# 2.2. Режим, пространство и контекст

Для возможности функционирования системы Unix аппаратная часть компьютера должна поддерживать по крайней мере два режима выполнения: более привилегированный **режим ядра** и менее привилегированный **режим задачи**. Как и следовало ожидать, прикладные программы работают в режиме задачи, а функции ядра выполняются в режиме ядра.

Большинство реализаций Unix используют **виртуальную память**.

* Адреса, выделенные программе, не ссылаются непосредственно на физическое размещение в памяти. Каждому процессу предоставляется собственное виртуальное адресное пространство, а ссылки на виртуальные адреса памяти транслируются в их фактическое нахождение в физической памяти при помощи набора **карт трансляции адресов.**

Ядро системы UNIX является **реентерабельным**, то есть к ядру могут обращаться одновременно несколько процессов. Фактически они могут выполнятТЬ одни и те же последовательности инструкций параллельно.

**Реентерабельностью** называется возможность единовременного обращения различных процессов к одному и тому же машинному коду, загруженному в память компьютера в единственном экземпляре.

“Реентабельный код – код чистых процедур” - лек

# 2.3.3. Полномочия пользователя. стр 68

Каждый пользователь системы имеет:

**идентификатором пользователя** (user ID, UID)

**идентификатором группы** (user group ID,или GID).

идентификаторы определяют принадлежность файлов, права доступа, а также возможность посылки сигналов другим процессам.

Супер пользователь:

UID = 0

GID = 1

**реальным (real) и действительным (effective)**

Каждый процесс обладает двумя идентификаторами пользователя — реальным (real) и действительным (effective)

Эффективный UID, GID влияют на создание файлов и доступ к ним.

* Во время создания файла ядро устанавливает ему атрибуты владельца файла как эффективные UID и GID процесса, из которого был сделан вызов на создание файла. Во время доступа процесса к файлу ядро использует эффективные идентификаторы процесса для определения, имеет ли он право обращаться к этому файлу

Реальный UID и реальный GID идентифицируют владельца процесса и влияют на право отправки сигналов.

* Процесс, не имеющий привилегий суперпользователя, может передавать сигналы другому процессу только в том случае, если реальный или эффективный UID отправителя совпадает с реальным UID получателя.

Управляющая информация о процессе поддерживается с помощью двух структур данных, имеющихся у каждого процесса: области u и структуры ргос.

# 2.4. Выполнение в режиме ядра. стр 72

## перевести систему в режим ядра.

три различных типа событий, которые могут **перевести систему в режим ядра**.

1. прерывания устройств
2. исключительные ситуации или просто исключения
3. ловушки или программные прерывания

## Прерывания

**Прерывания** — это асинхронные события, происходящие в периферийных устройствах, таких как диски, терминалы или аппаратный таймер.

* Так как прерывания не зависят от текущего процесса, они должны обрабатываться в системном контексте, при этом доступ в адресное пространство или область и процесса им не требуется.
* По этой же причине прерывания не должны производить блокировку, так как они могут заблокировать произвольный процесс.

## Исключительные состояния

**Исключительные состояния** возникают по ходу процесса по причинам, зависящим от него самого, например при попытке деления на ноль или обращения по несуществующему адресу.

* Обработчик исключительных состояний работает в контексте процесса и может обращаться к адресному пространству или области и процесса, а также блокировать процесс, если это необходимо.

## Программные либо системные прерывания (traps или ловушки)

**Программные либо системные прерывания (traps или ловушки)** происходят во время выполнения процессом особых инструкций, например в процессе перехода в системные вызовы, и обрабатываются синхронно в контексте процесса.

# 2.4.1. Интерфейс системных вызовов

Программный интерфейс ОС определяется набором системных вызовов, предоставляемых ядром пользовательским процессам.

*Системные вызовы выполняются в режиме ядра*, но в контексте процесса. Следовательно, они имеют возможность доступа к адресному пространству и области и вызывающего процесса. С другой стороны, они могут обращаться и к стеку ядра этого процесса.

# 2.4.2. Обработка прерываний

* Основная функция прерываний в компьютере заключается в том, чтобы **позволить взаимодействовать периферийным устройствам с процессором**, информируя его о завершении работы задачи, ошибочных состояниях и других событиях, требующих немедленного внимания.
* Прерывания происходят независимо от текущих действий системы или какого-либо процесса.

Функция, запускаемая для обслуживания прерывания, называется **обработчиком прерывания**, или процедурой обслуживания прерывания.

* Обработчик работает в режиме ядра и системном контексте. Так как прерванный процесс обычно не имеет никакого отношения к произошедшему в системе прерыванию, обработчик не должен обращаться к контексту процесса.
* По той же причине он также не обладает правом блокировки.
* Время, потраченное на обработку прерывания, является частью кванта времени, отведенного процессу, даже если производимые действия не имеют ни малейшего к нему отношения.

Так, **обработчик прерываний системного таймера использует тики** (промежутки времени между двумя прерываниями таймера) текущего процесса и потому нуждается в доступе к его структуре ргос.

Ядро также поддерживает прог**раммные или системные прерывания**, которые могут генерироваться при выполнении специальных инструкций.

* Такие прерывания могут использоваться, к примеру, в переключателе контекстов или в планировании задач, функционирующих в режиме разделения времени и имеющих низкий приоритет.
* Несмотря на то что описанные прерывания происходят синхронно с нормальной работой системы, они обрабатываются точно так же, как и обычные.

# 2.8.5. Завершение процесса стр 91

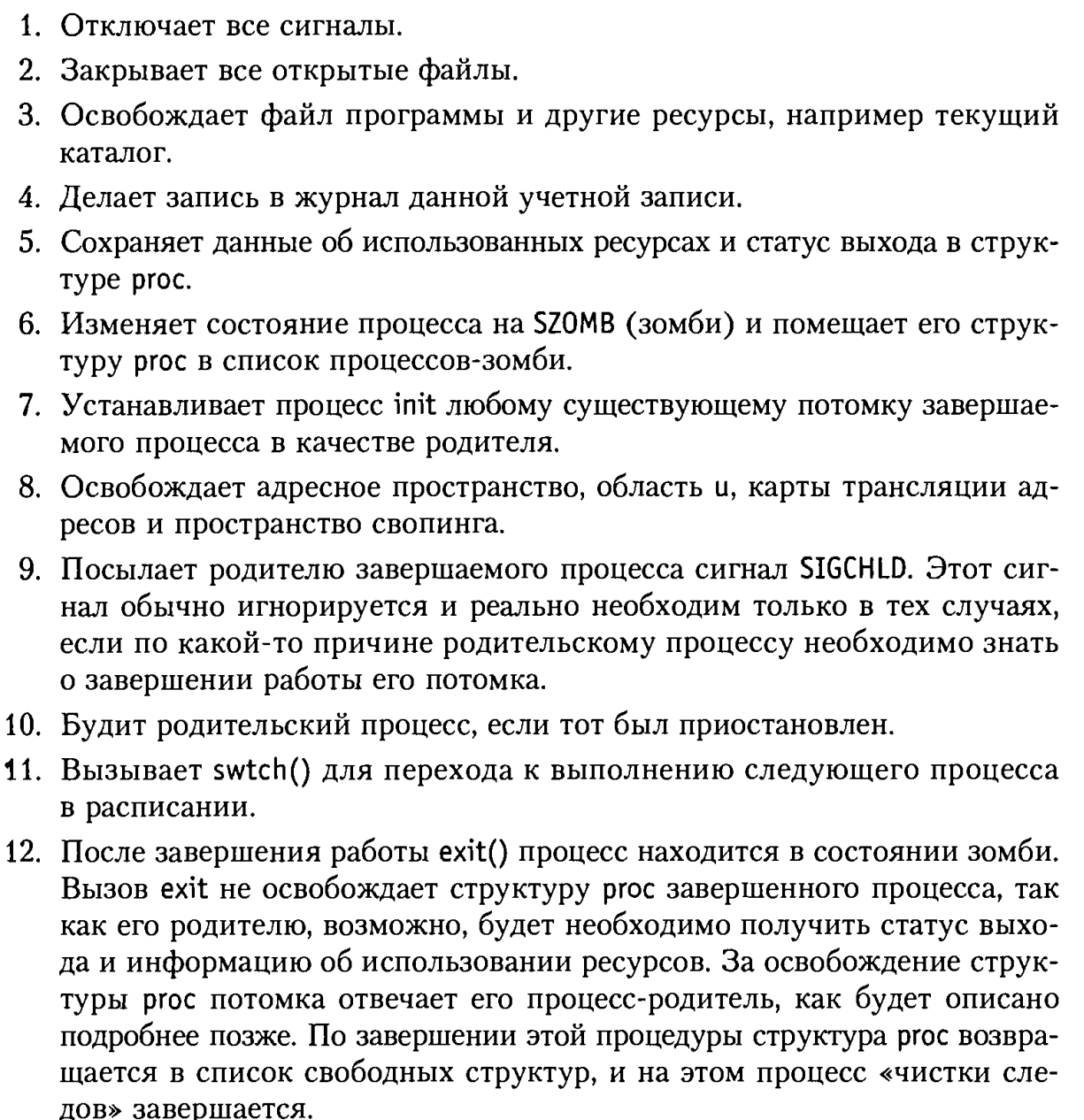
**Функция ядра exit()** предназначена для завершения процесса. Она вызывается изнутри, когда процесс завершается по сигналу.

С другой стороны, программа может выполнить системный вызов exit, который, в свою очередь,

вызовет функцию exit().

Функция exit() производит следующие действия:

**(Устанавливает процесс init любому существующему потомку завершаемого процесса в качестве родителя)**



# 3.1.3. Одновременность и параллельность

**Параллельность** многопроцессорного приложения — это достигнутая им степень параллельного выполнения, которая в любом случае будет ограничиваться количеством процессоров, физически доступных приложению.

Термин **одновременность** описывает максимальную степень параллельности, которую теоретически может достигать приложение с неограниченным количеством доступных процессоров.

стр 109

# 4.2.4. Спящие процессы и сигналы

Система UNIX поддерживает два вида сна: прерываемый и непрерываемый.

## Непрерываемый сон

Процесс, находящийся в состоянии сна до события, которое может произойти в ближайшее время (например, завершение дискового ввода-вывода), пребывает в непрерываемом сне и не может быть побеспокоен поступающими сигналами.

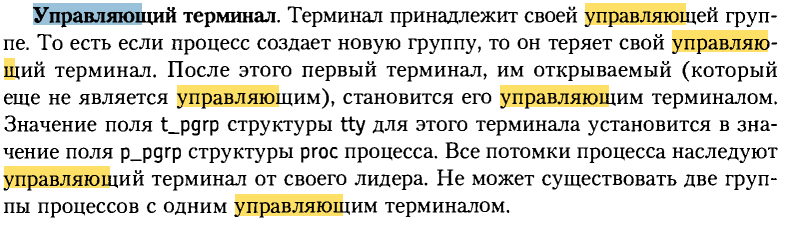
Если сигнал предназначается для процесса, находящегося в непрерываемом сне, то такой сигнал будет помечен как ожидающий, и больше никаких действий со стороны этого сигнала по отношению к процессу происходить не будет. Процесс не будет уведомлен о сигнале и после выхода из состояния сна до тех пор, пока он не окажется в состоянии возврата в режим задачи или не будет блокирован по прерываемому событию.

## Прерываемый сон

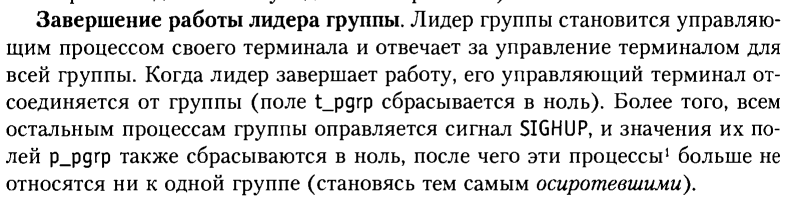
Процесс, ожидающий такого события, как ввод-вывод терминала, который может не произойти в течение продолжительного времени, находится в прерываемом состоянии сна и будет разбужен посланным ему сигналом.

Если процесс находится в состоянии, когда он будет заблокирован по прерываемому событию, то перед этой процедурой он проверит наличие сигналов. Если таковые будут найдены, процесс обработает их и прервет выполнение системного вызова. Если сигнал будет создан уже после того, как процесс был блокирован, ядро системы разбудит такой процесс. Пробуждение и начало работы процесса может быть вызвано двумя причинами: либо произошло ожидаемое событие, либо его сон был прерван сигналом,

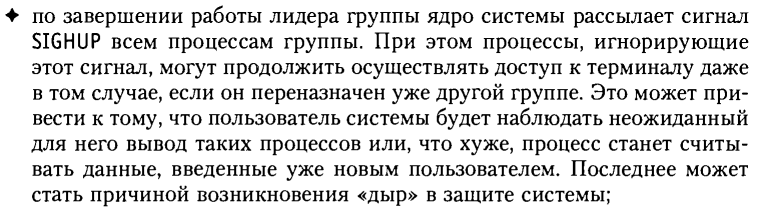
стр 171

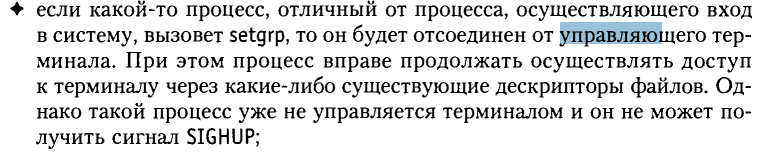


cnh 172

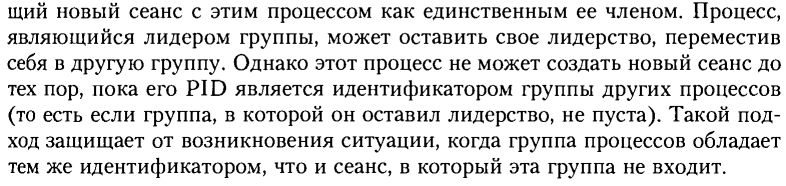


ограничения модели SVR3:





про ограничение setsid стр 179



# 6.3. System V IPC

3 механизма межпроцессного взаимодействия:

* семафоры
* очереди сообщений
* сегмент разделяемой памяти

Каждый ресурс IPC обладает набором атрибутов.

* **Ключ** (key). Поддерживаемое пользователем целое число, идентифицирующее конкретный экземпляр ресурса.
* **Создатель** (creator). Пользовательские и групповые идентификаторы процесса, создавшего ресурс.
* **Владелец** (owner). Пользовательские и групповые идентификаторы владельца ресурса. При создании ресурса его владелец и создатель одинаковы. Однако процесс, обладающими правами изменения владельца, может указать позже нового владельца ресурса. Такими правами обладают процессы создателя, текущего владельца и суперпользователя.
* **Права** (permissions). Права файловой системы на чтение/запись/выполнение для владельца, группы и других пользователей.

# 15.6.1. Идентификаторы и ключи. Раго

Каждой структуре IPC (очереди сообщений, семафору или сегменту разделяемой памяти) в ядре соответствует неотрицательное целое число – идентификатор.

В отличие от дескрипторов файлов, идентификаторы IPC не являются маленькими целыми числами. Каждый раз, когда создается какая либо структура IPC, идентификатор, присваиваемый этой структуре, все время увеличивается, пока не достигнет максимально возможного значения для целых чисел, после чего сбрасывается в ноль.

**Идентификатор** – это **внутреннее имя объекта IPC**. Процессам же необходим **механизм внешних имен**, чтобы организовать **взаимодействие через определенный объект IPC**. Для этого каждому объекту IPC ставится в соответствие ключ, который и выступает в роли внешнего имени.

Всякий раз, когда создается структура IPC (msgget, semget или shmget), обязательно должен быть указан ключ. Тип данных ключа является одним из основных системных типов данных – **key\_t**, который часто определяется в заголовочном файле как длинное целое со знаком. Ядро выполняет преобразование этого ключа во внутренний идентификатор