Сигналы.

(5 программа из 2 л/р)

В UNIX сигналы – базовое средство информирования процессов о событиях в системе;

В Windows нет сигналов, но есть event; (события)

Важнейшее событие ОС – завершение процесса

(нормальное, Kill, Ctrl+C, Ctrl+Z, Sigterm)

2 коммерчески значимые линии UNIX:

1) System V (11, 5m);
2) UNIX BSD;

Они различаются по API;

Процесс может определить собств. реакцию на полученный сигнал, может игнорировать его, может реагировать по умолчанию (как определено в системе) или определить свою реакцию на полученный сигнал;

handler - классическое название обработчика сигнала

Сист. вызов signal() выполняет регистрацию нашего обработчика сигнала на сигнал;

Не входит в POSIX ⇒ не стоит использовать в переносимом ПО;

А sigaction() входит в POSIX;

Зачем процессу определять свои обработчики сигналов?
Потому что с помощью техники сигналов можно существенно изменять ход выполнения программы;

signal и sigaction есть в ANSI C
⇒ есть всегда

Ещё:
В POSIX есть sigjump();

Процессы-демоны.

Книга (13 глава) „UNIX" Стивена Рago

#ps -ajx - увидеть все процессы, в т.ч. сист. демонов;

0 - init; 1 - процесс, открывший терминал,
2 - kthreadd daemon (занимается диспетчеризацией)

Поток - единица диспетчеризации ⇒ kthreadd daemon

ОС поддерживает потоки уровня ядра
⇒ kthread daemon;

Потоки очень дорогие;

Многопоточность приложений - большая проблема

При запуске потоков резко увеличивается размер кучи;
Память - ценнейший ресурс системы;

Оптимизация больше вакса для серверов, чем для ПК;

13.3 Правила программирования демонов.

1) Сброс маски режима создания файлов: ф-ция umask (0);
расш. umask

все правила реализованы в ф-ции daemonize (вызывается из main, т.е. из предка)

↳ В её начале вызывается umask
затем fork и предок завершается; ⟹

Предок, вызвавший fork, ② создаёт группу процессов и становится её лидером

⇒ Потомок теряет группу, становится сиротой и его усыновляет терминальный процесс;
Это делается для того, чтобы процесс потомок гарантированно не был лидером группы;
Это явл. условием вызова ф-ции setsid();

3) setsid();
set session id

В UNIX до сих пор есть часть кода, относящаяся к телетайпу

Иерархическая машина UNIX

В отличие от System V, в Linux fork надо вызывать 1 раз, а не 2 (чтобы потомок не стал лидером группы)

После setsid() осиротевший потомок станет демоном, т.е. он станет лидером группы, лидером сессии, но в группе и сессии он будет единственным и утратит управляющий терминал;

У сист. демонов в TTY стоит ?, а

в state – S (interruptable sleep), s (session leader), иногда l (много-потоковый)

Из-за того, что демон становится лидером группы и лидером сессии у него будет 3 одинаковых идентификатора: собственный, группы и сессии;

↳ Особенность демонов

4) chdir("/"); //переход в корневую директорию;

Файловая система - ключевая подсистема ОС;

В UNIX (Linux) поддерживается большое кол-во файловых систем;
ext 2 - классическая;
ext 4;
nfs;
fdt32;
ntfs - Windows;

В основе лежит монтирование файловых систем (базовое действие UNIX);

PDP 11 ⟹ СМ 11
(сшф калька СССР)

Windows упрощает работу неподготовленных пользователей ⟹ коммерческий успех

Файлы при подключении (её собств. ф. с.) монтируется к дереву каталогов UNIX;

Работать можно только со смонтированной ф. с.;

Сист. демоны предназначены для выполнения определённых действий в системе;

Пробуждаются от interruptable sleep, чтобы выполнить важную системную задачу;

Пока демон работает, отмонтировать ф. с. нельзя;

5) Потомок наследует от предка дескрипторы открытых файлов, маску создания файлов, сигнальную маску;

Umask — предок сбрасывает файловую маску, и потомок наслед. её, чтобы демон мог создавать файлы с любыми правами доступа;

Можно было и после fork маску сбросить;

В любой ОС есть (простые) файлы (информация о нём м.б. прочитана, пока он хранится на диске), а есть открытые файлы (когда процесс открывает файлы, то для него создаются специальные системные таблицы, т.к. они требуются для специальных действий с ними);

Предок мог открыть любое кол-во файлов и установить для них опр. права доступа, но потомку, который стал демоном, это не нужно;

Поэтому определяется макс. номер дескриптора открытого файла: вызывается функция getrlimit(RLIMIT_NOFILE, &r1)

struct rlimit

→ в ней есть поле r1.rlimit_max

Это соответствует с RLIM_INFINITY

Уст-ся значение и в цикле от 0 до 1024 все дескрипторы открытых файлов закрываются (в т.ч. и 0, 1, 2 – stdin, stdout и stderr)

При создании любого процесса для него автоматически открываются файлы с дескрипторами 0, 1, 2 (stdin, stout, stderr); и в цикле они закрываются...

б) Открыть дескрипторы 0, 1 и 2 и направить их в /dev/null, т.к. некоторые демоны открывают дескрипторы 0, 1 и 2. Если это произойдёт, ошибок быть не должно;

fd 0 = open("/dev/null", O_RDWR);

fd 1
fd 2 } (dup) — полезная функция

Сигналы сопровождают события в системе; когда вызывается setsid(), демон утрачивает управляющий терминал, а его потеря — событие в системе, которое сопровождается сигналом SIGHUP, который приводит к завершению процесса. Чтобы этого не произошло, в демоне этот сигнал игнорируется, т.е. в sigaction() передаётся структура sigaction, у которой в поле handler устанавливается SIGIGN;

В конце daemonize инициализируется файл журнала:

openlog(), syslog()

Проработать [§13.4] журналирование ошибок

При запуске лаб демонстрировать журнал ошибок;

Также из main вызывается already_running()

§13.5 Демоны в единственном экземпляре.

Почему у демона нет управляющего терминала?

Чтобы через терминал нельзя было влиять на его работу; (защита от дурака)

Наши демоны, в отличие от сист. демона, запускаются из командной строки;
(а системные создаются с помощью сценария инициализации)

Также его можно убить с помощью k.ill-9 (а системного нельзя)

Почему демон должен выполняться в единственном экземпляре?

Потому что нам не нужны n демонов, выполняющих одну и ту же задачу;

К тому же непонятно, как эту задачу распределять между n-ным кол-вом демонов;

В книге предлагается путей, обеспечивающий выполнение демона в единственном экземпляре:

Необходимо создать файл блокировки:

Open(/* имя */);

в директории /var/run/<имя файла>
↳ доступна только с правами суперпользователя

Это единственное место в коде демона, где требуются права суперпользователя;

Файл создаётся в пространстве ядра, чтобы его нельзя было легко удалить;

Файл блокировки - гарантия того, что демоны выполняются в единств. экземпляре;

Создание файла не явл. критически важным действием, он создаётся для надёжности; (можно и обычный файл создавать)

Итого:

Open(..., LOCK MODE);

lockfile(); // ф-ция Стивена

Есть ещё lockfd() (можно использовать её)

Что будет происходить:

Первый экземпляр создаст lockfile и заблокирует его для записи;

Следующие экземпляры попытаются писать в этот же файл, но не смогут и завершатся с ошибкой;

Таким образом гарантируется, что демон будет запущен в единств. экземпляре;

§13.6 Соглашения для демонов.

Листинг 13.3 – демон, перечитывающий конфигурационный файл;

В демоне (уже в main) создаётся доп. поток, т.е. вызывается pthread_create();

Чтобы можно было анализировать сигналы, необходимо восстановить реакцию процесса на сигнал по умолчанию, т.е. в поле handler загрузить SIG_DFL;

Листинг 13.4 – также созд. доп. поток, но мы будем его создавать;

Т.к. созд. доп. поток, то в программе всего будет 2 потока: главный и дополнительный
⟹ в state демона будет ещё буква l;

Лаба 4:
упр. 3.1 (3.2 уже сделали в л/р), 3.3