# Реализация системных структур ОС для управления процессами и операций над ними.

**Операционные системы**

Беспутин Глеб Антонович

# Содержание

# Список иллюстраций

# Список таблиц

# 1 Цель работы

Цель данного доклада заключается в изучении и анализе системных структур операционных систем, которые отвечают за управление процессами и операциями над ними.

# 2 Задание

1)Рассмотреть основные понятия и принципы управления процессами в операционных системах.

2)Исследовать операции над процессами, которые выполняются операционной системой, такие как создание, остановка, приоритеты и планирование процессов.

# 3 Текст доклада

## 3.1 Режимы и состояния процесса

Процессом называется последовательность операций при выполнении программы, которые представляют собой наборы байтов, интерпретируемые центральным процессором как машинные инструкции (т.н. “текст”), данные и стековые структуры. Выполнение процесса заключается в точном следовании набору инструкций, который является замкнутым и не передает управление набору инструкций другого процесса; он считывает и записывает информацию в раздел данных и в стек, но ему недоступны данные и стеки других процессов. Одни процессы взаимодействуют с другими процессами и с остальным миром посредством обращений к операционной системе.

Существует два основных режима (или уровня) выполнения процесса : - режим задачи (пользователя) - режим ядра.

В режиме задачи процесс выполняет инструкции прикладной программы,допустимые на непривилегированном уровне защиты процессора, при этом процессу недоступны системные структуры данных. Когда процессу требуется получение каких-либо услуг ядра ОС, он делает системный вызов, который выполняет инструкции ядра, находящиеся на привилегированном уровне. Несмотря на то, что выполняются инструкции ядра, это происходит от имени процесса, сделавшего системный вызов. Выполнение процесса при этом переходит в режим ядра. Таким образом, ядро системы защищает собственное адресное пространство от доступа прикладного процесса, который может нарушить целостность структур

данных ядра и привести к разрушению ОС. Более того, часть процессорных инструкций, например, изменение регистров, связанных с управлением памятью, могут быть выполнены только в режиме ядра

## 3.2 Контекст процесса

Каждому процессу соответствует контекст или образ, включающий в себя следующие три компонента : 1) пользовательский контекст — содержимое виртуального адресного пространства, сегментов программного кода, данных, стека, разделяемых сегментов и сегментов файлов, отображаемых в виртуальную память;

2) регистровый контекст — содержимое аппаратных регистров (регистр счетчика команд, регистр состояния процессора, регистр указателя стека и регистры общего назначения), на платформе x86-IA32 регистровый контекст сохраняется в так называемом сегменте состояния задачи (англ.TSS — task state segment);

3) системный контекст — структуры данных ядра ОС, связанные с процессом. Системный контекст процесса состоит из двух частей: статической и динамической. Для каждого процесса имеется одна статическая часть системного контекста и переменное число динамических частей. Статическая часть контекста включает следующие атрибуты процесса:

- Идентификатор процесса PID (от англ. Process Identifier) — уникальный номер, идентифицирующий процесс в системе. По сути, это номер строки в таблице процессов — специальной внутренней структуре ядра ОС, хранящей информацию о процессах. В любой момент времени ни у каких двух процессов номера не могут совпадать, однако после завершения процесса его номер освобождается и может быть в дальнейшем использован для

идентификации любого вновь запущенного процесса. Идентификатор родительского процесса PPID (от англ. Parent Process Identifier). В ОС UNIX процессы выстраиваются в иерархию — новый процесс может быть создан только одним из уже существующих процессов, который выступает для него родительским. Очевидно, что в такой схеме должен присутствовать один процесс с особым статусом: он должен быть порожден ядром операционной системы и будет являться родительским для всех остальных процессов в системе. В ОС UNIX такой процесс имеет собственное имя — init. Идентификаторы пользователя и группы. Идентификаторы пользователя UID и группы GID, от имени которых запущен процесс — наследуются от родительского процесса; а также эффективный идентификатор пользователя EUID и эффективный идентификатор группы EGID, которые фактически используются ядром ОС для определения прав доступа процесса в системе. В общем случае, для процесса значения UID и GID могут не совпадать с EUID и EGID соответственно.

- Идентификаторы группы процессов (PGID) и сеанса (SID)
- Приоритет процесса — число, используемое при планировании исполнения процесса в операционной системе.
- Таблица дескрипторов открытых файлов — список структур ядра, описывающий все файлы, открытые этим процессом для ввода-вывода.
- Состояние процесса. Каждый процесс в любой момент времени находится в одном из нескольких определенных состояний
- Терминальная линия (TTY). Терминал или псевдотерминал , связанный с процессом. С этим терминалом по умолчанию связаны стандартные потоки: входной, выходной и поток сообщений об ошибках.
- Временные параметры процесса, включая время выполнения в режиме задачи, время выполнения в режиме ядра.
- Переменные окружения — набор строковых переменных, определяющих окружение процесса.

- Другие системные параметры, включая сигналы, ожидающие доставки; размер адресного пространства процесса; указатель на локальную таблицу дескрипторов сегментов; список областей памяти процесса; код возврата, передаваемый родительскому процессу и пр.

Динамическая часть контекста процесса — это один или несколько стеков, которые используются процессом при выполнении в режиме пользователя и в режиме ядра (в процессе прерываний и системных вызовов)

## 3.3 Создание и завершение процесса

Каждый процесс в ОС UNIX, за исключением первоначального (нулевого), является объектом, создаваемым в результате выполнения системного вызова fork. Процесс, инициировавший вызов fork(), называется родительским (процесс-родитель), а вновь создаваемый процесс называется порожденным или дочерним (процесс-потомок). Каждый процесс имеет одного родителя, но может породить множество дочерних процессов.Завершение выполнения процесса инициирует системный вызов exit(), аргументом которого является целое число типа int, передаваемое в родительский процесс как код завершения1 процесса-потомка. В ОС UNIX принято, что в случае успешного завершения процесс возвращает код 0, иные значения трактуются как признак завершения процесса с какой-либо ошибкой. Процессы могут вызывать функцию exit() как в явном, так и в неявном виде. Так, во всех программах на языке Си компилятором автоматически создается специальная процедура startup, вызывающая функцию main(). При завершении выполнения программы, т.е. на выходе из функции main(), в процедуре startup выполняется вызов exit() с кодом завершения, который был указан в операторе return функции main(). Процесс-родитель может синхронизировать продолжение своего выполнения с моментом завершения процесса-потомка, если воспользуется системной функцией wait().Функция wait() приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не прекратит выполнение

или до появления сигнала2 , который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.Возможны ситуации, когда дочерний процесс к моменту вызова функции wait() уже завершился. Такой дочерний процесс называется процессомзомби, или просто зомби (англ. zombie process, а также англ. defunct process); он еще присутствует в списке процессов ОС, чтобы дать родительскому процессу возможность получить код его завершения. В таком случае функция wait() немедленно возвращается, а сис темные ресурсы, связанные с дочерним процессом, освобождаются.Системная функция exec()1 дает возможность процессу запускать другую программу, при этом соответствующий этой программе исполняемый файл будет загружен в пространство памяти данного процесса. Содержимое пользовательского контекста после вызова функции становится недоступным, за исключением передаваемых функции параметров, которые переписываются ядром из старого адресного пространства в новое.

## 3.4 Переменные окружения

Каждый запускаемый в ОС процесс содержит некое информационное пространство, именуемое окружением, в котором можно задавать именованные хранилища данных — переменные окружения (другой термин — переменные среды). Для переменных окружения определены операции записи любой текстовой информации (присвоение значения переменной окружения) и чтения этой информации. При создании нового процесса окружение процесса-потомка создается как точная копия окружения процесса-родителя. Переменные окружения дочернего процесса копируются из родительского, поэтому дочерний процесс не может непосредственно повлиять на переменные окружения родительского процесса. Родительский процесс может заранее установить список или модифицировать значения наследуемых переменных для дочернего процесса. Переменные окружения устанавливаются пользователем или сценариями оболочки. Начальный набор переменных инициализируется стартовыми сцена-

риями ОС и сценариями, запускаемыми при регистрации пользователя в системе . Переменные окружения имеют большое значение в ОС UNIX, так как хранят множество настроек как системы в целом, так и отдельных программ.

## 3.5 Приоритет процессов

Процессы в ОС UNIX имеют различный приоритет выполнения, определяемый в каждый момент планировщиком процессов. Так, системные процессы, как правило, имеют изначально более высокий приоритет по сравнению с пользовательскими. В ОС UNIX традиционно используются динамически изменяющиеся приоритеты. При образовании процесса ему назначается некоторый устанавливаемый системой статический или относительный приоритет — параметр nice number (NI), учитываемый планировщиком процессов при определении очередности их запуска. Реальным критерием планирования выступает динамический приоритет или приоритет выполнения — параметр priority (PRI), а статический приоритет составляет основу начального значения динамического приоритета процесса и не изменяется ядром на всем протяжении жизни процесса. Диапазон значений приоритета различен, в зависимости от версии ОС UNIX и используемого планировщика. В ОС Linux и Solaris параметр nice number может принимать значения от -20 до 20, при этом наивысшему приоритету соответствует минимальное значение nice number, низшему — максимальное значение nice number. Пользовательские процессы (выполняемые от имен обычного пользователя) могут принимать значения nice number от 0 до 20, системные процессы (выполняемые от имени суперпользователя root) обычно запускаются с большим приоритетом (с отрицательным значе- ´ нием nice number). Диапазон значений приоритета выполнения PRI может отличаться от диапазона nice number (например, в ОС Solaris диапазон PRI составляет от 0 до 159). Отдельно стоит выделить так называемый приоритет реального времени, который получают наиболее важные системные процессы, критичные к каким-либо временным задержкам.

Такой приоритет обозначается как RT (от ´ англ. Real Time). Для изменения стартового значения относительного приоритета процесса при запуске программы используется команда nice, а для изменения относительного приоритета выполняемого процесса — команда renice. В приложениях приоритет может быть изменен системным вызовом nice().

# 4 Выводы

1)Рассмотрел основные понятия и принципы управления процессами в операционных системах.

2)Исследовал операции над процессами, которые выполняются операционной системой, такие как создание, остановка, приоритеты и планирование процессов.

# Список литературы