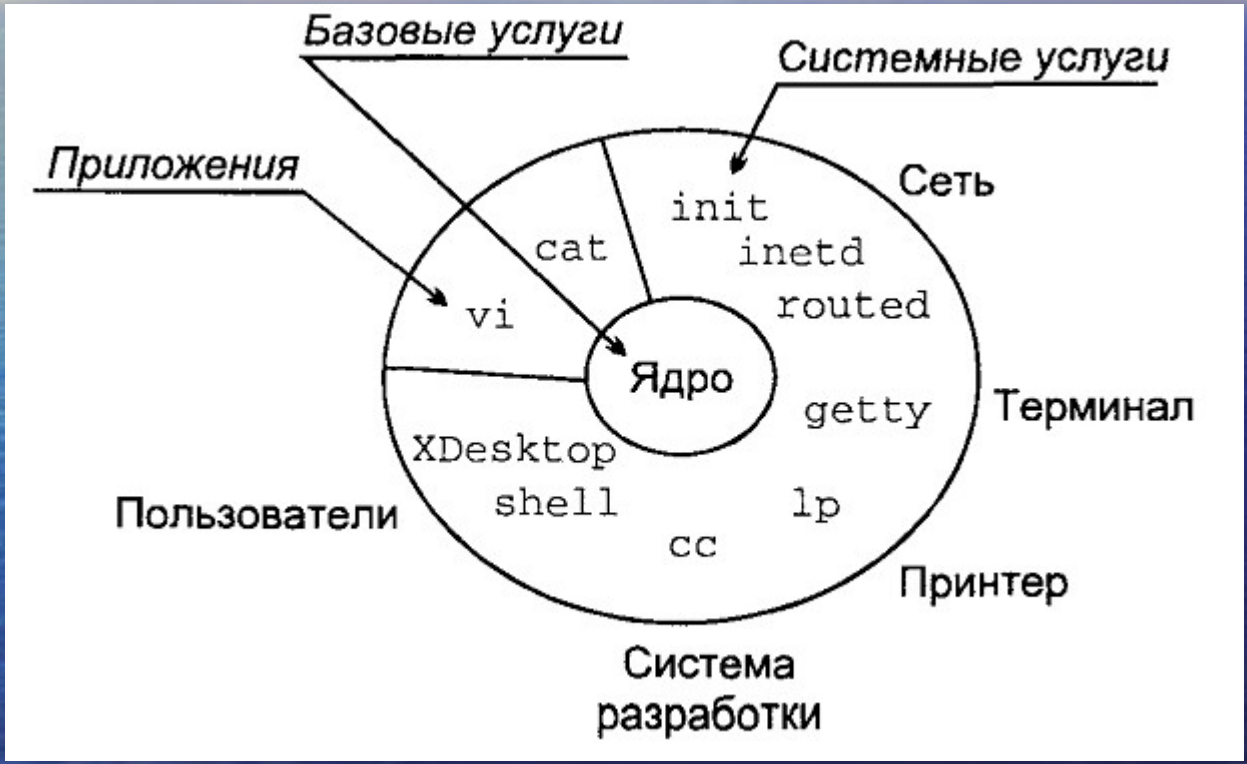
Введение в архитектуру UNIX

Причины популярности UNIX

- Код написан на Си
- Многозадачная, многопользовательская с широким спектром услуг
- Наличие стандартов
- Мощный модульный пользовательский интерфейс
- Иерархическая файловая система
- Большое количество свободно распространяемых приложений

Архитектура UNIX

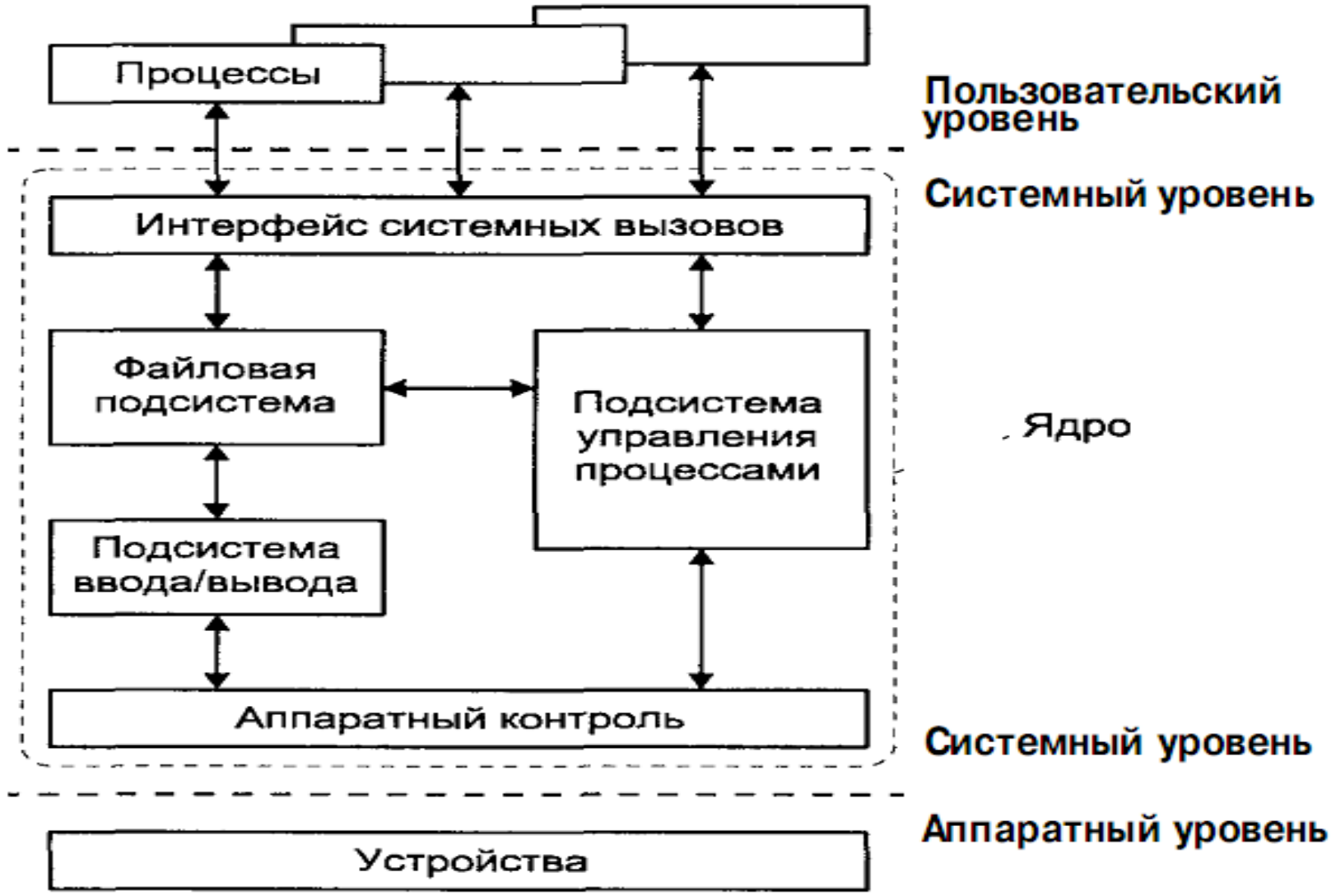


Ядро системы

обеспечивает базовую функциональность:

управление процессами, распределение памяти, доступ к файлам и периферийным устройствам

Структура ядра



Основные подсистемы ядра

- Файловая подсистема
- Подсистема управления процессами и памятью
- Подсистема ввода/вывода

Файловая подсистема

- Поддержка унифицированного интерфейса к обычным файлам и периферийным устройствам
- Проверка прав доступа

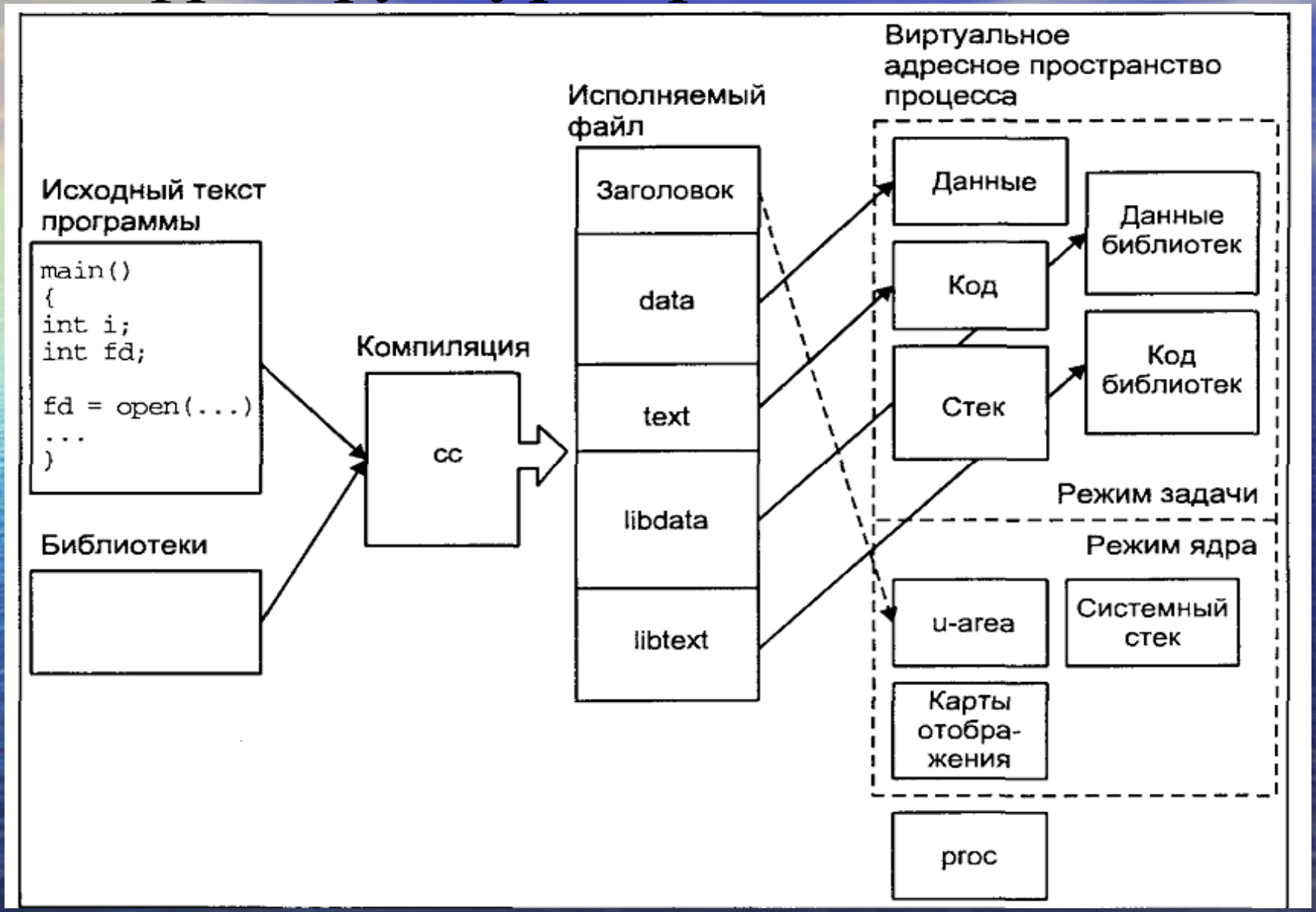
Подсистема управления процессами

- Создание и удаление процессов
- Распределение системных ресурсов
- Синхронизация процессов
- Межпроцессное взаимодействие

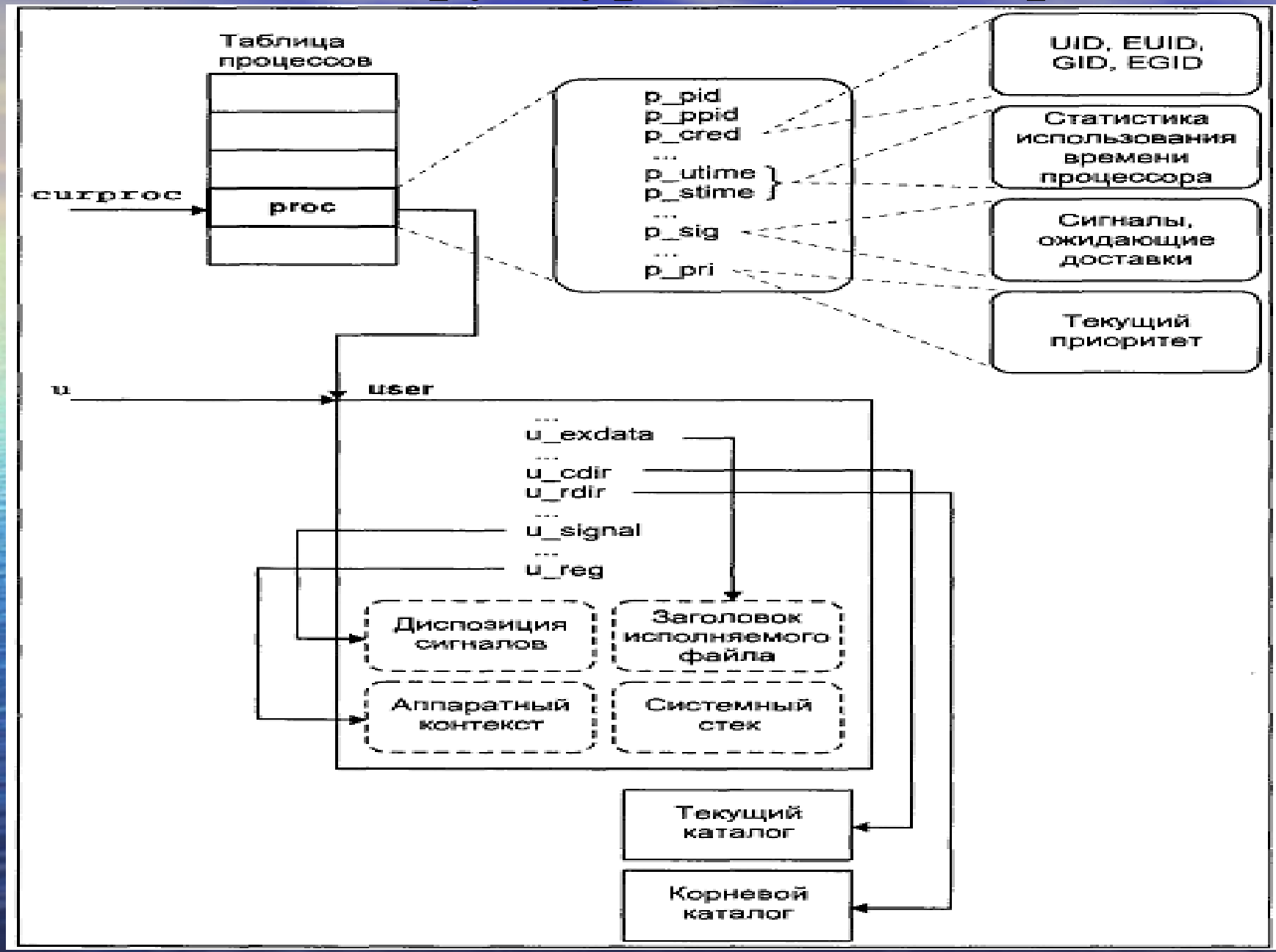
Подсистема ввода/вывода

- Обеспечение работы с периферийными устройствами
- Буферизация данных
- Взаимодействие с драйверами

Инфраструктура процесса ОС UNIX



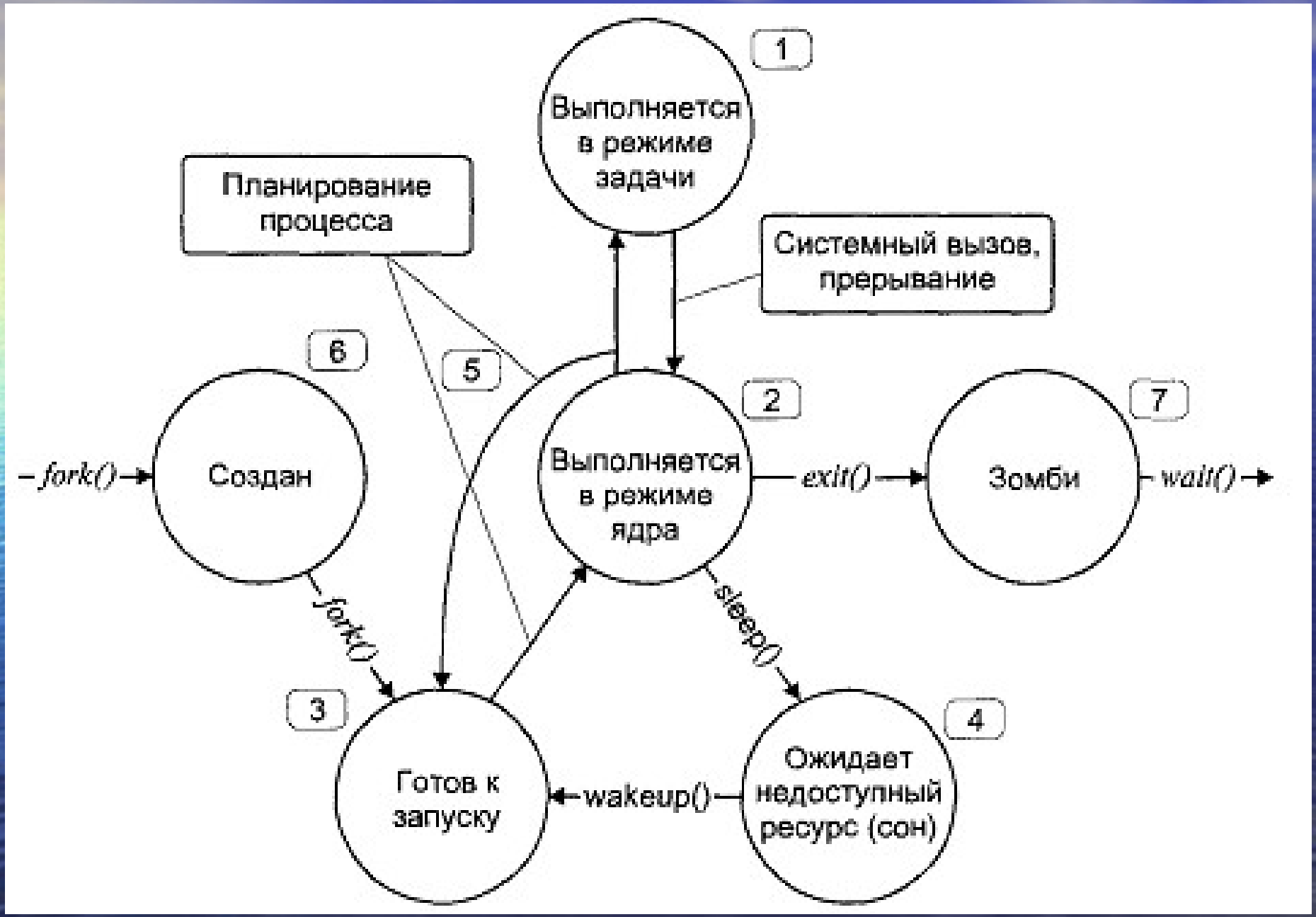
Основные структуры данных процесса



Структура proc

char	p_stat	Состояние процесса (выполнение, приостановлен, сон и т. д.)
char	p_pri	Текущий приоритет процесса
unsigned int	p_flag	Флаги, определяющие дополнительную информацию о состоянии процесса
unsigned short	p_uid	UID процесса
unsigned short	p_suid	EUID процесса
int	p_sid	Идентификатор сеанса
short	p_pgrp	Идентификатор группы процессов (равен идентификатору лидера группы)
short	p_pid	Идентификатор процесса (PID)
short	p_ppid	Идентификатор родительского процесса (PPID)
sigset_t	p_sig	Сигналы, ожидающие доставки
unsigned int	p_size	Размер адресного пространства процесса в страницах
time_t	p_utime	Время выполнения в режиме задачи
time_t	p_stime	Время выполнения в режиме ядра
caddr_t	p_ldt	Указатель на LDT процесса
struct pregion	*p_region	Список областей памяти процесса
short	p_xstat	Код возврата, передаваемый родительскому процессу
unsigned int	p_utbl[ ]	Массив записей таблицы страниц для u-area

Граф состояний процесса



Состояния процесса (1)

- Режим задачи. Выполнение прикладных инструкций процесса
- Режим ядра. Выполнение системных инструкций от имени процесса
- Готов к запуску. В очереди на выполнение

Состояния процесса (2)

- Сон (ожидание недоступного ресурса)
- При переходе из режима ядра в режим задачи может произойти переключение контекста
- Создан (fork)
- Зомби (exit или по сигналу)

Контекст процесса

- АП в режиме задачи
- Управляющая информация
- Окружение процесса
- Аппаратный контекст

Переключение контекста

- Текущий процесс переходит в состояние сна, ожидая недоступного ресурса
- Текущий процесс завершает свое выполнение
- После пересчета приоритетов в очереди на выполнение находится более высокоприоритетный процесс
- Происходит пробуждение более высокоприоритетного процесса

Прерывание от таймера (1)

- Обновление статистики использования процессора для текущего процесса
- Выполнение ряда функций, связанных с планированием процессов
- Проверка превышения процессорной квоты для данного процесса
- Обновление системного времени и других, связанных с ним таймеров

Прерывание от таймера (2)

- Обработка отложенных вызовов
- Обработка алармов
- Пробуждение системных процессов (своппер, диспетчер страниц)

Нотация главного тика

Планирование процессов

- Системы пакетной обработки данных
- Интерактивные системы (системы разделения времени)
- Системы реального времени

Системы пакетной обработки данных

- Пропускная способность – максимальной количество задач в единицу времени
- Оборотное время – минимизация времени, затрачиваемого на ожидание обслуживания и обработку задачи
- Использование процессора – поддержка постоянной занятости процессора

Системы разделения времени

- Время отклика – быстрая реакция на запросы
- Соразмерность – выполнение пожеланий пользователя

Системы реального времени

- Окончание работы к сроку – предотвращение потери данных
- Предсказуемость – предотвращение деградации качества в мультимедийных системах

Классы приложений

- Интерактивные
- Фоновые
- Реального времени

Принципы управления памятью

- Примитивное управление памятью (специализированные микропроцессорные системы)
- Расширенное управление (чаще всего виртуальная память)

Примитивное управление

- Нет защиты программ друг от друга и от ОС
- Заранее на этапе компиляции надо знать физические адреса
- Объем физической памяти будет ограничивать число процессов

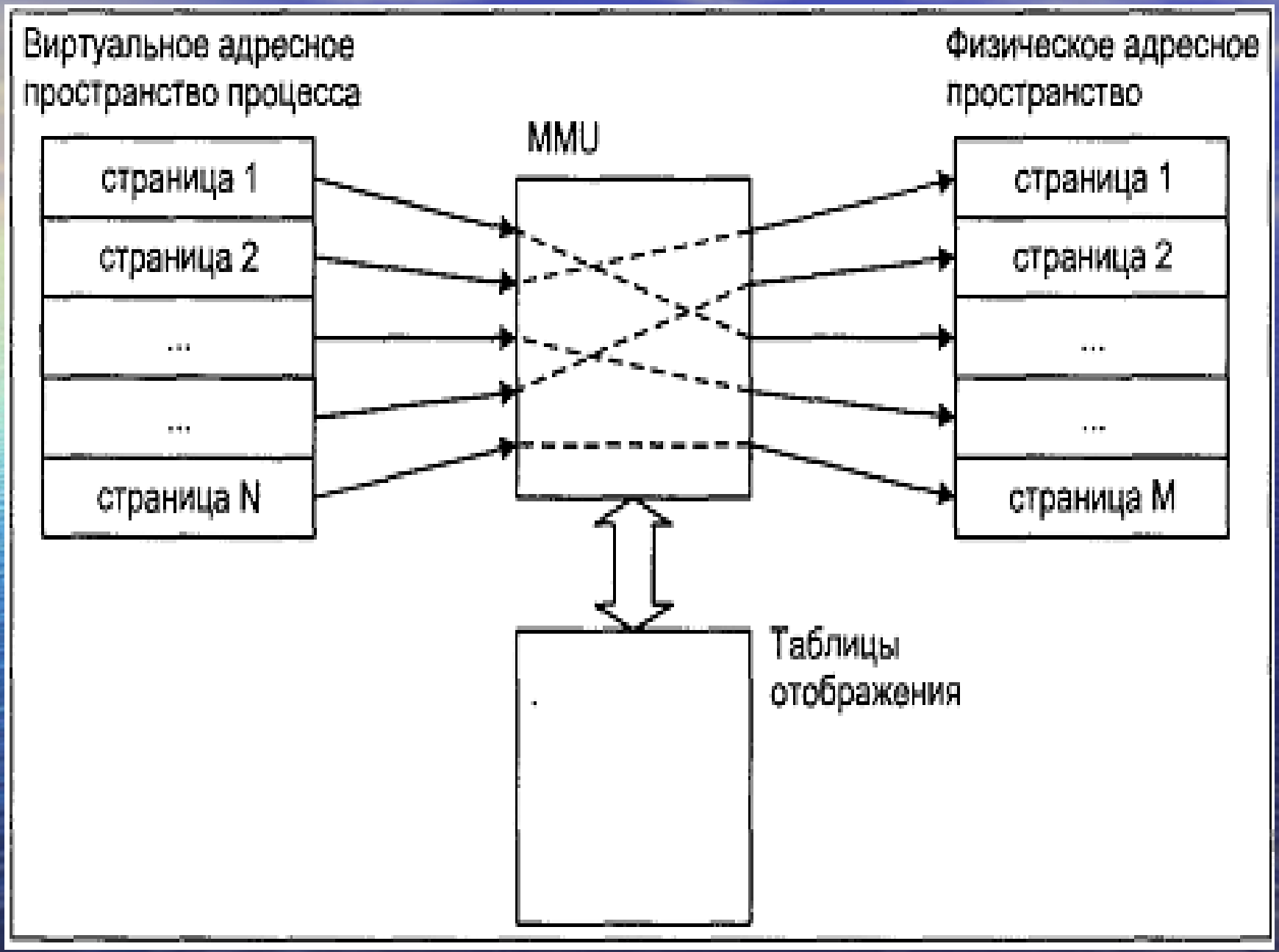
Виртуальная память (1)

- Выполнение задач, размер которых превышает размер ОП
- Выполнение частично загруженных в память задач. Ускорение времени их запуска
- Размещение нескольких задач одновременно в памяти

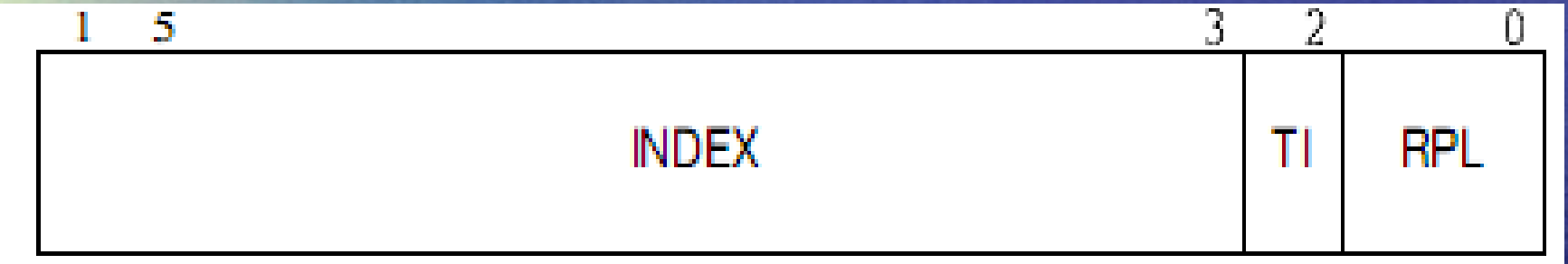
Виртуальная память (2)

- Размещение задач в произвольном месте ОП
- Размещение задачи в нескольких различных частях ОП
- Совместное использование областей памяти (сегмент кода)

Виртуальная и физическая память



Селектор сегмента



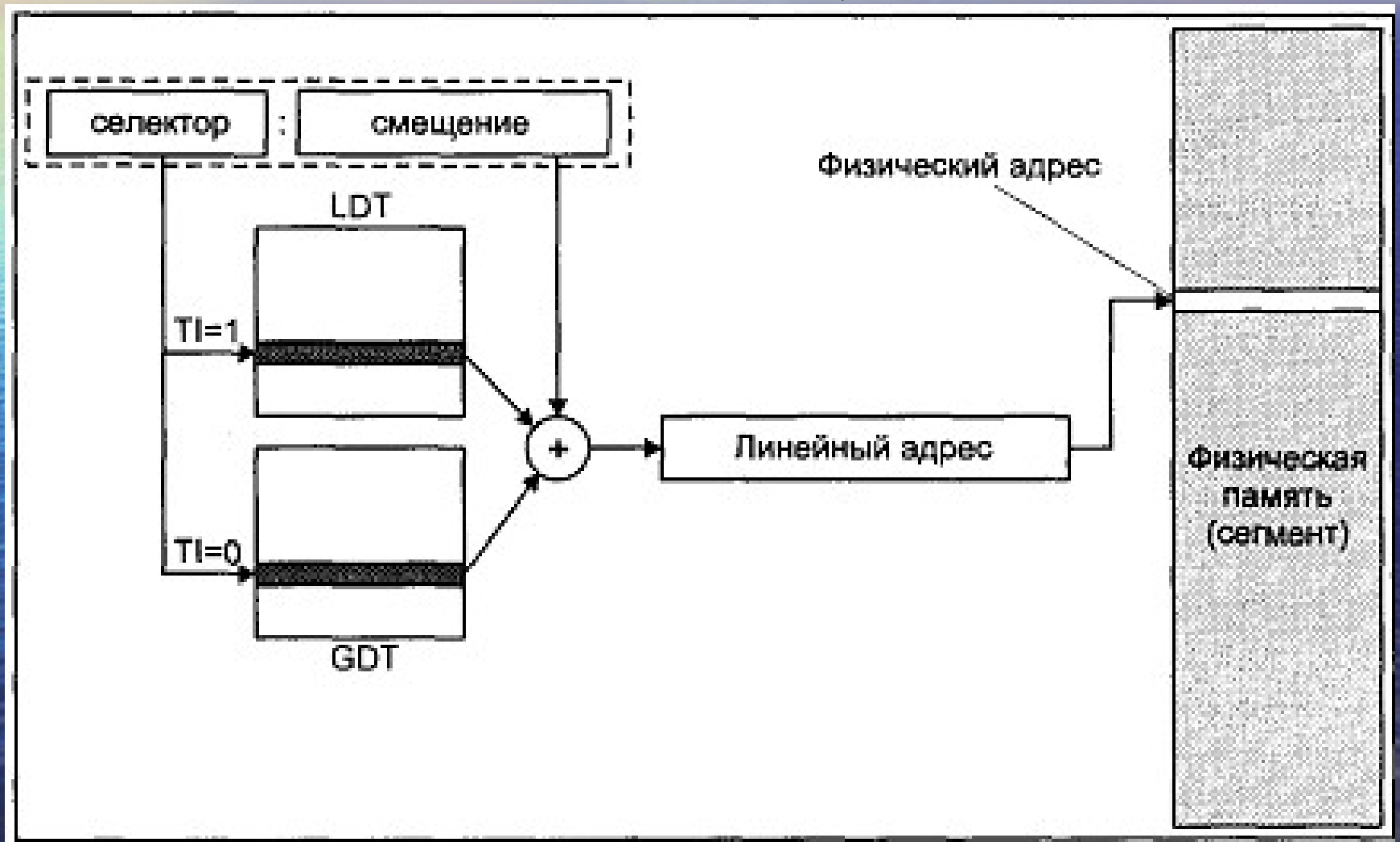
Дескриптор сегмента (1)

Базовый адрес	В этом поле хранится 32-битный адрес начала сегмента. Процессор добавляет к нему смещение и получает 32-битный линейный адрес.
Предел	Это поле определяет размер сегмента. Если результирующий линейный адрес выходит за пределы сегмента, процессор генерирует особую ситуацию. Границы сегмента позволяют процессору обнаруживать такие распространенные ошибки, как переполнение стека, неверные указатели, неверные адреса вызовов и переходов. В случае, когда операционная система считает, что обращение за пределы сегмента не является ошибкой (например, при переполнении стека), она может расширить сегмент путем выделения дополнительной памяти и запросить выполнение команды вновь.

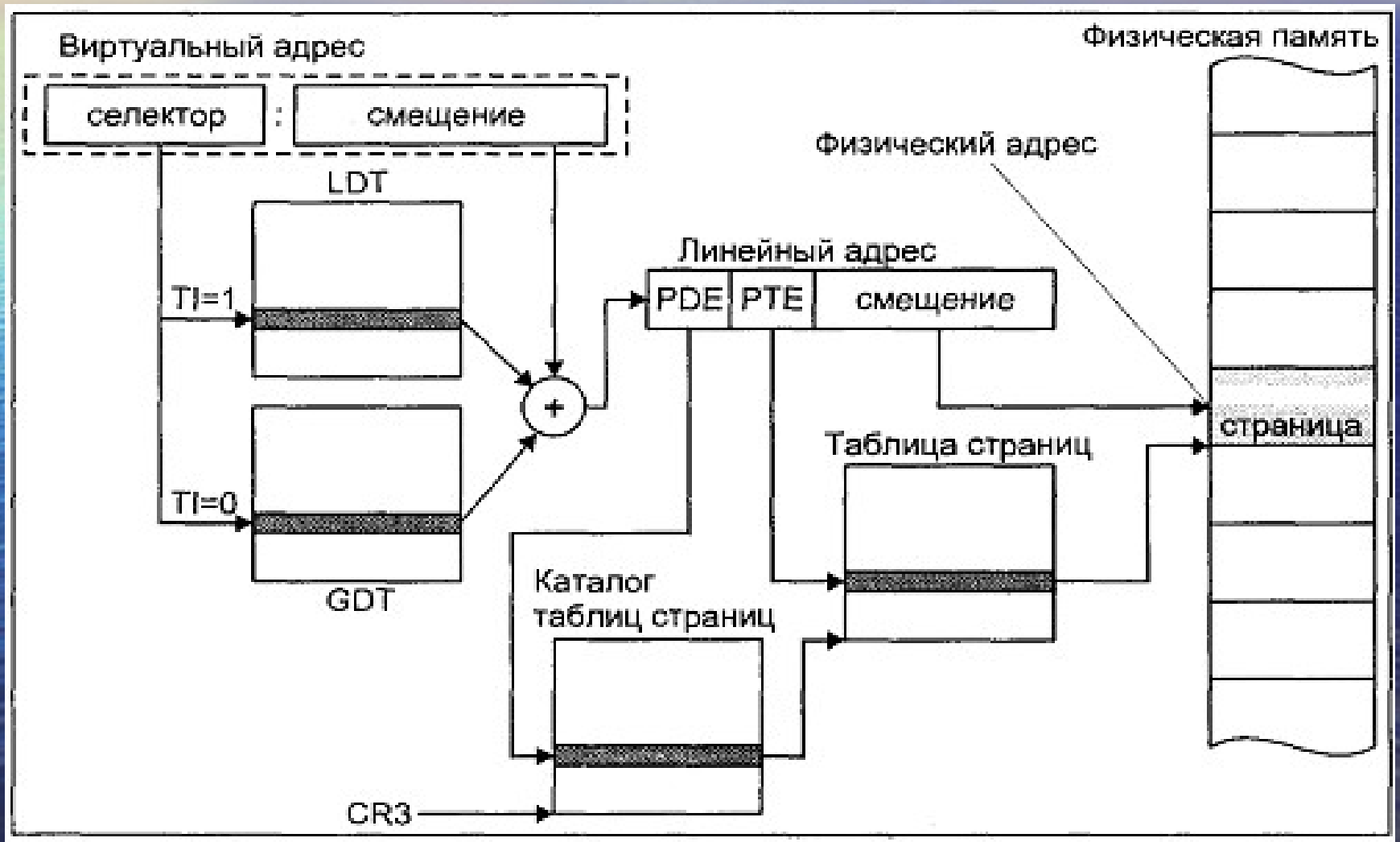
Дескриптор сегмента (2)

Привилегии	Это поле, имеющее название Descriptor Privilege Level (DPL), определяет уровень привилегий сегмента и используется совместно с полем RPL селектора для разрешения или запрещения доступа к сегменту. Для получения доступа к сегменту задача должна иметь по крайней мере такой же уровень привилегий, как и сегмент, т. е. RPL > DPL.
Признак присутствия	Этот бит обеспечивает один из механизмов реализации виртуальной памяти. Если бит не установлен, при попытке обращения к сегменту процессор генерирует особую ситуацию отсутствия сегмента, позволяя ядру подгрузить сегмент из вторичной памяти и вновь повторить инструкцию, не затрагивая при этом выполнение процесса. Однако в большинстве современных версий UNIX виртуальная память основана на страничном механизме, при котором сегмент всегда присутствует в памяти, а обмен между оперативной и вторичной памятью происходит на уровне страниц.
Тип	Это поле определяет тип сегмента. Процессор проверяет тип сегмента на соответствие исполняемой команде. Это, в частности, не позволяет интерпретировать информацию сегмента данных как инструкции процессора.
Права доступа	Это поле определяет права доступа, ограничивающие множество операций, которые можно производить с сегментом. Например, сегмент кода обычно отмечается как исполняемый и читаемый. Сегменты данных могут иметь право доступа только для чтения, или для чтения и записи.

Трансляция адреса с использованием механизма сегментации



Трансляция адреса с использованием страничного механизма



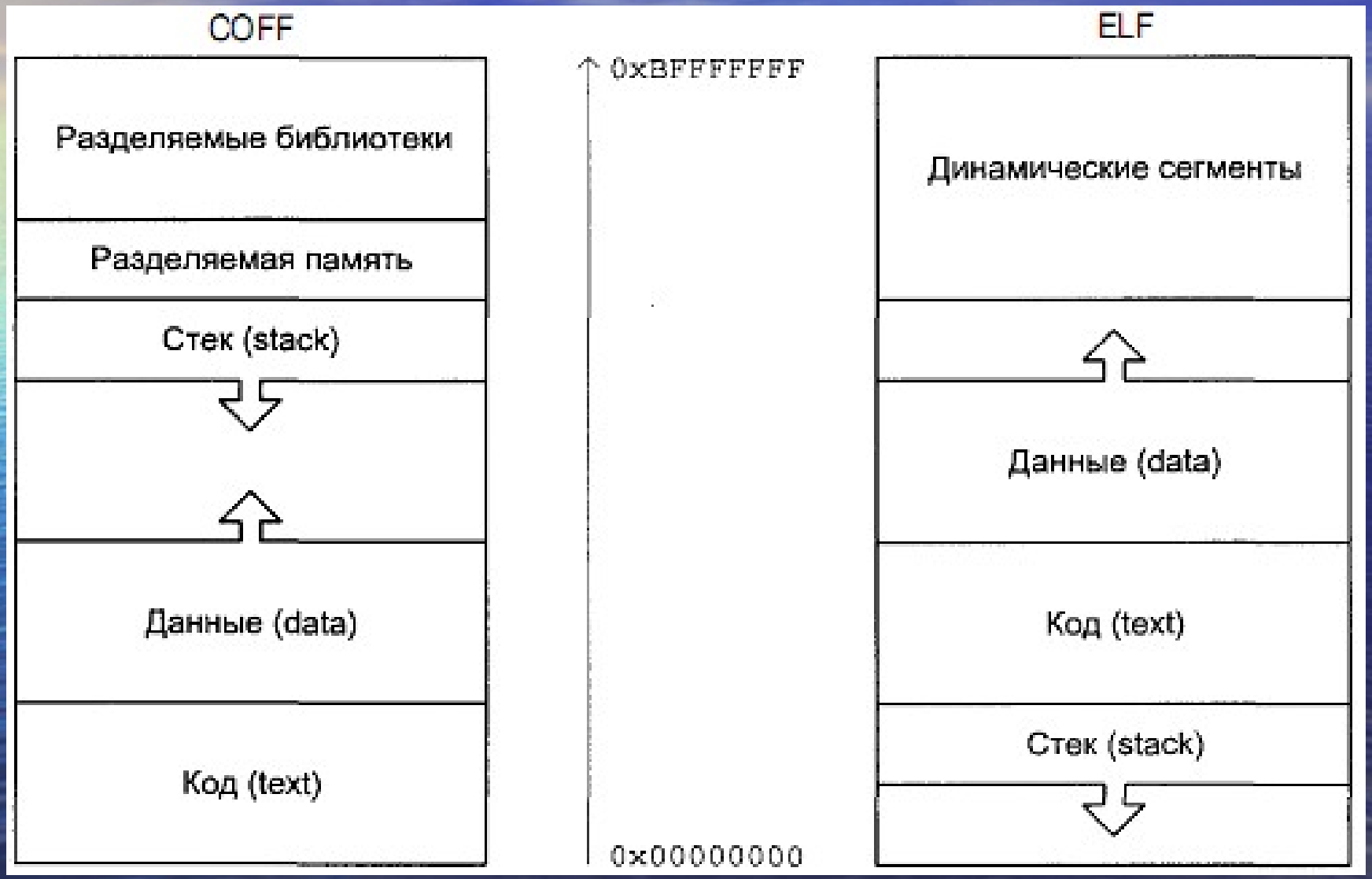
Адресное пространство процесса

32-разрядный линейный адрес (4Г)

Старший (1Г) АП ядра – 256 элементов каталога страниц

Младшие (3Г) АП задачи – 768 элементов каталога страниц

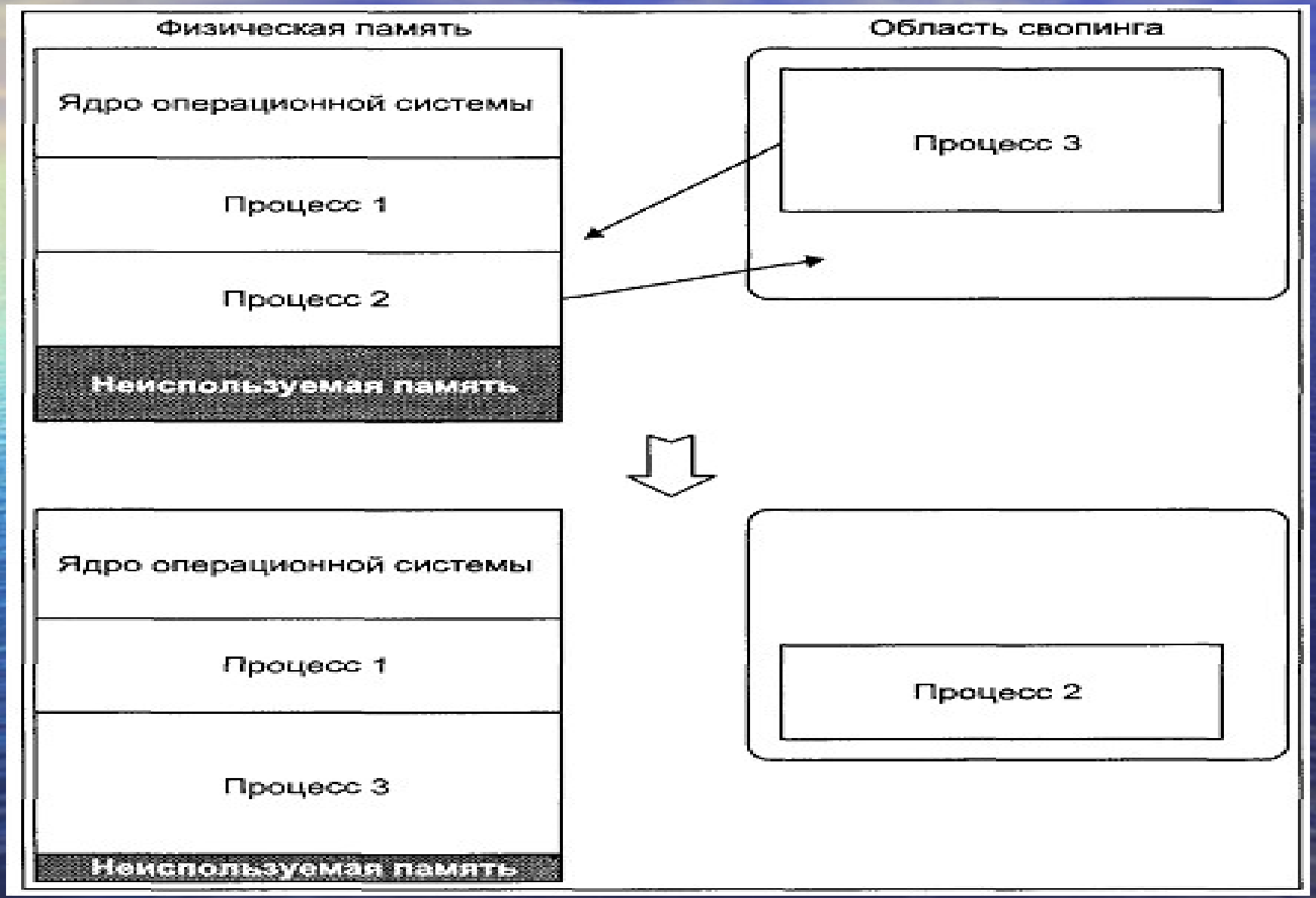
Виртуальная память процесса в режиме задачи



Страничный механизм в основном реализуется за счет аппаратной поддержки, но ОС отвечает за:

- Размещение в памяти каталога таблиц страниц и таблиц страниц
- Установка отображения путем записи соответствующих значений в таблицы страниц
- Обработка страничных ошибок
- Управление сверхоперативным кэшем
- Обеспечение обмена между ОП и ВП

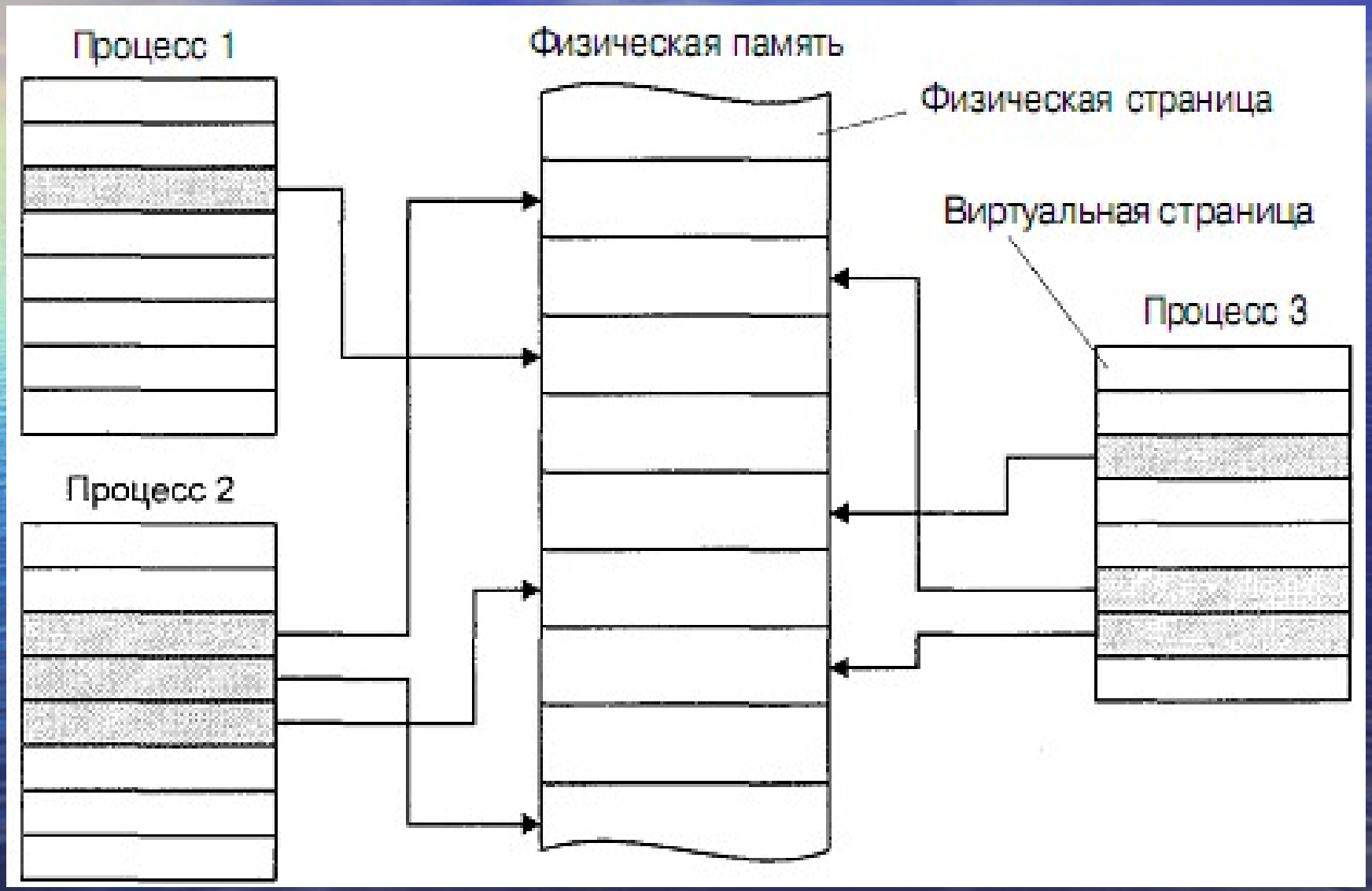
Ранние версии UNIX (свопинг)



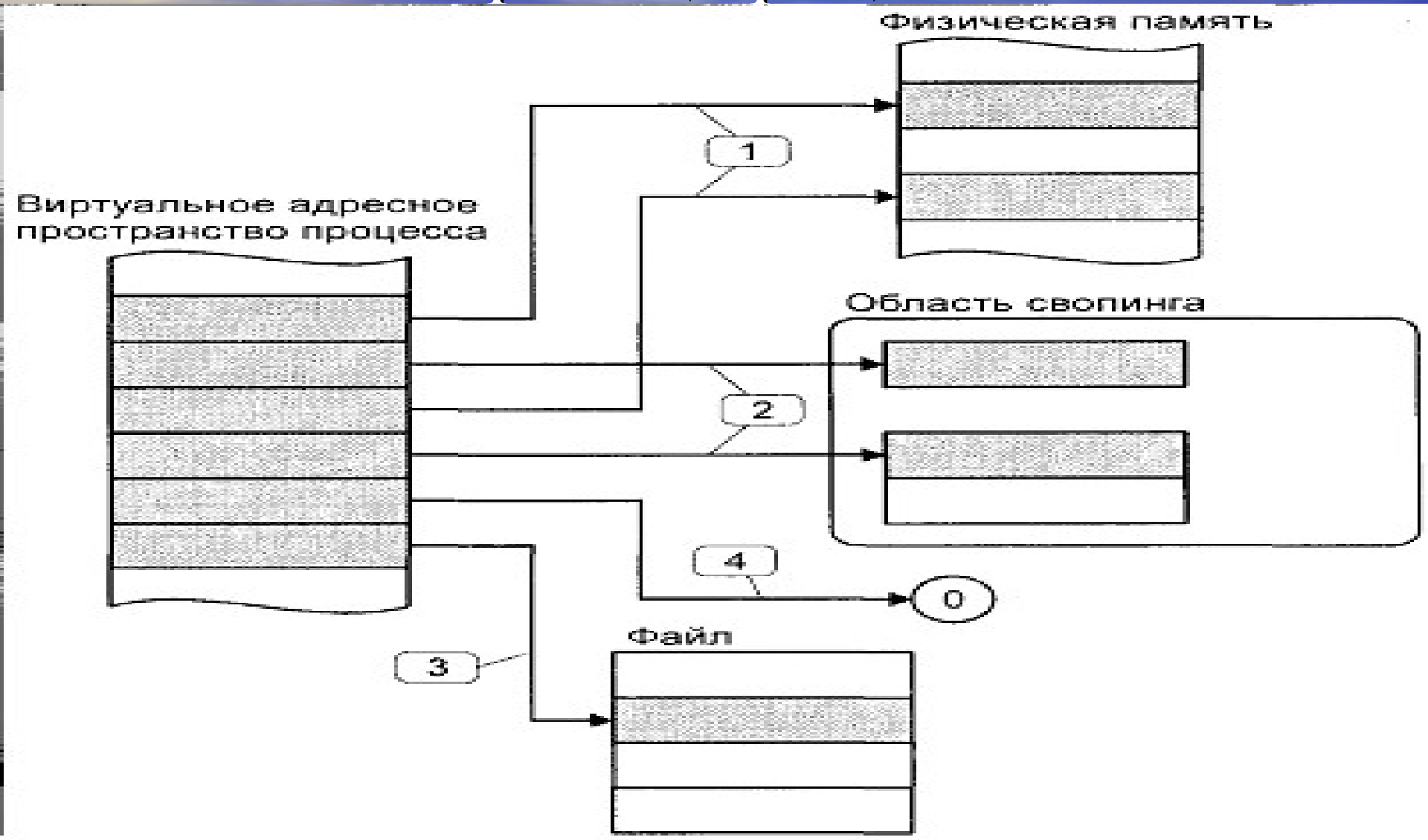
Принципы страничного замещения

- Принцип загрузки (fetch policy)
- Принцип размещения (placement policy)
- Принцип замещения (replacement policy)

Страничное замещение по требованию



Возможное местонахождение страниц процесса



Преимущества страничного замещения

- Размер программы ограничивается только разрядностью адреса
- Запуск программы происходит очень быстро
- Большее число программ может быть загружено и выполняться одновременно
- Перемещение отдельных страниц между ОП и ВП требует меньших затрат

Алгоритмы замещения страниц (1)

- Оптимальный алгоритм (практически неосуществим)
- NRU (Not Recently Used) не использовавшаяся в последнее время страница
- FIFO первым прибыл, первым обслужен
- Вторая попытка (усовершенствованный FIFO)

Алгоритмы замещения страниц (2)

- Часы (другая реализация алгоритма «Вторая попытка»)
- LRU (Least Recently Used) страница, не использовавшееся больше всего
- NFU (Not Frequently Used) редко использовавшаяся страница
- Старение
- Рабочий набор
- WSClock

Оптимальный алгоритм

Выгрузить страницу, которая не будет использована больше всего.

Можно реализовать только для конкретной программы с заранее заданным входным набором данных

NRU (1)

Используется 2 бита:

R (Referenced) обращение

M (Modified) изменение

NRU (2)

4 класса страниц:

1. Не было обращений и изменений
2. Не было обращений, страница изменена
3. Было обращение, страница не изменена
4. Произошло и обращение и изменение

NRU (3)

Удаление страницы в непустом классе с наименьшим номером

FIFO

Ведется список всех страниц, находящихся в данный момент в памяти.

При страничном прерывании выгружается страница из головы списка, а новая добавляется в хвост списка.

Вторая попытка

Модифицированный алгоритм FIFO.

При попытке удаления у страницы проверяется бит R. Если он равен 0, то страница удаляется, если – нет, то бит сбрасывается и страница помещается в начало списка как только что загруженная.

Часы

Модифицированный алгоритм «Вторая попытка».

Список страниц является кольцевым, что снижает затраты на перемещение страниц из головы в хвост списка.

LRU

Выгружается из памяти страница, не использовавшаяся больше всего.

Сложно реализовать алгоритм, так как необходимо после каждого обращения к памяти перестраивать список страниц.

Существуют различные аппаратные реализации

NFU

Для каждой страницы ведется счетчик. После каждого прерывания от таймера к счетчику прибавляется значение бита R.

Для выгрузки выбирается страница с наименьшим значением счетчика.

Алгоритм «ничего не забывает» в пределах каждого процесса

Старение

Модификация алгоритма NFU.

Каждый счетчик перед прибавлением R сдвигается вправо на один разряд. Прибавление осуществляется в крайний левый бит. Практика показывает, что 8 бит достаточно

Рабочий набор

Все программы характеризуются локальностью обращений к памяти.

Можно использовать опережающую подкачку страниц из «рабочего набора» текущего процесса.

Трудность алгоритма определения «рабочего набора»

WSClock

Модификация алгоритма «рабочего набора». Используются «часы» в виде кольцевого списка. С каждой странице связано время ее загрузки с диска

Лучшими алгоритмами являются:

старение и WSClock

Создание процесса
Системный вызов fork()

Создается точная копия родительского процесса за некоторыми отличиями

Отличия родительского и дочернего процессов (1)

- Уникальный идентификатор PID
- Отличается идентификатор родителя PPID
- Дочерний процесс получает собственную копию u-area (файловые дескрипторы, но записи разделяются)

Отличия родительского и дочернего процессов (2)

- Очищаются ожидающие доставки сигналы
- Обнуляется временная статистика (время выполнения в режиме ядра и режиме задачи)
- Блокировки памяти и записей не наследуются

Отличия родительского и дочернего процессов (3)

- Возвращаемое значение fork()

  родительский процесс – PID потомка

  дочерний процесс – 0

Действия, выполняемые fork() (1)

- Резервирует место в области свопинга для сегмента данных и стека
- Размещает и инициализирует новую запись в таблице процессов и присваивает PID

Действия, выполняемые fork() (2)

- Размещает карты отображения, необходимые для трансляции адреса
- Размещает u-area и копирует ее содержимое с родительского процесса

Действия, выполняемые fork() (3)

- Инициализирует аппаратный контекст, копируя его с родительского процесса
- Устанавливает возвращаемое значение fork()
- Состояние – готов к запуску

Запуск новой программы Системный вызов exec()

Не порождает нового процесса

Происходит замещение кода текущего процесса

Анализирует содержимое исполняемого файла

Действия, выполняемые exec() (1)

- Производит трансляцию имени файла
- Анализирует заголовок файла
- Обрабатывает биты setuid и setgid
- Сохраняет аргументы вызова и переменные окружения

Действия, выполняемые exec() (2)

- Резервирует место в области свопинга для сегмента данных и стека
- Освобождает старые области процесса и свопинга

Действия, выполняемые exec() (3)

- Размещает и инициализирует карты отображения, необходимые для трансляции адреса (может совместно использоваться сегмент кода с процессом, выполняющим ту же программу)

Действия, выполняемые exec() (4)

- Копирует сохраненные аргументы и переменные окружения в новый стек процесса
- Устанавливает обработчики сигналов
- Инициализирует аппаратный контекст процесса

Завершение выполнения процесса

Добровольно exit()

Принудительно по сигналу

Процесс освобождает все ресурсы и переходит в состояние зомби

Действия, выполняемые при завершении (1)

- Отключает все сигналы
- Закрывает все открытые файлы
- Сохраняет статистику использования вычислительных ресурсов и код возврата в записи таблицы процессов

Действия, выполняемые при завершении (2)

- Состояние – зомби
- Процесс init – родитель для всех потомков данного процесса
- Освобождает адресное пространство процесса, u-area, карты отображения и области свопинга

Действия, выполняемые при завершении (3)

- Отправляет сигнал о завершении потомка родительскому процессу
- Пробуждает родительский процесс, если тот ожидает завершения потомка
- Происходит переключение контекста