ОПЕРАЦИОННЫЕ СИСТЕМЫ

Лекция 5

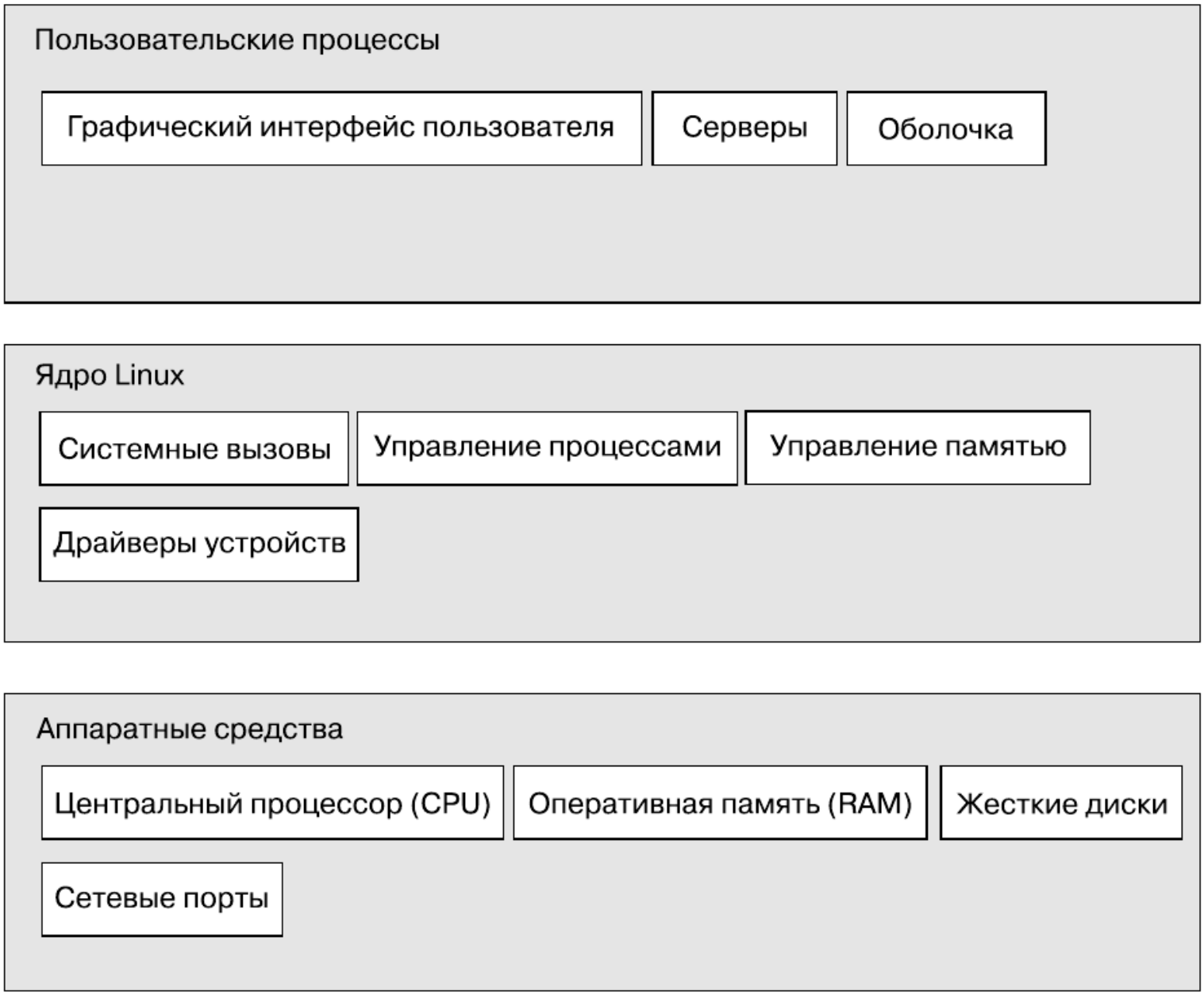
Гурген Аракелов

27 октября 2016 г.

Лаборатория Касперского

LINUX

УРОВНИ АБСТРАКЦИИ LINUX



ЯДРО LINUX


Системные вызовы
Компонент ввода-вывода
Виртуальная файловая система
Терминалы
Дисциплина линии связи
Драйверы символьных устройств
Сокеты
Сетевые протоколы
Драйверы сетевых устройств
Файловые системы
Обобщенный уровень блоков
Планировщик ввода-вывода
Драйверы блочных устройств
Компонент управления памятью
Виртуальная память
Замена страницы подкачки
Кэш страниц
Компонент управления процессами
Обработка сигналов
Создание/ завершение процессор и потоков
Планирование процессора
Прерывания
Диспетчер

LINUX. ФУНКЦИИ ЯДРА.

Пространство ядра и пользовательское пространство.

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе
3. Ядро выбирает новый процесс, который будет работать в следующий квант времени

LINUX. ЯДРО. УПРАВЛЕНИЕ ПРОЦЕССАМИ.

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе
3. Ядро выбирает новый процесс, который будет работать в следующий квант времени
4. Ядро готовит память и подготавливает процессор

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе
3. Ядро выбирает новый процесс, который будет работать в следующий квант времени
4. Ядро готовит память и подготавливает процессор
5. Ядро задает процессору квант времени, который будет работать выбранный процесс

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе
3. Ядро выбирает новый процесс, который будет работать в следующий квант времени
4. Ядро готовит память и подготавливает процессор
5. Ядро задает процессору квант времени, который будет работать выбранный процесс
6. Ядро переводит процесс в режим пользователя и передает управление процессу.

Что происходит при переключении процессов:

1. Процессор прерывает текущий процесс, переключается в режим ядра и передает управление ядру
2. Ядро сохраняет информацию о текущем процессе
3. Ядро выбирает новый процесс, который будет работать в следующий квант времени
4. Ядро готовит память и подготавливает процессор
5. Ядро задает процессору квант времени, который будет работать выбранный процесс
6. Ядро переводит процесс в режим пользователя и передает управление процессу.

LINUX. ЯДРО. УПРАВЛЕНИЕ ДРАЙВЕРАМИ УСТРЙОСТВ.

Драйвера устройств, как правило, доступны только в режиме ядра.

Linux. Ядро. Системные вызовы.

Системные вызовы — выполняют специальные задачи, которые пользовательский процесс не может выполнить сам.

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов
2. Каждому пользовательскому процессу необходима своя область памяти

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов
2. Каждому пользовательскому процессу необходима своя область памяти
3. Процессы не должны иметь доступ к памяти, предназначенной для другого процесса

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов
2. Каждому пользовательскому процессу необходима своя область памяти
3. Процессы не должны иметь доступ к памяти, предназначенной для другого процесса
4. Пользовательские процессы должны иметь возможность совместно использовать память

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов
2. Каждому пользовательскому процессу необходима своя область памяти
3. Процессы не должны иметь доступ к памяти, предназначенной для другого процесса
4. Пользовательские процессы должны иметь возможность совместно использовать память
5. Часть памяти должна быть доступна только для чтения

При работе с памятью необходимо учитывать следующие условия:

1. Ядро должно обладать собственной памятью, недоступной для процессов
2. Каждому пользовательскому процессу необходима своя область памяти
3. Процессы не должны иметь доступ к памяти, предназначенной для другого процесса
4. Пользовательские процессы должны иметь возможность совместно использовать память
5. Часть памяти должна быть доступна только для чтения
6. Система может использовать больше памяти чем есть в наличии, привлекая в качесвте вспомогательного пространства дисковые устройства.

Процессы в Linux

Процессы в Linux

Каждый процесс выполняет одну программу.

Процессы в Linux

Каждый процесс выполняет одну программу.
Изначально, каждый процесс имеет один поток выполнения

Процессы в Linux

Каждый процесс выполняет одну программу.
Изначально, каждый процесс имеет один поток выполнения
Процессы идентифицируются своими **PID**.

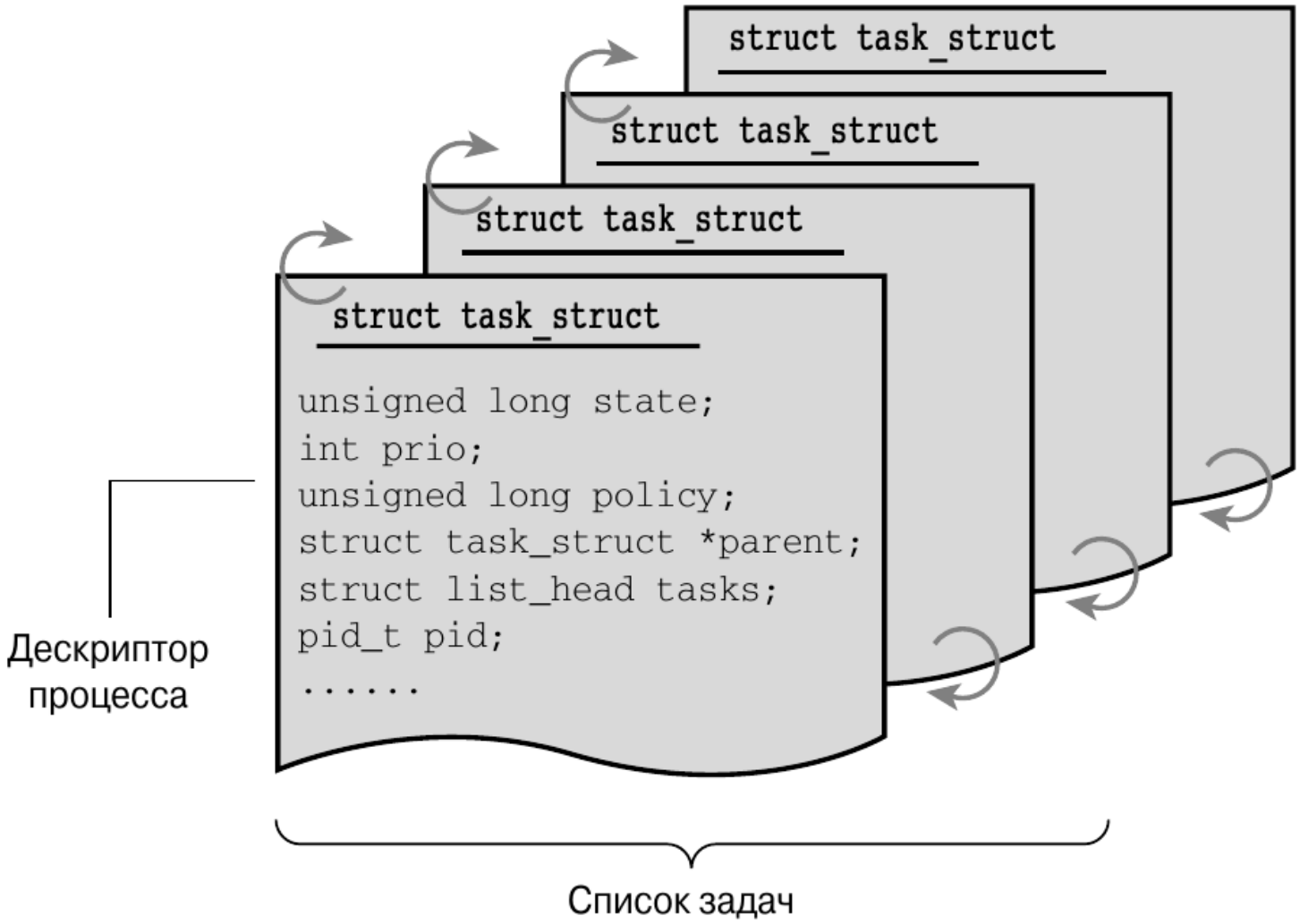
Процессы в Linux

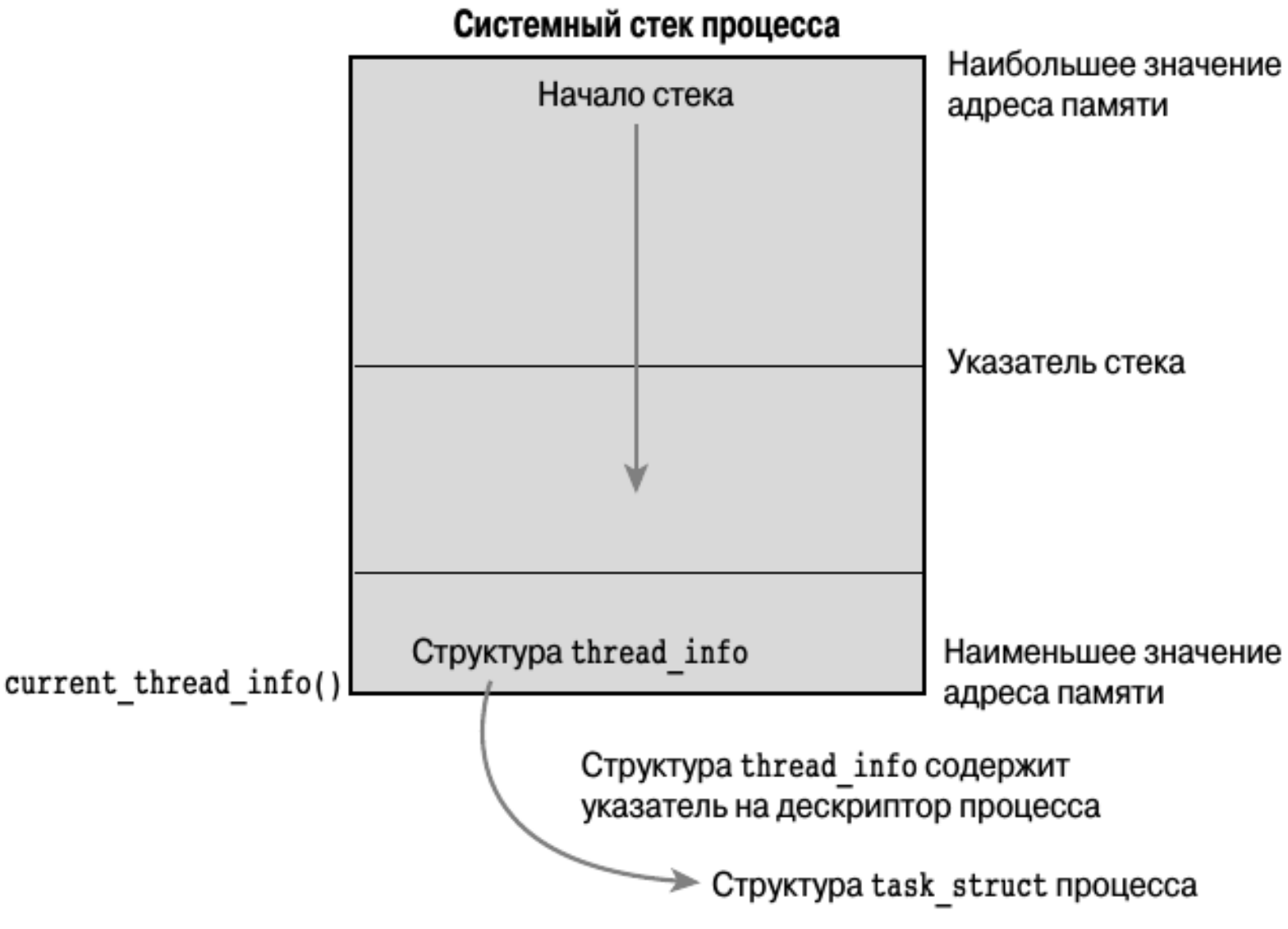
Внутренне Linux представляет любой контекст исполнения как задачу, описываемую структурой

`task_struct`.

Для каждого процесса в памяти всегда находится его дескриптор.
Выделение памяти под дескриптор происходит особым образом.

ПРОЦЕССЫ В LINUX




Системный стек процесса
Начало стека
Наибольшее значение адреса памяти
Указатель стека
Структура thread_info
Наименьшее значение адреса памяти
current_thread_info()
Структура thread_info содержит указатель на дескриптор процесса
Структура task_struct процесса

Процессы в Linux. Реализация

Все процессы организованны в двунаправленный список структур задач.
Существует возможность отображения PID на адрес соответствующей структуры

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры
5. Состояние системного вызова

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры
5. Состояние системного вызова
6. Таблица дескрипторов файлов

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры
5. Состояние системного вызова
6. Таблица дескрипторов файлов
7. Учетные данные

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры
5. Состояние системного вызова
6. Таблица дескрипторов файлов
7. Учетные данные
8. Стек ядра

Процессы в Linux. Реализация

Категории информации в дескрипторе процесса:

1. Параметры планирования
2. Образ памяти
3. Сигналы
4. Машинные регистры
5. Состояние системного вызова
6. Таблица дескрипторов файлов
7. Учетные данные
8. Стек ядра
9. Разное

Процессы в Linux. Состояния

В Linux каждый процесс гарантированно находится в одном из 5 состояний:

1. TASK_RUNNING

Процессы в Linux. Состояния

В Linux каждый процесс гарантированно находится в одном из 5 состояний:

1. TASK_RUNNING
2. TASK_INTERRUPTIBLE

Процессы в Linux. Состояния

В Linux каждый процесс гарантированно находится в одном из 5 состояний:

1. TASK_RUNNING
2. TASK_INTERRUPTIBLE
3. TASK_UNINTERRUPTIBLE

Процессы в Linux. Состояния

В Linux каждый процесс гарантированно находится в одном из 5 состояний:

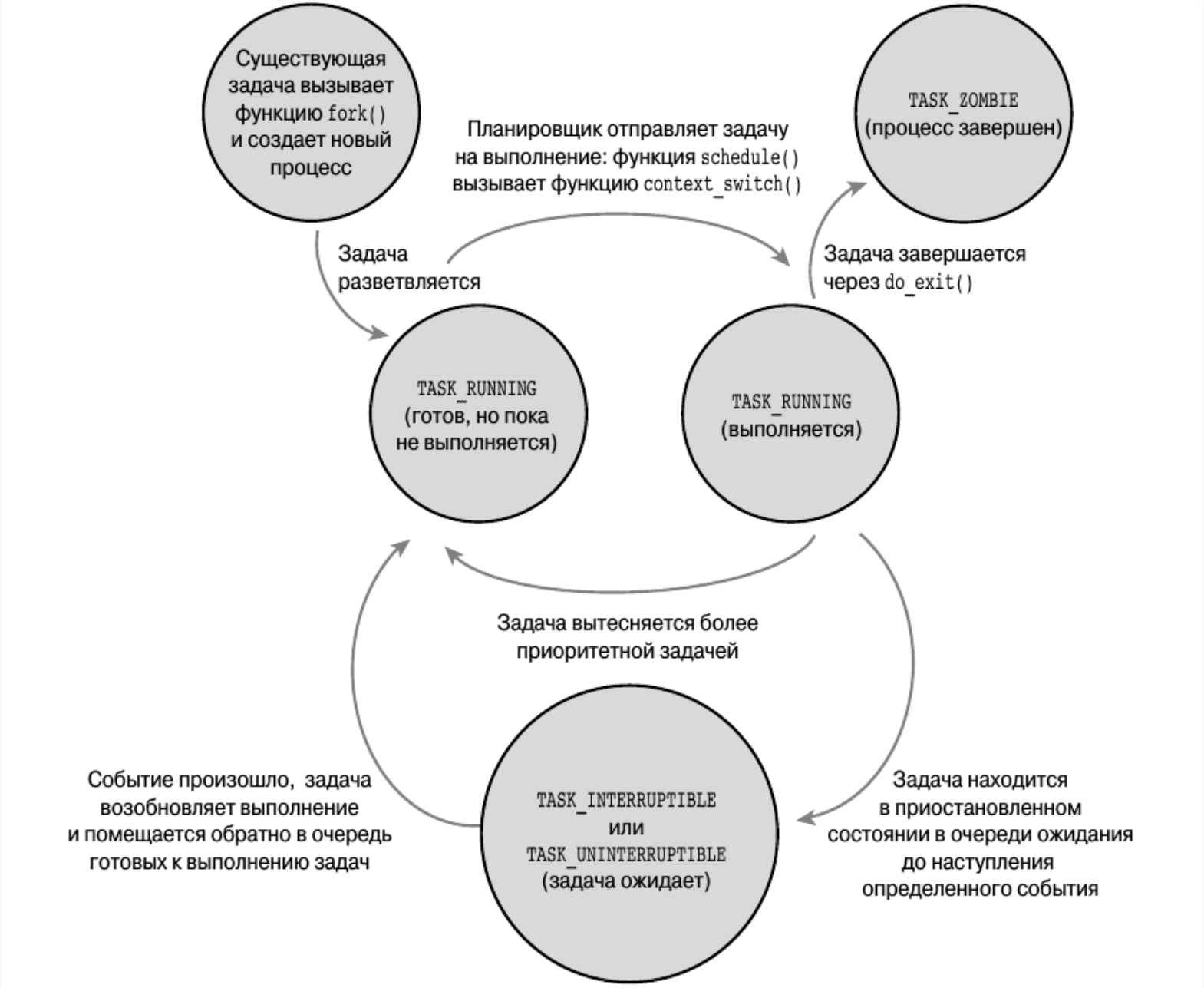
1. TASK_RUNNING
2. TASK_INTERRUPTIBLE
3. TASK_UNINTERRUPTIBLE
4. TASK_TRACED

Процессы в Linux. Состояния

В Linux каждый процесс гарантированно находится в одном из 5 состояний:

1. TASK_RUNNING
2. TASK_INTERRUPTIBLE
3. TASK_UNINTERRUPTIBLE
4. TASK_TRACED
5. TASK_STOPPED

ПРОЦЕССЫ В LINUX



Процессы в Linux

Работа ядра от имени процесса в контексте процесса.

Процессы в Linux

Иерархия процессов, процесс **init**, файлы сценариев начальной загрузки.

Процессы в Linux

init_task

Процессы в Linux. Создание нового процесса.

`fork()`, `clone()`, `do_fork()(kernel/fork.c)`, `copy_process()`.

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского
4. Дочерний процесс переводится в состояние TASK_UNINTERRUPTIBLE

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского
4. Дочерний процесс переводится в состояние TASK_UNINTERRUPTIBLE
5. Вызывается copy_flags(). (PF_SUPERPRIV,PF_FORKNOEXEC).

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского
4. Дочерний процесс переводится в состояние TASK_UNINTERRUPTIBLE
5. Вызывается copy_flags(). (PF_SUPERPRIV,PF_FORKNOEXEC).
6. alloc_pid()

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского
4. Дочерний процесс переводится в состояние TASK_UNINTERRUPTIBLE
5. Вызывается copy_flags(). (PF_SUPERPRIV,PF_FORKNOEXEC).
6. alloc_pid()
7. Настройка общих ресурсов

Процессы в Linux. Создание нового процесса.

Что происходит внутри copy_process():

1. Вызывается dup_task_struct(), которая создает новый стек ядра, структуры thread_info и task_struct.
2. Проверяется, не будет ли исчерпан лимит на количество процессов для данного пользователя.
3. Дочерний процесс делается отличным от родительского
4. Дочерний процесс переводится в состояние TASK_UNINTERRUPTIBLE
5. Вызывается copy_flags(). (PF_SUPERPRIV,PF_FORKNOEXEC).
6. alloc_pid()
7. Настройка общих ресурсов

Дочерний процесс преднамеренно запускается на выполнение раньше родительского. Зачем?

ПРОЦЕССЫ В LINUX. РЕАЛИЗАЦИЯ

```
1 #include <unistd.h>
2
3 int main(int argc, char* argv[])
4 {
5   fork();
6   return 0;
7 }
8
```

Процессы в Linux. Способы общения

Основные средства общения процессво в Linux:

Процессы в Linux. Способы общения

Основные средства общения процессво в Linux:

1. Трубы (pipes)
   `ls | grep`
2. Сигналы (Посылаются только своей группе)

LINUX. ПОЛЬЗОВАТЕЛИ

Пользователь — это сущность, которая может запускать процесс и обладать файлами.
У каждого процесса из пространства пользователя существует пользователь–**владелец**.

Вопросы?