УО «ВГУ им. П.М. Машерова»
Кафедра прикладной математики и механики

В.В. Новый

Операционные системы. Процессы

Практикум по дисциплине «Операционные системы» для студентов 2 курса специальности Прикладная математика (1-31 03 03)

Репозиторий ВГУ

Витебск, 2010

Лабораторная работа №1. Процессы в Windows

Системные и пользовательские процессы. Системные вызвовы для создания и завершения процессов.

Задание 1. Разработайте приложение, демонстрирующее создание одного дочернего процесса и устанавливающее для него в качестве рабочего каталога каталог «C:\TEMP». В качестве запускаемых приложений использовать приложение из списка: MS Word, MS Excel, MS Access, MS Paint, WinRAR, «control inetcpl.cpl», «control firewall.cpl», «control appwiz.cpl», «control access.cpl», «control sysdm.cpl»

Задание 2. Дан некоторый файл с данными (документ MS Word, рабочая книга MS Excel, точечный рисунок, и т.д.). Разработайте приложение, демонстрирующее создание процесса на основе зарегистрированной в системе ассоциации с расширением заданного файла данных.

Задание 3. Разработка графической оболочки для программы UPX.
Разработайте графическое приложение – управляющую оболочку для программы UPX. Приложение должно позволять: выбрать сжимаемый файл, выбрать режим работы – компрессию или декомпрессию файла, выбрать степень сжатия файла, показать результат выполнения операции – успех (код возврата UPX = 0), неудача (код возврата UPX=1), предупреждение (код возврата UPX = 2).

Дополнительный материал

В ОС Windows существуют следующие системные вызовы для создания процессов:

- `CreateProcess`; (аналоги: `CreateProcessAsUser`, `CreateProcessWithLogonW`, `CreateProcessWithTokenW`);
- `ShellExecute`, `ShellExecuteEx`;
- `WinExec`.

Системный вызов `WinExec` в настоящее время объявлен Microsoft устаревшим и, согласно документации MSDN, должен использоваться только для совместимости с 16-битным кодом. Тем не менее, так как он в конечном итоге обращается к функции CreateProcess, в некоторых случаях допускается его использование:

```
UINT WinExec(
  LPCSTR lpCmdLine,
  UINT uCmdShow
);
```

Здесь,
`LPCSTR lpCmdLine` – имя программы (полный путь или короткое имя);
`UINT uCmdShow` – способ отображения окна, совпадающий с параметром функции `ShowWindow`, изученной Вами ранее.

Запуск приложения ассоциированного с расширением указанного файла в Windows:

```
HINSTANCE ShellExecute(
      HWND hwnd,
    LPCTSTR lpOperation,
    LPCTSTR lpFile,
    LPCTSTR lpParameters,
    LPCTSTR lpDirectory,
```

Репозиторий ВГУ

```
    INT nShowCmd
);
```
Здесь,
`HWND hwnd` – дескриптор окна родительского приложения;
`LPCTSTR lpOperation` – операционная строка, задающая действие выполняемое с файлом (каталогом): открытие в приложении, печать или открытие каталога (“open” = `NULL`, “print”, “explore”);
`LPCTSTR lpFile` – указатель на строку, содержащую имя файла, по ассоциации с которым будет запущено приложение;
`LPCTSTR lpParameters` – указатель на строку, содержащую параметры командной строки (обычно NULL);
`LPCTSTR lpDirectory` – указатель на строку, содержащую текущий каталог для запускаемого приложения;
`INT nShowCmd` – способ отображения окна, совпадающий с константой для `ShowWindow`.

Существует аналог данной функции: `ShellExecuteEx`, - отличающийся от `ShellExecute` способом задания параметров. Для данного системного вызова они задаются через структуру `SHELLEXECUTEINFO`.

После завершения работы, процесс (выполняющийся поток) может вызвать функцию ExitProcess указав в качестве параметра код завершения (exit code):
```
VOID ExitProcess(UINT uExitCode);
```
Равносильно выполнению в программе оператора `return` с кодом возврата.

Другой процесс может определить код возврата:
```
BOOL GetExitCodeProcess(
     HANDLE hProcess,
     LPDWORD lpExitCode
);
```
Дескриптор опрашиваемого процесса `hProcess` должен обладать правами доступа `PROCESS_QUERY_INFORMATION`.
`lpExitCode` указывает на переменную, которая принимает код завершения. (`STILL_ACTIVE` = процесс еще не завершился).

Репозиторий ВГУ

Лабораторная работа №2. Процессы и потоки в Linux

Реализация процессов в Linux. Создание и завершение процесса.

Задание 1. Разработайте приложение, запускающее новый процесс (например, Midnight Commander – mc).
Задание 2. Разработайте приложение, выводящее на экран свои PID и PPID.
Задание 3. Разработайте приложение завершающее указанный с помощью PID процесс.

Указания: при написании программ предусмотрите обработку возможных ошибок.

Дополнительный материал

Справочная информация к прочтению:

- firststeps_linux.chm
- Консольная команда man.

Для получения информации о системном вызове или команде используйте команду man с именем вызова или команды в качестве параметра. Например:

```
man exec
```

В случае, если существует несколько одноименных объектов команде man можно указать требуемый раздел руководства использовав следующий формат:

```
man n <вызов>
```

,где n – задает секцию справочного руководства и может принимать следующие значения:

1 – исполнимые программы или команды оболочки (shell);
2 – системные вызовы;
3 – библиотечные вызовы (например, printf);
4 – специальные файлы (например, файлы устройств);
5 – форматы файлов и соглашения;
и т.д.

Например, для вывода справки по системному вызову open:

```
man 2 open
```

Для получения списка процессов и другой информации о них могут быть использованы команды ps (консольная команда), top (или ее графическая версия gtop).

При работе с системными вызовами может понадобится подключить дополнительные заголовочные файлы. Их имя можно узнать во встроенной справочной системе.

Для работы с процессами могут быть использованы следующие функции:

Репозиторий ВГУ

```
pid_t fork(void);
void _exit(int exit_code);
int execl(const char* path, const char* arg, …);
```

Здесь pid_t – тип, задающий ProcessID запускаемого процесса (определен в <sys/types.h>).

Обратите внимание, что функция fork() уникальна – она возвращает управление дважды: в родительский процесс и в дочерний процесс. В родительский процесс возвращается pid дочернего процесса, а в дочерний – 0. В редком случае ошибки – отрицательное значение.

```
pid_t wait(int *status);
```

Системный вызов wait позволяет родительскому процессу дождаться завершения дочернего процесса и помещает код возврата в переменную status.

Сигналы – это простейшая форма межпроцессного взаимодействия в стандарте POSIX. Они позволяют одному процессу быть прерванным асинхронным образом по инициативе другого процесса или ядра для того, чтобы обработать какое-то событие. Сигналы используются для решения таких задач, как завершение процессов и сообщения демонам о необходимости перечитать конфигурационный файл. При получении сигнала процесс может сделать одно из 3:

- проигнорировать сигнал;
- перехватить сигнал;
- позволить ядру выполнить действие по умолчанию, которое зависит от конкретного полученного сигнала.

Для отправки сигналов используется системный вызов kill().

Номера сигналов и их имена описаны в /usr/include/bits/signum.h

Для использования системного вызова kill() необходимо подключить заголовочные файлы sys/types.h и signal.h.

Формат:

```
int kill(pid_t pid, int sig);
```

,где pid – идентификатор целевого процесса, sig – сигнал, который передается. Возвращаемое занчение: -1 при ошибке, 0 при успехе. Процесс может послать сигнал сам себе.

Приложение. Некоторые сигналы.

SIGABRT Функция abort () посылает сигнал процессу, который ее вызвал, прерывая процесс со сбросом файла дампа ядра. Под Linux библиотека C вызывает abort (), когда происходит сбой утверждения (assertion).

SIGALRM Вызывается, когда предупреждение, установленное alarm (), устаревает.

SIGBUS Когда процесс нарушает ограничения, накладываемые оборудованием, но не связанные с защитой памяти, посылается этот сигнал. Обычно это случается на традиционных платформах Unix, когда выполняется попытка "невыровненного" доступа, но ядро Linux исправляет такие попытки и продолжает выполнять процесс.

SIGCHLD Этот сигнал посылается процессу, когда один из его дочерних процессов устаревает или остановлен. Это позволяет процессу избежать появления "зомби" за счет вызова одной из функций wait() из обработчика

Репозиторий ВГУ

сигнала. Если родитель всегда ожидает завершения дочерних процессов, прежде чем продолжить работу, этот сигнал может быть проигнорирован. Это отличается от сигнала SIGCLD, представленного в ранних версиях System V.

SIGCLD устарел и более не должен применяться.

SIGCONT Этот сигнал перезапускает приостановленный процесс. Также он может быть вызван процессом, позволяющим выполнить действие после перезапуска. Большинство редакторов перехватывают этот сигнал и обновляют терминал после перезапуска.

SIGFPE Этот сигнал посылается, когда процесс вызывает арифметическое исключение. Все исключения плавающей точки, такие как переполнение и потеря значимости, вызывают этот сигнал, как это происходит при делении на 0.

SIGHUP Когда терминал отсоединяется, лидер сеанса, ассоциированного с терминалом, получает этот сигнал, если только на терминале не выставлен флаг CLOCAL. Если лидер сеанса завершается, SIGHUP отправляется лидеру каждой группы процессов в данном сеансе. Большинство процессов прерываются при получении SIGHUP, поскольку это значит, что пользователя уже нет в системе. Многие процессы-демоны интерпретируют SIGHUP как запрос на закрытие и повторное открытие журнальных файлов, а также на перечитывание конфигурационных файлов.

SIGILL Процесс пытается запустить некорректную аппаратную команду.

SIGINT Этот сигнал посылается всем процессам в группе процессов переднего плана, когда пользователь нажимает клавиатурную комбинацию прерывания (обычно ^C).

SIGIO Произошло асинхронное событие ввода-вывода. По вопросам асинхронного ввода-вывода обращайтесь к соответствующим источникам. **SIGKILL** Этот сигнал генерируется только вызовом kill () и разрешает пользователю безусловно прервать процесс.

SIGPIPE Процесс выполнил запись в канал, который не имеет читателя.

SIGPROF Завершилось действие таймера профилирования. Это сигнал обычно используется профилировщиками, которые проверяют другие характеристики процесса времени выполнения. Профилировщики обычно используются для оптимизации времени выполнения программ, помогая программистам находить узкие места. Простейшим профилировщиком является утилита gprof, входящая в состав всех дистрибутивов Linux.

SIGPWR Система обнаружила надвигающуюся потерю питания. Обычно этот сигнал отправляется процессу init демоном, отслеживающим источники питания машины, позволяя корректно завершить работу до отключения питания.

SIGQUIT Этот сигнал посылается всем процессам в группе процессов переднего плана, когда пользователь нажимает клавиатурную комбинацию завершения (обычно ^/)

SIGSEGV Этот сигнал посылается, когда процесс пытается прочитать неотображаемую память, выполнить страницу памяти, которая не была отображена с привилегиями на выполнение, или же выполнить запись в память, к которой не имеет прав доступа на запись.

SIGSTOP Этот сигнал генерируется только вызовом kill (), и дает возможность пользователю безусловно остановить процесс.

SIGSYS Когда программа пытается выполнить несуществующий системный вызов, ядро прерывает программу с помощью этого сигнала. Это никогда не должно происходить в программах, которые осуществляют системные вызовы посредством системой библиотеки C.

Репозиторий ВГУ

SIGTERM Этот сигнал генерируется только вызовом kill () и дает возможность пользователю элегантно прервать процесс. Процесс должен прекратиться насколько возможно быстро, немедленно после получения сигнала.
SIGTRAP Когда программа проходит через точку прерывания, этот сигнал посылается процессу. Обычно он перехватывается процессом отладчика, который установил точку прерывания.
SIGTSTP Этот сигнал посылается всем процессам в группе процессов переднего плана, когда пользователь нажимает клавиатурную комбинацию прерывания (обычно ^Z).
SIGTTIN Этот сигнал посылается фоновому процессу, который пытается осуществить чтение из контролируемого им терминала.
SIGTTOU Этот сигнал посылается фоновому процессу, который пытается осуществить запись на контролируемый им терминал.
SIGURG Этот сигнал посылается, когда по сокету принимается экстренное сообщение.
SIGUSR1 Для этого сигнала нет предопределенного назначения; процессы могут использовать его для собственных нужд.
SIGUSR2 Для этого сигнала нет предопределенного назначения; процессы могут использовать его для собственных нужд.
SIGVTALRM Отправляется, когда истекает период действия таймера, установленного вызовом settimer().
SIGWINCH Когда окно терминала изменяет размер, например, когда пользователь растягивает окно xterm, все процессы в группе процессов переднего плана получают этот сигнал.
SIGXCPU Процесс превысил свой мягкий лимит использования ресурсов процессора. Этот сигнал посылается раз в секунду до тех пор, пока данный процесс не превысит жесткий лимит использования ресурсов процессора. Как только это произойдет, процесс прерывается сигналом SIGKILL.
SIGXFSZ Когда программа превышает лимит максимального размера файла, ей посылается этот сигнал, что обычно уничтожает процесс. Если сигнал перехвачен, то системный вызов, который послужил причиной превышения лимита на размер файла, возвращает ошибку EFBIG.

Репозиторий ВГУ