# ОПЕРАЦИОННЫЕ СИСТЕМЫ И СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

## Лекция № 04 — Иерархия процессов

**Преподаватель: Поденок Леонид Петрович, 505а-5**

**+375 17 293 8039 (505а-5)**
**+375 17 320 7402 (ОИПИ НАНБ)**
**prep@lsi.bas-net.by**

**ftp://student:2ok*uK2@Rwox@lsi.bas-net.by**

**Кафедра ЭВМ, 2024**

2023.03.07

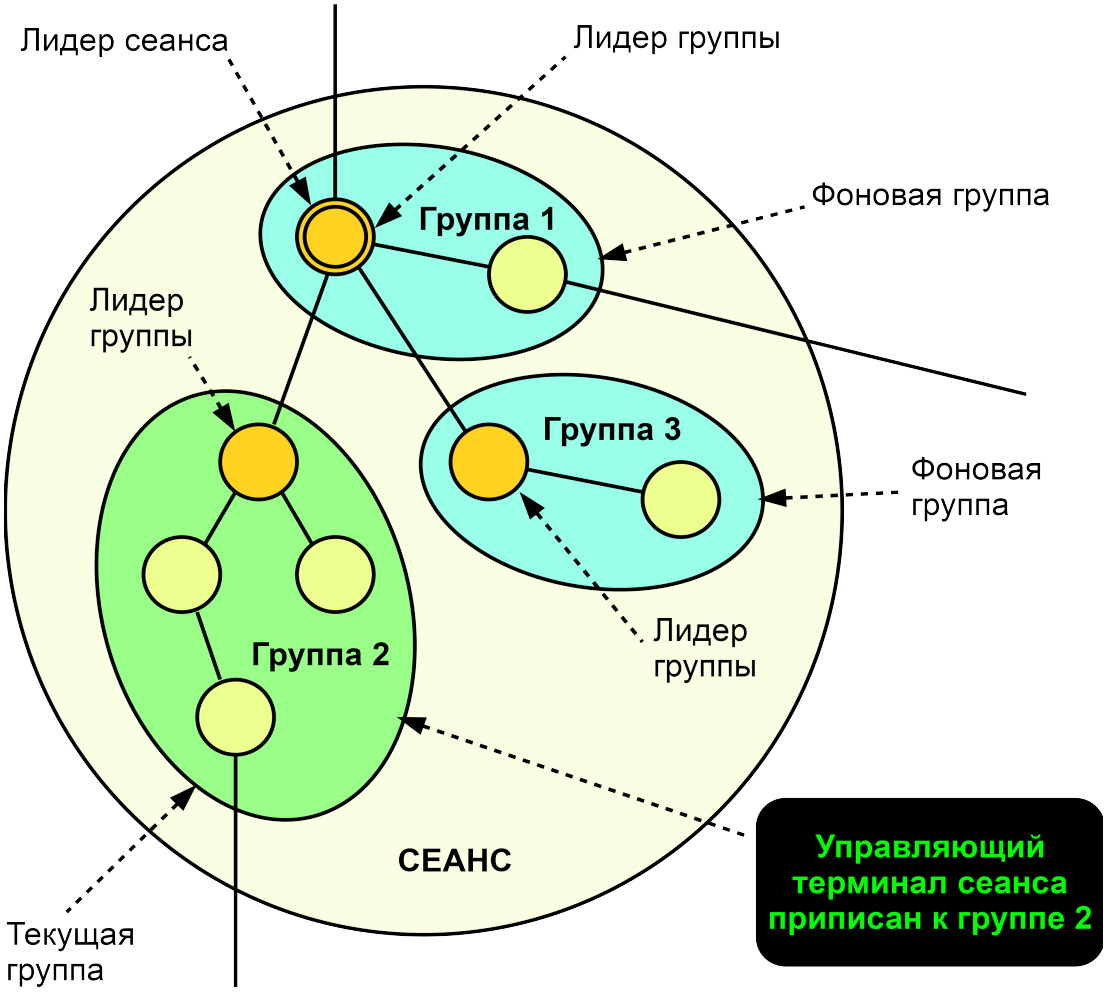
# Оглавление

# Группы процессов

Понятия группы процессов, сеанса, лидера группы, лидера сеанса, управляющего терминала сеанса. Системные вызовы **`getpgrp()`**, **`setpgrp()`**, **`getpgid()`**, **`setpgid()`**, **`getsid()`**, **`setsid()`**.

Все процессы в системе связаны родственными отношениями,
образуя генеалогическое дерево или лес из таких деревьев, где в качестве узлов деревьев выступают сами процессы, а связями служат отношения родитель-потомок.

Все эти деревья принято разделять на группы процессов.



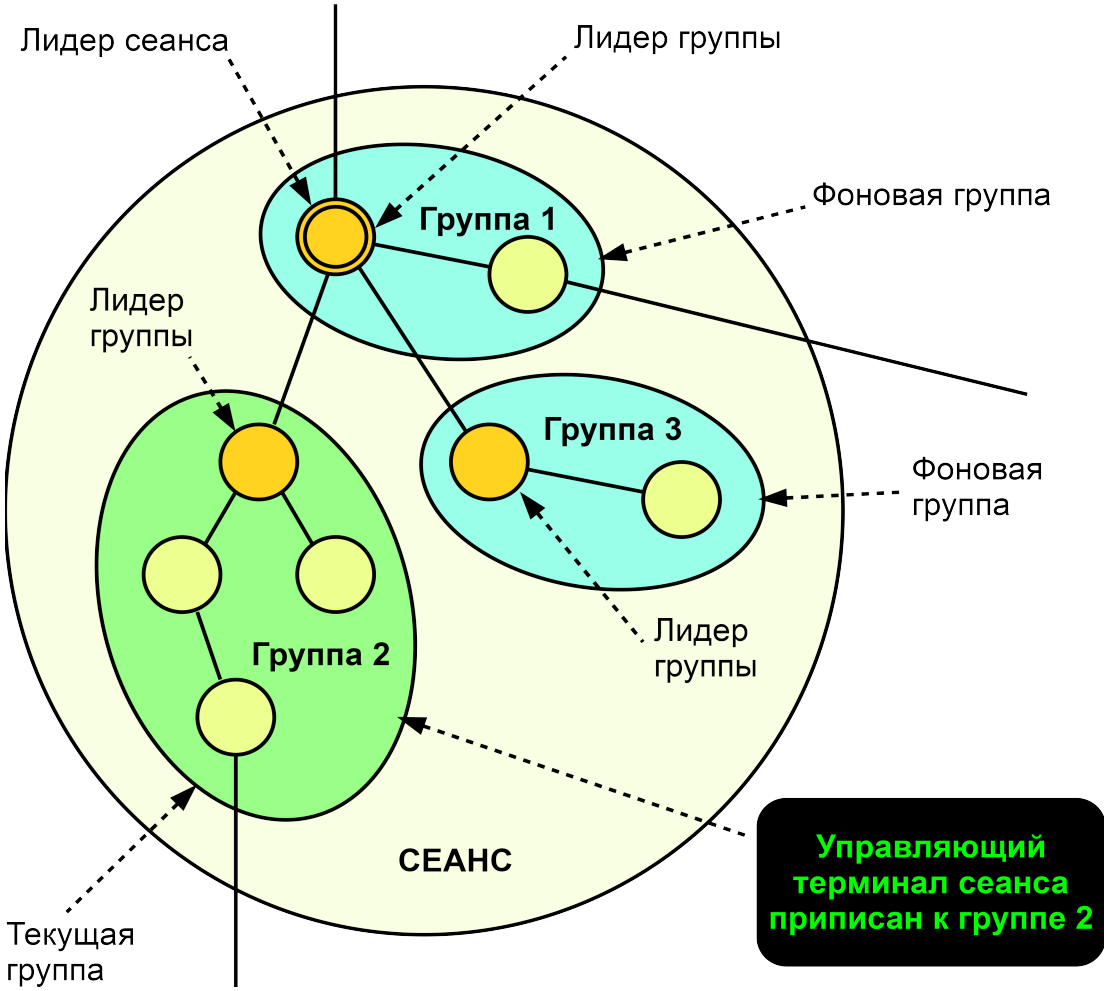
## Группа процессов

Группа процессов включает в себя один или более процессов и существует, пока в группе присутствует хотя бы один процесс.

Каждый процесс обязательно включен в какую-нибудь группу.

При рождении нового процесса он попадает в ту же группу процессов, в которой находится его родитель (**`cgid = pgid`**).

Процессы могут мигрировать из группы в группу по своему собственному желанию или по желанию другого процесса (в зависимости от версии UNIX).




Многие системные вызовы могут быть применены не к одному конкретному процессу, а ко всем процессам в некоторой группе (**`waitpid`**). Поэтому способ объединения процессы в группы определяется тем, как их собираются использовать.

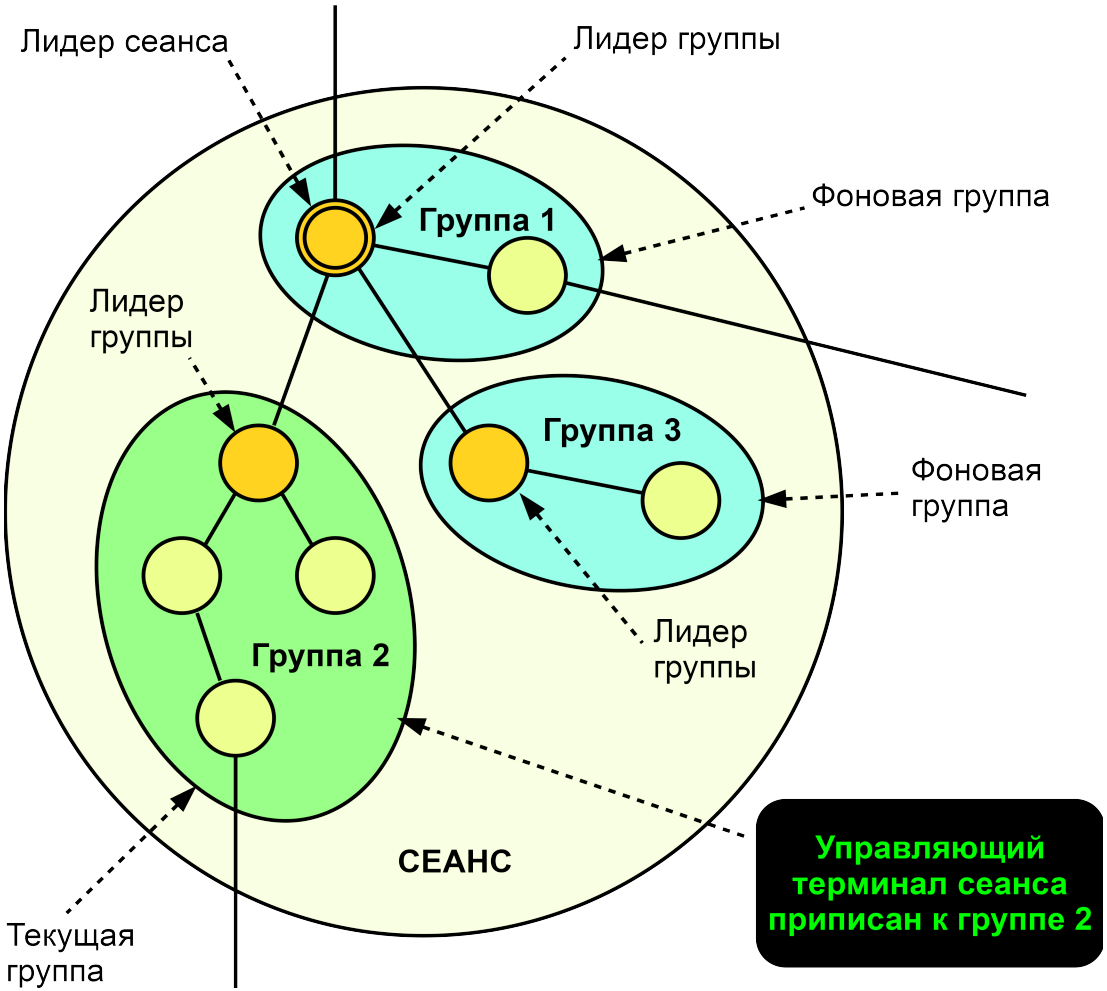
## Сеансы

Группы процессов объединяются в сеансы.

Понятие сеанса изначально было введено в UNIX для логического объединения групп процессов, созданных в результате каждого входа и последующей работы пользователя в системе.

В связи с этим с каждым сеансом в системе может быть связан терминал, называемый управляющим терминалом сеанса, через который обычно и общаются процессы сеанса с пользователем.

**Сеанс не может иметь более одного связанного с ним управляющего терминала, и управляющий терминал связан ровно с одним сеансом.**




В то же время могут существовать сеансы, вообще не имеющие управляющего терминала.

Определенные входные последовательности от управляющего терминала вызывают отправку сигналов всем процессам в группе процессов переднего плана[1], связанной с управляющим терминалом.

1 foreground processes

## Идентификаторы процесса

### ID (PID) процесса

Каждый процесс имеет уникальный неотрицательный целочисленный идентификатор (PID), который ему назначается при создании с помощью **fork(2)**.

Процесс может узнать свой PID с помощью вызова **getpid(2)**.

PID имеет тип **pid_t** (определён в **<sys/types.h>**).

PID используется в различных системных вызовах для указания процесса, с которым работает вызов, например, **kill(2)**, **ptrace(2)**, **setpriority(2)**, **setpgid(2)**, **setsid(2)**, **sigqueue(3)** и **waitpid(2)**.

PID процесса сохраняется после вызова **execve(2)**.

### Родительский ID (PPID) процесса

ID родительского процесса — это ID процесса, который создал данный процесс с помощью **fork(2)**. Процесс может получить свой PPID с помощью **getppid(2)**. PPID имеет тип **pid_t**.

PPID процесса сохраняется после вызова **execve(2)**.

### ID группы процессов и сеанса

У каждого процесса есть ID сеанса и ID группы процессов – они тоже имеют тип **pid_t**.

Процесс может получить ID ***своего сеанса*** с помощью **getsid(2)**, а ID ***своей группы*** процессов с помощью **getpgrp(2)**.

Потомок, создаваемый с помощью **fork(2)**, наследует ID сеанса и группы процессов своего родителя. Идентификатор сеанса и группы сохраняется после **execve(2)**.

**Группа процессов переднего плана (или задание переднего плана)**[2] – группа процессов, процессы-члены которой имеют определенные привилегии, недоступные процессам в фоновых группах процессов в отношении доступа их к управляющему терминалу.

Каждый сеанс, который установил соединение с управляющим терминалом, имеет не более одной группы процессов сеанса в качестве группы процессов переднего плана этого управляющего терминала.

**Группа фоновых процессов (или фоновое задание)** – любая группа процессов, кроме группы процессов переднего плана[3], которая является членом сеанса, установившего соединение с управляющим терминалом.

Сеансы и группы процессов — это абстракции, предназначенные для поддержки управления заданиями оболочки.

Группа процессов (иногда называемая «заданием» (job)) — это набор процессов, у которых одинаковый ID группы процессов.

Оболочка создаёт новую группу процессов для процессов, используемых в одной команде или конвейере (например, два процесса, созданные командой «**`dirwalk | wc`**», помещаются в одну группу процессов).

Членство в группе процессов может быть установлено с помощью **`setpgid(2)`**.

Процесс, чей ID процесса совпадает с его ID группы процессов, называется лидером группы процессов этой группы.

---

2 foreground

3 foreground

## Сеанс

Сеанс – это набор процессов, у которых одинаковый ID сеанса.

Все члены группы процессов имеют одинаковый ID сеанса – они всегда принадлежат одному сеансу и сеансы и группы процессов формируют из процессов жёсткую двухуровневую иерархию.

Новый сеанс создаётся вызовом **`setsid(2)`**.

***ID созданного сеанса совпадает с PID процесса***, который вызвал **`setsid(2)`**. Создатель сеанса также называется лидером сеанса.

**Управляющий процесс** – лидер сеанса, установивший соединение с управляющим терминалом. Если терминал впоследствии перестает быть управляющим терминалом для этого сеанса, лидер сеанса перестает быть управляющим процессом.

**Все процессы в сеансе используют общий управляющий терминал.**

Управляющий терминал назначается в момент, когда лидер сеанса впервые открывает терминал (если при вызове **`open(2)`** не указан флаг **`O_NOCTTY`**).

Терминал может быть управляющим терминалом не более чем для одного сеанса.

В сеансе может быть только одно активное задание (foreground job), все остальные задания в сеансе считаются фоновыми заданиями (background jobs).

Только активное задание может читать данные из терминала.

Когда процесс в фоне пытается прочитать данные с терминала, его группе процессов посылается сигнал **`SIGTTIN`**, который приостанавливает (suspends) задание.

Если у терминала установлен флаг **`TOSTOP`** (**`termios(3)`**), то только активное задание может писать в терминал.

Попытка записи из фонового задания приводит к генерации сигнала **SIGTTOU**, который приостанавливает задание.

Если нажимаются клавиши терминала, которые генерируют сигнал (например клавиша interrupt, обычно это комбинация Ctrl-C), то сигнал посылается процессам в активном задании.

С членами группы процессов могут работать различные системные вызовы и библиотечные функции и операции **fcntl(2)**, (**kill(2)**, **waitpid(2)**...).

## Идентификаторы пользователя и группы

С каждым процессом связаны идентификатор пользователя и различных групп. Эти идентификаторы представляются в виде целых чисел с типами **`gid_t`** и **`uid_t`**, соответственно (определены в **`<sys/types.h>`**). В Linux каждый процесс имеет следующие идентификаторы пользователя и групп:

**1) Реальный ID пользователя (real user) и реальный ID группы**.

Эти ID определяют кто владелец процесса.

Реальный ID пользователя и группы процесса можно получить с помощью вызовов **`getuid(2)`** и **`getgid(2)`**.

**2) Эффективный[4] ID пользователя (effective user) и эффективный ID группы.**

Эти ID используются ядром для определения прав, которые будет иметь процесс при доступе к общим ресурсам, таким как очереди сообщений, общая память и семафоры.

**В большинстве систем UNIX эти ID также определяют права доступа к файлам (см. п. 4)**).

Однако в Linux для этой задачи используются ID файловой системы.

Эффективный ID пользователя и группы процесса можно получить с помощью вызовов **`geteuid(2)`** и **`getegid(2)`**.

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

uid_t getuid(void);
uid_t geteuid(void);
uid_t getgid(void);
uid_t getegid(void);
```

4 действующий

**3) Сохранённые set-user-ID и set-group-ID.**

Эти ID используются в программах с установленными битами **set-user-ID** и **set-group-ID** для сохранения копии соответствующих эффективных ID, которые устанавливаются в момент запуска программы (**execve(2)**).

Программа с **set-user-ID** может повышать и понижать права, переключая свой ID эффективного пользователя туда и обратно между значениями её ID реального пользователя и сохранённым **set-user-ID**.

Такое переключение производится с помощью вызовов **seteuid(2)**, **setreuid(2)** или **setresuid(2)**.

Программа с **set-group-ID** выполняет аналогичные задачи с помощью **setegid(2)**, **setregid(2)** или **setresgid(2)**.

Сохранённый **set-user-ID** и **set-group-ID** процесса можно получить с помощью **getresuid(2)** и **getresgid(2)**.

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

int seteuid(uid_t euid); // меняет эффективный идентификатор пользователя
int setegid(gid_t egid); // меняет эффективный идентификатор группы

// меняет действительный и эффективный ID пользователя вызывающего процесса
int setreuid(uid_t ruid, uid_t euid);
int setregid(gid_t rgid, gid_t egid);
```

4) **ID пользователя файловой системы и ID группы файловой системы**[5].

Эти ID используются для определения прав доступа к файлам (`path_resolution(7)`).

Каждый раз при изменении ID эффективного пользователя (группы) ядро также автоматически изменяет ID пользователя (группы) файловой системы на то же значение. Следовательно, ID файловой системы, обычно, равны соответствующим эффективным ID, а семантика проверки прав доступа к файлам в Linux такая же как и у других систем UNIX.

ID файловой системы можно сделать отличным от эффективных ID с помощью вызова **`setfsuid(2)`** и **`setfsgid(2)`**.

```
#include <sys/fsuid.h> // Только в Linux

int setfsuid(uid_t fsuid);
int setfsuid(uid_t fsuid);
```

5 есть только в Linux

**5) ID дополнительных групп (supplementary group)**.

Определяет ID добавочных групп, которые используются при проверке доступа к файлам и другим общим ресурсам.

В ядрах Linux до версии 2.6.4 процесс может быть членом 32 дополнительных групп.

Начиная с версии 2.6.4 процесс может быть членом 65536 дополнительных групп. В помощью вызова **sysconf(_SC_NGROUPS_MAX)** можно узнать количество дополнительных групп, в которых процесс может быть членом.

Процесс может получить список ID дополнительных групп с помощь **getgroups(2)**, и изменить этот список с помощью **setgroups(2)**.

Дочерний процесс, созданный **fork(2)**, наследует копии ID пользователя и все группы своего родителя.

При **execve(2)** сохраняются ID реального пользователя и группы процесса, а также ID дополнительных групп, в то же время эффективный и сохранённый ID могут измениться (описано в **execve(2)**).

Кроме целей, отмеченных выше, идентификаторы пользователя процесса также используются:

- при определении права на отправку сигналов (**kill(2)**;
- при определении права на установку параметров планировщика процесса (значение уступчивости, политика и приоритет планирования в реальном времени, увязывание ЦП, приоритет ввода-вывода);
- при проверке ограничения по ресурсам (**getrlimit(2)**);
- при проверке ограничения на количество экземпляров **inotify**[6], которые процесс может создать (**inotify(7)**).

6 Наблюдает за событиями файловой системы

## getpgid() – получить идентификатор группы процессов

Каждая группа процессов в системе получает свой собственный уникальный номер – **gid**. Узнать этот номер можно с помощью системного вызова **getpgid()**.

Используя его, процесс может узнать номер группы для себя самого или для процесса из своего сеанса.

Данный системный вызов присутствует не во всех версиях UNIX. Это следствие разделения UNIX на линию BSD и линию System V. Вместо вызова **getpgid()** в таких системах существует системный вызов **getpgrp()**, который возвращает номер группы только для текущего процесса. Linux поддерживает оба системных вызова.

## setpgid() – установить идентификатор группы процессов

Для перевода процесса в другую группу процессов, возможно с одновременным ее созданием, применяется системный вызов **setpgid()**.

Перевести в другую группу процесс может либо самого себя (не во всякую и не всегда), либо своего процесса-потомка, который не выполнял системный вызов **exec()**, т.е. не запускал на выполнение другую программу.

При определенных значениях параметров системного вызова создается новая группа процессов с идентификатором, совпадающим с идентификатором переводимого процесса, состоящая первоначально только из одного этого процесса.

Новая группа может быть создана только таким способом, поэтому идентификаторы групп в системе уникальны. Переход в другую группу без создания новой группы возможен только в пределах одного сеанса.

В некоторых разновидностях UNIX системный вызов **setpgid()** отсутствует, а вместо него существует системный вызов **setpgrp()**, который умеет только создавать новую группу процессов с идентификатором, совпадающим с ID текущего процесса, и переводить текущий процесс в нее.

В некоторых разновидностях UNIX, где совместно сосуществуют вызовы **setpgrp()** и **setpgid()**, например в Solaris, вызов **setpgrp()** ведет себя иначе – он аналогичен **setsid()**.

**Лидер группы**

Процесс, идентификатор которого совпадает с идентификатором его группы, называется лидером группы (**pid=gid**).

Одно из ограничений на применение вызовов **setpgid()** и **setpgrp()** состоит в том, что **лидер группы не может перебраться в другую группу**.

**Сеанс**

Каждый сеанс в системе также имеет свой собственный номер – **sid**.

Для того, чтобы узнать его можно воспользоваться системным вызовом **getsid()**. В разных версиях UNIX на него накладываются различные ограничения.

В Linux такие ограничения отсутствуют.

**Лидер сеанса**

Системный вызов **setsid()** приводит к созданию новой группы, состоящий только из процесса, который его выполнил (он становится лидером новой группы), и нового сеанса, идентификатор которого совпадает с идентификатором процесса, сделавшего вызов (**pid=gid=sid**).

Такой процесс называется лидером сеанса.

**Этот системный вызов может применять только процесс, не являющийся лидером группы.**

## Установить/получить ID группы процесса

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

int   setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);

pid_t getpgrp(void);                  // по версии POSIX.1
pid_t getpgrp(pid_t pid);             // по версии BSD

int   setpgrp(void);                  // по версии System V
int   setpgrp(pid_t pid, pid_t pgid); // по версии BSD
```

Все перечисленные интерфейсы доступны в Linux и используются для получения и установки идентификатора группы процессов (PGID).

Для получения PGID вызывающего процесса предпочтительней использовать версию POSIX.1 – **`getpgrp(void)`**, а для установки PGID вызывающего процесса – **`setpgid()`**.

Вызов **`setpgid()`** устанавливает PGID у процесса с идентификатором **`pid`** равным **`pgid`**.

Если значение **`pid`** равно 0, то используется идентификатор вызывающего процесса.

Если значение **`pgid`** равно 0, то PGID процесса, указанного в **`pid`**, становится равным его идентификатору процесса.

Если **setpgid()** используется для перевода процесса из одной группы в другую (это делают некоторые оболочки командной строки для объединения каналов процессов), то обе группы процессов должны быть частью одного сеанса (**setsid(2)**). В этом случае в **pgid** указывается существующая группа процессов, в которую нужно выполнить перевод и идентификатор сеанса этой группы должен совпадать с идентификатором сеанса переводимого процесса.

В версии POSIX.1 вызов **getpgrp()** без аргументов возвращает PGID вызывающего процесса.

В версии System V вызов **setpgrp()** без аргументов эквивалентен **setpgid(0, 0)**.

**Вовращают**

При успешном выполнении **setpgid()** и **setpgrp()** возвращают 0.

В случае ошибки возвращается -1, а **errno** устанавливается в соответствующее значение.

Вызов **getpgrp()** (POSIX.1) всегда возвращает PGID вызывающего процесса.

Вызовы **getpgid()** и **getpgrp()** (BSD) при успешном выполнении возвращают ID группы процессов. При ошибке возвращается -1, а значение **errno** устанавливается соответствующим образом.

## setsid() – создает сеанс и устанавливает идентификатор группы процесса

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

pid_t setsid(void);
```

Если вызывающий процесс не является лидером группы процессов, вызов **setsid()** создаёт новый сеанс. Вызывающий процесс становится лидером нового сеанса (то есть, его ID сеанса становится равным ID самого процесса).

Вызывающий процесс также становится лидером группы процессов новой группы процессов в сеансе (то есть, его ID группы процессов становится равным ID самого процесса).

Вызывающий процесс будет единственным в новой группе процессов и новом сеансе.

Изначально, новый сеанс не имеет управляющего терминала.

Управляющий терминал назначается в момент, когда лидер сеанса впервые открывает терминал (если при вызове **open(2)** не указан флаг **O_NOCTTY**).

Терминал может быть управляющим терминалом не более чем для одного сеанса.

**Возвращает**

При успешном выполнении возвращается идентификатор (нового) сеанса вызывающего процесса. В случае ошибки возвращается **(pid_t)-1**, а **errno** устанавливается в соответствующее значение.

**Ошибки**

**EPERM** – вызывающий процесс является лидером группы процессов.

Лидер группы процессов — это процесс, идентификатор группы процессов которого равен идентификатору самого процесса (PID).

Отказ лидера группы процессов выполнять **setsid()** предотвращает возможность того, что сам лидер группы процессов переместится в новый сеанс, в то время как другие процессы в группе останутся в первоначальном сеансе, что поломало бы жёсткую двухуровневую иерархию сценариев и групп процессов.

Для того, чтобы **setsid()** выполнился успешно, следует вызвать **fork(2)** и в родителе **_exit(2)**, а затем в дочернем процессе (который по определению не может быть лидером группы процессов) следует вызвать **setsid()**.

Если сеанс имеет управляющий терминал, у которого не установлен флаг **CLOCAL** и возникает зависание (hangup) терминала, то лидеру сеанса посылается **SIGHUP**.

Если завершается процесс, который является лидером сеанса, то сигнал **SIGHUP** посылается каждому процессу в приоритетной (foreground) группе процессов управляющего терминала.