ОС UNIX

Домашнее задание 5 – 7

Задача №5

- Написать программу, которая запускает другие программы и следит за их состоянием.
- У программы есть конфиг. файл, в котором прописано:
  - имя исполняемого файла;
  - параметры для передачи исполняемому файлу,
  - способ наблюдения за ним: дождаться завершения, либо дождаться завершения и запустить заново еще одну копию.
- Например:

  /bin/ls -l wait

  /bin/sleep 15 respawn

Задача №5

- Программа должна прочитать свой конфиг, с помощью вызова fork() создать нужное количество своих копий и запомнить идентификаторы процессов, которые стартовали.
- В каждой из этих копий стартовать указанную программу с параметрами.
- Если fork прошел успешно, то в спец. файл (/var/run/имяфайла.pid, или в /tmp) записать идентификатор запущенного процесса.

Задача №5

- Если процесс завершился, то:
  - если необходимо просто ожидать завершения, надо уничтожить соотв. ему файлы и больше не стартовать;
  - если программу необходимо перезапустить, то запустить новый процесс с тем же исполняемым файлом и обновить соответствующий pid-файл.

Задача №5

- Дополнительное требование 1.
  - Возможно, что у какого-то из процессов испорчен/не существует исполняемый файл, либо заданы неверные параметры запуска. В этом случае если в течении определенного времени (например, 5 сек) будет больше 50 попыток неудачного запуска программы, то необходимо выдать сообщение (в лог) о том, что этот файл испорчен и в течении часа не повторять попыток его запуска.

Задача №5

- Дополнительное требование 2.
  - После своего запуска наша программа должна закрыть терминал, установить себе рабочий каталог в корень и в дальнейшем все свои сообщения выводить через систему логирования вызовом syslog (т.е. запускаться сразу как демон).

Задача №5

- Дополнительное требование 3.
  - Получив сигнал -HUP, программа должна перечитать конфиг. файл, завершить все дочерние процессы и запустить новые в соответствии с новым конфиг. файлом.

Задача №5

- Ключевые слова:
  - init: wait, respawn или xinetd
  - fork(), exec(), waitpid()
  - signal(), kill –hup
- Шаблон запуска потомков прикреплен к посту в вк.
- Пример шаблона для создания демона есть в книге Робачевского (2е издание стр. 211).

Задача №6

- Создать собственную модель файловых блокировок на основе файлов myfile.lck.
- myfile.lck хранит:
  - идентификатор процесса, который этот файл создал;
  - тип операции блокировки (read/write).

Задача №6

- Написать программу, которая при передаче ей имени файла myfile:
  - проверяла бы существование файла myfile.lck и давала разрешение на запись.
  - при разрешении на запись создавала бы файл myfile.lck с перечисленными свойствами;
  - при запрете записи дожидалась пока не освободится файл, после чего сразу редактировала его.

Задача №4

- Пример. Имеем файл с паролями вида:

  name password

  имя 1231

  root пароль

  noname6 psswd6

  фантазии нехватка

  maincpp hisPassword

  cat catPassword

  user userPassword

  ночь спатьхочуя

  надо дописатькод

  newName17 newPassword17

  надо придумать

  овсе ок

  добрых снов

Задача №4

- Пример:
  - подаём программе на вход, например, пользователя, изменённый пароль и редактируемый файл;
  - программа открывает файл, создает файл блокировки, редактирует исходный файл и удаляет блокировку;
  - параллельно вторым экземпляром программы проверяем работоспособность блокировки, вставляя для наглядности sleep() во время записи.

Задача №6

- Дополнительное требование:
  - реализовать блокировку файлов кусками;
  - в файле .lck будет записываться информация о заблокированном куске;
  - использовать файл .lck.lck для блокировки файла первичного уровня на время записи информации о заблокированном куске.

Задача №7

- Написать параллельный обработчик данных ☺
- Он состоит из 3 частей:
  - **Программа src.** Читает входной файл, извлекает из него куски, и разбрасывает в несколько файлов вывода (подготавливает данные для обработчика).
  - **Обработчики hdl.** Однотипные, каждый из них обрабатывает свой файл и результат работы сохраняет во второй промежуточный файл.
  - **Программа dst**. Собирает результат.

Задача №7

- Предлагается сэкономить на промежуточных файлах, сделав лаунчер, который делает из себя несколько форков, соединяя компоненты через неименованные каналы.
- **В результате программы считают, что они читают и пишут в файл, но фактически они работают с каналами**.

Задача №7

- Проблема: разные части данных могут обрабатываться с разной скоростью – асинхронный ввод-вывод.
- Программы src и dst должны воспользоваться вызовом select() – смотреть, какой из клиентов готов к приему данных и писать в соответствующий pipe.

Задача №7

- Способы передачи количества входных файлов:
  - количество строго фиксировано; ☹
  - через аргумент командной строки – причем желательно начать с 4го файла, оставив стандартные stdin, stdout, stderr;
  - через вызов записи нуля байт - до тех пор, пока не вернется ошибка.
- Возможные обработчики:
  - поблочное перемножение матриц;
  - сжатие изображений;
  - обработка текста;
  - whatever you want.

Задача №7

- Схема.