

ОП.14
ОСНОВЫ
функционирования UNIX -
СИСТЕМ

3АНЯТИЕ 02

Основы организации UNIX

Рассмотрим основные принципы, на которых базируется операционная система UNIX:

1. Операционная система должна быть **максимально надежной** — это означает, в первую очередь, что система должна **сохранять работоспособность** при **возникновении** каких-либо **неисправностей** (отказы в работе аппаратных средств, ошибки в функционировании программного обеспечения).

Основы организации UNIX

В более узком смысле **надежность** означает, что система должна по возможности обеспечивать некоторый **минимальный уровень функционирования** при возникновении неисправностей, а также способность к быстрому восстановлению. Например, при отказе **одного из жестких дисков** система должна выполнить ряд **восстановительных операций**, чтобы остаться **в работоспособном состоянии** и обеспечивать работу пользователей — при этом самым существенным является **минимизация потерь критически важных данных** пользователей.

Основы организации UNIX

Принципиально **избежать отказов нельзя** — аппаратные средства имеют свойство **давать сбои**, а приложения пользователя, впрочем, как и программные модули самой системы, могут содержать ошибки. Важным является то, с какими **"потерями"** система выходит из этой ситуации. **Самым неприятным** следствием сбоя в работе может быть потеря критически **важных данных пользователей**, поэтому способность быстрого **восстановления системы после сбоев**, включая восстановление целостности данных, является **наиболее существенным** критерием при оценке надежности.

Основы организации UNIX

Операционная система UNIX изначально проектировалась как **отказоустойчивая система**.

Несмотря на некоторые различия в программной архитектуре, во всех современных UNIX-системах предусмотрены базовые механизмы, гарантирующие **высокий уровень надежности**.

Основы организации UNIX

Вот основные из них:

- **разделение программного кода операционной системы и пользовательских приложений.** В практическом плане это означает, что **большая часть** программного кода операционной системы, включая драйверы устройств и системные сервисы, а также системные области данных и стека, **изолирована от пользовательского программного кода** и реализована в форме *ядра*.

Основы организации UNIX

Если, например, **пользовательский процесс** попытается напрямую, минуя стандартные вызовы операционной системы, **обратиться к жесткому диску**, то механизмы защиты ядра **не позволят выполнить эту операцию**, поскольку могут быть разрушены важные дисковые структуры, что сделает систему неработоспособной.

Доступ **пользовательских приложений** к ресурсам системы возможен только посредством определенных стандартных функций, **предоставляемых системой UNIX** специально для этих целей.

Основы организации UNIX

- использование **механизмов безопасности** для доступа пользовательских процессов к ресурсам операционной системы.

Каждый процесс **может обращаться** только к тем ресурсам (файлам и устройствам), для которых **система предоставляет ему доступ**. При этом каждому пользовательскому процессу назначается **индивидуальный уровень привилегий**, определяющий, какие операции и с какими ресурсами процесс может выполнять;

Основы организации UNIX

- **использование механизмов резервного копирования и восстановления данных, включая полное восстановление системы.**

Основы организации UNIX

UNIX является **многопользовательской ОС**, обеспечивая возможность одновременной работы нескольких пользователей в одной системе. Поскольку пользователи, работающие в системе, могут использовать **один и тот же ресурс одновременно**, например, дисковый файл, система предусматривает **механизмы разделения доступа** к общим ресурсам. То есть, одни пользователи могут **читать и записывать данные** в файл, в то время как другие могут только **читать** данные из файла или вообще не иметь доступа к данным.

Основы организации UNIX

Операционные системы, допускающие работу в таком режиме, называются ***многopользовательскими***.

Операционная система UNIX выполняет **множество процессов одновременно**, включая как системные, так и пользовательские процессы.

Операционная система управляет процессом в течение **всего жизненного цикла**, от начала создания и до его уничтожения.

Основы организации UNIX

В UNIX **все ресурсы**, к которым может иметь доступ процесс, являются **объектами файловой системы**. Такой **единый подход** лежит в основе построения и функционирования файловой системы UNIX.

UNIX рассматривает дисковые файлы программ или данных, принтеры, жесткие диски, терминальные линии и т. д. как **объекты файловой системы**, доступ к которым осуществляется с помощью системных вызовов.

Основы организации UNIX

Такой механизм очень удобен, поскольку позволяет использовать **единый программный интерфейс** как для работы с **файлами данных**, так и с **устройствами**, независимо от физической природы объектов.

Кроме этого, можно использовать единые подходы для **установки атрибутов безопасности** для объектов файловой системы.

Основы организации UNIX

Перечисленные особенности UNIX делают ее очень **удобной средой** для функционирования **пользовательских приложений** и позволяют легко переносить ее на разные аппаратные платформы.

Посмотрим, как отображены вышеперечисленные **возможности** операционной системы UNIX в ее **программной архитектуре**:

Основы организации UNIX



Основы организации UNIX

Основой операционной системы UNIX является **ядро**, которое взаимодействует, с одной стороны, с **аппаратными средствами**, а с другой стороны обеспечивает работу **пользовательских программ**.

Часть функций операционной системы выполняется **системными процессами**, реализованными, как правило, в форме **процессов-серверов**, более известных под названием "**демон**" (daemon), и его основная функция — **выполнение запросов** на обслуживание клиентских процессов.

Основы организации UNIX

Уровень **интерфейса системных вызовов** отвечает за **взаимодействие** программ пользователя и операционной системы.

Ни одна программа пользователя **не может получить доступ** к ресурсам системы иначе, как посредством **системных вызовов**.

В качестве **примеров** системных вызовов можно привести низкоуровневые функции **ввода/вывода**, такие как `open ()`, `read ()`, `write ()` и `close ()`.

Основы организации UNIX

Системные вызовы обеспечивают выполнение целого ряда операций:

- трансляции операций пользователя в запросы к драйверам устройств;
- создания, запуска и уничтожения процессов;
- ввода/вывода;
- доступа к файлам и дисковым устройствам;
- поддержки терминальных устройств.

Основы организации UNIX

Ядро

Ядро операционной системы UNIX отвечает за выполнение **базовых функций системы**:

- управление процессами;
- управление файловой системой;
- управление устройствами ввода/вывода;
- Управление безопасностью системы.

Функционирование операционной системы в основном определяется ядром.

Основы организации UNIX

Ядро

Во многих версиях операционной системы UNIX ядро реализовано в виде большого исполняемого дискового файла, который запускается в момент инициализации системы — так называемое *монолитное ядро*.

Монолитное ядро включает **все необходимые модули** для работы, в том числе и **драйверы устройств**.

Для **изменения функциональности** такого ядра, например, при необходимости **включить новый драйвер** устройства, исходный текст ядра нужно полностью **перекомпилировать**.

Основы организации UNIX

Ядро

Модульное ядро содержит базовые компоненты, необходимые для загрузки системы, при этом **дополнительные модули** программного кода (обычно это драйверы устройств) загружаются в оперативную память и **подключаются к ядру динамически**, по мере необходимости, например, при включении аппаратного устройства.

Такие дополнительные модули реализованы в виде загружаемых модулей ядра.

Основы организации UNIX

Ядро

Преимуществом модульного ядра является то, что базовый модуль ядра имеет небольшой размер, **быстрее загружается** и требует меньше ресурсов операционной системы.

В то же время **монолитное ядро работает чуть быстрее**, поскольку не требуется переключений контекста выполняемых процессов (что имеет место в случае загрузки/выгрузки дополнительных модулей) и дополнительной синхронизации, как при использовании отдельных модулей.

Основы организации UNIX Ядро

В **большинстве** современных операционных систем UNIX используется **комбинированный тип ядра**, которое, обладая широким диапазоном функциональности, допускает загрузку дополнительных модулей во время работы операционной системы и **сочетает преимущества** модульного и монолитного ядра.

Основы организации UNIX

Ядро

Еще одна **особенность** функционирования ядра современных UNIX-систем — возможность одновременного выполнения **нескольких потоков**.

Поток можно представить как **отдельный выполняющийся фрагмент** программного кода в рамках одного процесса.

При этом ядро управляет **синхронизацией** выполнения потоков.

В этом случае говорят о **многопоточности ядра**, которая позволяет повысить эффективность функционирования как самого ядра, так и операционной системы в целом

Основы организации UNIX

Ядро

Высокая **эффективность многопоточности** объясняется тем, что синхронизация отдельных потоков одного и того же процесса выполняется с **минимальными издержками** и требует **значительно меньше времени**, чем запуск отдельного процесса.

Большинство современных операционных систем UNIX **поддерживает выполнение не только отдельных потоков ядра**, но и **многопоточность на уровне пользовательских программ**, **повышая их производительность**.

Основы организации UNIX Ядро

В этом случае многопоточные приложения выполняются как совокупность **элементарных** (lightweight) **процессов**, которые используют общее адресное пространство, общие страницы памяти и открытые файлы.

Естественно, что для получения выигрыша в производительности выполняющаяся программа должна поддерживать реализацию многопоточности.

Основы организации UNIX

Ядро

Еще одной **особенностью** ядра UNIX является и то, что оно функционирует в режиме ***невывтесняющей многозадачности*** (non-preemptive multitasking).

Это означает, что система **не может прерывать** выполнение процесса, выполняющегося в режиме ядра.

Основы организации UNIX

Ядро

Ядро UNIX обеспечивает выполнение следующих функций:

- **синхронизация** процессов (создание, выполнение, остановка и завершение процессов);
- **планирование** приоритетов выполнения (выделяется интервал времени);
- **распределение** памяти выполняемым процессам — при этом каждому процессу выделяется определенный **объем** памяти.

Основы организации UNIX

Ядро

- **создание, изменение и удаление** файловых систем, расположенных на устройствах постоянного хранения информации (**выделение дискового пространства**), а также управление данными пользователей.
- **управление доступом** процессов к периферийным устройствам ввода/вывода, таким как терминалы, накопители на магнитных лентах и сетевое оборудование.

Основы организации UNIX

Ядро

Ядро операционной системы является **прозрачным** для пользовательской программы.

Это означает, что детали взаимодействия программы пользователя и операционной системы **скрыты от пользователя**.

Например, если программа пользователя обращается к какому-либо файлу и записывает в него данные, то ядро системы выполняет приблизительно такую последовательность действий:

1. **Определяет местоположение файла на носителе.**

Основы организации UNIX

Ядро

2. Получает **информацию о расположении** требуемых данных в физических секторах накопителя.
3. Определяет **место записи блока данных** в физическую область дискового пространства и т. д.
4. Наконец, ядро **вызывает драйвер** устройства и **передает** ему **параметры** и код операции (записи данных), после чего и выполняется **запись данных**.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Программа или, по-другому, **приложение** представляет собой **исполняемый файл**, вызываемый операционной системой для выполнения.

Процесс в UNIX, как и в других операционных системах, — это **выполняющийся образ** исполняемого файла.

В обычном смысле, когда мы говорим, что "**программа выполняется**" или "**программа завершилась**", имеется в виду **процесс** или группа процессов.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Термин "**процесс**" — это ключевое понятие в UNIX.

В первом приближении процессом называют загруженный в память **двоичный образ дискового файла**.

Процессор интерпретирует его как **совокупность машинных инструкций, данных и стековых структур**.

Работу самой операционной системы можно представить как функционирование **совокупности процессов**. Например, при входе пользователя в операционную систему начинает выполняться процесс *shell*, при запуске какой-либо команды, например, `ls -l`, также порождается процесс.

Основы организации UNIX

Ядро

Программы, процессы и потоки

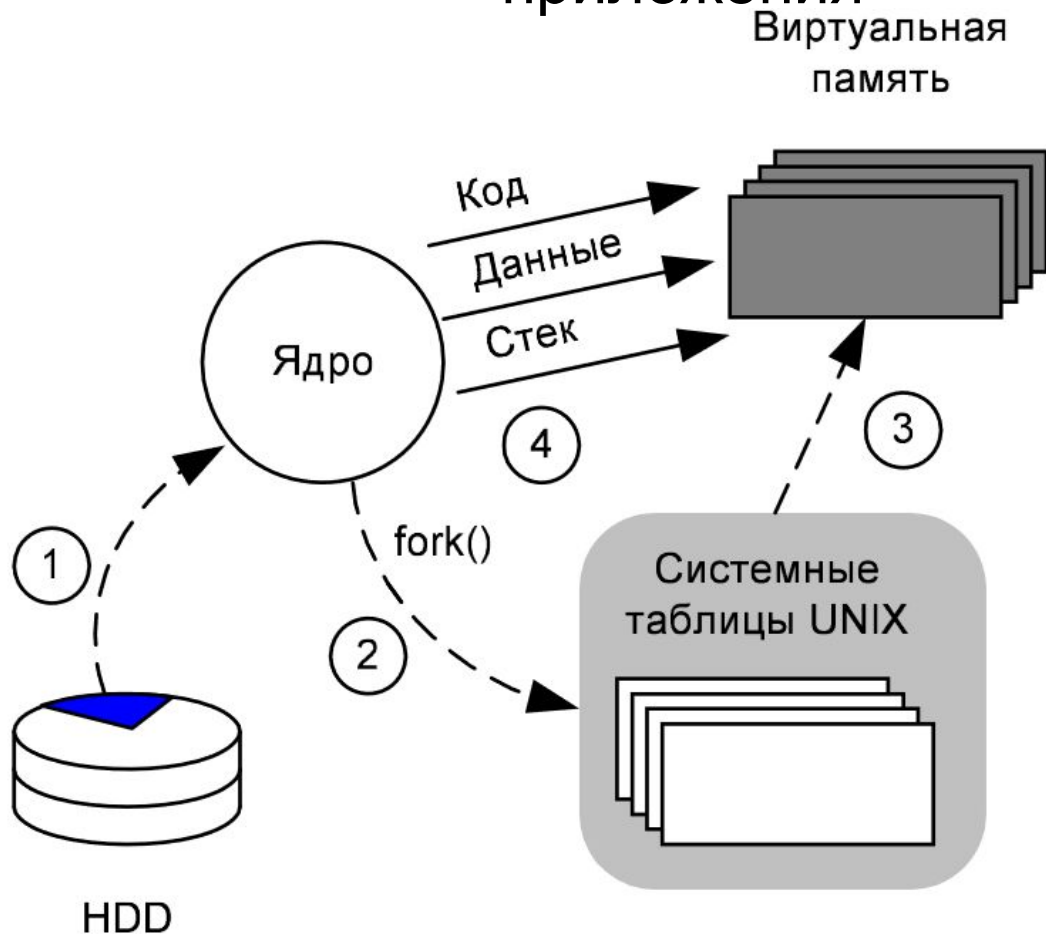
Обобщая, можно сказать, что всякий раз, когда вызывается команда UNIX или **запускается пользовательская программа, порождается новый процесс.**

Из этого правила есть исключения, например, запуск команды `cd`, но подобные случаи мы рассматривать не будем.

Основы организации UNIX Ядро

Программы, процессы и потоки

Операции, выполняемые при запуске
приложения



1. ОС считывает исполняемый файл программы с диска.

2. Если формат файла программы является корректным, выполняет системный вызов `fork()`. При этом **ядро заносит информацию**, относящуюся к созданному процессу, в специальные системные.

3, 4. А также выделяет **виртуальную память** для кода, данных и стека текущего процесса, после чего управление передается первой инструкции процесса.

Основы организации UNIX

Ядро

Программы, процессы и потоки

В простейшем варианте исполняемый файл программы выполняется как **один процесс**, хотя в большинстве других случаев таких процессов может быть **множество**.

Следует сказать, что и само **ядро** системы функционирует как **совокупность многих взаимосвязанных процессов**.

Ядро системы идентифицирует каждый процесс по его номеру, который называется **идентификатором процесса** (Process ID, **PID**). Каждому процессу присваивается уникальный идентификатор PID, позволяющий ядру **различать процессы**.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Завершение выполнения процесса инициируется системным вызовом `exit ()`.

Ядро операционной системы **отслеживает окончание работы** процесса и **освобождает** использовавшийся им идентификатор.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Все **процессы**, выполняющиеся в среде операционной системы UNIX, могут функционировать либо в **пользовательском режиме** (User Mode), либо в **режиме ядра** (Kernel Mode).

Пользовательский режим не позволяет напрямую обращаться к аппаратным ресурсам системы и системным структурам данных.

Процесс, выполняющийся в режиме **ядра, имеет такую возможность.**

Основы организации UNIX

Ядро

Программы, процессы и потоки

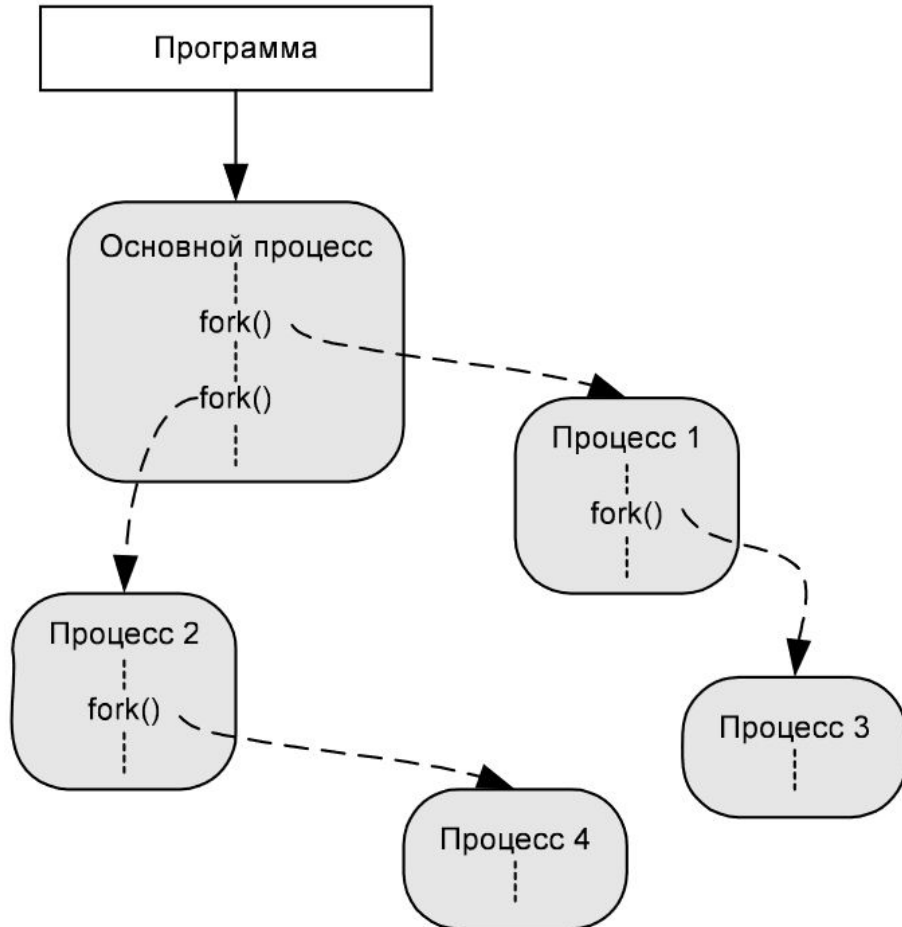
До сих пор мы рассматривали механизм создания и выполнения **одного процесса**.

В большинстве случаев **работающий процесс** может **порождать другие процессы**, например, при необходимости параллельной обработки данных или клиентских запросов (если это процесс-сервер наподобие `telnet`, `ftp` и т. д.).

Как и в рассмотренной ранее схеме запуска процесса, здесь используется системный вызов `fork ()`.

Основы организации UNIX Ядро

Программы, процессы и потоки



При запуске приложения **создается основной процесс**, который создает (порождает) процессы 1 и 2, процесс 1 создает процесс 3, а процесс 2 создает процесс 4.

В этом случае говорят, что основной процесс является **родительским** для процессов 1 и 2, процесс 2 является **родительским** для процесса 4, а процесс 1 является **родительским** для процесса 3. В свою очередь, процессы 1 и 2 называются **дочерними** или **порожденными** основным процессом, а процессы 3 и 4 — порожденными для процессов 1 и 2 соответственно.

Основы организации UNIX

Ядро

Программы, процессы и потоки

В UNIX предусмотрен еще один механизм, обеспечивающий параллельное выполнение программного кода, но уже в рамках одного и того же процесса — **механизм потоков** (threads).

Потоки широко используются как функциями ядра, так и системными сервисами UNIX.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Существенным **отличием** потоков от процессов является то, что поток выполняется **в контексте данного процесса**.

Для потока **не выделяется** отдельное адресное пространство и не создаются копии таблиц текущего процесса.

Файловые **дескрипторы** (некий атрибут объекта) и **переменные** в памяти являются **общими** для процесса и потока.

При **закрытии**, например, файлового дескриптора в потоке он **закрывается** и для процесса.

Основы организации UNIX

Ядро

Программы, процессы и потоки

Для создания потоков во всех версиях операционной системы UNIX используется функция `pthread_create()` библиотеки C («Си»), одним из параметров которой является адрес функции потока.

Функция потока, созданная с помощью функции `pthread_create()`, собственно и выполняет всю работу.

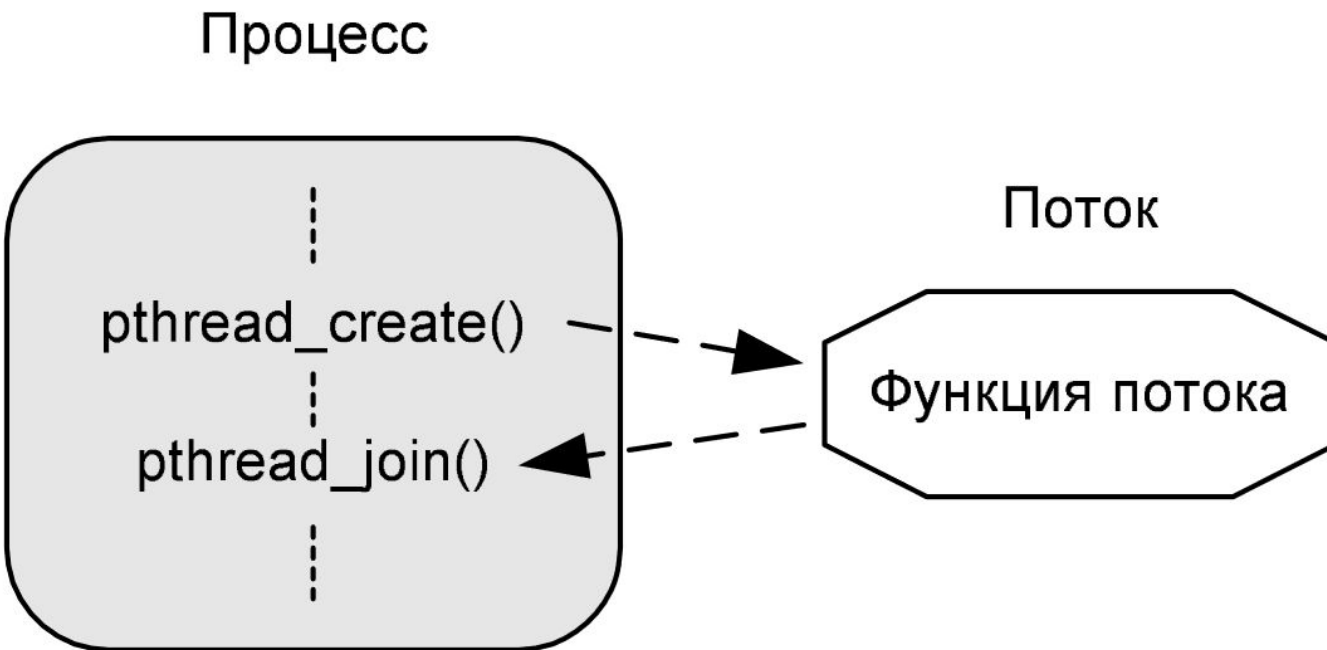
Основы организации UNIX

Ядро

Программы, процессы и потоки

Процесс может **ожидать завершения** выполнения потока, выполнив функцию `pthread_join()`.

Сама функция потока может **завершаться** обычным образом — при этом инструкция `return` или просто достижение конца функции **приводит к неявному завершению** работы потока.



Основы организации UNIX

Ядро

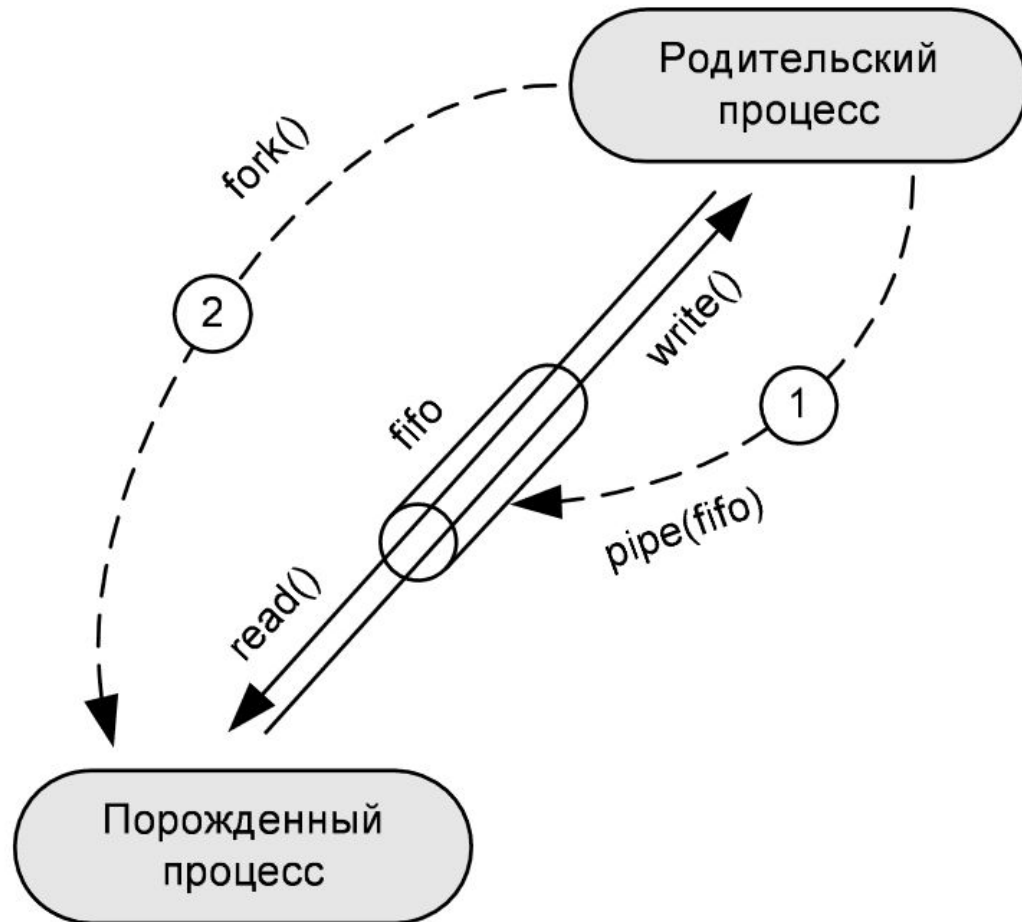
Взаимодействие процессов

Ранее мы рассмотрели, каким образом **создается** порожденный процесс, но при этом возникает закономерный вопрос: каким образом данные одного процесса **могут быть доступны** другому процессу?

В операционной системе UNIX для этого предусмотрен **механизм обмена**, известный под названием **неименованного канала**.

Основы организации UNIX Ядро Взаимодействие процессов

двунаправленный канал передачи данных



Вначале родительский процесс создает **неименованный канал** при помощи системного вызова `pipe()` (1).

Созданный таким образом канал является **двунаправленным**, т. е. процесс может как записывать в него данные, так и считывать их из него.

Неименованный канал характеризуется двумя дескрипторами — для **ВХОДНЫХ** и **ВЫХОДНЫХ** данных.

Основы организации UNIX

Ядро

Взаимодействие процессов

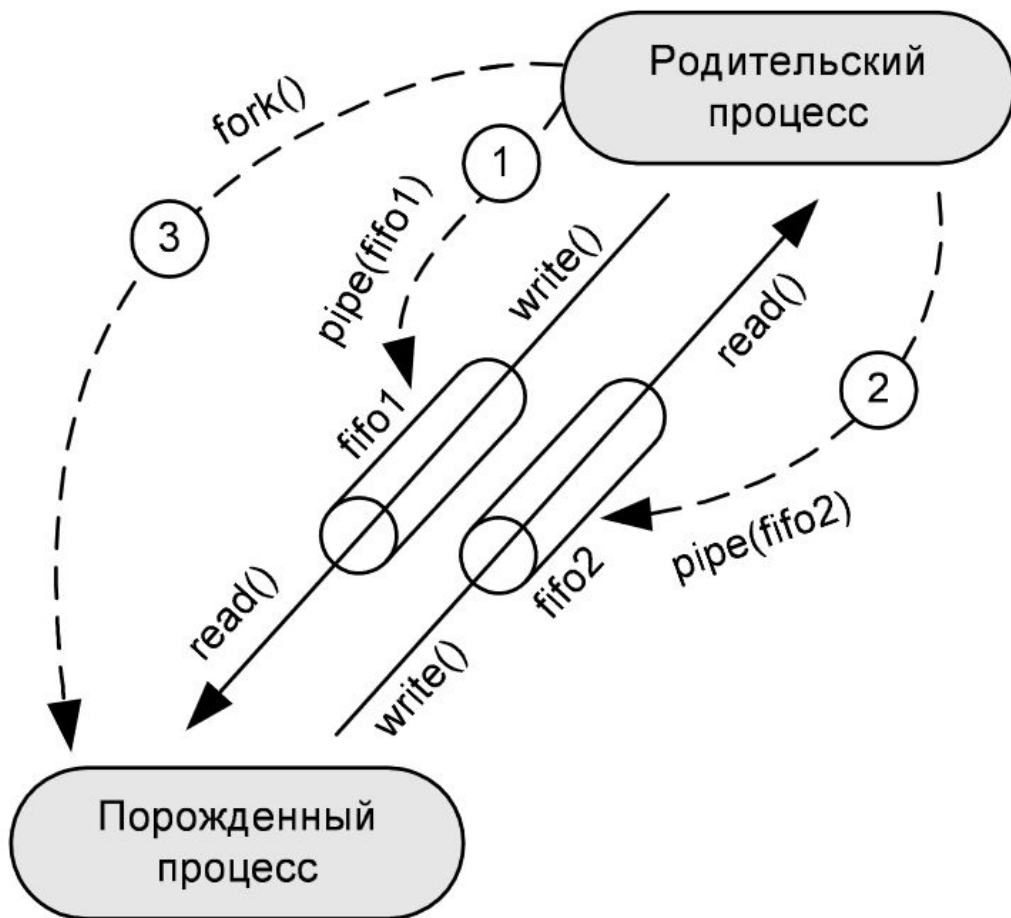
Двунаправленный канал передачи данных, такой канал, в котором оба процесса могут одновременно как записывать данные, так и читать их из него.

Такая конфигурация **используется редко** из-за сложностей синхронизации передачи/приема данных.

Более приемлемой является конфигурация обмен данными при помощи **двух однонаправленных** каналов.

Основы организации UNIX Ядро Взаимодействие процессов

два однонаправленный канал передачи данных



Основной процесс создает **два однонаправленных** канала с дескрипторами `fifo1` и `fifo2` при помощи двух вызовов функции `pipe (1, 2)`.

При этом канал `fifo1` можно настроить так, чтобы в него мог **записывать** данные **родительский** процесс, а **читал** их **порождённый** (дочерний) процесс.

Канал `fifo2` можно настроить для работы **в обратном направлении**: записи данных дочерним и чтения данных родительским процессом.

Основы организации UNIX

Ядро

Взаимодействие процессов

До сих пор мы рассматривали взаимодействие родительского и порожденного процессов. Во многих случаях процессы должны **обмениваться данными с другими процессами**, независимыми от них.

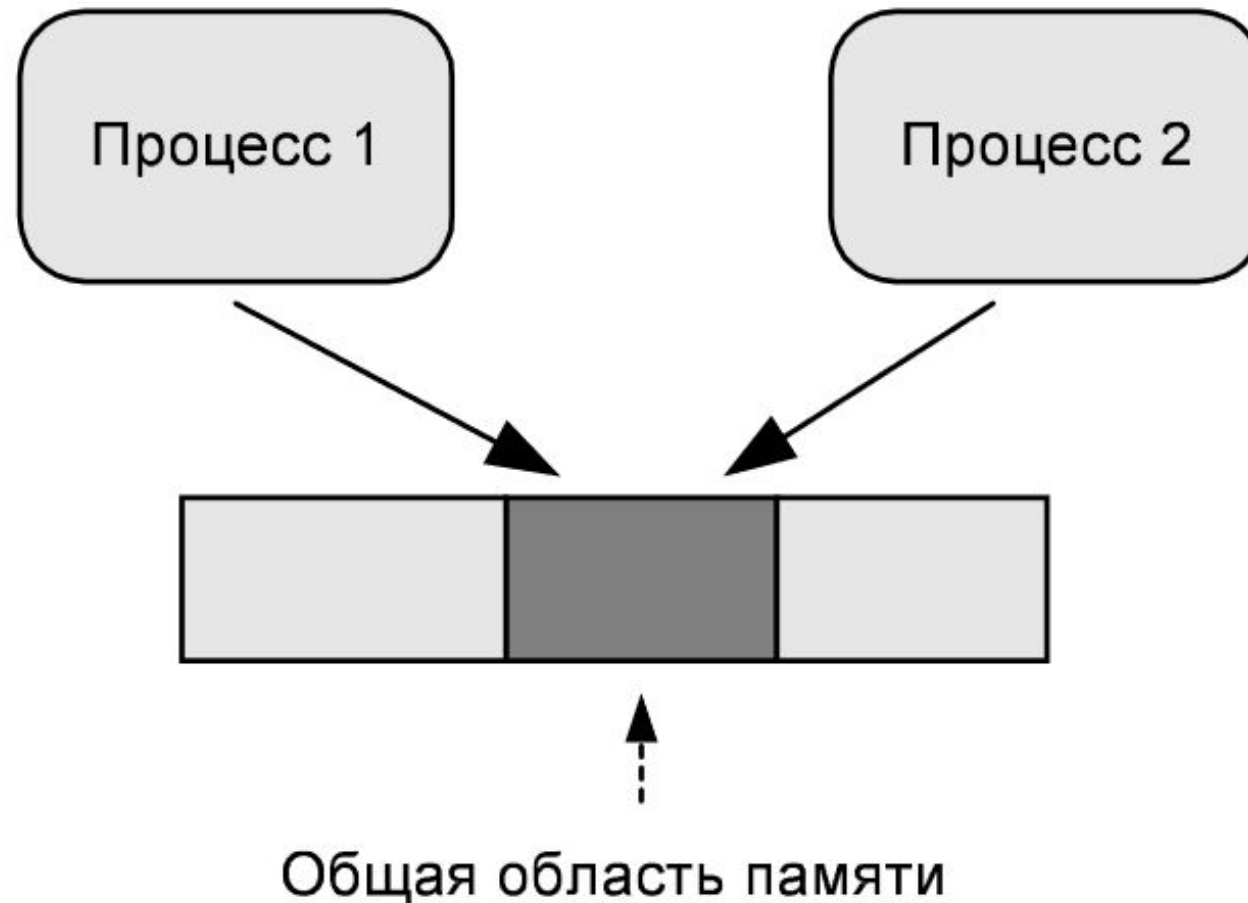
Для этого может использоваться несколько механизмов.

Процессы могут обмениваться данными, используя **общие (разделяемые) области памяти**.

Основы организации UNIX

Ядро

Взаимодействие процессов



Основы организации UNIX

Ядро

Взаимодействие процессов

При этом данные одного процесса могут быть доступны другому процессу.

При использовании такого механизма следует учитывать то, что оба процесса могут запросить одни и те же данные, поэтому следует предусмотреть механизмы синхронизации доступа (можно использовать стандартные средства операционной системы UNIX).

Основы организации UNIX

Ядро

Взаимодействие процессов

Для межпроцессного взаимодействия в UNIX очень широко используется механизм так называемых **сокетов** (гнезд), который мы подробно рассмотрим при анализе сетевых возможностей операционной системы.

Основы организации UNIX

Системные процессы

Следующий уровень функциональности, который мы рассмотрим, — **системные процессы**, которые создаются при запуске системных программ.

К системным программам или, по-другому, к **системному программному обеспечению** относят командные оболочки *shell* (интерпретаторы), команды и утилиты системного администрирования, драйверы и протоколы коммуникаций.



Основы организации UNIX Системные процессы

Операционная система UNIX включает ряд стандартных системных программ для выполнения задач **администрирования, конфигурирования и поддержки** файловой системы.

Кроме того, к этой группе программ следует отнести утилиты:

- настройки параметров конфигурации системы;
- перекомпоновки ядра (если она необходима) и добавления новых драйверов устройств;
- создания и удаления учетных записей пользователей;
- создания и подключения физических файловых систем;
- установки параметров контроля доступа к файлам.

Основы организации UNIX Системные процессы

В качестве **пользовательских** программ могут выступать **командные файлы** (скрипты), написанные с помощью командного интерпретатора *shell*, или **разработанные** на одном из языков высокого уровня (C, Pascal, Fortran) приложения.

Командные интерпретаторы представляют собой **высокоуровневую** среду программирования.

Они и очень удобны для **создания** собственных **командных файлов** (по аналогии с MS-DOS), позволяющих разрабатывать довольно сложные программы.

Возможности командных интерпретаторов **намного превышают** те, которые имеет, например, интерпретатор командной строки MS-DOS, что делает их полноценным инструментом разработки программ.

Основы организации UNIX Системные процессы

Системные процессы являются частью ядра и всегда расположены в **оперативной памяти**.

Они не имеют **выполняемых файлов** и запускаются особым образом при инициализации ядра системы.

Исполняемые **инструкции и данные** таких процессов постоянно **находятся в ядре системы**, поэтому они могут вызывать другие функции и обращаться к данным, недоступным для остальных процессов.

Основы организации UNIX

Системные процессы

К системным процессам относится и процесс на чальной инициализации `init`, являющийся прародителем всех остальных процессов.

Несмотря на то, что `init` не является частью ядра и запускается из выполняемого файла, его функционирование критически важно для всей системы в целом. Программа `/sbin/init`, запускающая процесс `init`, порождает процессы для запуска системы на основе записей, находящихся в файле `/etc/inittab`. При этом процесс `init` анализирует записи в файле `/etc/inittab` и определяет последовательность запуска, остановки и перезапуска остальных процессов.

Основы организации UNIX Системные процессы

К системным процессам относится и процесс на чальной инициализации `init`, являющийся прародителем всех остальных процессов.

Несмотря на то, что `init` не является частью ядра и запускается из выполняемого файла, его функционирование критически важно для всей системы в целом.

Программа `/sbin/init`, запускающая процесс `init`, порождает процессы для запуска системы на основе записей, находящихся в файле *`/etc/inittab`*.

Основы организации UNIX Системные процессы

При этом процесс `init` анализирует записи в файле *`/etc/inittab`* и определяет последовательность запуска, остановки и перезапуска остальных процессов.

Ещё одна группа процессов, которые выполняются в системе, относится к ***прикладным процессам***.

Основы организации UNIX

Системные процессы

Как правило, это процессы, созданные в контексте пользовательского сеанса работы, важнейшим из которых является командный интерпретатор *shell*, **обеспечивающий** выполнение команд пользователя в системе UNIX.

Пользовательские процессы могут **выполняться** как в **интерактивном** (приоритетном), так и в **фоновом** режимах.

Интерактивные процессы монополюно владеют терминалом, и пока такой процесс не завершит свое выполнение, пользователь не имеет доступа к командной строке.

Список литературы:

1. Юрий Магда. UNIX для студентов, Санкт-Петербург «БХВ-Петербург», 2007.
2. Unix и Linux: руководство системного администратора, 4-е издание, 2012, Э. Немет, Г. Снайдер, Т. Хейн, Б. Уэйли
3. Организация UNIX систем и ОС Solaris 9, Торчинский Ф.И., Ильин Е.С., 2-е издание, исправленное, 2016.

Спасибо за внимание!

Преподаватель: Солодухин Андрей Геннадьевич

Электронная почта: asoloduhin@kait20.ru