Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра измерительных информационных технологий

Шмаков В.Э.

Администрирование в информационных системах

Методические указания к лабораторным работам

Санкт-Петербург
2015

СОДЕРЖАНИЕ

Занятие первое

- Войдите в систему,
  login:gr401_1 *(или login: gr401_2)*
  pw : uheggf,frfkfdhjd

- Запустите терминал нажатием комбинации клавиш *Ctrl + Alt + t*
  (попутно можете познакомиться с графическим интерфейсом
  Ubuntu 14.04, если это интересно).

- Выполните на терминале команды *shell* , которые упоминались в конце
  лекции №1 :
  pwd , ***who*** , ***ls*** , ***cd*** , ***mkdir*** , ***rm***
  полное описание синтаксиса и семантики этих и любых других команд
  находятся в системе помощи, вызываемой с терминала в виде
  man <*команда****>*** или запускайте *Firefox* и используйте всю мощь
  Интернета для поиска информации.
  Анализируйте выполнение этих команд, а наиболее значимые
  скриншоты (снимается нажатием Alt+PrntScrn) помещайте в отчет.
  Отчет создавайте в приложении *LibreOffice Writer* и сохраняйте в своем
  Documents каталоге.

- Создайте дерево каталогов глубиной вложения до трех уровней.
  Создайте в каталогах текстовые файлы. Способ создания нового файла с
  терминала выберите самостоятельно (см. *Google*).

- Запустите с терминала *MidnightCommander* вводом команды *mc*
  и ознакомьтесь с его основными возможностями.
  Наполните созданные на предыдущем шаге текстовые файлы
  каким-либо содержанием, используя какой-либо редактор
  (например, вызов редактора из *мс* происходит по нажатию *F4)*.
  Просмотр содержимого файлов из *mc* — по *F3*, убрать панели *mc*
  с терминала — *Ctrl+o* . Подробно описывать работу с *mc* не надо.

- Выполните на терминале команды *shell* , которые упоминались в
  лекциях *№1* и *№2*:
  cat , ***cp*** , ***find***, ***link***, ***chmod***
  Для манипуляций с помощью этих команд используйте текстовые файлы,
  созданные и наполненные на предыдущем шаге.

- Попытайтесь создать на своем дереве какой-нибудь каталог с правами
  доступа, как это было в примере с каталогом *darkroom* в лекции *№2*.

Занятие второе

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для второго занятия (файлы на разделяемом ресурсе).

- Компилируйте файлы примеров с помощью строчного компилятора ***g++*** . В простейшем случае одномодульного проекта применяется команда ***g++ <****имя .cpp файла****>*** , исполняемый файл генерируется только в случае отсутствия ошибок компиляции и, по умолчанию, именуется ***a.out*** Запускается исполняемый файл с терминала с указанием полного пути (например, ***./a.out*** , если он в текущем каталоге) или из М*C.* Сведения о полной функциональности ***g++*** можете почерпнуть в справке.

- Скомпилируйте и выполните примеры программ *forkdemo.cpp* , *tinymenu.cpp* , *tinyexit.cpp* , *procgroup.cpp , wait_parent.cpp* . Процесс *wait_parent* при исполнении запускает процесс *wait_child* . Программа *wait_child.cpp* компилируется с опцией : *g++ wait_child.cpp -o wait_child*

  Пояснения к примерам этих программ можно найти в тексте лекции *№4.* Результаты выполнения программ анализируйте и помещайте вместе с соответствующими скриншотами и с пояснениями в отчет *№2.*
  При необходимости конвертации текстовых файлов из формата DOS в Unix и, наоборот, используйте команды *dos2unix* и *unix2dos* .

- Модифицируйте программу *forkdemo.cpp* (или создайте собственную) так, чтобы ввод/вывод на терминал отсутствовал, а при проходе по циклу была временная задержка, например, *sleep(7)* . Запустите эту программу в фоновом режиме (*background*), введя при запуске символ **&** после пробела и зафиксировав значение *PID*, назначенное системой фоновому процессу при запуске.
  Выполните на терминале команды ***ps*** , ***top*** , ***uptime*** , ***pstree*** .
  Снимите свой фоновый процесс командой ***kill*** с соответствующими параметрами. Скриншоты вместе с пояснениями к выполнению процессов и команд (а также исходные тексты программ, составленных самостоятельно) приведите в отчете.

- Что произойдет, если процесс потомок сменит текущий каталог – будет ли изменен текущий каталог для родителя?
  Создайте программу, подтверждающую ответ и приведите в отчете.

- Проиллюстрируйте как процесс-родитель и процесс-потомок разделяют один и тот же дескриптор и смещение текстового файла. Для этого составьте программу, в которой процесс-родитель должен открывать текстовый файл и запускать потомка. Потомок должен читать порцию данных из открытого файла и выводить на консоль. По завершению потомка родитель должен читать из того же файла и выводить результат на консоль. Можете использовать вызов *sleep()* для синхронизации доступа родителя и потомка к файлу.

Занятие третье

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для третьего занятия (файлы на разделяемом ресурсе).

- Скомпилируйте и выполните программу *whosortpipe.cpp .* Сопоставьте результат выполнения программы с выполнением тех же команд из *shell* в конвейерном режиме (|).
  Анализ результатов работы этой и всех последующих программ с соответствующими скриншотами (а также самостоятельно составленные исходные тексты программ) приводите в отчете №3.

- Программу *cmdpipe.cpp* запускайте после компиляции, задавая ей при стартах в качестве параметров пары команд *shell* для конвейеризации (*who* и *sort* ; *last* и *sort* ; *last* и *more* ; *pstree* и *more*). Сопоставьте результаты запусков программы с выполнением тех же пар команд из *shell* в конвейерном режиме.
  Можно ли с помощью вызова *popen()* создать программу, организующую конвейер из трех команд *shell*, передаваемых ей в качестве параметров командной строки при запуске?
  Если да, то создайте такую программу, если нет, дайте обоснованный ответ, почему нельзя.

- Напишите программу (например, на основе вызовов *pipe()*), воспринимающую варьируемое количество команд, передаваемых ей при запуске в качетве параметров. Каждая последующая команда должна быть соединена с предыдущей с помощью конвейера. Так, при запуске программы в виде :
  $ *./a.out last sort more*
  должны выполняться действия эквивалентные запуску команд из shell :
  $ *last* | *sort* | *more*

- Разберите и выполните пример клиент-серверного взаимодействия, организованного на конвейерах различного типа.
  Исходный текст примера содержится в файлах *pipe_server.cpp* , *pipe_client.cpp* и *pipe_local.h .* Север запускается в фоновом режиме*.* Проанализируйте результаты функционирования данной системы и ее недостатки. Программа сервер этого примера исполняет каждый командный запрос поочередно. Если какой-либо запрос потребует много времени, все остальные клиентские процессы будут ожидать обслуживания.
  Желательно модифицировать программу *pipe_server.cpp* так, чтобы при получении нового сообщения от очередного клиента сервер порождал очередной дочерний процесс для выполнения задачи обслуживания данного запроса (выполнения переданной от клиента команды и переправки клиенту результата).

Занятие четвертое

- Создайте несколько символьных переменных среды (переменных окружения). Составьте командный файл, выводящий на консоль значения этих переменных. Выполните операцию конкатенации (склеивания) значений переменных и выведите полученный результат на консоль. Выделите из конкатенированной переменной среды подстроку и выведите ее на консоль. Замените выделенную подстроку на какое-либо другое значение и выведите измененное значение переменной среды на консоль.
  Создайте несколько других переменных среды в интерпретации, как числовые переменные. В другом командном файле выполните с этими числовыми переменными все допустимые арифметические операции, выводя на консоль результаты операций и соответствующие комментарии.

- Создайте командный файл (основной), выдающий при старте сообщение и затем вызывающий другой командный файл (имя которого задается при старте основного файла в качестве параметра командной строки), который выдает свое сообщение и приостанавливается до нажатия любой клавиши.
  При возврате управления в вызывающий (основной) файл из него должно выдаваться еще одно сообщение, подтверждающее возврат.

- Составьте командный файл, выводящий на экран различия содержимого двух каталогов, имена которых передаются в качестве параметров. Отличия искать в именах файлов, их размерах и атрибутах.

- Разработайте командный файл сценария для поиска текстовых файлов, содержащих заданную последовательность символов (эта последовательность передается при запуске, в качестве первого параметра командной строки). В качестве второго параметра передается имя файла результатов, который должен быть создан в сценарии для записи в него имен найденных текстовых файлов и номеров их строк, в которых
  содержится заданная последовательность символов.

- Создайте командный файл, который синхронизирует содержимое заданного каталога с эталонным. После запуска и отработки командного файла в заданном каталоге должен оказаться тот же набор файлов, что и в эталонном (если файла нет – он копируется из эталонного каталога, если найдется файл, которого нет в эталонном - удаляется). Если файл с некоторым именем есть и в заданном и в эталонном каталогах, то он перезаписывается только в случае, если в эталонном более новая версия файла.
  Имена каталогов должны передаваться командному файлу при запуске в качестве параметров командной строки.

Занятие пятое

- Создайте учетные записи для нескольких пользователей (не задавая им прав администратора) и объедините их в две группы. Заходя в систему под разными аккаунтами, создайте в соответствующих домашних каталогах файлы, варьируя при этом права доступа для пользователя, для группы, для всех. Убедитесь, что права доступа разделяются в соответствии с тем, как это задано.
  Проведите операцию слияния файлов с различными правами доступа и проверьте какие при этом получаются права у результирующего файла.

- Запустите в фоновом (background) режиме командный файл (процесс), выдающий в цикле с некоторой задержкой сообщение на консоль. Запустите другой командный файл (процесс), требующий диалога, в обычном режиме (foreground). Убедитесь в том, что вывод этих двух процессов на консоль перемежается.
  Остановите фоновый процесс сигналом *kill*. Запустите его снова, организовав предварительно перенаправление его вывода в файл. Убедитесь, что теперь вывод двух процессов разделен.

- Доработайте предыдущее задание так, чтобы показать возможность перевода фонового процесса в диалоговый режим (foreground) для выполнения операции ввода с клавиатуры и, затем, возврата его обратно в фоновый (background) режим (команды *fg, bg, jobs*).
  Продемонстрируйте возможность оставления фонового процесса на исполнение после завершения пользовательского сеанса работы в ОС.

- Разработайте командный файл для выполнения архивации каталога через определенные интервалы времени. Имя архивируемого каталога, местоположение архива и время архивации передаются при запуске командного файла в виде параметров командной строки.

Занятие шестое

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для четвертого занятия (файлы на разделяемом ресурсе).

- Программа *sigint.cpp* осуществляет ввод символов со стандартного ввода. Скомпилируйте и запустите программу и отправьте ей сигналы *SIGINT* (нажатием *Ctrl-C*) и *SIGQUIT* (нажатием *Ctrl*-\). Проанализируйте результаты.

- Запустите программу *signal_catch.cpp* , выполняющую вывод на консоль. Отправьте процессу сигналы *SIGINT* и *SIGQUIT* , а также *SIGSTOP* (нажатием *Ctrl-Z*) и *SIGCONT* (нажатием *Ctrl-Q*) .
  Проанализируйте поведение процесса и вывод на консоль, а также сравните с программой из предыдущего пункта. Приведите результаты анализа и скриншоты в отчете №4 .

- Скомпилируйте и запустите программу *sigusr.cpp .* Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой *kill* различные сигналы, в том числе и *SIGUSR1* , проанализируйте реакцию на них.

- Составьте программу, запускающую процесс-потомок. Процесс-родитель и процесс-потомок должны генерировать (можно случайным образом) и отправлять друг другу сигналы (например *SIGUSR1 SIGUSR2*).
  Каждый из процессов должен выводить на консоль информацию об отправленном и о полученном сигналах.
  Для организации обработчиков сигналов предпочтительно использовать системный вызов *sigaction()* и сответствующую структуру данных.
  Обеспечьте корректное завершение процессов.

- Модифицируйте программу *занятия №3* (файлы *pipe_server.cpp* , *pipe_client.cpp* и *pipe_local.h*) сделав ее более стабильной в работе.
  В числе недостатков, которые желательно устранить, можно указать следующие:

  - если клиент завершается по получению сигнала *SIGINT* (*Ctrl+C),* то *private FIFO* не удаляется из системы (исправляется посредством организации перехвата сигнала с выполнением необходимых действий);
  - клиентский процесс при его инициализации может обрушиться, если сервер окажется недоступен (исправляется путем попытки запуска сервера из клиента, если сервер не активен).

Занятие седьмое

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для шестого занятия (файлы на разделяемом ресурсе).

- Скомпилируйте и выполните программу *gener_sem.cpp*, иллюстрирующую создание наборов с семафорами или получение доступа к ним.
  Запустите программу несколько раз и после каждого ее завершения выполните команду *ipcs -s .* Поясните зависимость процедуры создания семафоров от используемых в вызове *semget()* флагов.

- Удалите созданные на предыдущем шаге семафоры с помощью команды *ipcrm* с соответствующей опцией и значением *id* семафора или ключа .

- Скомпилируйте s*emdemo.cpp*, демонстрирующую организацию разделения доступа к общему ресурсу между несколькими процессами с помощью технологии семафоров.
  Запустите сразу несколько процессов на разных терминалах и проанализируйте их взаимодействие и соблюдение очередности в попытках получения общего ресурса.

- Скомпилируйте программу *semrm.cpp* и произведите с ее помощью удаление созданного на предыдущем шаге семафора.
  Поясните почему данная программа удаляет только те семафоры, которые были созданы при выполнении программы *semdemo.cpp* .

- Попробуйте удалить семафор с помощью запуска *semrm.cpp* во время исполнения s*emdemo.cpp* и проанализируйте ситуацию.

- Попытайтесь улучшить программу s*emdemo.cpp* , например, предоставив процессу возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), организовав при этом завершение процесса по вводу какого-либо символа.

- Составьте программу, позволяющую мониторить количество процессов, (типа s*emdemo*) находящихся в состоянии ожидания освобождения ресурса («*Trying to lock...*») в каждый момент времени. Программа строится на основе вызова *semctl()* с соответствующими параметрами и запускается на отдельном терминале.

Занятие восьмое

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для пятого занятия (файлы на разделяемом ресурсе).

- Скомпилируйте и выполните программу *gener_mq.cpp* , создающую несколько очередей сообщений. После завершения программы выполните команду *ipcs* и поясните отличие результата от того, что был при вызове подобной команды из программы.

- Скомпилируйте программы *sender.cpp* и *receiver.cpp* , задав соответствующим исполняемым файлам разные имена (*g++ <имя .cpp файла> -o <имя .out файла>*).
  Запустите процессы на разных тоерминалах и передайте текстовые сообщения от процесса *sender* процессу *receiver.*
  Проанализируйте, что происходит с ресурсом *Message Queue* после завершения каждого из процессов (командой *ipcs*). При этом выполните различные виды завершения отправкой сигналов *SIGQUIT* (нажатием *Ctrl*-\) и *SIGINT* (нажатием *Ctrl-C*) .

- Что происходит, если процесс *receiver* запускается уже после того, как процесс *sender* отправил в очередь одно или множество сообщений ?

- Запустите несколько процессов *receiver* на различных терминалах и, отправляя сообщения процессом *sender ,* проанализируйте ситуацию.

- Модифицируйте программы *sender.cpp* и *receiver.cpp* так, чтобы организовать отправку сообщений двух типов через одну и ту же очередь для двух различных процессов получателей.
  Для этого необходимо управлять
  параметром в поле *mtype* структуры *my_msgbuf* на передающей стороне и параметром *msgtyp* в системном вызове *msgrcv()* на приемной стороне.

Занятие девятое

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для седьмого занятия (файлы на разделяемом ресурсе).

- Скомпилируйте и выполните программу *gener_shm.cpp*, демонстрирующую создание сегментов разделяемой памяти.
  Запустите программу несколько раз и после каждого ее завершения выполните команду *ipcs -m .* Поясните зависимость процедуры создания сегментов разделяемой памяти от используемых в вызове *shmget()* флагов.

- Удалите созданные на предыдущем шаге сегменты разделяемой памяти с помощью команды *ipcrm* с соответствующей опцией и значением *id* сегмента или ключа .

- Скомпилируйте s*hmdemo.cpp*, осуществляющую операции записи в разделяемую память без разделения доступа к этому общему ресурсу. Символы записываемые в общую память передаются в качестве параметра командной строки при запуске процесса s*hmdemo*. Запуск этого процесса без параметров приводит к выводу на консоль текущего содержимого сегмента общей памяти.

- Запустите несколько раз процессы типа s*hmdemo* с различными значениями параметров и проиллюстрируйте возможности чтения и записи в сегмент общей памяти независимо исполняемыми процессами. Затем удалите сегмент памяти командой *ipcrm .*

- Скомпилируйте и выполните программу *attach_shm.cpp*, иллюстрирующую передачу символьной информации между двумя процессами (родственными) через сегмент общей памяти с модификацией этой информации.
  Проанализируйте значения выводимой информации о границах сегментов в системной памяти.
  За счет чего по завершении данной программы сегмент общей памяти не присутствует в системе?

- Составьте программу, создающую три разделяемых сегмента памяти размером 1023 байта каждый. Укажите в вызове *shmat()* параметр *shmaddr = 0* при привязке сегментов.
  Разместит ли система сегменты в последовательных участках?
  Позволит ли система ссылку или изменение 1024-го байта любого из этих участков?

Занятие десятое

- Войдите в систему и скопируйте в свой *HOME* каталог набор исходных файлов для занятия (файлы на разделяемом ресурсе).

- Скомпилируйте и выполните программу *socketpair.cpp* иллюстрирующую создание простейшего вида сокета и обмен данными двух родственных процессов. Проанализируйте вывод на консоль. Существует ли зависимость обмена от различных соотношений величин временных задержек (в вызовах *sleep()*) в процессе родителе и в процессе потомке ?

- Скомпилируйте программы *echo_server.cpp* и *echo_client.cpp* , задавая им при компиляции разные имена (размещаем файлы в одном каталоге). Запустите программы сервера и клиента на разных терминалах. Вводите символьную информацию в окне клиента и проанализируйте вывод.
  Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия ?
  С чем связано создание специального файла в текущем каталоге во время исполнения программ ?

- Скомпилируйте с разными именами программы *sock_c_i_srv.cpp* и *sock_c_i_clt.cpp* (в них используется общий *include* файл *local_c_i.h*). Запустите программы сервера и клиента на разных терминалах. При запуске клиента указывайте в качестве параметра командной строки имя хоста *localhost* . Вводите символьную информацию в окне клиента и поясните вывод.
  Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия ?

- Модифицируйте программу *echo_server.cpp* так, чтобы при ответе на запросы клиента, что-либо выводилось в окне сервера.
  Испытайте работу эхо сервера при одновременной работе с несколькими клиентами.

Занятие одинадцатое

- Войдите в систему и скопируйте в свой *HOME* каталог исходный файл для этого занятия (файл на разделяемом ресурсе).

- Скомпилируйте и запустите программу *server_game.cpp ,* иллюстрирующую обмен данными с клиентскими приложениями по итеративной схеме.

- Запустите другой терминал и проверьте с него наличие в системе созданного сервером сокета и то, что он находится в состоянии *LISTEN*. Для этого выполните команду *netstat -a | grep 1066* Проанализируйте вывод данной команды и объясните ее смысл.

- Запустите в качестве клиентского процесса утилиту *telnet* со следующими параметрами

telnet localhost 1066

*(*При организации коммуникации по сети, на разных компьютерах, вместо *localhost* при запуске клтента указывается *IP-адрес* компьютера, на котором был запущен сервер. *)*

- Диалог с сервером заключается в угадывании слова. Оно вводится по буквам с клиентского терминала. При этом сервер вместо неугаданных букв выдает символы ”-” , а также считает число оставшихся неудачных попыток (всего их предусмотрено 12).

- Завершите серверное приложение с помощью сигнала *kill* , и затем определите командой *netstat -a | grep 1066* когда исчезает из системы соединение на сокетах. Во время сеанса обмена также примените команду *netstat -a | grep 1066* чтобы исследовать состояние сединения.

- Проделайте все заново, но запускайте не одно клиентское приложение (в виде *telnet*) , а несколько экземпляров с разных терминалов, и попытайтесь работать с них одновременно.
Проанализируйте, как будет обслуживать запросы в этом случае сервер.

- Модифицируйте программу *server_game.cpp* так, чтобы запросы от каждого из клиентов могли обслуживаться конкурентно (путем запуска для каждого нового соединения собственного нового процесса на сервере. Возможно также улучшить качество самой игровой функции *guess_word()* сервера.
Проанализируйте, как обслуживаются запросы в случае конкурентной схемы работы сервера.

Занятие двенадцатое

- Войдите в систему и определите IP адрес вашего компьютера.

  В дальнейшем также понадобится пароль администратора для вашего компьютера, можете использовать pw: *2015nf,kbws2015*

- Просмотрите текущие правила, установленные в iptables. Результаты выполнения ваших команд, а также последствия применения вводимых вами правил протоколируйте в отчете о лабораторной работе.
  Никогда не сохраняйте вводимые правила командами типа *save* .

- Введите правило, блокирующее весь входящий трафик на ваш компьютер. Поверьте наличие введенного правила в системе.

- Добавьте правила для фильтрации входящего трафика (с использованием целей DROP и ACCEPT) так, чтобы веб-трафик проходил, а остальной был блокирован. Просмотрите детализированный список правил и проверьте их действие.

- Закройте вход ICMP пакетам, отправляемым с какого-либо другого компьютера лаборатории. Создав соответствующее правило (испытайте цели DROP и REJECT), проверьте его действие утилитой *ping* .

- Составьте правило для журналирования тех входящих пакетов, которые отбрасываются вашим фильтром.

- Организуйте на вашем компьютере форвардинг с какого-либо порта внешнего интерфейса на определенный порт другой машины лаборатории.

ТРЕБОВАНИЯ К ОТЧЕТАМ

По результатам выполнения каждой из лабораторной работ необходимо составлять отчет. Отчет должен быть оформлен с титульным листом с названием университета, института и кафедры. На титульном листе указывается номер и название лабораторной работы в соответствии с оглавлением, а также номер группы и ф.и.о. студента исполнителя работы.

В отчете приводится информация, удостоверяющая выполнение заданий лабораторной работы: скриншоты с результатами исполнения команд и программ, исходные тексты тех С-программ или фрагментов, которые были составлены самим студентом, скрипты (BASH и др.) командных файлов, ответы на поставленные в заданиях вопросы и т.д. Результаты выполнения заданий должны предваряться в отчете текстом, формулирующим само задание.

Защита лабораторной работы происходит по предъявлению оформленного отчета и сопровождается демонстрацией исполнения программ и командных файлов.

ЛИТЕРАТУРА

1. Дэвид Тейнсли. Linux и UNIX : программирование в shell. Руководство разработчика. - К: Издательская группа BHV, 2001.

2. Chris Brown. UNIX Distributed Programming. Prentice Hall International (UK) Limited, 1994.

3. Андрей Робачевский. Операционная система UNIX. - СПб.: БХВ — Санкт-Петербург, 1999.

4. Теренс Чан. Системное программирование на C++ для UNIX. - К.: Издательская группа
   BHV, 1999.

5. Уильям Стивенс. UNIX : взаимодействие процессов. – СПб.: Питер, 2002.