Содержание

1 Управление заданиями (job control)

«Управление заданиями» обозначает специальный протокол, с помощью которого пользователь может переключаться между несколькими группами процессов (заданиями) в пределах одной сессии.

1.1 Основные концепции

Основное назначение интерактивного интерпретатора команд состоит в чтении команд с терминала пользователя и создании процессов для запуска программ, заданных этими командами. Для этого используются вызов `fork` в сочетании с вызовом функции семейства `exec`.

Одна команда может запускать единственный процесс, но часто одна команда использует несколько процессов. Например, если используется оператор `|` командного интерпретатора, явно задаётся запуск нескольких программ в отдельных процессах. Даже если запускается одна программа, она может для своей работы использовать несколько процессов. Например, типичная команда компиляции `gcc -c foo.c` использует 4 процесса (хотя в любой момент времени работают только два из них). Назначание утилиты `make` — запускать другие программы в отдельных процессах.

Процессы, относящиеся к одной команде, называются *группой процессов* (process group) или *заданием* (job). Пользователь может оперировать с ними одной командой. Например, комбинация Ctrl-C посылает сигнал `SIGINT`, чтобы завершить все процессы в *основной* (foreground) группе процессов данного терминала.

Сессия (session) — это бóльшая группа процессов. Как правило, все процессы, запущенные в течение одного сеанса работы пользователя с системой на одном терминале, принадлежат одной сессии.

Каждый процесс относится к некоторой группе процессов. Когда создаётся новый процесс, он становится членом той же группы процессов и сессии, что его родительский процесс.

Процесс можно поместить в другую группу процессов, используя функцию `setpgid`, если процесс принадлежит к той же сессии, что и группа процессов, в которую он помещается.

Единственный способ перевести процесс в другую сессию заключается в том, чтобы сделать его начальным процессом новой сессии или *лидером сессии* (session leader), используя функцию `setsid`. Для лидера сессии создаётся новая группа процессов, из которой он уже не может быть перемещён в другую группу.

Обычно новые сессии создаются при входе пользователя в систему, и лидером сессии является процесс интерпретатора команд, запущенный в результате успешного входа.

Интерпретатор команд, поддерживающий управление заданиями, должен регулировать, какое из заданий может использовать терминал в любой данный момент времени. В противном случае может работать несколько заданий, пытающихся одновременно считать данные с терминала, и возникнет путаница, какое из них должно получить ввод пользователя. Чтобы предотвратить это, интерпретатор команд должен взаимодействовать с драйвером терминала, используя специальные функции, описанные в этом документе.

Интерпретатор команд может предоставлять неограниченный доступ к управляющему терминалу только одной группе процессов одновременно. Эта группа процессов называется *основным заданием* (foreground job) данного управляющего терминала. Другие группы процессов, запущенные интерпретатором команд, которые работают без такого доступа к терминалу, называются *фоновыми заданиями* (background job).

Если фоновое задание пытается считать данные с управляющего терминала, оно останавливается драйвером терминала. Кроме того, если установлен режим `TOSTOP` терминала, то же самое произойдет при записи данных на терминал фоновым заданием. Пользователь может остановить основное задание, нажав комбинацию Ctrl-Z (символ `SUSP`), а программа может остановить любое задание, послав ему сигнал `SIGSTOP`. Интерпретатор команд должен отслеживать факт остановки задания, оповещать пользователя об этом и предоставлять возможности для пользователя продолжить остановленное задание и переключать задания между основным и фоновым режимом.

1.2 Дополнительность управления заданиями

Не все операционные системы поддерживают управление заданиями, хотя большинство современных (**Linux**, **FreeBSD**, **Solaris**) поддерживают.

Чтобы проверить, поддерживает ли система управление заданиями, можно использовать макрос `_POSIX_JOB_CONTROL`. Если этот макрос определён в файле `<unistd.h>`, система поддерживает управление заданиями.

Если управление заданиями не поддерживается, в сессии может быть только одна группа процессов, которая ведёт себя, как будто всегда выполняется как основная группа. Функции создания новых групп процессов завершатся неудачно с кодом ошибки `ENOSYS`. Макросы, определяющие названия различных сигналов, связанных с управлением заданиями, определены, даже если система не поддерживает управление заданиями. Однако система никогда не генерирует такие сигналы, а попытка послать такой сигнал, проверить или установить реакцию на него завершается с ошибкой или ничего не делает.

1.3 Управляющий терминал

Одним из атрибутов процесса является его *управляющий терминал*. Сыновние процессы, созданные с помощью `fork`, наследуют управляющий терминал процесса-отца. Таким способом все процессы в сессии наследуют управляющий терминал от лидера сессии.

Лидер сессии, который управляет терминалом, называется *управляющим процессом* терминала.

Пользователь обычно не должен заботится о конкретном механизме, использующемся для того, чтобы выделить управляющий терминал сессии, поскольку это делается системой, когда пользователь входит в систему.

Некоторый процесс может отсоединиться от своего управляющего терминала, вызвав `setsid`, чтобы стать лидером новой сессии.

1.4 Доступ к управляющему терминалу

Процессы основного задания управляющего терминала имеют неограниченный доступ к терминалу, в отличие от фоновых процессов.

Когда процесс фонового задания пытается прочитать данные с его управляющего терминала, группе процессов посылается сигнал `SIGTTIN`. Обычно этот сигнал вызывает остановку всех процессов в этой группе (если только они обрабатывают сигнал и не останавливаются). Если процесс, который попытался прочитать данные, игнорирует или блокирует этот сигнал, `read` завершается с ошибкой `EIO`.

Аналогично, когда процесс фонового задания пытается записать на свой управляющий терминал, его группе процессов может быть послан сигнал `SIGTTOU`. Режим посылки этого сигнала устанавливается битом `TOSTOP` флагов локального режима терминала. Этот флаг по умолчанию не установлен, поэтому запись на управляющий терминал разрешена. Кроме того, запись на управляющий терминал разрешена, если сигнал `SIGTTOU` игнорируется или блокируется записывающим процессом.

Большинство операций с терминалом, которые может выполнить процесс, рассматриваются как чтение или запись. В описании каждой операции это указано.

1.5 Сиротские группы процессов

Когда управляющий процесс завершает работу, его терминал становится свободным и может использоваться для создания на нём новой сессии. Например, с того же терминала может зайти другой пользователь. Поэтому могут возникнуть неприятности, если какие-то процессы из старой сессии будут пытаться использовать этот терминал.

Чтобы предотвратить эту проблему, группы процессов, которые продолжают работать после того, как лидер сессии завершился, помечаются как *сиротские группы процессов* (orphaned process groups).

Когда группа процессов становится сиротой, всем процессам в группе посылается сигнал `SIGHUP`. Обычно это вызывает завершение работы программы. Однако если процесс игнорирует сигнал или устанавливает для него обработчик сигнала, процесс продолжает работу как процесс-сирота даже после того, как его управляющий процесс завершился, но он более не имеет доступа к терминалу.

1.6 Функции управления заданиями

1.6.1 Идентификация управляющего терминала

Функция `isatty` позволяет провериить, является ли данный файловый дескриптор дескриптором терминала.

```
#include <unistd.h>
int isatty(int fd);
```

Функция возвращает 1, если файловый дескриптор `fd` является дескриптором терминала, и 0 в противном случае. В случае ошибки (неверный файловый дескриптор) возвращается -1.

Функция `ttyname` возвращает имя специального файла устройства терминала.

```
#include <unistd.h>
char *ttyname(int fd);
```

Если файловый дескриптор `fd` связан с терминальным устройством, функция `ttyname` возвращает указатель на статическую строку символов, оканчивающуюся символом-терминатором `'\0'`. Эта строка содержит имя специального файла устройства данного терминала. Функция возвращает `NULL`, если файловый дескриптор не связан с терминальным устройством, или имя файла устройства не может быть определено.

Функция `ctermid` позволяет определить имя файла устройства управляющего терминала процесса.

```
#include <stdio.h>
char *ctermid(char *str);
```

Функция `ctermid` возвращает строку, содержащую имя файла устройства, соответствующего управляющему терминалу текущего процесса. Если `str` не равно `NULL`, это должен быть массив, который может хранить по крайней мере `L_ctermid` символов. Имя терминала будет записано в эту строку, и функция возвратит указатель на её начало. В противном случае функция возвращает указатель на статическую строку, которая может быть переписана при последующих вызовах этой функции.

Если имя управляющего терминала по какой-либо причине не может быть определено, функция возвращает пустую строку. Даже если функция вернула имя файла, доступ к соответствующему файлу не гарантируется.

Макрос `L_ctermid` задаёт длину строки, необходимую, чтобы хранить имя файла, возвращаемое `ctermid`.

1.6.2 Работа с группами процессов

```
#include <sys/types.h>
#include <unistd.h>

pid_t setsid(void);
pid_t getsid(pid_t pid);
int   setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
```

Функция `setsid` создаёт новую сессию. Текущий процесс становится лидером сессии и помещается в новую группу процессов, идентификатор которой совпадает с идентификатором данного процесса. Изначально группа процессов не содержит других процессов, и сессия не содержит других групп процессов. Процесс помечается, как не имеющий управляющего терминала.

Функция `setsid` в случае успеха возвращает идентификатор новой группы процессов. В случае ошибки возвращается -1, и переменная `errno` содержит код ошибки `EPERM`. Это значит, что процесс уже является лидером сессии.

Функция getsid возвращает идентификатор группы процессов лидера сессии указанного процесса pid. Если pid равен 0, берётся идентификатор текущего процесса. В случае ошибки возвращается -1, и устанавливается переменная errno.

ESRCH Процесса с таким pid не существует.

EPERM Текущий процесс и процесс с заданным идентификатором pid принадлежат разным сессиям, а конкретная реализация не позволяет в этом случае получать информацию об управляющем процессе.

Функция getpgid возвращает идентификатор группы процессов процесса с заданным идентификатором pid. Если аргумент pid равен 0, возвращается информация для текущего процесса. В случае ошибки возвращается -1.

Функция setpgid помещает процесс pid в группу процессов pgid. И pid, и pgid могут быть равны 0, что означает идентификатор текущего процесса. При необходимости создаётся новая группа процессов. Если pid не совпадает с идентификатором самого процесса, это должен быть идентификатор одного из сыновних процессов, который ещё не выполнил к этому моменту вызов exec. В случае успешного завершения функция возвращает 0. При неудаче функция возвращает -1, а переменная errno устанавливается в код ошибки. Если система не поддерживает управление заданиями, функция также завершается с ошибкой.

EACCES Сыновний процесс, заданный аргументом pid уже выполнил exec.

EINVAL Недопустимое значение pgid.

ENOSYS Система не поддерживает управление заданиями.

EPERM Процесс, заданный аргументом pid, является лидером сессии, не находится в той же самой сессии, что текущий процесс, значение pgid не соответствует группе процессов в той же самой сессии, что текущий процесс.

1.6.3 Управление доступом к терминалу

```
#include <sys/types.h>
#include <unistd.h>

pid_t tcgetpgrp(int fd);
int   tcsetpgrp(int fd, pid_t pgid);
pid_t tcgetsid(int fd);
```

Функция tcgetpgrp возвращает идентификатор основной группы процессов терминала, заданного открытым файловым дескриптором fd.

Если основная группа процессов отсутствует, то есть когда все процессы бывшей основной группы процессов завершили работу, и ни одно другое задание не было поставлено как основное, функция возвращает число, большее 1, которое не соответствует никакой существующей группе процессов.

В случае ошибки возвращается значение -1. Переменная errno может принимать следующие значения.

EBADF Аргумент fd не является открытым файловым дескриптором.

ENOSYS Система не поддерживает управление заданиями.

ENOTTY Терминал, ассоциированный с fd, не является управляющим терминалом данного процесса.

Функция tcsetpgrp используется, чтобы установить основную группу процессов терминала. Аргумент fd задаёт файловый дескриптор терминала, аргумент pgid определяет группу процессов. Текущий процесс должен принадлежать той же сессии, что и группа процессов pgid, и иметь тот же самый управляющий терминал.

С точки зрения доступа к терминалу, эта функция рассматривается как запись на терминал. Если она вызывается из фонового процесса управляющего терминала, обычно всем процессам в этой группе посылается сигнал SIGTTOU. Если текущий процесс блокирует или игнорирует сигнал SIGTTOU, операция выполняется успешно, и сигнал не посылается.

В случае успеха функция возвращает 0. Значение -1 обозначает ошибку, а переменная errno в этом случае может принимать следующие значения.

EBADF Аргумент fd не является открытым файловым дескриптором.

EINVAL Недопустимый аргумент pgid.

ENOSYS Система не поддерживает управление заданиями.

ENOTTY Терминал, ассоциированный с fd, не является управляющим терминалом данного процесса.

EPERM Группа процессов pgid не принадлежит к той же сессии, что текущий процесс.

Функция tcgetsid позволяет получить идентификатор группы процессов сессии, для которой терминал, заданный его файловым дескриптором fd, является управляющим терминалом. Если вызов завершился успешно, возвращается идентификатор группы процессов. В противном случае возвращается значение -1, а переменная errno устанавливается в код ошибки.

1.7 Работа интепретатора команд

В данном разделе подробно описываются действия, которые должен выполнять интепретатор команд, реализующий управление заданиями.

1.7.1 Инициализация интерпретатора команд

Когда интерпретатор команд, обычно выполняющий управление заданиями, запускается, он должен проверять, что он был запущен другим интерпретатором команд, который сам выполняет управление заданиями.

Интерпретатор команд может быть запущен в интерактивном или в неинтерактивном режиме. Неинтерактивный режим включается, когда выполняется одно из следующих условий:

- Один из стандартных потоков не ассоциирован с терминалом.
- В командной строке задан файл, который необходимо выполнить.
- В командной строке задана команда, которую необходимо выполнить.
- Интерактивный режим отключён специальной опцией командной строки.

Интерактивный интерпретатор команд должен убедиться, что он запущен как основное задание управляющего терминала, перед тем, как включать своё управление заданиями. Для этого интерпретатор запрашивает идентификатор своей группы процессов с помощью getpgid и сравнивает его с идентификатором основной группы процессов управляющего терминала, полученной с помощью tcgetpgrp.

Если интерпретатор не является основным заданием, он должен остановить себя, послав сигнал SIGTTIN своей группе процессов. Он не должен пытаться сам сделать свою группу основной для терминала, это должен сделать пользователь. Если интерпретатор был продолжен, он должен повторить проверку и остановить себя, если он снова не основной процесс.

После того, как интерпретатор был сделан основным процессом терминала, он может включить своё управление заданиями. Для этого вызывается `setpgid`, чтобы создать отдельную группу процессов для интепретатора, затем вызывается `tcsetpgrp`, чтобы сделать эту группу процессов основной.

Когда интерпретатор включает управление заданиями, он должен игнорировать сигналы управления заданиями, чтобы случайно не остановить самого себя. Для этого обработчики соответствующих сигналов устанавливаются в `SIG_IGN`.

Интерпретатор команд, работающий в неинтерактивном режиме, не должен поддерживать управление заданиями. Все запускаемые им процессы должны оставаться в той же группе процессов, в которой находится сам интерпретатор. Таким образом родительский интерпретатор сможет рассматривать неинтерактивный интерпретатор команд и все его сыновние процессы как одно задание.

1.7.2 Запуск заданий

После того, как интерпретатор команд включил управление заданиями, он может обрабатывать команды пользователя и запускать новые задания.

Для создания новых процессов в группе процессов используются вызовы `fork` и `exec`. Однако, поскольку обычно в этой операции участвуют сразу несколько процессов, необходимо выполнять операции в правильном порядке, чтобы избежать временны́х ошибок.

Существует два варианта, как можно структурировать дерево отношения «отец-сын» для процессов. Во-первых, все процессы в группе процессов может создавать сам командный интерпретатор; во-вторых, командный интерпретатор может создать один процесс, который будет отцом всех остальных процессов в группе. Обычно первый подход проще в реализации.

После того, как каждый процесс создан с помощью `fork`, он должен поместить себя в новую группу процессов, вызвав `setpgid`. Первый процесс в новой группе становится лидером группы процессов, а его идентификатор процесса становится идентификатором всей группы процессов.

Сам интерпретатор тоже должен вызвать `setpgid`, чтобы поместить каждый из его сыновних процессов в новую группу процессов. Это нужно для того, чтобы избежать временно́й ошибки: каждый сыновний процесс должен быть помещён в новую группу процессов до того, как он начнёт выполнять новую программу, но и сам интерпретатор должен иметь все сыновние процессы в новой группе перед тем, как он продолжит выполнение. Если и интерпретатор, и каждый из сыновних процессов вызовут `setpgid`, это гарантирует, что всё будет сделано правильно независимо от того, какой из процессов сделает это первым.

Если задание запускается как основное задание, новая группа процессов должна быть сделана основной группой процессов управляющего терминала с помощью вызова функции `tcsetpgrp`. Точно так же, это действие должен сделать и сам интерпретатор, и каждый из его сыновних процессов.

После этого каждый сыновний процесс должен восстановить обработку сигналов. При инициализации процесс интерпретатора установил игнорирование сигналов управления заданиями. В результате каждый сыновний процесс унаследовал эту обработку сигналов. Для сыновних процессов это, без сомнения, нежелательно, поэтому каждый сыновний процесс должен явно установить действия для этих сигналов в `SIG_DFL`.

Наконец, каждый сыновний процесс должен вызвать `exec` и запустить требуемую программу. В этом месте должны быть обработаны перенаправления дескрипторов ввода/вывода.

Если командный интерпретатор не запущен как интерактивный, все манипуляции с изменением группы процессов и переустановкой сигналов выполняться не должны.

1.7.3 Переключение между основным и фоновым режимом

Когда запускается основное задание, интерпретатор должен дать ему доступ к управляющему терминалу, вызвав `tcsetpgrp`. После этого он должен ждать пока все процессы в основной группе процессов не завершатся или не будут остановлены.

После того, как все процессы в основной группе завершились или остановлены, интерпретатор должен снова взять терминал под контроль для своей собственной группы процессов, вызвав `tcsetpgrp`.

Фоновое задание может оставить терминал в непонятном состоянии, поэтому интерпретатор должен восстановить свои сохранённые режимы терминала перед тем, как перейти к вводу команд пользователя. В случае, если задание было остановлено, интерпретатор должен перед этим сохранить текущие режимы терминала, чтобы восстановить их позднее, когда задание будет продолжено. Для сохранения и восстановления режимов терминала используются функции `tcgetattr` и `tcsetattr`.

Если задание запускается в фоновом режиме, интерпретатор остается в основном режиме и продолжает ввод команд пользователя с терминала.

1.7.4 Остановленные и завершившиеся задания

Когда основное задание запущено, интерпретатор должен дождаться, пока все процессы задания либо завершатся, либо будут остановлены. Для этого интерпретатор может вызвать функции `waitpid` или `wait4`. Чтобы получать информацию об остановленных процессах нужно указать опцию `WUNTRACED`.

Кроме этого интерпретатор должен регулярно проверять статус фоновых заданий, чтобы сообщать о завершившихся или остановленных заданиях пользователю. Для этого могут использоваться функции `waitpid` или `wait4` с опцией `WNOHANG`. Такая проверка может быть помещена перед вводом новых команд пользователя.

Интерпретатор может получать асинхронные сообщения о том, что доступен статус сыновних процессов, если он установит обработчик сигнала `SIGCHLD`.

1.7.5 Продолжение остановленных заданий

Интерпретатор может продолжить остановленное задание, послав сигнал `SIGCONT` соответствующей группе процессов. Если задание продолжается как основное задание терминала, интерпретатор должен вызвать `tcsetpgrp`, чтобы дать заданию доступ к терминалу, и восстановить установки терминала. После того, как задание было продолжено в режиме основного задания терминала, интерпретатор команд должен подождать, пока все процессы задания завершатся или будут остановлены, как будто бы это задание было только что запущено как основное.