ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение высшего профессионального образования

Московский государственный институт электроники и математики (Технический университет)

Кафедра математического обеспечения систем обработки информации и управления

Программирование в операционной среде UNIX: обмен информацией между параллельными процессами, организация защиты файлов в файловой системе, обработка прерываний

Методические указания к лабораторным работам по дисциплине «Операционные системы»

Специальности:
230401 – Прикладная математика
220101 – Вычислительные машины, комплексы, системы и сети
071900 – Информационные системы и технологии
090102 – Компьютерная безопасность

Москва 2022

Составитель канд. техн. наук, проф. А.Ю. Истратов

УДК 681.1.5

Программирование в операционной среде UNIX: обмен информацией между параллельными процессами, организация защиты файлов в файловой системе, обработка прерываний. / Моск. гос. ин-т электроники и математики; Сост. –А.Ю. Истратов, 2015г. – с.19

Библиогр.: 5 назв.

Рассматриваются вопросы программирования в операционной системе UNIX. Излагаемый материал является общим для всех разновидностей UNIX – систем. Представлена информация о командах ОС UNIX, системных вызовах, среде пользователя. Приведены задания лабораторных работ и примеры выполнения.

Для студентов направлений и специальности «Информационные системы и технологии», «Компьютерная безопасность», «Прикладная математика»

???? ISBN 5–93378–087–1

1. КРАТКИЕ СВЕДЕНИЯ ОБ ОПЕРАЦИОННОЙ СИСТЕМЕ UNIX.

Операционные системы играют огромную роль в эксплуатации современных ЭВМ. ЭВМ представляют основные ресурсы для решения задачи. Однако, чтобы сделать эти ресурсы более доступными для программиста, требуется детально разработанная программа, называемая операционной системой (ОС). ОС обычно предоставляет средства для разработки и запуска программ на ЭВМ, средства управления пространством памяти ЭВМ, средства доступа к периферийным устройствам ЭВМ и к некоторой файловой системе. Кроме перечисленных выше средств, более сложные ОС могут предоставлять и другие возможности, делающие ЭВМ еще более удобной для использования: одновременный доступ к ЭВМ множества пользователей (разделение времени), средства связи между пользователями, одновременное выполнение множества различных программ (мультизадачность), средства разработки и поддержки файлов, средства, облегчающие разработку программного обеспечения, средства для ввода и обработки текстов и средства защиты.

В последние годы широкую известность получила операционная система UNIX, разработанная в начале 70-х годов К. Томпсоном и Р. Ритчи в Bell Telephone Laboratories (США) и первоначально предназначенная для проведения исследовательских работ. Однако, концептуальная целостность системы и целый ряд новых, нетрадиционных и прогрессивных решений, заложенных в нее при создании, показали преимущество ОС UNIX перед другими ОС этого класса, основные из которых следующие:

- структурированная многоуровневая файловая система;
- средства разделения времени;
- возможность для любого пользователя выполнять одновременно более одной программы;
- наличие механизмов, позволяющих одной программе передавать ее результаты в другую программу (необязательное использование дополнительного пространства памяти);
- возможность перевода выдачи результатов работы с одного периферийного устройства на другое;
- встроенный командный интерпретатор и язык shell;
- структурированный язык для системного программирования;
- расширенные средства для ввода, внесения изменений и обработки информации и др.

В нашей стране аналогом ОС UNIX являлась ОС ДЕМОС (ИНМОС). Сначала эта система была ориентирована на ЭВМ класса PDP-11 (СМ ЭВМ, Электроника), а затем перенесена и сферу ЕС ЭВМ, и на персональные компьютеры.

В настоящее время на принципах ОС UNIX разработано достаточно большое количество ОС: XENIX, UNIX HP, UNIX SCO, SOLARIS, LINUX,

KONIX, FREE BSD и др. Поэтому в дальнейшем под системой UNIX будут подразумеваться любые UNIX-подобные системы.

Все работы в системе UNIX представлены множеством конкурирующих процессов. Процесс - потребитель ресурсов, единица работы и управления. Логически, каждый процесс выполняется своим виртуальным процессором в своем виртуальном адресном пространстве. Каждый процесс имеет уникальный номер, называемый идентификатором процесса, который ставится процессу в соответствие при его рождении. Любые обращения к процессу возможны только через его идентификатор.

Работа процесса состоит в выполнении некоторого файла - программы, состоящей из трех сегментов: процедурного сегмента, сегмента данных и динамического сегмента. Процедурный сегмент содержит машинные команды и константы, сегмент данных - данные, которые могут изменяться в процессе работы, а динамический сегмент выделяется при загрузке выполняемого файла в основную память и содержит данные, не инициализируемые при компиляции. Логическое взаимодействие между процессами реализуется через механизм сигналов.

В UNIX реализован интерфейс для обмена информацией между процессами с дисковыми файлами и внешними устройствами. Это существенно упрощает работу с файлами на пользовательском уровне. Работа с любым внешним устройством с точки зрения пользователя рассматривается как работа с обычным файлом. При этом обмен информацией с любыми файлами осуществляется посредством одних и тех же операций ввода-вывода.

Обмен информацией между родственными процессами может выполняться как традиционными методами, через общие файлы, так и с помощью специального средства - программного канала.

Исключительно большую роль в UNIX играет файловая система. Она имеет многоуровневую структуру, в которой каждый уровень определяется файлом-каталогом. Такую структуру, с некоторыми оговорками, можно назвать древовидной. Помимо файлов-каталогов в состав файловой системы входят обычные и специальные файлы. Любой файл файловой системы может быть указан его полным именем, в котором перечисляются все каталоги, начиная с корневого, через которые проходит путь к данному файлу. Помимо этого, файл может указываться именем относительно текущего каталога. Текущим называется каталог, в котором в данный момент находится пользователь и, который обычно указывает файлы этого пользователя или группы пользователей.

Файловая система обеспечивает защиту файлов от несанкционированного доступа на уровнях пользователя или группы пользователей.

Как с каждым процессом связан идентификатор процесса, так и с каждым файлом в UNIX связан индексный дескриптор, содержащий его основные характеристики. Такой подход делает программы полностью независимыми от местоположения объектов, с которыми они работают.

Все действия в рамках UNIX координируются ядром системы. Ядро во время работы постоянно находится в оперативной памяти и содержит системные программы и управляющие структуры данных. В его функции входит:

- диспетчеризация процессов, включая распределение ресурсов для них;
- выполнение системных вызовов, запросов от процессов к операционной системе;
- управление устройствами при выполнении операций ввода-вывода;
- управление файловой системой;
- распределение и перераспределение оперативной памяти.

Ядро всегда работает в режиме "система". Оно обеспечивает мультипрограммную поддержку в многопользовательском (многотерминальном) режиме, называемым также режимом разделения времени. Количество функций ядра в UNIX минимизировано за счет того, что большое число системных задач решается специальными утилитами.

Пользователь взаимодействует с системой на уровне командного языка. В командном языке каждой команде соответствует свой выполняемый файл, поэтому набор команд может быть легко расширен за счет добавления новых команд и соответствующих им выполняемых файлов. Обработка команд командного языка производится интерпретатором команд shell. По построению командный язык напоминает процедурный язык программирования высокого уровня. В частности, он позволяет организовывать конвейеры команд и содержит конструкции типа условного оператора, цикла и т.п. Это делает программирование в UNIX двухуровневым. Программист может писать свои программы на выбранном языке программирования, например, Си, но он может написать программы и на командном языке. Допускается и смешанный вариант программирования, когда часть программ пишется на языке программирования, а затем обращение к ним производится из программ, написанных на командном языке. При этом надо иметь в виду, что программы, реализованные на командном языке, работают значительно медленнее из-за интерпретации каждой команды.

UNIX поддерживает ряд систем программирования, но наиболее удобной из них является система программирования Си.

Си - язык системного программирования, однако его с успехом можно применять как для задач вычислительного характера, так и для задач обработки данных. Обладая многими свойствами, присущими языкам типа Ассемблера, он остается в то же время процедурным языком программирования, и его использование существенно повышает производительность труда программистов.

Сама по себе система UNIX невелика по объему и довольно проста. Средства системы выразительны и имеют удобную мнемонику. Допускается изучение системы по частям, что дает возможность

пользователю ограничиться изучением только того набора средств, который ему в данный момент необходим. Как показала практика, неподготовленный пользователь выходит на режим самостоятельной работы через одну - две недели после начала освоения UNIX.

Инструментальность UNIX проявляется в ее насыщенности различными программными средствами, облегчающими процесс конструирования программного обеспечения. Инструментальные средства UNIX можно разделить на следующие группы:

- утилиты различного назначения, представляющие собой строительные блоки, которые с успехом могут использоваться при разработке сложных программных комплексов. Помимо этого, являясь наполнением некоторых команд командного языка, они в среде интерпретатора shell обеспечивают интерактивную работу пользователя за терминалом;
- средства работы с текстами (к ним относятся строковые и экранные редакторы, форматеры текстов и специальные утилиты, работающие с текстовой информацией);

- средства поддержки разработок программного обеспечения, к которым относятся координатор make и система контроля и документирования текстовых файлов sccs. Они используются для автоматизации процесса формирования сложных многомодульных программ в период их отладки и корректировке, документирование процесса разработки и ведения архива;
- средства генерации программ анализа - генераторы синтаксических и лексических анализаторов yacc и lex используются для автоматического построения соответствующих блоков транслирующих систем, а также специализированных программ обработки символьной информации;
- интерпретатор shell, являющийся важным инструментальным средством, позволяющим на уровне командного языка строить новые инструментальные средства проблемной направленности.

Свойство мобильности, которое присуще UNIX, означает возможность переноса системы с одной машинной архитектуры на другую с минимальными затратами. Мобильность UNIX обеспечивается разумным выбором соответствующей виртуальной UNIX-машины и уровнем языка, на котором система написана. Виртуальная машина должна быть достаточно абстрактной, чтобы при создании виртуальной системы можно было бы оперировать такими общими понятиями, как процесс, ресурс, файл и т. п., и в то же время не отрываться далеко от реальной ЭВМ, чтобы машинно-зависимая часть системы была наименьшей из возможных. В UNIX на уровне виртуальных понятий рассматриваются: управление процессами, распределение памяти, ввод-вывод, командный язык и др., а машинно-зависимая часть четко выделена.

Наряду с достоинствами, системе UNIX присущ ряд недостатков: не поддерживается режим реального времени; слабая устойчивость к аппаратным сбоям; снижение эффективности при решении однотипных задач. В UNIX слабо развиты средства взаимодействия и синхронизации процессов, отсутствуют средства гибкого управления их приоритетами. Все это приводит к низкой реактивности системы и делает ее использование в режиме реального времени нецелесообразным.

С другой стороны, поскольку UNIX, развивалась с ориентацией на область научных исследований, в ней отсутствовали некоторые возможности, необходимые в коммерческих применениях. Ряд разработчиков программного обеспечения предложили собственные коммерческие версии ОС UNIX, полученные путем ее расширения, например, системы IDRES, CROMIX, COGERENT, ONIX, XENIX. Последняя система - одна из наиболее широко известных, разработана фирмой Microsoft в начале 90-х годов с возможностями широкого коммерческого применения.

Из-за возрастающей популярности системы UNIX перечень прикладных программ, способных работать под ее управлением, постоянно увеличивается. Имеются программы обработки текстов, верстки газет, управления базами данных; трансляторы для языков Бейсик, Фортран, Кобол, Си/Си++, Паскаль, Ассемблеров и др. языков программирования. В настоящее время, благодаря техническому прогрессу, микро-ЭВМ имеют достаточно мощные вычислительные ресурсы, чтобы удовлетворить жестким требованиям функционирования ОС UNIX. Стало возможным выполнение большинства программ, которые широко применялись под управлением операционных систем CP/M, MS-DOS, PC-DOS, VAX, WINDOWS 2000. Вследствие возрастания коммерческого и прикладного значения ОС UNIX, а также удобства в эксплуатации, перспективы ее распространения и спроса становятся неограниченными.

2. ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ UNIX

Функционально программный интерфейс ОС UNIX может быть условно подразделен на две подсистемы: файловую систему и систему управления процессами. Первая представляет собой совокупность специально организованных наборов данных, хранящихся на внешних устройствах ЭВМ, и программных средств, гарантирующих доступ к этим данным и их защиту, а вторая обеспечивает разделение времени. Программный интерфейс, т.е. интерфейс между пользовательской программой и ядром ОС UNIX реализуется с помощью так называемых системных вызовов. Интерфейс между двумя пользовательскими программами (или между пользовательской программой и внешним устройствам, а также между двумя процессами) в ОС UNIX реализуется в рамках единой структуры данных, называемой файлом. Теоретически, файл ОС UNIX представляет собой последовательность байт данных,

завершающуюся символом логического конца файла. Физически, такая последовательность байт может представлять собой, например, набор блоков диска или магнитной ленты, либо область оперативной памяти. Среди менее привычных представлений файла в ОС UNIX можно назвать пользовательский терминал, печатающее устройство, трафик сети ЭВМ и т. п. Таким образом, если пользователь умеет программировать операцию ввода-вывода в файл, то сможет запрограммировать и операцию ввода-вывода на любое устройство.

Все файлы ОС UNIX имеют имена, которые могут быть использованы пользовательскими программами как средства получения доступа к данным, содержащимся в соответствующих файлах. Файлы ОС UNIX являются составляющими некоторой системы данных, называемой файловой системой ОС UNIX.

Система управления процессами реализует такие элементарные функции, как порождение процесса, завершение его функционирования и обмен данными между двумя функционирующими процессами. Кроме того, она осуществляет динамическое распределение оперативной памяти ЭВМ между двумя или несколькими процессами.

Интерфейс между любой пользовательской программой (процессом) и ядром ОС UNIX (программный интерфейс ОС UNIX), реализуется с помощью системных вызовов. Синтаксически, применение системного вызова (СВ) похоже на вызов подпрограммы, однако, код, реализующий этот СВ, находится в ядре ОС UNIX. При осуществлении СВ, как правило, используется механизм прерываний, реализуемый аппаратно. Более подробно эту процедуру можно описать так: при осуществлении СВ, в стек пользовательского процесса заносятся соответствующие параметры (как и в случае вызова подпрограммы), после чего на процессор вызывается инструкция программного прерывания (в случае вызова подпрограммы на процессор вызывается инструкция перехода по стартовому адресу вызываемой подпрограммы). В результате обработки прерывания аппаратурой центрального процессора ЭВМ, управление передается по адресу, хранящемуся в некоторой заранее определенной ячейке памяти ЭВМ, так называемом векторе прерывания, и, тем самым, начинается выполнение подпрограммы обработки прерывания, исполняемый код которой находится в ядре ОС UNIX. Подпрограмма обработки прерывания прежде всего извлекает из стека пользовательского процесса ранее помещенные туда параметры, а затем передает управление подпрограмме, реализующей системную функцию, соответствующую осуществленному СВ; исполняемый код этой подпрограммы также находится в ядре ОС UNIX. После того, как подпрограмма, реализующая указанную системную функцию, завершится, ядро ОС UNIX передаст управление в пользовательскую программу, осуществившую СВ, точно так же, как если бы завершилась подпрограмма, вызванная на выполнение пользовательской программой.

Системным вызовам соответствуют подпрограммы, объединенные в объектную библиотеку, которая компонуется с пользовательской программой по умолчанию. Каждая такая подпрограмма предоставляет возможность пользовательской программе осуществлять СВ.

В следующем разделе будут рассмотрены наиболее употребляемые системные вызовы.

3. ПРОГРАММНЫЙ ИНТЕРФЕЙС ОС UNIX

Интерфейс между пользовательской программой и ядром ОС UNIX охватывает более 1000 системных вызовов. Подробно перечислить и охарактеризовать их в методических указаниях не представляется возможным. Ниже будут приведены СВ, которые могут найти применение при подготовке к лабораторным работам (с полной информацией по системным вызовам можно познакомиться в [1], или с помощью подсказки ОС UNIX : man <имя СВ>) :

- alarm - посылает процессу сигнал побудки;
- fork, vfork - создает копию текущего процесса;
- getppid - возвращает идентификатор процесса-предка;
- getpid - возвращает идентификатор текущего процесса;
- kill - посылает сигнал одному или нескольким процессам;
- nice - устанавливает приоритет текущему процессу;
- plock - фиксирует в памяти текущий процесс;
- sleep - приостанавливает выполнение программы на заданный интервал времени;
- wait,waitpid - возвращает управление текущему процессу после завершения процесса-потомка;
- pause - приостанавливает функционирование текущего процесса;
- creat - создает и открывает файл для записи;
- open - открывает существующий файл;
- close - закрывает файл;
- link - создает жесткую ссылку на существующий файл;
- dup,dup2 - создают копию пользовательского дескриптора файла;
- lseek - перемещает указатель чтения-записи открытого файла;
- mknod - создает новый файл, каталог или специальный файл;
- pipe - осуществляет создание межпроцессного канала;
- read - осуществляет чтение из файла заданного числа байт;
- stat, fstat - осуществляет получение информации о индексном дескрипторе файла;
- write - осуществляет запись в файл заданного числа байт;
- umask - осуществляет получение информации о значении битов кода защиты созданного файла;
- sync - осуществляет принудительное завершение всех

операций ввода-вывода;

- system - выполняет указанную командную строку;
- exec, execl ... -осуществляют загрузку и выполнение программ;
- signal, sigset, sigaction - предоставляет процессу определить свою реакцию на получение того или иного сигнала.

Следует отметить, что, как правило, прерывание выполнения СВ в результате получения пользовательским процессом сигнала, невозможно (это определяется самой природой СВ - функционирование ядра ОС UNIX не может быть прервано), за исключением ряда СВ, связанных с выполнением операций ввода-вывода, таких как creat, open, close, read, write, а также СВ wait и pause. Возвращаемым значением СВ, выполнение которого было прервано, всегда является целое число –l, а значение глобальной переменной errno из файла <errno.h> устанавливается равным EINTR.

4. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС ОС UNIX

Пользовательский интерфейс ОС UNIX реализуется на уровне интерпретатора команд shell, - одной из наиболее важных и сложных программ системы. Она обеспечивает интерфейс "пользователь - ядро" и выполняет следующие основные функции:

- интерпретирует команды;
- обрабатывает имена файлов, определенные через метасимволы;
- осуществляет переадресацию ввода-вывода;
- создает среду пользователя;
- поддерживает командный язык.

Команды интерпретатора shell реализуются обычными программами, записанными на языке Си и собранными в одном из каталогов файловой системы UNIX. Условно все команды можно разбить на три группы: информационные, служебные и работы с файловой системой и файлами.

Информационные команды позволяют получать различного рода справочную информацию. Наиболее важные команды этой группы следующие(*):

- date - вывести или установить дату и время;
- ps - получить информацию о процессах;
- who - получить информацию о работающих в системе;
- tty - получить информацию о собственном терминале;
- cal - вывести календарь для данного месяца данного года.

Служебные команды дают возможность выполнить ряд специальных функций, таких как установка среды пользователя, печать параметров,

установка прав доступа и др. К наиболее важным командам этой группы относятся (*):

- stty - установить характеристики терминала;
- echo - выдать параметры;
- kill - завершить процесс;
- chmod - изменить права доступа.

Самая многочисленная группа команд обеспечивает работу с файловой системой и отдельными файлами. Здесь следует выделить команды (*):

- pwd - вывод полного имени текущего каталога;
- cd - изменение текущего каталога;
- ls - вывод информации о файлах текущего или заданного каталога;
- red - вызов экранного редактора для создания или корректировки файлов;
- lpr - построчная печать файлов;
- mv - пересылка и переименование файлов;
- cp - копирование файлов;
- rm - уничтожение файлов;
- cat - объединение файлов с последующим выводом результата на экран;
- pr - печать файлов;
- wc - вывод количества строк, слов и символов в одном или нескольких файлах;
- du - вывод информации о количестве блоков, занятых каждым файлом, и общего количества блоков для всех файлов;
- find - поиск требуемого файла;
- file - определение типа файла;
- grep - поиск строк файлов по шаблону;
- cmp - сравнение двух или нескольких файлов;
- sort - сортировка или соединение файлов с помещением результата в заданный файл.

ПРИМЕЧАНИЕ (*)

Более подробно с набором и аргументами команд ОС UNIX можно познакомиться в [2], [3] или с помощью подсказки ОС UNIX: man <имя команды>.

Всего в UNIX-системах реализовано более 1130 команд, оперируя которыми пользователь может вести активный диалог с системой и выполнять представительный набор функций.

5. ЦЕЛЬ ЛАБОРАТОРНЫХ РАБОТ

Цель лабораторных работ 1-3 заключается в приобретении знаний и навыков работы на процедурном программном уровне с ядром операционной системы, в организации обмена информацией между параллельными процессами, в организации защиты файлов файловой системы, в изучении механизма прерываний и переключения реакции обработки прерывания на пользовательскую программу. При подготовке и выполнении лабораторных работ будут широко использоваться знания, полученные в ходе изучения дисциплин "Теоретические основы ЭВМ" (внутреннее устройство аппаратуры ЭВМ) и "Алгоритмические языки и программирование" (язык программирования Си). Студенты познакомятся с такими понятиями, как создание и выполнение параллельных процессов, обмен информацией между двумя или несколькими процессами через программные каналы, через ядро операционной системы (в дополнение к обмену информацией через файлы), запуск из одной пользовательской программы другой пользовательской программы, передача при поступлении прерывания управления заданной функции.

Эти лабораторные работы помогут студентам сформировать взгляд на развитие современного математического обеспечения ЭВМ, путях его разработки и совершенствования; увидеть все преимущества и недостатки ОС UNIX в сравнении с другими операционными системами.

6. ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Пусть требуется выполнить следующее задание: написать программу, предоставляющую информацию обо всех активных процессах в системе UNIX. Передать эту информацию через межпроцессный канал в параллельный процесс и вывести в нем идентификаторы всех процессов, не управляемых пользовательскими терминалами. Предусмотреть в программе возможность неоднократного прерывания от клавиатуры.

Текст программы можно представить в следующем виде:

```
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
sigjmp_buf obl; /* область памяти для запоминания
                   состояния процесса */
int ll = 0;   /* счетчик прерывания */
main()
{
    void pol3(); /* подпрограмма обработки прерывания */
    char n1[100], n2[100]; /* символьные массивы для считывания
```

```
                              информации из канала */
int vv;     /* флаг завершения работы программы */
int gg = 0;
int g[2];   /* дескрипторы межпроцессного канала */
signal (SIGINT, po13); /* уведомление о том, что в случае
прихода сигнала прерывания SIGINT, управление передается
процедуре po13 */
sigsetjmp (obl,1); /* запоминание текущего состояния
процесса */
vv = 0;
do {
   sigsetjmp(obl,1);
   pipe(g); /* создание межпроцессного канала */
   if (fork() == 0)   /* распараллеливание процесса */
   {                  /* процесс-потомок */
       close(1);      /* закрытие стандартного вывода */
       close(g[0]);   /* закрытие межпроцессного канала на
       чтение */
       dup2(g[1],1);  /* дублирование дескриптора
       межпроцессного канала на стандартный вывод */
       close(g[1]);   /* удалить копию */
       execl("/bin/ps",  "ps",  "ax", 0); /* вывод всех
       активных процессов в системе в межпроцессный канал
       */
   }
   else
   {                  /* процесс-родитель */
       wait(&s);      /* ожидание окончания процесса-
       потомка */
       sigsetjmp(obl,1);
       close(g[1]);
       read(g[0], n, 80); /* считывание 1-ой записи из меж-
       процессного канала */
       read(g[0], nl, 80); /* считывание 2-й записи */
       rr = cmpstr(n, nl); /* сравнение двух строк */
       sigsetjmp(obl,1);
       while (rr != -1)
       {
           if(n1[7] == 'j') break; /* процесс управляется
              пользовательским терминалом */
           m = atoi(nl); /* определение первого целого
                   числа из строки nl */
           if(m == 0) vv = 1;
           if((gg == 0) && (m != 0)) printf ("%d\n",
           gg);
           if (gg != m) printf ("%d\n", m);
           gg = m;
           read (g[0], n, 80);
           rr = cmpstr(n, nl);
           strcpy(nl, n);
           sleep( 1);    /* ожидание 1 сек.  */
       }
```

```
        }
    }
    while (vv != 1);
    sigsetjmp(obl,1);
    printf("good bye !!!\n");
}
/* Подпрограмма обработки прерывания */
void po13()
{
    ll ++;
    signal (SIGINT, po13);
    if (11 > 9)  /* не больше 9ти прерываний */
    {
        printf ("good bye\n");
        exit(1);
    }
    printf(" П Р Е Р Ы В А Н И Е   !!!  \n");
    siglongjmp (obl, 1);    /* возвращение на последний
    setjmp */
}
/* Подпрограмма сравнения строк */
int cmpstr(vv,  nn)
{
    char vv[ ],  nn[ ];
    int  i,  ml,  m2;
    for (m1 = 0; vv[m1] != '\0'; ml ++);
    for (m2 = 0; vv[m2] != '\0'; m2 ++);
    if (m1 != m2) return (0); /* длины строк vv и nn разные */
    for (i = 0; i < ml; i ++)
    {
        if (vv[i] != nn[i]) return (i + 1); /* строки равны по
        длине, но отличаются (i + 1) символом */
        return(-1); /* строки одинаковы */
    }
}
```

7. ЭТАПЫ ПОДГОТОВКИ И ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Прежде чем приступить к выполнению лабораторной работы, необходимо познакомиться с выполнением системных вызовов и команд ОС UNIX ([1] - [5]). После этого целесообразно подготовить тексты программ.

Перед выходом на ЭВМ желательно познакомиться с описанием команд одного из текстовых редакторов (EE, VI, ED, EDIT, NANO или др.) [2], [3], [4].

При известных идентификаторе и пароле, а также работающем терминале процедура входа в систему осуществляется в следующей последовательности:

1. Установить связь между ЭВМ и Вашим терминалом. Для этого достаточно выполнить команду putty.exe с указанием IP адреса, номера порта и протокола обмена. После этого система ответит коротким простым сообщением:

ИМЯ :

2. Необходимо представить себя системе, т.е. вслед за подсказкой "ИМЯ"(“LOGIN:”) набрать с клавиатуры свой идентификатор и нажать клавишу <ENTER>. Система откликнется сообщением ПАРОЛЬ : (PASSWORD:)
3. Ввести свой пароль (он не отображается на экране).

При успешном выполнении этих операций на экране появится знак #, $ или %. Это сигнализирует об успешном входе в систему и является подсказкой интерпретатора shell (# - для привилегированного пользователя). Далее можно вводить любую команду системы, в частности, для ввода и редактирования текста, трансляции программы и т.п.

Например, vi a.c - запись текста программы в файл a.c с помощью экранного редактора vi.

После создания файла a.c и записи в него программы, можно приступать к его трансляции:

cc -o a.exe a. c,

где a.exe - имя выходного загрузочного файла; -o - ключ транслятора Си, указывающий, что имя выполняемого модуля будет a.exe.

Вслед за этапом трансляции следует этап выполнения. На экране набирается имя загрузочного файла (./a.exe) и нажатием клавиши <ENTER> он запускается на выполнение.

8. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

1. Написать программу, осуществляющую вывод в создаваемый по запросу файл через межпроцессный канал из параллельного процесса полного имени текущего каталога и списка файлов текущего каталога. Предусмотреть в программе возможность неоднократного прерывания от клавиатуры. При поступлении первого прерывания вывести дополнительно и количество блоков, отводимых под каждый файл текущего каталога. При большем количестве прерываний вывести только общее количество блоков, отведенных под каталог.
2. Написать программу, осуществляющую выполнение команды ОС UNIX ls таким образом, чтобы данные на стандартный ввод команды ls (ключи, имена файлов) в параллельном процессе потомке поступали из стандартного вывода основного процесса (где они вводились бы по запросу) через межпроцессный канал. Предусмотреть в программе возможность неоднократного прерывания от клавиатуры. При поступлении трех сигналов прерывания дополнительно вывести количество строк для каждого

файла, указанного в аргументах. В остальных случаях просто продолжать выполнение программы.

3. Написать программу подсчета количества блоков, занимаемых каждым файлом и общего количества блоков для всех файлов текущего каталога. Информацию переслать через межпроцессный канал. И при считывании ее из канала в параллельном процессе извлечь данные только о том файле, имя которого введено по запросу, а также об общем количестве блоков. Предусмотреть возможность неоднократного прерывания от клавиатуры. При поступлении пяти сигналов прерывания вывести информацию о количестве символов указанного файла В остальных случаях просто продолжать выполнение программы.
4. Написать программу поиска в файле, имя которого вводится по запросу неограниченное число раз, строки с заданным шаблоном. Полученную информацию через область внешних аргументов переслать в параллельный процесс и в нем вывести её в файл. Предусмотреть возможность неоднократного прерывания по сигналу <CTRL>+<C>. При поступлении более десяти сигналов прерывания окончить выполнение программы.
5. Написать программу вывода календаря текущего месяца текущего года и по введенной дате определить день недели. Предусмотреть возможность неоднократного прерывания выполнения программы от клавиатуры. При поступлении трех сигналов прерывания вывести информацию о том, сколько дней в указанном году.
6. Написать программу выдачи списка всех пользователей, работающих в настоящее время в системе (имена связанных с ними терминалов и время входа в систему не указывать). Передать полученную информацию через межпроцессный канал в параллельный процесс, где вывести на экран. Предусмотреть возможность неоднократного внешнего прерывания. При каждом прерывании вывести время его поступления.
7. Написать программу поиска и удаления по запросу во всех каталогах, имеющих общий родительский каталог, файлов с расширением txt. Предусмотреть возможность неоднократного прерывания по сигналу <CTRL>+<C>. При поступлении каждого сигнала прерывания выводить информацию о том, сколько файлов уже удалено.
8. Написать программу определения типа файла, указанного при запросе. Полученную информацию через файловую систему передать в параллельный процесс, где вывести на печать. Предусмотреть возможность поступления неоднократного прерывания от клавиатуры. При поступлении каждого сигнала прерывания выводить количество файлов (из числа указанных при запросе), содержащих текст программ на Си.
9. Написать программу, осуществляющую трансляцию и выполнение

файлов на языке программирования Си, имена которых вводятся по запросу. Предусмотреть возможность неоднократного прерывания от клавиатуры. При поступлении более пяти сигналов прерывания распечатать все файлы текущего каталога с расширением *.c.

10. Написать программу вывода информации о количестве блоков, занимаемых каждым файлом текущего каталога. Передать эту информацию через межпроцессный канал в параллельный процесс и в нем вывести имена тех файлов, размер которых превышает 4 блока. Предусмотреть возможность неоднократного внешнего прерывания. При поступлении каждого пятого прерывания выводить информацию об общем количестве блоков, отведенных под файлы текущего каталога.
11. Написать программу сортировки файлов текущего каталога по времени последнего доступа. Передать полученную информацию через межпроцессный канал в параллельный процесс и в нем вывести на экран имена только тех файлов, которые отмечены текущей датой. Предусмотреть возможность неоднократного, прерывания от клавиатуры. При поступлении каждого второго сигнала прерывания выводить количество символов (байтов) выделенных файлов.
12. Написать программу, предоставляющую информацию обо всех активных процессах в системе. Передать эту информацию через межпроцессный канал в параллельный процесс и вывести в нем идентификаторы всех процессов. Предусмотреть возможность неоднократного прерывания от клавиатуры. При поступлении четвертого прерывания вывести имена всех задействованных терминалов в системе в настоящее время.
13. Написать программу подсчета числа строк, слов и символов указанного по запросу текстового файла. Полученную информацию передать через межпроцессный канал в параллельный процесс и в нем вывести только информацию о количестве слов, записав ее при этом в файл. Предусмотреть возможность неоднократного прерывания по сигналу <CTRL>+<C>. При поступлении каждого нечетного прерывания выводить информацию обо всех текстовых файлах текущего каталога.
14. Написать программу, предоставляющую информацию о содержимом очереди на печать. Передать эту информацию через файловую систему в параллельный процесс и в нем вывести на экран дисплея содержимое тех файлов, которые хотят распечатать. Предусмотреть возможность неоднократного прерывания от клавиатуры. При поступлении 3-его прерывания вывести количество файлов в очереди на печать.
15. Написать программу вывода списка всех пользователей, работающих в настоящее время в системе. Передать полученную информацию через файловую систему в параллельный процесс и вывести на экран

только имена задействованных терминалов. Предусмотреть в программе неоднократное прерывание от клавиатуры. При поступлении второго прерывания вывести на экран дисплея содержимое корневого каталога.

16. Написать программу, создающую и заполняющую текстовый файл. Информацию этого текстового файла через межпроцессный канал передать в параллельный процесс. В нем после каждого символа текста вписать в файл порядковый номер и передать через межпроцессный канал в параллельный процесс, где разделить каждый символ со своим номером пробелом. Из текущего процесса переслать полученную информацию в исходный процесс и там вывести на печать. Предусмотреть возможность прерывания от клавиатуры.
17. Написать программу, определяющую права доступа для всех файлов текущего каталога. Передать эту информацию через файловую систему в параллельный процесс и вывести данные только о тех файлах, дата создания которых не превышает семи дней от текущей даты. Предусмотреть возможность прерывания от клавиатуры.
18. Написать программу копирования содержимого одного файла в другой, создавая последний в случае необходимости, и оформить ее в виде команды интерпретатора shell ОС UNIX с выдачей сообщения о количестве блоков копируемого файла (блок = 512 байт) и байтов. Предусмотреть аномальные ситуации и возможность прерывания от клавиатуры.
19. Написать программу, объединяющую два введенных по запросу файла в один. Переслать содержимое полученного файла через межпроцессный канал в параллельный процесс, где получить информацию о количестве строк, слов и символов этого файла. Предусмотреть возможность прерывания от клавиатуры. При поступлении первого прерывания вывести информацию о количестве строк, слов и символов первого файла, при поступлении второго прерывания вывести те же самые атрибуты второго файла
20. Написать программу сравнения и вывода результата двух введенных по запросу файлов. Информацию переслать через межпроцессный канал в параллельный процесс, где определить число блоков, занимаемых каждым из введенных файлов. Предусмотреть возможность прерывания по сигналу <CTRL>+<C>. Если поступило более 3-х сигналов прерывания, то создать новый каталог в текущем и записать туда сравниваемые файлы.
21. Написать программу, формирующую командный файл, транслирующий и запускающий на выполнение введенный по запросу и переданный через межпроцессный канал файл. Предусмотреть возможность неоднократного прерывания от клавиатуры. При поступлении 2-го прерывания возобновить

выполнение исходной программы.

22. Написать программу, посылающую почту всем соседним пользователям. Предусмотреть возможность прерывания от клавиатуры. При поступлении пятого прерывания вывести на экран имена терминалов, на которые посылалась почта.
23. Написать программу поиска в одном из соседних каталогов файлов размером более, чем в три блока, к которым обращались менее, чем пять дней назад. Полученную информацию через файловую систему послать в параллельный процесс. Предусмотреть возможность неоднократного внешнего прерывания. При поступлении 2-го прерывания вывести информацию об общем количестве блоков, занимаемых этими файлами.
24. Написать программу, осуществляющую копирование введенного по запросу файла. Информацию переслать через межпроцессный канал в параллельный процесс-потомок, где проверить, нет ли различий между исходным файлом и его копией, и исходный файл удалить, если нет различий. Предусмотреть возможность неоднократного прерывания по сигналу <CTRL>+<C>. При поступлении 1-го прерывания переименовать файл в исходный и распечатать его содержимое.
25. Написать программу нахождения файлов с расширением *.c, принадлежащих данному пользователю, и изменить у них код защиты на введенный. Предусмотреть возможность прерываний от клавиатуры. Первые пять прерываний игнорировать. При поступлении большего числа прерываний вывести количество файлов с измененным кодом защиты.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дансмур М., Дейвис Г. Операционная система UNIX и программирование на языке Си. - М.: Радио и связь, 1989 г.
2. Готье Р. Руководство по операционной системе UNIX - М.: Финансы и статистика, 1985 г.
3. Браун С. Операционная система UNIX. - М.: Мир, 1986 г.
4. Чан Т. Системное программирование на C++ для UNIX. – Киев: Издательская группа BHV, 1999 г.
5. Моли Б. UNIX/LINUX: Теория и практика программирования.- М: КУДИЦ_ОБРАЗ, 2004 г.
6. Стивене Р.У., Раго С.А. UNIX. Профессиональное программирование. 2-е издание. – СПб.: Символ-Плюс, 2007 г.
7. Готье Р. Руководство по операционной системе UNIX - М.: Финансы и статистика, 1985 г.

8. Браун С. Операционная система UNIX. - М.: Мир, 1986 г.
9. Петерсен Р. LINUX: руководство по ОС. Т1, Т2. - Киев: Издательская группа BHV, 1999 г.
10. Тейнсли Д. LINUX и UNIX: программирование в shell. Руководство разработчика. - Киев: Издательская группа BHV, 2001 г.
11. Кейт Хэвиленд, Дайна Грэй, Бен Салама Системное программирование в UNIX: Пер. с англ. - М., ДМК Пресс. - 368 с., ил. (Серия «Для программистов»), 2015 г.