла **until**

Инвертирует условие повторения по сравнению с while

Синтаксис:

```
until <команда>
do
<команды>
done
```

Пока "команда" не выполнится успешно, выполнять команды, завершаемые словом done.

Пример:

```
if test S# -eq 0
then
echo "Usage $0 file..." > &2
exit
fi
  until test S# -eq 0
  do
  if test -s $1
  then
echo "no file $1" > &2
else
sort +1 -2 $1 | tr -d ... (процедура)
fi
  shift (сдвиг аргументов)
  done
```

Исполняется аналогично предыдущему.

Оператор выбора **case**

Синтаксис:

```
case <string> in
```

string1) <если string = string1, то выполнить все следующие команды до ;; > ;;

string2) <если string = string2, то выполнить все следующие команды до ;; > ;;

```
string3) ... и т.д. ...
esac
```

Пример:

Пусть процедура имеет опцию -t, которая может быть подана как первый параметр:

```
.................
together = no
case $1 in
-t)          together = yes
shift ;;
-?)          echo "$0: no option $1"
exit ;;
esac
  if test $together = yes
  then
sort ...
fi
```

где ? - метасимвол (если -?, т.е. "другая" опция, отличная от -t, то ошибка). Можно употреблять все метасимволы языка Shell, включая ?, *, [-]. Легко добавить (в примере) другие опции, просто расширяя case.

Использование временных файлов в каталоге /tmp

Это специальный каталог, в котором все файлы доступны на запись всем пользователям.

Если некоторая процедура, создающая временный файл, используется несколькими пользователями, то необходимо обеспечить уникальность имен создаваемых файлов. Стандартный прием – имя временного файла $0$$, где $0 - имя процедуры, а $$ - стандартная переменная, равная уникальному идентификационному номеру процесса, выполняющего текущую команду.

Хотя администратор периодически удаляет временные файлы в /tmp, хорошей практикой является их явное удаление после использования.

Комментарии в процедурах

Они начинаются с двоеточия : , которое считается нуль-командой, а текст комментария - ее аргументом. Чтобы Shell не

интерпретировал метасимволы ($, * и т.д.), рекомендуется заключать текст комментария в одиночные кавычки.

В некоторых вариантах ОС UNIX примечание начинается со знака #.

Пример процедуры

```
:'Эта процедура работает с файлами, содержащими имена'
: 'и номера телефонов,'
:'сортирует их вместе или порознь и печатает результат на'
:'экране или на принтере'
:'Ключи процедуры:'
:'-t (together) - слить и сортировать все файлы вместе'
:'-p (printer) - печатать файлы на принтере'
if test $# - eq 0
then
echo "Usage: $ 0 file ... " > & 2
exit
fi
together = no
print = no
while test $# -gt 0
do case $1 in
-t)          together = yes
shift ;;
-p)          print = yes
shift ;;
-?)          echo "$0: no option $1"
exit ;;
*) if test $together = yes
then
sort -u +1 -2 $1 | tr ... > /tmp/$0$$
  if $print = no
then
cat /tmp/$0$$
  else
lpr -c /tmp/$0$$
fi
rm /tmp/$0$$
exit
else if test -s $1
then         echo "no file $1" > &2
```

```
  else      sort +1 -2 $1 | tr...> /tmp/$0$$
if $print = no
then        cat /tmp/$0$$
else        lpr -c /tmp/$0$$
fi
            rm /tmp/$0$$
  fi
shift
fi;;
  esac
done.
```

Процедура проверяет число параметров $# и, если оно равно нулю, завершается. В противном случае она обрабатывает параметры (оператор case). В качестве параметра может выступать либо ключ (символ, предваряемый минусом), либо имя файла (строка, представленная метасимволом *). Если ключ отличен от допустимого (метасимвол ? отличен от t и p), процедура завершается. Иначе в зависимости от наличия ключей t и p выполняются действия, заявленные в комментарии в начале процедуры.

Обработка прерываний в процедурах

Если при выполнении процедуры получен сигнал прерывания (от клавиши BREAK или DEL, например), то все созданные временные файлы останутся неудаленными (пока это не сделает администратор) ввиду немедленного прекращения процесса.

Лучшим решением является обработка прерываний внутри процедуры оператором trap:

Синтаксис:

trap 'command arguments' signals...

Кавычки формируют первый аргумент из нескольких команд, разделенных точкой с запятой. Они будут выполнены, если возникнет прерывание, указанное аргументами signals (целые):

2 - когда вы прерываете процесс;

1 - если вы "зависли" (отключены от системы)

и др.

Пример (развитие предыдущего):

```
case $1 in
.....
*) trap 'rm /tmp/*; exit' 2 1 (удаление временных файлов)
if test -s $1
```

.....
rm /tmp/*
Лучше было бы:
trap 'rm /tmp/* > /dev/null; exit' 2 1
так как прерывание может случиться до того, как файл /tmp/$0$$ создан и аварийное сообщение об этом случае перенаправляется на null-устройство.

Выполнение арифметических операций: expr

Команда expr вычисляет значение выражения, поданного в качестве аргумента, и посылает результат на стандартный вывод. Наиболее интересным применением является выполнение операций над переменными языка Shell.

Пример суммирования 3 чисел:

```
$ cat sum3
expr $1 + $2 + $3
$ chmod 755 sum3
$ sum3 13 49 2
64
$
```

Пример непосредственного использования команды:

```
$ expr 13 + 49 + 2 + 64 + 1
129
$
```

В expr можно применять следующие арифметические операторы: +, -, *, /, % (остаток). Все операнды и операции должны быть разделены пробелами.

Заметим, что знак умножения следует заключать в кавычки (одинарные или двойные), например: '*', так как символ * имеет в Shell специальный смысл.

Более сложный пример expr в процедуре (фрагмент):

```
num = 'wc -l < $1'
tot = 100
count = $num
avint = 'expr $tot / $num'
avdec = 'expr $tot % $num'
while test $count -gt 0
do ...
```

Здесь wc -l осуществляет подсчет числа строк в файле, а далее это число используется в выражениях.

Отладка процедур Shell

Имеются три средства, позволяющие вести отладку процедур.

- Размещение в теле процедуры команд echo для выдачи сообщений, являющихся трассой выполнения процедуры.

- Опция -v (verbose = многословный) в команде Shell приводит к печати команды на экране перед ее выполнением.

- Опция -x (execute) в команде Shell приводит к печати команды на экране по мере ее выполнения с заменой всех переменных их значениями; это наиболее мощное средство.

Утилита AWK

Awk - утилита, подобная grep. Однако, кроме поиска по образцу, она позволяет проверять отношения между полями строк (записей) и выполнять некоторые действия над строками (генерировать отчеты). Название не является акронимом, оно образовано первыми буквами фамилий авторов (A.V.Aho, P.Y.Weinberger, B.W.Kernighan).

Задание поиска-действия следует синтаксису:

/<образец>/{<действие>}

И образец, и действие могут отсутствовать. Найденные по образцу строки при отсутствии заданного действия выводятся в стандартный вывод (на экран).

Образец задается регулярным выражением, как и в grep. Если образец отсутствует, обрабатываются все строки.

Рассмотрим примеры действий, которые можно выполнить командой awk.

Перестановка полей строки выполняется с помощью ссылки на поле $n, где n - номер поля.

Например:

```
$ cat people
Mary Clark 101
Henry Morgan 112
Bill Williams 100
$ awk '{print $2 "," $1 "^I" $3}' people
Clark, Mary 101
Morgan, Henry 112
Williams, Bill 100
```

где ^(control - I) - знак табуляции для подвода каретки к очередной позиции табуляции (для выравнивания третьего поля).

Действия для awk могут быть заданы в файле.

Например:

```
$ cat swap
{print $2 "," $1 "^I" $3}
$ awk -f swap people
```

Awk имеет встроенные образцы и переменные. Образцы BEGIN и END означают начало и конец файла соответственно. Переменная NR (Number of Records) означает число записей (строк) в файле, NF - число полей в записи. Можно использовать переменные, объявленные пользователем. Пример, подсчитывающий среднее значение третьего поля файла tennis (программа действий для awk - в файле average):

```
$ cat > average
{total = total + $3}
END {print "Average value is", total/NR}
^D
$ awk -f average tennis
Average value is 8.9
$
```

Образец поиска в awk может содержать условные выражения. Пример, в котором в файле tennis пишутся все записи, значение третьего поля в которых не меньше 10:

```
$ awk '$3 >= 10 {print $0}'tennis
Steve Daniel 11
Hank Parker 18
Jack Austen 14
$
```

Знак $0 (доллар-ноль) есть ссылка на всю запись (строку). В общем случае выражение для условия подчиняется синтаксису, близкому к синтаксису выражений в языке С. Кроме того, в команде awk допустимо указывать отрезок образцов. Пример выборки всех записей, сделанных с 1976 до 1978 г.:

```
$ sort -n -o chard.s chard
$ awk '/1976/, /1978/ {if($2 < 8.00 print $0}' chard.s
1976 7.50 Chateau
1977 7.75 Chateau
```

1978 5.99 Charles

Как видно из примера, в программах действий для awk можно использовать управляющие структуры с синтаксисом, близким к языку C.

Пример цикла для печати полей всех записей файла в обратном порядке:

$ awk {for (i = NF; i > 0; --i) print $i} f1, где NF - число полей в записи.

Встроенные функции AWK

length(arg) - Функция длины arg. Если arg не указан, то выдает длину текущей строки.

exp(),log(),sqrt() - Математические функции экспонента, логарифм и квадратный корень.

int() - Функция целой части числа.

substr(s,m,n) - Возвращает подстроку строки s, начиная с позиции m, всего n символов.

index(s,t) - Возвращает начальную позицию подстроки t в строке s. (Или 0, если t в s не содержится.)

sprintf(fmt,exp1,exp2,...) - Осуществляет форматированную печать (вывод) в строку, идентично PRINTF.

split(s,array,sep) - Помещает поля строки s в массив array и возвращает число заполненных элементов массива. Если указан sep, то при анализе строки он понимается как разделитель.

Операции отношения awk

X = = Y – X равно Y?
X ! = Y – X не равно Y?
X > Y – X больше чем Y?
X = > Y X больше чем или равно Y?
X < Y – X меньше чем Y?
X < = Y – X меньше чем или равно Y?
X ~ Re – X совпадает с регулярным выражением Re?
X ! ~ Re – X не совпадает с регулярным выражением Re?

Старшинство операций в awk

Группа	Операции
1	= += -= *= /= %=
2	||

3	&&
4	> >= < <= == != ~ !~
5	Строка конкатенации «x» «y» становится «xy»
6	+ -
7	* / %
8	++ --

Стандартные переменные

ARGC – число аргументов в командной строке;
ARGV – массив с аргументами командной строки;
FILENAME – строка текущего файла ввода;
FNR – номер текущей записи в текущем файле;
FS – разделитель полей ввода;
NF – число полей в текущей записи;
NR – номер текущей записи;
OFMT – формат вывода чисел (по умолчанию % 6g);
OFS – разделитель полей ввода (по умолчанию пробел);
ORS – Разделитель выводимых записей (по умолчанию новая строка);
RS – Разделитель полей ввода (по умолчанию новая строка).

Список команд Shell

date — вывод даты;
who — вывод пользователей;
who am i— вывод собственного имени;
exit — выход из системы (для передачи кода завершения);
mail — почта;
write — передача сообщения другому пользователю;
man — информация о команде;
news — новости;
ed — текстовый редактор (a/... ./w имя/ctrl-d)
ls— перечень имен файлов в каталоге;
ls -t — перечень файлов во временном порядке;
ls -l — перечень файлов в полном виде;
ls -li — перечень файлов в расширенном виде;
cat — распечатка файла (cat>имя — создание файла);
pr — распечатка по 66 строк;
mv — перенос файла;
cp — копирование файла;
rm — удаление файла;

ln — назначение связи;
rmdir — удалить каталог;
mkdir — создать каталог;
pwd — определение своего рабочего каталога;
cd — смена каталога;
wc — подсчет числа строк, слов и символов;
tail +n — вывод файла начиная со строки с номером n;
cmp — поиск различий между файлами (до первого различия);
diff — поиск всех различий;
echo — вывод строки (` ` — результата, ' ' — команды);
echo $? — выдача кода завершения команды (0, 1, 2);
wait — ждать завершения всех процессов;
kill — убить процесс (kill -9 #_процесса);
ps — список процессов;
nohup — выполнение команды после отключения (nohup кмд&);
nice — запуск с пониженным приоритетом (nice кмд&);
at — запуск в определенное время (at команды ctrl-d);
export — сообщение интерпретатору о использовании переменных;
sh — переход в порожденный shell;
du — определение занятого пространства;
df — свободное пространство диска;
chmod — смена права доступа;
mesg — (n — запрет, y — разрешение) сообщения;
sleep — пауза;
set — показать все ранее определенные переменные;
set `` — установить значение переменной;
time — информация о времени выполнения команды;
uname — информация о системе (uname -a — полная);
read — присваивает переменной значение последующей строки;
touch — заменяет время модификации файла на настоящее;
for — цикл (for i in список/ do команды/ done);
case — выбор (case слово in/шаблон) команды ;;/esac);
if— условие (if команда / then команды, если условие верно / else команды, если условие ложно/fi;
while — цикл (while команда/do тело цикла, выполняется пока команда возвращает истина/done;
until — цикл (аналог while, но ждет ложь);

trap — последовательность действий, выполняемая при прерывании (trap 'rm -f $old; exit 1' 1 2 15), где

0 — выход из интерпретатора

1 — отбой

2 — прерывание (DEL)

3 — останов (ctrl-\; вызывает распечатку содержимого памяти программы)

9 — уничтожение

15 — окончание выполнения.

Встроенные переменные интерпретатора

$# — число аргументов;

$* — все аргументы, передаваемые интерпретатору ($@);

$- — флаги передаваемые интерпретатору;

$? — возвращение значения последней выполненной команды;

$$ — номер процесса интерпретатора;

$! — номер процесса последней команды, запущенной с &;

Правила сопоставления шаблонов в интерпретаторе

* — задание любой строки, в том числе и пустой.

? — любой одиночный символ;

"..." — задает в точности ...; ""(") защищает от спецсимволов;

\c — задает с буквально;

a|b — только для выражения выбора, a или b.

Значения переменных

$var — значение var;

${var-thing} — значение var, если оно определено, в противном случае thing;

${var=thing} — значение var, если var не определено, то присваивается значение thing;

${var?строка} — если var определено — $var, в противном случае выводится строка и инт. прекращает работу;

${var+thing} — thing, если $var определено, в противном случае ничего.

Метасимволы

| — конвейер (связь выходного потока одной программы с выходным потоком другой);
& — асинхронный запуск;
; — последовательное выполнение;
> — помещение выходного потока;
>> — добавление выходного потока;
* — любая строка;
? — любой символ;
[ccc] — задает любой символ из [ccc] в имени файла;
`...` — инициирует выполнение команды;
( ) — инициирует выполнение команды в порожденном shell;
{ } — инициирует выполнение команды в текущем shell;
$1 — заменяется аргументом командного файла;
$var — значение переменной var в программе на языке shell;
${var} — значение var;
\ — перевод строки;
'...' — непосредственное использование;
"..." — непосредственное использование, после того, как $`...` и \ будут интерпретированы;
— остальная строка — комментарий;
p1&&p2 — выполнить p1, в случае успеха p2;
p1||p2 — выполнить p1, в случае неудачи p2;
2>file — переключить поток диагностики на файл;
2>&1 — поместить стандартный поток диагностики в выходной
поток;
1>&2 — добавление выходного потока к стандартному потоку
диагностики.

План выполнения

1. Согласуйте с преподавателем вариант выполнения задания.
2. Согласно варианту, разработайте программный файл. При разработке учтите возможность неправильного запуска ваших

программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки.

Варианты заданий на выполнение

Вариант 1. Разработать программу, отправляющую почту (содержимое файла) группе пользователей, выбираемых из общего списка (хранящегося в другом файле) в интерактивном режиме. Например, вы отвечаете "Y" для тех, кому надо посылать, "N" — не надо, "Q" — конец выбора.

Вариант 2. Разработать программу, выводящую через определенный интервал времени информацию о пользователях в системе: кто вошел, кто вышел.

Вариант 3. Разработать программу, выполняющую в зависимости от ключа один из 3–х вариантов работы:

- с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;
- с ключом /b создает резервные копии указанных файлов;
- с ключом /d удаляет указанные файлы после предупреждения.

Вариант 4. Разработать программу создающую, копирующую или удаляющую файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 5. Разработать программу, добавляющую вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) /b /e).

Вариант 6. Разработать программу, регистрирующую время своего запуска в файле протокола run.log и автоматически запускающую некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 7. Разработать программу, копирующую произвольное число файлов, заданных аргументами из текущего каталога в указываемый каталог.

Вариант 8. Разработать программу, которая в интерактивном режиме могла бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 9. Разработать программу, которая бы запускала бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает.

Вариант 10. Разработать программу, которая получала бы в качестве параметра какой-либо символ и в зависимости от второго параметра вырезала или сохраняла в заданном файле все строки начинающиеся на этот символ.

Вариант 11. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос – копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да» то программа копирует файл.

Вариант 12. Разработать программу, которая бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 13. Разработать программу (аналог команды wc), которая бы получала бы в качестве аргумента имя текстового файла и выводила на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 14. Разработать программу (аналог команды tail), которая печатает конец файла. По умолчанию – 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 15. Разработать программу, которая склеивала бы текстовые файлы, заданные в качестве аргументов, и сортировала бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 16. Разработать программу, которая формировала бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 17. Разработать программу, разбирающую содержимое письма (файл или входной поток), выделяющую заголовок письма с адресом отправителя (поля From: или From) и отправляющую содержимое письма без заголовка обратно отправителю.

Вариант 18. Разработать программу, которая изменяет текстовый файл так, что четные и нечетные строки меняются местами.

Вариант 19. Разработать программу, которая бы в зависимости от параметров, строила бы выборку по какому бы условию (числовые значения) из табличного файла.

Вариант 20. Разработать программу, которая инвертирует текстовый файл или его строки.

2.3 Лабораторная работа «Управление процессами в ОС QNX»

Цель работы

Познакомиться с визуальным интерфейсом ОС QNX, возможностями различных функций управления процессами и изучить принципы работы с компилятором С в среде ОС QNX.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Создание процессов

На самом высоком уровне абстракции система состоит из множества процессов. Каждый процесс ответственен за обеспечение служебных функций определенного характера.

Разделение объектов на множество процессов дает ряд преимуществ:

1. возможность декомпозиции задачи и модульной организации решения;
2. удобство сопровождения;
3. надежность.

Любой поток может осуществить запуск процесса. Однако необходимо учитывать ограничения, вытекающие из основных принципов защиты.

Из курса «Операционные системы» вы уже должны быть знакомы с возможностями запуска процессов из командного интерпретатора *(shell).*

Например:

$ program1 — запуск приложения в режиме переднего плана;

$ program2 & — запуск приложения в режиме заднего плана.

$ nice program3 — запуск приложения с заниженным приоритетом.

Обычно разработчиков программного обеспечения не заботит тот факт, что командный интерпретатор создает процессы — это

просто подразумевается. Однако в большой мультипроцессорной системе вы можете пожелать, чтобы одна главная программа выполняла запуск всех других процессов вашего приложения.

Рассмотрим некоторые функции, которые ОС QNX использует для запуска других процессов:

- *system();*
- *fork();*
- *vfork();*
- *exec();*
- *spawn().*

Какую из этих функций применять, зависит от двух требований: переносимости и функциональности.

system() — самая простая функция; она получает на вход одну командную строку, такую же, которую вы набрали бы в ответ на подсказку командного интерпретатора, и выполняет ее.

Фактически, для обработки команды функция system() запускает копию командного интерпретатора.

fork() — порождает процесс, являющийся его точной копией. Новый процесс выполняется в том же адресном пространстве и наследует все данные порождающего процесса.

Между тем, родительский и дочерний процесс имеют различные идентификаторы процессов, так как в системе не может быть двух процессов с одинаковыми идентификаторами. Есть и еще одно отличие, это значение, возвращаемое функцией fork(). В дочернем процессе функция возвращает ноль, а в родительском процессе идентификатор дочернего процесса.

Пример, использования функции fork():

```
printf("PID родителя равен %d\n", getpid());
if (child_pid = fork()) {
printf("Это родитель, PID сына %d\n", child_pid);
} else {
printf("Это сын, PID %d\n", getpid());
}
```

vfork() — так же порождает процесс. В отличие от функции fork() она позволяет существенно сэкономить на ресурсах, поскольку она делает разделяемое адресное пространство родителя. Функция vfork() создает дочерний процесс, а затем приостанавливает

родительский до тех пор, пока дочерний процесс не вызовет функцию exec() или не завершится.

exec() — заменяет образ порождающего процесса образом нового процесса. Возврата управления из нормально отработавшего exec() не существует, т.к. образ нового процесса накладывается на образ порождающего процесса. В системах стандарта POSIX новые процессы обычно создаются без возврата управления порождающему процессу - сначала вызывается fork(), а затем из порожденного процесса — exec().

spawn() — создает новый процесс по принципу «отец»-«сын». Это позволяет избежать использования примитивов fork() и exec(), что ускоряет обработку и является более эффективным средством создания новых процессов. В отличие от fork() и exec(), которые по определению создают процесс на том же узле, что и порождающий процесс, примитив spawn() может создавать процессы на любом узле сети.

План выполнения

1. Познакомиться с интерфейсом ОС QNX.

Изучить процедуру компиляции (компилятор командной строки gcc). Повторить стандартный ввод – вывод, разбор аргументов и переменных среды. Исследовать работу функций по работе с файлами языка С.

2. Написать программу, которая бы запускала в памяти еще один процесс и оставляла бы его работать в бесконечном цикле. При повторном запуске программа должна убирать запущенный ранее процесс из памяти (можно использовать kill).

3. Подготовиться к ответам на теоретическую часть лабораторной работы.

Посмотреть задание на следующую лабораторную работу, с целью вспомнить численные методы, чтобы быть готовым к выполнению лабораторной работы.

2.4 Лабораторная работа «Управление потоками в ОС QNX»

Цель работы

Познакомиться возможностями функций управления потоками в ОС QNX.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Создание потоков

Процесс может содержать один или несколько потоков. Число потоков варьируется. Один разработчик программного обеспечения, используя только единственный поток, может реализовать те же самые функциональные возможности, что и другой, используя пять потоков. Некоторые задачи сами по себе приводят к многопоточности и дают относительно простые решения, другие в силу своей природы, являются однопоточными, и свести их к моногопоточночной реализации достаточно трудно.

Любой поток может создать другой поток в том же самом процессе. На это не налагается никаких ограничений (за исключением объема памяти). Как правило, для этого применяется функция POSIX *pthread_create()*:

```
#include <pthread.h>
int
pthread_create(pthread_t *thread,
const pthread_attr_t *attr,
                    void *(*start_routine) (void *),
void *arg);
```

thread — указатель на pthread_t, где храниться идентификатор потока;

attr — атрибутивная запись;

start_routine — подпрограмма с которой начинается поток;

arg — параметр, который передается подпрограмме start_routine.

Первые два параметра необязательные вместо них можно передавать NULL.

Параметр thread можно использовать для хранения идентификатора вновь создаваемого потока.

Пример однопоточной программы. Предположим, что мы имеет программу, выполняющую алгоритм трассировки луча. Каждая строка растра не зависит от остальных. Это обстоятельство (независимость строк растра) автоматически приводит к программированию данной задачи как многопоточной.

```
int main ( int argc, char **argv)
{
int x1;
… // Выполнить инициализации
for (x1 = 0; x1 < num_x_lines; x1++)
{
do_one_line (x1);
}
… // Вывести результат
}
```

Здесь видно, что программа независимо по всем значениям x1 рассчитывает необходимые растровые строки.

Пример первой многопоточной программы. Для параллельного выполнения функции do_one_line (x1) необходимо изменить программу следующим образом:

```
int main ( int argc, char **argv)
{
int x1;
… // Выполнить инициализации
for (x1 = 0; x < num_x_lines; x1++)
{
pthread_create (NULL, NULL, do_one_line,
(void *) x1);
}
```

```
… // Вывести результат
}
```

Пример второй многопоточной программы. В приведенном примере непонятно когда нужно выполнять вывод результатов, так как приложение запустило массу потоков, но не знает когда они завершаться. Можно поставить задержку выполнения программы (sleep 1), но это не будет правильно. Для этого лучше использовать функцию pthread_join().

Есть еще один минус у приведенной выше программы, если у нас много строк в изображении не факт, что все созданные потоки будут функционировать параллельно, как правило, процессоров системе, гораздо меньше. Для этого лучше модифицировать программу так, чтобы запускалось столько потоков, сколько у нас процессоров в системе.

```
int num_lines_per_cpu;
int num_cpus;

int main (int argc, char **argv)
{
int cpu;
pthread_t *thread_ids;

… // Выполнить инициализации

// Получить число процессоров
num_cpus = _syspage_ptr->num_cpu;

thread_ids = malloc (sizeof (pthread_t) *
 num_cpus);
num_lines_per_cpu = num_x_lines / num_cpus;

for (cpu = 0; cpu < num_cpus; cpu++)
{
pthread_create (&thread_ids [cpu], NULL,
do_one_batch,
(void *) cpu);
}

// Синхронизировать с завершением всех потоков
```

```
for (cpu = 0; cpu < num_cpus; cpu++)
{
pthread_join (thread_ids [cpu], NULL);
}


… // Вывести результат
}

void *do_one_batch (void *c)
{
int cpu = (int) c;
int x1;
for (x1 = 0; x1 < num_lines_per_cpu; x1++)
{
do_one_line(x1 + cpu * num_lines_per_cpu);
}
}
```

План выполнения

1. Выполнить задание согласно варианту.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

Варианты заданий на выполнение

Вариант 1. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки обменом «пузырьком».

Вариант 2. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки простых вставок.

Вариант 3. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод сортировки выбором.

Вариант 4. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом прямоугольников.

Вариант 5. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом трапеций.

Вариант 6. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать вычисления интегралов методом Симпсона.

Вариант 7. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) – золотого сечения.

Вариант 8. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) – общего поиска.

Вариант 9. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод оптимизации функций (min, max) – дихотомии.

Вариант 10. Написать программу, которая принимает в качестве параметров набор имен файлов данных (произвольное число) и запускает все файлы на параллельную обработку (используя треды, нити, потоки). В качестве обработки использовать метод решения системы нелинейных уравнений – метод касательных.

2.5 Лабораторная работа «Организация обмена сообщениями в ОС QNX»

Цель работы
Познакомиться с механизмами обмена сообщениями в ОС QNX.

Форма проведения
Выполнение индивидуального задания.

Форма отчетности
Защита программного кода командного файла.

Теоретические основы

Связь между процессами посредством сообщений

Механизм передачи межпроцессных сообщений занимается пересылкой сообщений между процессами и является одной из важнейших частей операционной системы, так как все общение между процессами, в том числе и системными, происходит через сообщения. Сообщение в QNX – это последовательность байтов произвольной длины (0-65535 байтов) и произвольного формата.

Протокол обмена сообщениями выглядит таким образом: задача блокируется для ожидания сообщения. Другая же задача посылает первой сообщение и при этом блокируется сама, ожидая ответа. Первая задача становится разблокированной, обрабатывает сообщение и отвечает, разблокируя при этом вторую задачу.

Сообщения и ответы, пересылаемые между процессами при их взаимодействии, находятся в теле отправляющего их процесса до того момента, когда они могут быть приняты. Это значит, что, с одной стороны, уменьшается вероятность повреждения сообщения в процессе передачи, а с другой – уменьшается объем оперативной памяти, необходимый для работы ядра. Кроме того, уменьшается число пересылок из памяти в память, что разгружает процессор. Особенностью процесса передачи сообщений является то, что в сети, состоящей из нескольких компьютеров под управлением QNX, сообщения могут прозрачно передаваться процессам, выполняющимся на любом из узлов.

Передача сообщений служит не только для обмена данными между процессами, но, кроме того, является средством синхронизации выполнения нескольких взаимодействующих процессов.

Рассмотрим более подробно функции Send(), Receive() и Reply().

Использование функции Send(). Предположим, что процесс А выдает запрос на передачу сообщения процессу В. Запрос оформляется вызовом функции Send().

Send (pid, smsg, rmsg, smsg_bn, rmsg_len)

Функция Send() имеет следующие аргументы:

- pid — идентификатор процесса-получателя сообщения (т.е. процесса В); pid — это идентификатор, посредством которого процесс опознается операционной системой и другими процессами;
- smsg — буфер сообщения (т.е. посылаемого сообщения);
- rmsg — буфер ответа (т.е. сообщения посылаемого в ответ);
- smsg_len — длина посылаемого сообщения;
- rmsg_len — максимальная длина ответа, который должен получить процесс А.

Обратите внимание на то, что в сообщении будет передано не более чем smsg_len байт и принято в ответе не более чем rmsg_len байт — это служит гарантией того, что буферы никогда не будут переполнены.

Использование функции Receive(). Процесс В может принять запрос Send(), выданный процессом А, с помощью функции Receive().

pid = Receive (0, msg, msg_len)

Функция Receive() имеет следующие аргументы:

- pid — идентификатор процесса, пославшего сообщение (т.е. процесса А);
- 0 — (ноль) указывает на то, что процесс В готов принять сообщение от любого процесса;
- msg — буфер, в который будет принято сообщение;
- msg_len — максимальное количество байт данных, которое может поместиться в приемном буфере.

В том случае, если значения smsg_len в функции Send() и msg_len в функции Receive() различаются, то количество передаваемых данных будет определяться наименьшим из них.

Использование функции Reply(). После успешного приема сообщения от процесса А процесс В должен ответить ему, используя функцию Reply().

Reply (pid, reply, reply_len)

Функция Reply() имеет следующие аргументы:

– pid — идентификатор процесса, которому направляется ответ (т.е. процесса А);

– reply — буфер ответа;

– reply_len — длина сообщения, передаваемого в ответе.

Если значения reply_len в функции Reply() и rmsg_len в функции Send() различаются, то количество передаваемых данных определяется наименьшим из них.

Дополнительные возможности передачи сообщений. В системе QNX имеются функции, предоставляющие дополнительные возможности передачи сообщений, а именно:

– условный прием сообщений;

– чтение и запись части сообщения;

– передача составных сообщений.

–

Обычно для приема сообщения используется функция Receive(). Этот способ приема сообщений в большинстве случаев является наиболее предпочтительным.

Однако иногда процессу требуется предварительно «знать», было ли ему послано сообщение, чтобы не ожидать поступления сообщения в RECEIVE-блокированном состоянии. Например, процессу требуется обслуживать несколько высокоскоростных устройств, не способных генерировать прерывания и, кроме того, процесс должен отвечать на сообщения, поступающие от других процессов. В этом случае используется функция Creceive(), которая считывает сообщение, если оно становится доступным, или немедленно возвращает управление процессу, если нет ни одного отправленного сообщения.

ВНИМАНИЕ. По возможности следует избегать использования функции Creceive(), так как она позволяет процессу

непрерывно загружать процессор на соответствующем приоритетном уровне.

Примеры обмена сообщениями

Клиент

Передача сообщения со стороны клиента осуществляется применением какой-либо функции из семейства *MsgSend()*.

Мы рассмотрим это на примере простейшей из них — *MsgSend():*

```
include <sys/neutrino.h>
int MsgSend(int cold, const void *smsg, int sbytes,
void *rmsg, int rbytes);
```

Для создания установки соединения между процессом и каналом используется функция *ConnectAttach()*, в параметрах которой задаются идентификатор процесса и номер канала.

```
#include <sys/neutrino.h>
int ConnectAttach( uint32_t nd,
                   pid_t pid,
                   int chid,
                   unsigned index,
                   int flags );
```

Пример:

Передадим сообщение процессу с идентификатором 77 по каналу 1:

```
#include <sys/neutrino.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

char *smsg = “Это буфер вывода”;
char rmsg[200];
int coid;
int main(void);
```

```
{
// Установить соединение
coid = ConnectAttach(0, 77, 1, 0, 0);
if (coid == -1)
{
fprintf(stderr, “Ошибка ConnectAttach к
 0/77/1!\n”);
  perror(NULL);
exit(EXIT_FAILURE);
}
// Послать сообщение
if(MsgSend(coid, smsg, strlen(smsg)+1,rmsg,
sizeof(rmsg)) == -1)
{
fprintf(stderr, “Ошибка MsgSendNn”);
perror(NULL);
exit(EXIT FAILURE);
}
if (strlen(rmsg) > 0)
{
printf(“Процесс с ID 77 возвратил \”%s\”\n”,
rmsg);
}
exit(0);
}
```

Предположим, что процесс с идентификатором 77 был действительно активным сервером, ожидающим сообщение именно такого формата по каналу с идентификатором 1.

После приема сообщения сервер обрабатывает его и в некоторый момент времени выдает ответ с результатами обработки. В этот момент функция MsgSend() должна возвратить ноль (0), указывая этим, что все прошло успешно.

Если бы сервер послал нам в ответ какие-то данные, мы смогли бы вывести их на экран с помощью последней строки в программе (с тем предположением, что обратно мы получаем корректную ASCIIZ-строку).

Сервер

Создание канала. Сервер должен создать канал — то, к чему присоединялся клиент, когда вызывал функцию *ConnectAttach().*

Обычно сервер, однажды создав канал, приберегает его «впрок». Канал создается с помощью функции *ChannelCreate()* и уничтожается с помощью функции *ChannelDestroy():*

```
#include <sys/neutrino.h>
int ChannelCreate(unsigned flags);
int ChannelDestroy(int chid);
```

Пока на данном этапе будем использовать для параметра flags значение 0 (ноль). Таким образом, для создания канала сервер должен сделать так:

```
int  chid;
chid = ChannelCreate (0);
```

Теперь у нас есть канал. В этом пункте клиенты могут подсоединиться (с помощью функции *ConnectAttach()*) к этому каналу и начать передачу сообщений. Сервер отрабатывает схему сообщений обмена в два этапа — этап «приема» (receive) и этап «ответа» (reply).

```
#include <sys/neutrino.h>
int MsgReceive(int chid, void *rmsg, int rbytes,
struct _msg_info *info);
int MsgReply(int rcvid, int status, const void
*msg, int nbytes);
```

Для каждой буферной передачи указываются два размера (в случае запроса от клиента это sbytes на стороне клиента и rbytes на стороне сервера; в случае ответа сервера это sbytes на стороне сервера и rbytes на стороне клиента). Это сделано для того, чтобы разработчики каждого компонента смогли определить размеры своих буферов — из соображений дополнительной безопасности.

В нашем примере размер буфера функции MsgSend() совпадал с длиной строки сообщения.

Пример (Структура сервера):

```
#include <sys/neutrino.h>
#include <sys/iomsg.h>
#include <errno.h>
#include <stdio.h>
```

```
#include <process.h>
void main(void)
{
int  rcvid;
int   chid;
char  message [512];

// Создать канал
chid = ChannelCreate(0);

// Выполняться вечно — для сервера это обычное дело
while (1)
{
// Получить и вывести сообщение
rcvid=MsgReceive(chid, message,
sizeof(message), NULL);
            printf (“Получил сообщение, rcvid %X\n”,
rcvid);
            printf (“Сообщение такое: \”%s\”,\n”,
message);

// Подготовить ответ — используем тот же буфер
strcpy (message, “Это ответ”);
            MsgReply (rcvid, EOK, message,
sizeof (message));
}
}
```

Как видно из программы, функция MsgReceive() сообщает ядру том, что она может обрабатывать сообщения размером вплоть до sizeof(message) (или 512 байт).

Наш клиент (представленный выше) передал только 28 байт (длина строки).

Определение идентификаторов узла, процесса и канала (ND/PID/CHID) нужного сервера

Для соединения с сервером функции ConnectAttach() необходимо указать дескриптор узла (Node Descriptor — ND), идентификатор процесса (process ID — PID), а также идентификатор канала (Channel ID — CHID).

Если один процесс создает другой процесс, тогда это просто — вызов создания процесса возвращает идентификатор вновь созданного процесса. Создающий процесс может либо передать собственные PID и CHID вновь созданному процессу в командной строке, либо вновь созданный процесс может вызвать функцию ***getppid()*** для получения идентификатора родительского процесса, и использовать некоторый «известный» идентификатор канала.

```
#include <process.h>
pid_t getpid( void );
```

Вопрос: «Как сервер объявляет о своем местонахождении?»

Существует множество способов сделать это; мы рассмотрим только три из них, в порядке возрастания «элегантности»:

1. Открыть файла с известным именем и сохранить в нем ND/PID/CHID. Такой метод является традиционным для серверов UNIX, когда сервер открывает файл (например, /etc/httpd.pid), записывает туда свой идентификатор процесса в виде строки ASCII и предполагают, что клиенты откроют этот файл, прочитают из него идентификатор.

2. Использовать для объявления идентификаторов ND/PID/ CHID глобальные переменные. Такой способ обычно применяется в многопоточных серверах, которые могут посылать сообщение сами себе. Этот вариант по самой своей природе является очень редким.

3. Занять часть пространства имен путей и стать администратором ресурсов.

План выполнения

1. Необходимо создать два приложения - клиент и сервер, которые бы обменивались между собой сообщениями, используя стандартные механизмы операционной системы QNX/Neutrino2.

При реализации клиент-серверных приложений требуется предусмотреть возможность работы с сервером нескольких клиентов.

В клиенте имеется возможность консольного ввода команд, которые должны обрабатываться сервером. Сервер должен обрабатывать не менее трех команд посылаемых клиентом:

- help — помощь по командам;
- list /dir — получение содержимого указанного каталога;
- get filename — передача клиенту указанного файла.

Также сервер должен отвечать клиенту, если тот послал ему не верную команду.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

2.6 Лабораторная работа «Управление таймером и периодическими уведомлениями в ОС QNX»

Цель работы

Познакомиться с механизмом управления временем ОС QNX — системным таймером.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Управление таймером

В QNX управление временем основано на использовании системного таймера. Этот таймер содержит текущее координатное универсальное время (UTC) относительно 0 часов 0 минут 0 секунд 1 января 1970 г. Для установки местного времени функции управления временем используют переменную среды TZ.

Процесс может создать один или несколько таймеров. Таймеры могут быть любого поддерживаемого системой типа, а их количество ограничивается максимально допустимым количеством таймеров в системе.

Функция создания таймера позволяет задавать следующие типы механизма ответа на события:

– перейти в режим ожидания до завершения. Процесс будет находиться в режиме ожидания начиная с момента установки таймера до истечения заданного интервала времени;

– оповестить с помощью proxy. Proxy используется для оповещения процесса об истечении времени ожидания;

– оповестить с помощью сигнала. Сформированный пользователем сигнал выдается процессу по истечении времени ожидания.

Вы можете задать таймеру следующие временные интервалы:

– *абсолютный.* Время относительно 0 часов, 0 минут, 0 секунд, 1 января 1970 г.;

– относительный. Время относительно значения текущего времени.

Можно также задать повторение таймера на заданном интервале. Например, вы установили таймер на 9 утра завтрашнего дня. Его можно установить так, чтобы он срабатывал, каждые пять минут после истечения этого времени. Можно также установить новый временной интервал существующему таймеру. Результат этой операции зависит от типа заданного интервала:

- для абсолютного таймера новый интервал замещает текущий интервал времени;
- для относительного таймера новый интервал добавляется к оставшемуся временному интервалу.

При использовании множества параллельно работающих таймеров — например, когда необходимо активизировать несколько потоков в различные моменты времени — ядро ставит запросы в очередь, отсортировав таймеры по возрасту их времени истечения (в голове очереди при этом окажется таймер с минимальным временем истечения). Обработчик прерывания будет анализировать только переменную, расположенную в голове очереди.

При использовании периодических и однократных таймеров у вас появляется выбор:

- послать импульс;
- послать сигнал:
- создать поток.

В данной лабораторной работе будем использовать вторую возможность.

Пример 1. Данная программа запускает в цикле ожидания на 10 секунд и прерывает это ожидание с помощью сигнала.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>
#include <signal.h>
#include <time.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>

int main(void)
{
struct itimerspec timer; // структура с описанием
```

```
 //таймера
timer_t timerid;      // ID таймера

extern void handler(); //Обработчик таймера
struct sigaction act;  //Структура описывающая
//действие
  //по сигналу
sigset_t set; //Набор сигналов нам необходимый
 //для таймера

sigemptyset( &set ); //Обнуление набора
sigaddset( &set, SIGALRM); //Включение в набор
  //сигнала
       //от таймера
act.sa_flags = 0;
act.sa_mask = set;
act.sa_handler = &handler; // Вешаем обработчик
  // на действие
sigaction( SIGALRM, &act, NULL); //Зарядить сигнал,
// присваивание структуры
// для конкретного сигнала
// (имя сигнала,
// структура-действий)

//Создать таймер
if (timer_create (CLOCK_REALTIME, NULL, &timerid)
 == -1)
{
 fprintf (stderr, "%s: ne udalos timer %d\n",
 "TM", errno);
}


// Данный макрос для сигнала SIGALRM не нужен
// Его необходимо вызывать
// для пользовательских сигналов
// SIGUSR1 или SIGUSR2. Функция timer_create()
// в качестве второго параметра должна
// использовать &event.

// SIGEV_SIGNAL_INIT(&event, SIGALRM);
```

```
timer.it_value.tv_sec=  3; //Взвести таймер
  //на 3 секунды
timer.it_value.tv_nsec= 0;
timer.it_interval.tv_sec=  3; //Перезагружать
//таймер
timer.it_interval.tv_nsec= 0; // через 3 секунды

timer_settime (timerid, 0, &timer, NULL);
//Включить таймер

for (;;)
{
sleep(10);              // Спать десять секунд.
// Использование таймера задержки
printf("More time!\n");
}
exit(0);
}

void handler( signo )
{
//Вывести сообщение в обработчике.
printf( "Alarm clock ringing!!!.\n");
// Таймер заставляет процесс проснуться.
}
```

Здесь *it_value* и *it_interval* принимает одинаковые значения. Такой таймер сработает один раз (с задержкой *it_value*), а затем будет циклически перезагружаться с задержкой *it_interval*.

Оба параметра *it_value* и *itinterval* фактически являются структурами типа ***struct timespec*** — еще одного POSIX-объекта. Эта структура позволяет вам обеспечить разрешающую способность на уровне долей секунд. Первый ее элемент, *tv_sec*, — это число секунд, второй элемент, *tv_nsec*, — число наносекунд в текущей секунде. (Это означает, что никогда не следует устанавливать параметр *tv_nsec* в значение, превышающее 1 миллиард — это будет подразумевать смещение на более чем 1 секунду).

Пример 2:

```
t_value.tv_sec = 5;
it_value.tv_nsec = 500000000;
it_interval.tv_sec = 0;
it_interval.tv_nsec = 0;
```

Это сформирует однократный таймер, который сработает через 5,5 секунды. (5,5 секунд складывается из 5 секунд и 500,000,000 наносекунд.)

Мы предполагаем здесь, что этот таймер используется как относительный, потому что если бы это было не так, то его время срабатывания уже давно бы его истекло (5.5 секунд с момента 00:00 по Гринвичу, 1 января 1970).

Пример 3:

```
it_value.tv sec = 987654321;
it_value.tv_nsec = 0;
it_interval.tv_sec = 0;
it_iriterval. tv_nsec = 0;
```

Данная комбинация параметров сформирует однократный таймер, который сработает в четверг, 19 апреля 2001 года

В программе могут быть использованы следующие структуры и функции:

TimerCreate(), TimerCreate_r() — Функция создание таймера

```
#include <sys/neutrino.h>
int TimerCreate( clockid_t id,
const struct sigevent *event );
int TimerCreate_r( clockid_t id,
const struct sigevent *event );
```

Обе функции идентичны, исключение составляют лишь способы обнаружения ими ошибок:

- TimerCreate() — если происходит ошибка, то возвращается значение -1.
- TimerCreate_r() — при возникновении ошибки ее конкретное значение возвращается из секции Errors.

Struct sigevent — Структура, описывающая событие таймера

```
#include <sys/siginfo.h>
union sigval {
   int      sival_int;
   void     *sival_ptr;
};
```

Файл <sys/siginfo.h> определяет также некоторые макросы для более облегченной инициализации структуры *sigevent*. Все макросы ссылаются на первый аргумент структуры sigevent и устанавливают подходящее значение sigev_notyfy (уведомление о событии).

SIGEV_INTR — увеличить прерывание. В этой структуре не используются никакие поля. Инициализация макроса: *SIGEV_INTR_INIT(event)*

SIGEV_NONE — Не посылать никаких сообщений. Также используется без полей. Инициализация: *SIGEV_NONE_INIT(event)*

SIGEV_PULSE — посылать периодические сигналы. Имеет следующие поля:

int sigev_coid — ID подключения. По нему происходит связь с каналом, откуда будет получен сигнал;

short sigev_priority — установка приоритета сигналу;

short sigev_code — интерпретация кода в качестве манипулятора сигнала. sigev_code может быть любым 8-битным значением, чего нужно избегать в программе. Значение sigev_code меньше нуля приводит к конфликту в ядре.

Инициализация макроса: *SIGEV_PULSE_INIT(event, coid, priority, code, value)*

SIGEV_SIGNAL — послать сигнал процессу. В качестве поля используется: int sigev_signo — повышение сигнала. Может принимать значение от 1 до -1. Инициализация макроса: *SIGEV_SIGNAL_INIT(event, signal)*

SignalAction(), SignalAction_r() // функция определяет действия для сигналов.

```
#include <sys/neutrino.h>

int SignalAction( pid_t pid,
                  void (*sigstub)(),
                  int signo,
                  const struct sigaction* act,
                  struct sigaction* oact );
```

```
int SignalAction_r( pid_t pid,
        void* (sigstub)(),
        int signo,
        const struct sigaction* act,
        struct sigaction* oact );
```

Все значения ряда сигналов идут от_SIGMIN (1) до _SIGMAX (64).

Если act не NULL, тогда модифицируется указанный сигнал. Если oact не NULL, то предыдущее действие сохраняется в структуре, на которую он указывает. Использование комбинации act и oact позволяет запрашивать или устанавливать (либо и то и другое) действия сигналу.

Структура *sigaction* содержит следующие параметры:

- *void (*sa_handler)()* — возвращает адрес манипулятора сигнала или действия для неполученного сигнала, действие-обработчик.
- *void (*sa_sigaction) (int signo, siginfo_t *info, void *other)* — возвращает адрес манипулятора сигнала или действия для полученного сигнала.
- *sigset_t sa_mask* — дополнительная установка сигналов для изолирования (блокирования) функций, улавливающих сигнал в течение исполнения.
- int sa_flags — специальные флаги, для того, чтобы влиять на действие сигнала. Это два флага: SA_NOCLDSTOP и SA_SIGINFO.
  - SA_NOCLDSTOP используется только когда сигнал является дочерним (SIGCHLD). Система не создает дочерний сигнал внутри родительского, он останавливается через SIGSTOP.
  - SA_SIGINFO сообщает Neutrino поставить в очередь текущий сигнал. Если установлен флаг SA_SIGINFO, сигналы ставятся в очередь и все передаются в порядке очередности.

Добавление сигнала на установку:

```
#include <signal.h>
int sigaddset( sigset_t *set,
        int signo );
```

Функция ***sigaddset()*** — добавляет signo в set по указателю. Присвоение сигнала набору. ***sigaddset()*** возвращает — 0, при удачном исполнении; -1, в случае ошибки;

Функция ***sigemptyset()* —** обнуление набора сигналов

```
#include <signal.h>
int sigemptyset( sigset_t *set );
```

Возвращает — 0, при удачном исполнении; -1, в случае ошибки.

План выполнения

1. Модифицировать программу, созданную в третьей лабораторной работе, так чтобы в клиенте работали два таймера с разной периодичностью, например. 3 и 4,44 секунды. При срабатывании каждого таймера серверу посылался бы сигнал с номером сигнала таймера.
2. Подготовиться к ответам на теоретическую часть лабораторной работы.

2.7 Лабораторная работа «Использование среды визуальной разработки программ в ОС QNX»

Цель работы

Ознакомление со средой визуальной разработки программ и построение пробного приложения в Phab (Photon Application Builder).

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

Основы работы с Phab

Для запуска среды Phab необходимо выбрать Launch=>Development=>Phab.

После запуска идем в пункты меню File=>New и в окне диалога выбираем тип проекта. Далее сохраняем проект под некоторым именем в своей домашней директории

Компиляция, компоновка и запуск. Осуществляются следующим образом:

1. Application=>Generate;
2. Выбрать платформу gcc;
3. Выбрать Make – создание исполняемого модуля;
4. Выбрать Run – запуск приложения.

Запустите приложение из терминального режима независимо от Phab.

Для того, что бы узнать какие объекты, функции и сообщения необходимы для построения приложения, используйте Launch=>HelpViewer. Ключевым словом может служить «widget».

Компоновка формы:

Чтобы создать рабочую форму с полями ввода и кнопкой, нужно сделать следующее:

1. Разместить элементы на форме.
2. Задать им уникальные имена.
3. Создать действие на нажатие кнопки:

a. Дать имя компилируемому модулю, в котором будет находиться обработчик кнопки.
b. Применить настройки.
c. Создать новый модуль.

4. Написание обработчика кнопки делаются непосредственно в созданном модуле.

Функции, используемые для работы с сообщениями:

*PtGetResource//*взять данные по ресурсу из компоненты формы, например, из поля для ввода текста изъять сам текст.

#define PtGetResource(widget, type, value, len) ...

widget – название ресурса (в данном случае – название поля, компоненты, в которую вводится сообщение, посылаемое клиентом серверу);

type – тип ресурса, например Pt_ARG_COLOR, Pt_ARG_TXT.

value – адрес, то есть куда отправлять сообщение, либо в какую переменную записать.

len – определяется в зависимости от типа ресурса, здесь это длина посылаемого сообщения.

Для того, чтобы взять текст, посланный сервером клиенту в ответ на его сообщение, и поместить в окно редактирования ввода, необходимо использовать функцию

*SetResource//*установить ресурс для данного элемента формы (например, для поля ввода текста)

#define PtSetResource(widget, type, value, len) ...

Пример:

```
PtWidget_t *widget;

PtSetResource( widget, Pt_ARG_FILL_COLOR, Pg_BLUE, 0 );
```

Обе функции возвращают значение 0 при удачной работе и -1 при возникновении ошибки.

План выполнения

1. Построить пробные приложения в Phab (Photon Application Builder).

Создать клиентское приложение, использующее графический интерфейс: 2 поля для ввода текста и кнопка отправки сообщения. Это приложение должно получать от пользователя текст через поле ввода и отправлять его по нажатию кнопки серверу в виде сообщения.

Сервер должен получать сообщение и отвечать ровно через 4,75 секунды.

Ответное сообщение сервера должно приходить во второе поле формы.

2. Подготовиться к ответам на теоретическую часть лабораторной работы.

2.8 Лабораторная работа «Улучшение навыков программирования в ОС QNX»

Цель работы

Осмысление знаний о механизмах и ресурсах ОС QNX и с получение дополнительного опыта программирования в среде ОС QNX.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

Защита программного кода командного файла.

Теоретические основы

В зависимости от варианта задания дополнительные теоретические основы можно получить, используя систему помощи QNX, которая включает как описание функционала операционной системы, так и описание функционала среды разработки.

План выполнения

Выполнить задание согласно Вашего варианта (Вариант должен быть согласован с преподавателем).

Защита программного кода, выполненного в ходе лабораторной работы.

Варианты заданий на выполнение

Вариант 1. Система безопасности летательного аппарата

Описание: Система должна следить за температурой носовой части, передней кромки левого и правого крыла. Всего три датчика температуры. Датчик носовой части должен опрашиваться с частотой 4Гц, датчики крыльев - 2Гц. Датчик возвращает значение температуры в диапазоне 0..65535К.

Задание: Написать программы сервера, моделирующие датчики и клиента - системы безопасности. Пусть значения температуры изменяются по закону косинуса (в случае отсутствия библиотеки тригонометрических функций, следует реализовать функцию косинуса с помощью разложения ряда) в заданном диапазоне.

Программа-клиент должна осуществлять опрос серверов и выводить на экран значение температуры в шесть столбцов (временная отметка, температура). Предусмотреть возможность отказа датчика, клиент не должен при этом блокироваться. Вместо отказавшего датчика в столбце должна выводиться -1.

При запуске должно быть три процесса сервера и один процесс клиент.

Смоделировать отказ датчика можно путем уничтожения одного или нескольких процессов-серверов (kill). Датчик считается потерянным, если он не ответил на два опроса подряд. Но датчик может восстановить свою работу. Моделируется запуском процесса-сервера.

Опции: Значения температуры выводятся разными цветами в зависимости от диапазона температуры.

0-256 - фиолетовый
257-512 - синий
..
.. - 65535 - красный

Вариант 2. Базовая станция сотовой связи

Описание: Процесса моделируют базовые станции сотовой связи (БСС). Станция «ведет» не более 64 терминальных устройств мобильных телефонов (МТ).

Каждый МТ регистрируется на БСС. Моделирующий процесс при запуске оповещает станцию о номере своего процесса и номере телефона. Это пока не вызов, это - регистрация.

При вызове вызывающий МТ должен сообщит БСС номер вызываемого абонента. Станция ищет данный номер среди зарегистрированных, если таковой находится, то станция разрешает передавать данные.

Задание: Написать программы сервера-БСС и клиента-МТ. Клиент должен узнавать номер процесса БСС через пространство имен (см. руководство по QNX/Neutrino). Далее клиент отправляет импульс (или пару импульсов последовательно) в которых оповещает сервер о номере своего процесса и номере телефона. Вызов осуществляется импульсом специального формата (определить самостоятельно). По этому вызову сервер отыскивает абонента и организует канал связи между абонентами.

По окончании связи станция должна определить, сколько продолжался сеанс. В случае если пытается зарегистрироваться 65-ый

по счету МТ. То сервер должен ответить соответствующим импульсом, по которому МТ «поймет», что в доступе отказано.

Вариант 3. Сетевой морской бой

Описание: Для игры в морской бой, запускаются две программы, которые представляют собой графические окна с двумя матрицами-полями 5х5. Одно поле противника, другое свое. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, расставить корабли перед боем (по пять кораблей на поле), начать игру, сдаться. После начала игры пользователи поочередно делаю выстрелы по полям врага, как только, у кого-либо потоплены все корабли, выдается сообщение, что бой закончен: Вы проиграли/выиграли.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер ждет, когда к ней подключаться два клиента. Далее сервер ожидает, когда клиенты проинициализируют свои игровые поля, информация о расположении кораблей храниться на сервере. После инициализации своих игровых полей, любой из клиентов может послать серверу сигнал о начале игры. Сервер случайным образом, выбирает игрока, который начинает первый ход, далее идет игра по правилам морского боя. Сервер получает информацию о выстреле, проверяет его результативность и отвечает клиентам, которые делают отметку в окне на игровом поле. Если у какого-либо клиента потоплены все корабли, то сервер прекращает игру и выдает сообщение о результате игры клиентам.

После окончания игры клиенты могут вновь без перезапуска программы начать игру, проинициализировав игровые поля.

Вариант 4. Сетевые крестики-нолики

Описание: Для игры в крестики-нолики, запускаются две программы, которые представляют собой графические окна с матрицей 3х3. В окне есть кнопки, нажатие на которые реализуют функции: подключиться к серверу, начать игру, сдаться. После начала игры пользователи поочередно делаю ходы, как только, кто-либо проиграл, выдается сообщение, что игра закончена: Вы проиграли/выиграли.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер ждет, когда к ней подключаться два клиента. Далее сервер случайным образом, выбирает игрока, который начинает первый ход и его символ (X или 0), далее идет игра по обычным правилам. Сервер получает информацию о ходе, проверяет

его результативность и отвечает клиентам, которые делают отметку в окне на игровом поле. Если какой-либо клиент выиграл, то сервер прекращает игру и выдает сообщение о результате игры клиентам.

После окончания игры клиенты могут вновь без перезапуска программы начать игру.

Вариант 5. Банкомат

Описание: Пользователь банкомата может, через банкомат идентифицироваться, посмотреть свой счет, получить информацию об операциях с ним (пополнение или изъятие денег), снять деньги или перевести на другой счет.

Задание: Написать консольное приложение-сервер, исполняющее роль банка, и оконное приложение-клиент, исполняющее роль банкомата. На сервере храниться перечень счетов клиентов, их пароли, количество денег и последние десять операций. Приложение клиент имеет оконный интерфейс, через который серверу посылаются запросы.

Вариант 6. Информационная система «Выборы»

Описание: Предварительный подсчет голосов за кандидатов. Число голосов на каждом из 5-х избирательных пунктах постепенно увеличивается. Центризбирком, опрашивает избирательные пункты и выводит результат по каждому из кандидатов. На экране изображаются кандидаты и кол-во голосов по каждому из них. Если у первого больше всего голосов, то он рисуется выше других (не по росту, а по расположению на экране); если у третьего кол-во голосов меньше всех, то он рисуется ниже всех; соответственно второй выше третьего, но ниже первого. Все кандидаты разных цветов.

Задание: Написать консольное приложение-сервер, исполняющее роль избирательного участка, и оконное приложение-клиент, исполняющее роль Центризбиркома. Число голосов на серверах, растет по таймеру. Клиент, также по таймеру, опрашивает сервера.

Вариант 7. Обмен сообщениями со спутником

Описание: В окне приложения нарисована планета, вокруг нее вращается спутник, в поле окна задается сектор контакта со спутником. Когда спутник заходит в сектор общения он начинает посылать сигнал о готовности к общению. Если в окне нажать кнопку «Опрос спутника» спутник вернет свои координаты, которые

отобразятся в окне. Если спутник, находится вне сектора контакта, то данная функция не доступна.

Задание: Написать консольное приложение-сервер, исполняющее роль спутника, и оконное приложение-клиент, исполняющее роль окна на станции наблюдения. Координаты спутника изменяются непосредственно на сервере, а клиент их постоянно опрашивает. Проверяет на вхождение в сектор и отображает спутник на экране.

Вариант 8. Реализовать ЧАТ для пользователей.

Описание: При запуске чата, происходит регистрация пользователя, после соединения с сервером, в окне приложения, показывается список пользователей чата. В программе также имеют два способа по обмену сообщениями: публичный (послать сообщение всем пользователям) и приватный (послать сообщение только конкретному пользователю).

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Клиент – это непосредственно программа ЧАТ, а сервер – программа, которая хранит список, присоединившихся пользователей их ID. Клиент формирует сообщение, состоящее из типа (публичное или приватное), имени пользователя (если сообщение приватное) и текста сообщения. Далее клиент отправляет сообщение серверу. Сервер переправляет это сообщение или конкретному клиенту, или всем пользователям. Сообщения в зависимости от типа, раскрашиваются в разный цвет (посланные или принятые, приватные или публичные).

Вариант 9. Мониторинг состояния доменной печи

Описание: При строительстве доменной печи в ее стенки закладываются термодатчики. Компьютер с заданной периодичностью опрашивает эти датчики и следит за состоянием стенок печи. В случае прогорания стенки печи выдается сигнал тревоги.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер исполняет роль датчика. В нем в специальной переменной хранится информация о длине термодатчика. С определенным интервалом времени длина термодатчика уменьшается. Клиент – это оконное приложение, в котором нарисован план печи с установленными термодатчиками. Клиент опрашивает датчики/сервера об их длине. И отображает полученную информацию на экране. Если длина датчика в пределах 71-100%, то он отображается

зеленым цветом. Если длина датчика в пределах 31-70%, то он отображается желтым цветом. Если длина датчика в пределах 1-30%, то он отображается красным цветом. Если длина датчика достигла 15%, то на кран выдается красное окно с сообщением об опасности.

В клиенте также отображаются и сами значения длин датчиков.

Клиент может работать с независимым количеством датчиков.

Вариант 10. Управление полетом

Описание: Диспетчерская станция управления полетами на земле ведет мониторинг за полетами самолетов с земли. Один раз в секунду опрашивая самолеты об их координатах и высоте. Если самолеты находятся в опасной близости, то диспетчер может подать самолету команду об изменении направления движения. Если диспетчер не подал команду об изменении полета, то может произойти авиакатастрофа.

Задание: Написать консольное приложение-сервер и оконное приложение-клиент. Сервер – это самолеты, при первом запуске у сервера генерируется случайная координата и высота зоны обслуживания диспетчерской станции. Далее генерируется направление полета (точка с координатами на краю зоны обслуживания). Самолет меняет свое местоположение вдоль направления полета. Клиент – это диспетчерская станция, в которой идет отображение плоскости полета вдоль земли и плоскости с разрезом высот. Если самолеты находятся в опасной близости, то они окрашиваются в желтый цвет. Если произошло столкновение, то они окрашиваются в красный цвет, и сервера самолетов, попавших в аварию завершают работу.

3 Методические указания к самостоятельной работе

3.1 Общие положения

Целями самостоятельной работы является систематизация, расширение и закрепление теоретических знаний, приобретение навыков - научно-исследовательской и производственно-технологической деятельности.

Самостоятельная работа по дисциплине «Операционные системы» включает следующие виды активности студента:

- проработка лекционного материала;
- подготовка к лабораторным работам;
- подготовка к экзамену.

3.2 Проработка лекционного материала

Для проработки лекционного материала студентам рекомендуется воспользоваться конспектом, сопоставить записи конспекта с соответствующими разделами учебных пособий [1-3]. Целесообразно ознакомиться с информацией, представленной в файлах, содержащих презентации лекций, предоставляемых преподавателем. Для проработки лекционного материала студентам, помимо конспектов лекций, рекомендуются следующие главы учебных пособий [1-3] по разделам курса:

Глава 1 [1]: Введение в операционные среды, системы и оболочки (Основные понятия. Классификация операционных систем. Классификация построений ядер операционных систем. Представление об интерфейсах прикладного программирования. Платформенно-независимый интерфейс POSIX. Основные принципы построения операционных систем).

Глава 1 [2]: Организация вычислительных задач (Процессы. Ресурсы. Режим мультипрограммирования. Потоки. Волокна. Планирование процессов и диспетчеризация задач. Взаимодействие и синхронизация задач. Прерывания. Управление задачами в ОС Windows).

Глава 3 [1]: Интерфейсы операционных систем (Интерфейс командной строки ОС Windows. Интерфейс командной строки ОС Unix).

Глава 3 [3]: Организация операционных систем реального времени (Функциональные требования ОСРВ. Архитектуры построения ОСРВ. Разделение ОСРВ по способу разработки).

Глава 4 [3]: Стандарты на ОСРВ (SCEPTRE. POSIX. DO-178B. ARINC-653. OSEK).

Глава 5 [3]: Обзор ОСРВ (Классификация ОСРВ в зависимости от происхождения. Системы на основе обычных ОС. Самостоятельные ОСРВ. Специализированные ОСРВ).

Глава 6 [3]: Микроядро ОС QNX Neutrino (Потоки и процессы. Механизмы синхронизации. Межзадачное взаимодействие. Управление таймером. Сетевое взаимодействие. Первичная обработка прерываний. Диагностическая версия микроядра).

Глава 7 [3]: Администратор процессов и управление ресурсами в ОС QNX (Управление процессами. Обработка прерываний. Администраторы ресурсов. Файловые системы. Инсталляционные пакеты. Символьные устройства. Сетевая подсистема. Технология JumpGate. Графический интерфейс пользователя).

При изучении учебно-методического пособия [1-2] студенту рекомендуется самостоятельно ответить на вопросы, приводимые в конце каждой главы. Рекомендуется сформулировать вопросы преподавателю и задать их либо посредством электронной образовательной среды вуза, либо перед началом следующей лекции.

3.3 Подготовка к лабораторным работам

Для подготовки к лабораторным работам «Файлы пакетной обработки в ОС Windows» студентам необходимо изучить раздел 2.1 учебного пособия [1] и пункт 3.1 данных методических указаний.

Для подготовки к лабораторным работам «Программирование на языке SHELL в ОС Unix» студентам необходимо изучить раздел 2.2 учебного пособия [1] и пункт 3.2 данных методических указаний.

Для подготовки к лабораторным работам «Управление процессами в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [3] и пункт 2.3 данных методических указаний.

Для подготовки к лабораторным работам «Управление потоками в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [3] и пункт 2.4 данных методических указаний.

Для подготовки к лабораторным работам «Организация обмена сообщениями в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [3] и пункт 2.5 данных методических указаний.

Для подготовки к лабораторным работам «Управление таймером и периодическими уведомлениями в ОС QNX» студентам необходимо изучить главу 6 учебного пособия [3] и пункт 2.6 данных методических указаний.

Для подготовки к лабораторным работам «Использование среды визуальной разработки программ в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [3] и пункт 2.7 данных методических указаний.

Для подготовки к лабораторным работам «Улучшение навыков программирования в ОС QNX» студентам необходимо изучить главу 7 учебного пособия [3] и ознакомиться с пунктом 2.8 данных методических указаний.

3.4 Подготовка к экзамену

Для подготовки к экзамену рекомендуется повторить соответствующие тематике разделы учебных пособий [1-3]. Экзаменационные вопросы представлены в рабочей программе изучаемой дисциплине, размещенной на образовательном портале ТУСУРа: https://edu.tusur.ru/.

Список литературы

1. Гриценко, Ю. Б. Операционные системы. Ч.1.: учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2009. — 187 с. — Режим доступа: https://edu.tusur.ru/publications/25.

2. Гриценко, Ю. Б. Операционные системы. Ч.2.: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2009. — 230 с. — Режим доступа: https://edu.tusur.ru/publications/31.

3. Гриценко, Ю. Б. Системы реального времени: Учебное пособие [Электронный ресурс] / Ю. Б. Гриценко. — Томск: ТУСУР, 2017. — 253 с. — Режим доступа: https://edu.tusur.ru/publications/6816Зыль С.Н. Операционная система реального времени QNX: от теории к практике. — СПб.: БХВ-Петербург, 2004. — 192с.: ил.