# 1. Подсистема управления процессами и потоками, ее функции и их реализация.

Одной из основных подсистем любой современной мультипрограммной операционной системы (ОС), непосредственно влияющей на функционирование компьютера, является подсистема управления процессами и потоками. Её основными функциями являются:

* создание процессов и потоков;
* изоляция процессов;
* обеспечение потоков и процессов ресурсами;
* планирование выполнения и диспетчеризация потоков;
* синхронизация процессов и потоков;
* организация взаимодействия между процессами;
* завершение и уничтожение процессов и потоков.

Процессы создаются в результате 5 основных событий:

* загрузка (инициализация операционной системы);
* запрос пользователя на создание процесса;
* запрос на создание процесса от уже работающего процесса;
* создание процесса ядром операционной системы для собственных нужд;
* запуск пакетного задания.

Как правило, в процессе загрузки ОС уже создаются несколько служебных процессов. Некоторые высокоприоритетные процессы обеспечивают взаимодействие с пользователями и выполняют назначенные им другие специфические задачи. Другие процессы могут являться фоновыми, не быть связанными с конкретными пользователями и выполнять особые функции – например, управление выводом на печать, электронной почтой, передачей данных по сети, Web-страницами, запланированным запуском программ и т.д. Такие процессы называют демонами (службами).

Процесс соответствует тому, что пользователь привык воспринимать как программу (приложение). В интерактивных системах пользователь может запустить процесс, введя с клавиатуры имени соответствующего исполняемого файла программы. В системах пакетной обработки - как правило, на мэйнфреймах - пользователи могут отправлять задания (в том числе удаленно). Эти задания накапливаются в очереди, из которой их считывает ОС и, по мере освобождения необходимых ресурсов, создает новые процессы.

Во всех перечисленных случаях, новый процесс формируется с технической точки зрения одинаково - текущий процесс выполняет системный запрос на создание нового процесса. Подсистема управления процессами и потоками обеспечивает процессы необходимыми им ресурсами. Операционная система поддерживает в памяти специальные информационные структуры, с помощью которых ведёт учёт выделенных каждому процессу ресурсов. Она может назначать процессу ресурсы в монопольное или совместное с другими процессами пользование. Некоторые из ресурсов выделяются системой процессу при его создании, а некоторые – динамически, т.е. во время выполнения, по его запросам. Ресурсы могут выделяться на все время жизни процесса или только на определенный период использования. Для выполнения своих функций подсистема управления процессами и потоками взаимодействует с подсистемами ОС, ответственными за управление другими ресурсами ЭВМ - подсистема управления памятью, файловая система и т.д..

Завершение процесса, как правило, происходит вследствие одного из следующих действий:

* обычное завершение;
* выход по ошибке;
* уничтожение другим процессом.

Каждый раз, когда процесс завершается, система предпринимает шаги, чтобы "зачистить следы" его пребывания в системе. Подсистема управления процессами освобождает области оперативной памяти, отведенные под коды, данные и системные информационные структуры процесса, а также закрывает все файлы, с которыми работал процесс,.

Процесс принято рассматривать как способ объединения взаимосвязанных ресурсов в одну группу. В эту группу входят: адресное пространство, содержащее текст программы, данные, а также другие ресурсы. - дочерние процессы, открытые файлы, учётная информация, обработчики сигналов, и другие. Ресурсами, объединенными в процессы, управлять гораздо проще. Для того чтобы отдельные процессы не могли вмешаться в распределение ресурсов или повредить области программ и данных друг друга, важнейшей задачей является изоляция одного процесса от другого.

Поток выполнения (от англ. thread – нить) – наименьшая единица обработки, исполнение которой может быть назначено ядром ОС. В большинстве случаем, поток находится «внутри» процесса. Несколько потоков могут существовать в рамках одного и того же процесса и совместно использовать ресурсы – например, память. В частности, потоки разделяют инструкции процесса, т.е. код программы и его контекст (значения переменных). Поток имеет свой счетчик команд, отслеживающих порядок выполнения действий. В него входят:

* регистры, в которых хранятся текущие переменные;
* стек, который содержит параметры всех функций, которые были вызваны, но еще не завершили своё выполнение, и локальные переменные.

Концепция потока добавляет в модель процесса возможность выполнения в его среде нескольких программ, которые работают параллельно.

Любой процесс содержит хотя бы один поток. Следует различать концепции поток и процесса:

* процессы используются для группирования ресурсов;
* потоки являются объектами, поочередно исполняющимися на центральном процессоре (при наличии в процессоре нескольких ядер или технологии hyper-threading, потоки выполняются одновременно).

Выполнение программ на центральном процессоре происходит последовательно, команда за командой. Поскольку команды в этой последовательности могут принадлежать разным процессам, то возникает необходимость в планировании процессорного времени и справедливого распределении его между потоками разных процессов. Расчет моментов времени, в которые необходимо прервать выполнение текущего потока и передать возможность выполнения другому потоку, называется планированием (англ. scheduling). Планирование потоков осуществляется на основе информации, хранящейся в информационных структурах (описателях, дескрипторах) процессов и потоков. При планировании учитываются приоритеты потоков, накопленное время выполнения, время ожидания в очереди, интенсивность обращения к вводу-выводу и другие факторы.

Диспетчеризация - это собственно сам процесс управления выполнением. т.е. реализация найденного в результате планирования решения. Диспетчеризация проходит в три этапа:

* сохранение контекста текущего потока;
* загрузка контекста потока, выбранного в результате планирования;
* запуск нового потока на выполнение.

Для общения друг с другом процессы могут использовать широкий спектр возможностей: каналы (pipe, в UNIX), почтовые ящики (mailbox, в Windows), вызов удаленных процедур (RPC, remote procedure call), сокеты (sockets). Эти механизмы характеризуются высокой степенью изоляции взаимодействующих сторон, что обеспечивает сохранность их данных и отсутствие проблем при совместном доступе к ресурсам ОС. К сожалению, эти средства взаимодействия не могут похвастать высокой производительностью.

В то же время, потоки выполняются в рамках одного процесса и имеют полный доступ ко всем его ресурсам. С одной стороны, это обуславливает высокую скорость доступа к общим данным, с другой – влечет за собой проблемы совместного доступа.

Очень важно согласовать обращение потоков к совместно используемым данным для предотвращения "состояния гонки" (когда несколько потоков пытаются изменить одну и ту же область данных или ресурс операционной системы). Типичная ситуация – когда один поток начинает читать данные из некоторого массива данных, ОС приостанавливает его выполнение и отдает процессорное время другому потоку, который производит запись в ту же самую область памяти. После этого ОС возвращает управление первому потоку, который продолжает считывание данных с того же места, на котором его прервали (и не знает о факте прерывания), а данные уже «испорчены» другим потоком.

Таким образом, синхронизация потоков является одной из важнейших функций подсистемы управления процессами и потоками. Современные операционные системы предоставляют множество механизмов синхронизации, включая семафоры, мьютексы, критические области и события (условные переменные).

Параллельное выполнение нескольких задач в рамках одного интерактивного приложения повышает эффективность работы пользователя. Например, при работе с текстовым процессором, пользователю удобно совмещать набор текста с такими продолжительными операциями, как проверка орфографии, разбиение на страницы, автоперенос по слогам или автосохранение. Современные электронные таблицы пересчитывают данные в фоновом режиме, если пользователь меняет значение какой-либо ячейки. В подобных случаях потоки используются как средство распараллеливания вычислений.

# 2. Стратегии управления вталкиванием, размещением и выталкиванием информации: краткая характеристика.

В течение всего процесса эволюции электронно-вычислительных машин, разработчики программного обеспечения сталкивались с проблемой нехватки оперативной памяти для размещения команд и особенно данных для обработки. Очень часто размер программы, которую хотелось бы загрузить в оперативную память для выполнения, превышает размер доступной физической памяти. Это приводит к поиску менее производительных алгоритмов, которые вынуждены либо подгружать новые данные по мере необходимости, либо вычислять их каждый раз из промежуточных данных. Иначе говоря, для сокращении требуемой алгоритмом памяти, приходится жертвовать скоростью его выполнения.

Одним из решений было разбиение целой программы на части, называемые оверлеями. Первый оверлей (с индексом 0) начинал выполняться первым. Перед своим завершением он запускал другой оверлей, и т.д. Все оверлеи хранились на диске. Операционная система управляла лишь их перемещением между оперативной памятью и диском, а разбиение программы на части и планирование их загрузки в память должен был осуществлять программист.

По мере совершенствования методов организации вычислительного процесса, появилась технология, известная под названием виртуальная память. Вообще, виртуальным называется ресурс, который пользователю или программе представляется обладающим свойствами, которыми он в действительности не обладает. Например, пользователю может быть предоставлена в распоряжение виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную оперативную память. При написании программ, разработчик не учитывает особенностей физического распределения памяти, а считает, что в его распоряжении имеется однородная память большого объема. В действительности же все используемые программой данные хранятся на одном или нескольких разнородных запоминающих устройствах - как правило, это физическая оперативная память и внешние накопители на магнитных дисках (т.н. файл подкачки). Непрерывные адреса большой виртуальной памяти отображаются в реальную память и, при обращении к ним программы, перенаправляются в нужную ячейку физической оперативной памяти, при необходимости подгружая в оперативную память необходимые блоки с внешних накопителей. Если при этом ограниченного объема физической оперативной памяти не хватает, часть её содержимого, согласно некоторой выбранной стратегии (как правило, менее всего или ранее всего использованные), выгружаются на внешний накопитель

Можно сказать, что концепция виртуальной памяти состоит в отделении адресного пространство (АП) процесса от адресов реальной оперативной памяти (РОП).

Управление виртуальной памятью имеет своей целью:

* облегчить жизнь программиста, сняв с него проблемы, связанные с ограниченным объемом РОП;
* обеспечить возможность создания и выполнения на ЭВМ программ максимально допустимого размера;
* устранить временн*у*ю фрагментацию, максимально эффективно используя ОП ЭВМ, которая образуется при отсутствии необходимого участка программы/данных в РОП и определяется временем, необходимым для размещения необходимых данных в РОП.

В ОС с реальной оперативной памятью, привязка адресного пространства к реальным адресам осуществляется статически до выполнения программы на все время выполнения.

В ОС с виртуальной памятью пространство процесса (образ процесса) во время выполнения хранится во внешней памяти ЭВМ и динамически загружается по частям в любое свободное место реальной памяти, по мере необходимости. Однако программа ничего не знает об этом – она написана и выполняется так, как будто полностью находится в РОП.

Виртуальная память представляет собой модель оперативной памяти, реализованную при помощи внешней памяти.

Механизм отображения (mapping) виртуальных адресов в реальные называется динамическим преобразованием адресов (ДПА).

Виртуальная память представлена в контексте выполняемой программы одним непрерывным блоком. Таким образом, при обращении к виртуальной памяти, кажущиеся смежными виртуальные адреса не обязательно будут являться таковыми в физическом адресном пространстве. Эта технология освобождает программиста от необходимости скрупулезно учитывать размещение, загрузку и выгрузку своих данных в РОП. Он имеет возможность писать программы наиболее естественным образом, концентрируя внимание на детали алгоритма и структуре программы и игнорируя особенности аппаратной архитектуры ЭВМ.

Механизм динамической адресации предполагает ведение таблиц, показывающих, какие ячейки ВП в текущий момент времени находятся в РОП и где именно. Поскольку индивидуальное отображение элементов информации (пословное или побайтовое) не имеет смысла (так как под таблицы отображения адресов потребовалось бы РОП больше чем под процессы), то отображение адресов выполняется на уровне блоков ОП, как изображено на рисунке 1.

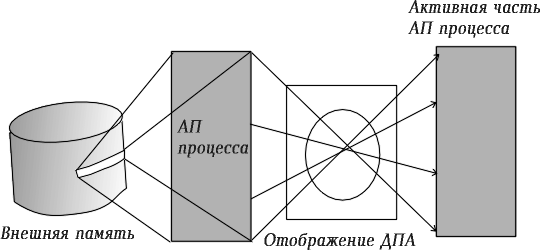


Рисунок 1. Динамическое преобразование адресов

При таком подходе возникают следующие вопросы:

* какую часть процессов держать в ОП, в некоторые моменты времени, выталкивая одни участки РОП и размещая другие;
* каким сделать размер блока.

Увеличение размера блока приводит к уменьшению размера таблицы отображения блоков, но увеличивает время обмена и, наоборот, уменьшение размера блока приводит к увеличению таблиц и уменьшению времени обмена с внешней памятью.

Блоки могут быть фиксированного размера (страницы) и переменного размера (сегменты). В этой связи существует четыре способа организации виртуальной памяти:

* динамическая страничная организация.
* сегментная организация.
* комбинированная сегментно-страничная организация.
* двухуровневая страничная организация.

Виртуальные адреса в страничных и сегментных системах являются двухкомпонентными и представляют собой упорядоченную пару (p,d), где p- номер блока (страницы либо сегмента), в которой размещается элемент, а d- смещение относительно начального адреса этого блока. Преобразование виртуального адреса V=(p,d) в адрес реальной памяти r осуществляется следующим образом. При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы отображения блоков данного процесса. В соответствии с номером блока p, из таблицы отображения блоков, считывается строка, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти. К считанному физическому адресу размещения выбранного блока добавляется размер смещения d и вычисляется требуемый реальный адрес, как показано на рисунке 2.

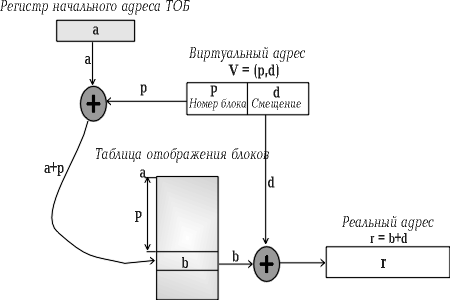


Рисунок 2. Преобразование виртуального адреса в реальной адрес памяти

Для управления виртуальной памятью существует три категории стратегий, направленных на снижение времени ожидания страниц и обеспечивающие расположение в памяти только используемых блоков.

**Стратегия вталкивания** определяет, когда следует переписать страницу/сегмент из внешней памяти в ОП.

*Вталкивание по запросу* – система ожидает ссылки на страницу или сегмент от выполняющегося процесса (прерывание по отсутствию страницы).

К преимуществам можно отнести минимальные накладные расходы на определение требуемых страниц, а также гарантию расположения в ОП только нужных страниц.

Недостаток этого подхода в том, что подкачка по одному блоку приводит к увеличению общего времени ожидания.

*Упреждающее вталкивание*предполагает, что система может предвидеть необходимость использования в дальнейшем страницы/сегмента. Если вероятность обращений к блоку высока и есть свободная ОП, то соответствующий блок переписывается в ОП.

Достоинство этой стратегии состоит в том, что сокращается время ожидания.

В настоящее время быстродействие аппаратуры увеличивается, и неоптимальные решения не приводят к существенному уменьшению эффективности вычислительных систем.

**Стратегия размещения** определяет, куда поместить поступающую страницу/сегмент. В страничных системах - тривиально: в любой свободный блок (страница имеет фиксированный размер). В сегментных системах те же самые стратегии, что и для РОП (в первую подходящую область, в наиболее подходящую, в наименее подходящую).

**Стратегия выталкивания (замещения)** определяет, какую страницу/сегмент удалить из ОП для освобождения места поступающей страницы.

Здесь основная проблема "пробуксовки", при которой вытолкнутая страница в следующий момент должна вновь размещаться в РОП.

Рассмотрим процедуры определения блоков для выталкивания из ОП.

а) выталкивание случайной страницы- в реальных системах не применяется;

б) выталкивание первой пришедшей страницы (FIFO - очередь) - для ее реализации необходимо устанавливать временные метки страниц;

в) выталкивание дольше всего неиспользованных страниц - для реализации необходимо поддерживать обновляемые временные метки, что обуславливает собой основной недостаток - существенные издержки, т.к. требуется постоянное обновление временных меток;

г) выталкивание реже всего используемых страниц - предполагает наличие счетчиков страниц (менее интенсивно, нежели обновляемые временные метки);

д) выталкивание не использующихся в последнее время страниц - самый распространенный алгоритм с малыми издержками, реализуется двумя аппаратными битами на страницу.