1 Управление процессами - основа функционирования ОС. Состояния процесса. Условия перехода из одного состояния в другое.

ПРОЦЕСС — ОСНОВА ФУНКЦИОНИРОВАНИЯ ВС

Процесс — это теоретическое понятие, на основании которого можно описать то, что происходит в системе при выполнении некоторых действий (программы).

Для пользователей знать, что происходит в системе при выполнении программы, не обязательно — наглядно показывает результат. Однако, для разработчиков систем необходимо знать, что происходит при выполнении той или иной операции, т.е. изменение состояния машины на физическом уровне. Понятие процесса, его развитие во времени и в пространстве в распределенных системах обработки информации помогают создать некоторую условную модель выполнения программы.

Состояние машины определяется состоянием процессора (содержимым адресуемых и внутренних регистров) и состоянием памяти (содержимым ячеек памяти). Это состояние меняется при выполнении процессором некоторых операций. Выполнение операции — это некоторое действие, результатом которого является переход за конечное время из начального состояния машины (до выполнения операции) в конечное (после ее выполнения). В глобальном масштабе такой операцией может быть и вся программа. Однако, на таком уровне мы не можем ввести понятие процесса. Тогда рассмотрим каждую такую операцию как неделимую, т.е. выполняемую процессором за один шаг (единицу времени, машинный такт).

С этой точки зрения программу можно рассматривать как последовательность элементарных (неделимых) операций. Будем отслеживать выполнение программы в течение времени T (от начала до конца выполнения). За это время процессор успеет выполнить N операций (N сколь угодно большое), которые будем идентифицировать по номеру (1,..,N).

В таких элементарных операциях можно выделить два момента: начало операции — H и конец операции — K. В эти моменты, времена которых обозначили соответственно tH и tK, будем отмечать состояния машины. Эти два момента считаются *событиями*. Таким образом, события позволяют отмечать изменения состояния машины.

При выполнении программы имеем последовательность операций 1,2,..,N, причем tH(1)<tK(1)=<tH(2)<...<tK(N) (хотя конец предыдущей операции и начало следующей — это в принципе одно и то же). Такая последовательность и называется *последовательным процессом* (или просто *процессом*). События H(1), K(1), H(2),..., K(N) образуют *временной след* (трассу или историю) процесса.

Таким образом, можно выделить три наиболее употребляемых определения процесса.

Процесс :

- **это любая выполняемая работа в системе;**
- **это динамический объект системы, которому она выделяет ресурсы;**
- **это траектория процессора в адресном пространстве ВС.**

Управление процессами, или Всё о PCB

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью специальных структур данных, образующих окружение процесса, среду исполнения или образ процесса. Образ процесса состоит из двух частей: данных режима задачи и режима ядра. Образ процесса в режиме задачи состоит из сегмента кода программы, которая подчинена процессу, данных, стека, библиотек и других структур данных, к которым он может получить непосредственный доступ. Образ процесса в режиме ядра состоит из структур данных, недоступных процессу в режиме задачи, которые используются ядром для управления процессом. Оно содержит различную вспомогательную информацию, необходимую ядру во время работы процесса.

Каждому процессу в ядре операционной системы соответствует блок управления процессом (**PCB** – process control block). Вход в процесс (фиксация системой процесса) –это создание его блока управления (PCB), а выход из процесса – это его уничтожение, т. е. уничтожение его блока управления.

Таким образом, для каждой активизированной задачи система создает свой PCB, в котором, в сжатом виде, содержится используемая при управлении информация о процессе.

PCB – это системная структура данных, содержащая определённые сведения о процессе со следующими полями:

1. Идентификатор процесса (имя);
2. Идентификатор родительского процесса;
3. Текущее состояние процесса (выполнение, приостановлен, сон и т.д.);
4. Приоритет процесса;
5. Флаги, определяющие дополнительную информацию о состоянии процесса;
6. Список сигналов, ожидающих доставки;
7. Список областей памяти выделенной программе, подчиненной данному процессу;
8. Указатели на описание выделенных ему ресурсов;
9. Область сохранения регистров;
10. Права процесса (список разрешенных операций);

Состояния процессов

Во время своего существования процесс может находиться в четырех состояниях: подготовленном, готовом, активном и заблокированном (рис.2.10). В каждый конкретный момент времени процесс находится в одном из этих состояний, которое фиксируется в блоке управления процессом (PCB).

Процесс находится в подготовленном состоянии, если он находится в системе (как правило на внешнем носителе информации), но ему не выделены ресурсы системы.

Процесс находится в готовом состоянии или активизирован, если ему уже выделены ресурсы (программа, необходимая для выполнения, находится в оперативной памяти), однако ему не выделено время процессора для выполнения (процессор занят другим процессом).

Процесс, которому выделяется время процессора, переводиться в активное состояние или состояние выполнения.

Во время выполнения процесса ему могут понадобиться дополнительные ресурсы, но для их получения необходимо некоторое время. Т.е. процесс должен ожидать наступления некоторого события или окончания выполнения конкретной операции (например, ввода/вывода для получения/выдачи данных). Такой процесс переводится в заблокированное состояние.

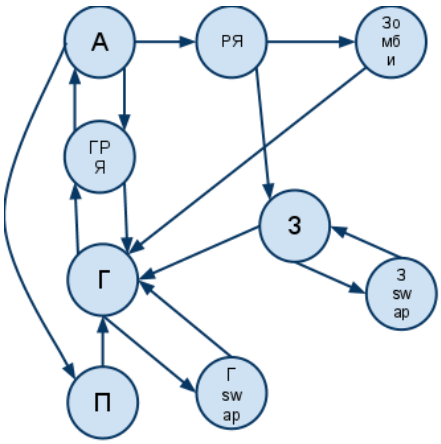
Процессы при поступлении в систему находятся в подготовленном состоянии и накапливаются во входных очередях заданий. Следует различать процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и процессы, находящиеся в подготовленном состоянии после загрузки операционной системы. Процессы первого вида, как правило, принадлежат пользователям, а второго вида ОС. Такие процессы находятся в подготовленном состоянии до тех пор, пока для выполнения функций системы они не будут активизированы. Пользовательские процессы активизируются или делается попытка их активизации при наличии в системе ресурсов. Попытка активизации процесса производится системой при наличии в системе оперативной памяти, достаточной для размещении программы, подчиненной процессу.

Поэтому в системе может находиться одновременно довольно много подготовленных и готовых процессов, для которых организуются очереди (одна для готовых и одна для подготовленных процессов). Дисциплины управления ими могут быть различными и выбор дисциплины обслуживания определяется требованиями, предъявляемыми к системе планирования.

Если система определила необходимость активизации процесса и выделяет нужные ему ресурсы, кроме времени процессора, то она переводит его в готовое состояние. При этом система определяет (если может) имеются ли в системе ресурсы, необходимые для выполнения данного процесса. Процесс, который переводится в готовое состояние, должен иметь больший приоритет, чем процессы, требующие таких же, как и он, типов ресурсов (т.е. он находится в начале очереди

подготовленных процессов). При этом сортировку по приоритетам процессов в очереди и планирование перехода в готовое состояние (следя за освобождением ресурсов) должен осуществлять планировщик данной очереди.
Если процессор освободился, то первый (наиболее приоритетный) процесс из очереди готовых процессов получает время процессора и переходит в активное состояние. Выделение времени процессора процессу осуществляет диспетчер.
Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.
Процесс, который требует дополнительных ресурсов, кроме времени процессора, либо выполнения некоторой операции другим процессом, переводится в заблокированное состояние и ожидает наступления события, которое он потребовал. Как только произошло ожидаемое событие, процесс переводится в готовое состояние.
Если процесс требует действия или ресурса, которых в данный момент операционная система не может выполнить, он переводится в подготовленное состояние и считается приостановленным.
Задача ОС заключается в слежении за состоянием всех процессов, находящихся внутри системы, особенно в активизированном состоянии, поскольку в активном состоянии процесс является монополистом по отношению к процессору. Поэтому ему и выделяется квант времени, чтобы процесс не блокировал процессор или не занимал его на длительное время, иначе могут возникнуть тупиковые ситуации. В некоторых ОС процесс, требующий наименьшего времени обслуживания на процессоре, имеет наибольший приоритет. Тогда быстрые заявки быстрее покинут систему, а процессы, требующие на обслуживание довольно большого кванта времени при большой загрузке системы вообще могут не обслужиться. С этой точки зрения наиболее приоритетными оказываются системные процессы, так как в ОС все заложено для их быстрого выполнения.
Приоритеты программ обработки прерываний зависят от употребляемости соответствующих ресурсов. Наибольший приоритет имеет программа обработки прерывания по таймеру (вызывается 18,2 раза в секунду) и наименьший прерывание по вводу/выводу. Пользовательские процессы имеют различные приоритеты в зависимости от порождающих их процессов и от времени выполнения на процессоре. Также необходимо следить за очередью блокированных процессов, с тем, чтобы среди них не было "бесконечно" ожидающих процессов. Вполне возможно, что событие, которое они ожидают, вообще может не произойти тогда их надо вывести из системы. Также может оказаться, что блокированный процесс имеет слишком низкий приоритет и он может бесконечно долго находиться в режиме бесконечного откладывания в очереди блокированных процессов тогда необходима система динамического, адаптивного изменения приоритетов.
В общем случае, система должна следить за процессами в очередях готовых и заблокированных процессов, чтобы не было бесконечно ожидающих процессов или процессов, монопольно удерживающих выделенные им ресурсы. Из сказанного следует, что эффективность системы управления процессами в значительной степени влияет на характеристики ВС, связанные с пропускной способностью. Плохая организация управления процессами может привести к тому, что некоторые процессы могут так и остаться в подготовленном состоянии.

Состояние процесса



Есть два рассказа о состоянии процессов. Простой и сложный. Для сложного используется картинка выше (9 состояний). Для простого - ниже. Симоненко больше любит сложный (а шо поделать).

Итак. Сложный вариант.

П – подготовка. На этом этапе лежат задания. Т.е. программа (что делать) и данные (над чем делать). Заданию ещё не выделили ресурсы. Когда ему планировщик выделит ресурсы, задание станет процессов и перейдёт в готовое состояние.

Г – готовность. Процесс размещен в ОП, ему выделены ресурсы, сформирован PCB. Так может случиться, что процесс своппируют на диск вместе с его ресурсами (ну, разве что ОП из ресурсов вычеркнут).

Гswap – процесс готов, но своппирован на диске.

ГРя – готовность в режиме ядра. Это некое абстрактное состояние, когда процесс уже имеет ресурсы, но ещё не допущен к процессору. С этого состояния процесс может или получить долгожданный доступ к процессору, или вернуться в очередь готовых.

А – активность. Процесс занял процессор и, собственно, выполняется. Если у процесса окончились выделенные ему кванты времени, он возвращается в ГРя. Если внезапно окончились его ресурсы – в очередь подготовленных. Если произошёл системный вызов – в Ря.

Ря – режим ядра. Сюда попадает процесс, пока выполняется системный вызов (например, ядро открывает файл). Процесс может поступить к заблокированным, или стать зомби.

Зомби – процесса в системе нет, но его PCB ещё есть. Такая ситуация возникает, когда процесс завершился, а породивший его процесс ещё не знает об этом.

Заблокированные процессы находятся в состоянии ожидания. Кому не повезёт – освободят ОП и будут своппированны на диск.

Зswap – процесс с диска можно опять вернуть в ОП в очередь заблокированных.

Если система определила необходимость активизации процесса и выделяет нужные ему ресурсы, кроме времени процессора, то она переводит его в готовое состояние.

Если процессор освободился, то первый (наиболее приоритетный) процесс из очереди готовых процессов получает время процессора и переходит в активное состояние. Выделение времени процессора процессу осуществляет диспетчер. В состоянии исполнение происходит непосредственное выполнение программного кода процесса. Выйти из этого состояния процесс может по трем причинам:

- операционная система прекращает его деятельность;
- он не может продолжать свою работу, пока не произойдет некоторое событие, и операционная система переводит его в состояние ожидание;
- в результате возникновения прерывания в вычислительной системе (например, прерывания от таймера по истечении предусмотренного времени выполнения) его возвращают в состояние готовность
- процесс не выполнился за выделенный ему квант времени, тогда он переходит снова в готовое состояние.

Из состояния ожидание процесс попадает в состояние готовность после того, как ожидаемое событие произошло, и он снова может быть выбран для исполнения.

Если процесс требует действия или ресурса, которых в данный момент операционная система не может выполнить, он переводится в подготовленное состояние и считается приостановленным.

В общем случае, система должна следить за процессами в очередях готовых и заблокированных процессов, чтобы не было бесконечно ожидающих процессов или процессов, монопольно удерживающих выделенные им ресурсы.

2 Модель непротиворечивости - причинная непротиворечивости

По традиции непротиворечивость всегда обсуждается в контексте операций чтения и записи над совместно используемыми данными, доступными в распределенной памяти (разделяемой) или в файловой системе (распределенной).

Модель непротиворечивости (consistency model), по существу, представляет собой контракт между процессами и хранилищем данных. Он гласит, что если процессы согласны соблюдать некоторые правила, хранилище соглашается работать правильно. То есть, если процесс соблюдает некоторые правила, он может быть уверен, что данные которые он читает являются актуальными. Чем сложнее правила - тем сложнее их соблюдать процессу, но тем с большей вероятностью прочитанные данные действительно являются актуальными.

- **причинная**

Модель причинной непротиворечивости (causal consistency) представляет собой ослабленный вариант последовательной непротиворечивости, при которой проводится разделение между событиями, потенциально обладающими причинно-следственной связью, и событиями, ею не обладающими. Если событие B вызвано предшествующим событием A или находится под его влиянием, то причинно-следственная связь требует, чтобы все окружающие наблюдали сначала событие A, а затем B.

Для того чтобы хранилище данных поддерживало причинную непротиворечивость, оно должно удовлетворять следующему условию: *операции записи, которые потенциально связаны причинно-следственной связью, должны наблюдаться всеми процессами в одинаковом порядке, а параллельные операции записи могут наблюдаться на разных машинах в разном порядке.*

Основная идея: Последовательность выполнения задач, которые связанны между собой (к примеру, одна задача пользуется результатами другой) должны быть одинаково видна всем процессам, причем задачи в ней должны идти в последовательности, которую налагает на них зависимость (то есть если сначала выполняется А и его результат передается Б, то все процессы видят последовательность А Б)

3 Применение ассоциативной памяти для повышения эффективности управления вычислительным процессом.

Системы с ассоциативной памятью представляют собой такие параллельные системы типа ОКМДР, которые оперируют с данными при доступе к ним при помощи тэгов пли выборки по содержимому ячеек памяти в большей степени, чем при помощи адресов. Ассоциативная память реализует принцип естественного параллелизма, за счет этого улучшение вычислений. Применяется в машинах Data flow. Но эта память дорогостоящая. Пример машины с ассоц. Памятью – STARAN.

В машинах преобразование адреса идёт на обратном уровне. В машинах используются двухуровневые кэши. Один из кэшей является ассоциативной памятью, в которой реализуется TLB-буфер (буфер быстрого преобразования адреса). Ассоциативная память ведёт обращение не по адресу, а по содержимому.

Для ускорения поиска в таблицах, используют аппаратные средства, а именно Translation Lookaside Buffer - TLB. Это ассоциативная память, в которую копируется часть, наиболее интенсивно используемых страниц (а такое множество можно выделить, исходя из принципа локальности). Сначала соответствующая страница ищется в TLB и только в случае если ее там нет, начинается поиск в таблицах страниц.

Компьютер снабжается небольшим аппаратным устройством, служащим для отображения виртуальных адресов в физические без прохода по таблице страниц, называемое буфером быстрого преобразования адреса (TLB - Translation Lookaside Buffer) или ассоциативной памятью. Оно обычно находится внутри диспетчера памяти и состоит из некоторого кол-ва записей.Каждая из них содержит информацию об одной странице: номер вирт. страницы, бит изменения, код защиты, и номер физ. блока, в котором находится эта страница. Когда виртуальный адрес представляется представляется диспетчером памяти для отображения в физический, аппаратура убеждается, что его номер находится в буфере TLB путем одновременного сравнения со всеми записями. Если найдено совпадение, то страничный блок берется прямо из буфера TLB, не выполняя поиска в таблице страниц.

Если адрес не найден в буфере быстрого преобразования адреса, выполняется поиск в таблице страниц. Затем он удаляет одну из записей из буфера TLB, заменяя ее только что найдена.

Короче, смысл тот же, что в кэш-памяти - очень быстрый доступ к небольшому объему частоиспользуемых страниц

4 Физические и логические единицы информации работы файловых систем.

Иерархии данных

Файл – именованная логически связанная совокупность данных,которая с т.з. пользователя представляет собой единое целое.

Логическая иерархия данных

- базы данных / базы знаний
- файловые системы – совокупность файлов
- файл – совокупность записей
- запись – совокупность полей, логическая единица объёма данных, к которым можем обратиться за 1 обращение к памяти
- поле – смысловая единица инфы, различаемая на уровне программы, совокупность идентификаторов (символов)
- символьный набор – внутренний, определяются кодировкой внутри машины; внешний определяет общение между машинами; совокупность байт
- байт (символ) – последовательность бит
- бит – неделимая единица инфы

Физическая иерархия данных

- том – объём инфы, доступный одному устройству чтения/записи (“что можно открутить и унести”). Бывают многофайловые тома и многотомные файлы. Совокупность физ. записей

- запись – объём данных, записываемых/считываемых за 1 раз с диска. Объём физ. и лог. записей может не совпадать (физ. больше). Для связи физ-лог записей используют операции блокирование и разблокирование записи. Запись = кластер, совокупность секторов, это наименьшее место на диске, которое может быть выделено для хранения файла.
- сектор – наименьший физический блок памяти на диске, определённого размера (обычно 512 байт).

5 Работа системы при подготовке и выполнении команда ввода вывода.

Есть три фундаментально различных способа осуществления операций ввода- вывода: программный ввод-вывод, ввод- вывод, управляемый с помощью прерываний, и ввод-вывод, использующий DMA.

Как обрабатывается прерывание ввода/вывода (из шпоры)

1. Обращение к супервизору
2. Сохранение текущего PSW в старый PSW. Выполнение дешифрации прерывания
3. Сохранение нового PSW в текущий
4. Переход по SVC на обработчик прерывания
5. Подготовка операций ввода/вывода
6. В адресное слово канала записывается адрес начала канальной программы
7. Запуск канальной программы
8. Команда SIO
9. Передача ус-вом сигнала об усп./неусп. старте – в слове сост. Канала CSW.
10. Обработчик прерываний запрашивает ответ о успешном старте
11. Ввод-вывод
12. Обработчик анализирует CSW на сигнал о завершении ввода/вывода, записывает старый PSW в текущий

Все. Дальше – продолжение основной программы

Существенная часть операционной системы занимается вводом-выводом. Операция ввода-вывода может выполняться тремя способами. Во-первых, при помощи программного ввода-вывода, при котором центральный процессор вводит или выводит каждый байт или слово, находясь в цикле ожидания готовности устрой ства ввода-вывода. Второй способ представляет собой управляемый прерывания ми ввод-вывод, при котором центральный процессор начинает передачу ввода-вывода для символа или слова, после чего переключается па другой процесс, пока прерывание от устройства не сообщит ему об окончании операции ввода-вывода. Третий способ заключается в использовании прямого доступа к памяти (DMA), при котором отдельная микросхема управляет переносом целого блока данных, и инициирует прерывание только после окончании операции переноса блока.

Ввод-вывод можно разбить на четыре уровня иерархии: процедуры обработки прерываний, драйверы устройств, независимое от устройств программное обеспе чение ввода-вывода и библиотеки ввода-вывода и спулеры, работающие в пространстве пользователя.

Драйверы устройств управляют деталями работы устройств и предоставляют однородные интерфейсы к остальной части операционной системы. Независимое от устройств программное обеспечение ввода-вывода занимается буферизацией и сообщением об ошибках.

Компилятор перед любой командой в/в ставит системный взвод
(с точки зрения машины это прерывание SVC-обращение к супервизору). Система должна среагировать на прерывание при этом сбрасывается текущее значение PSW в старое.
-передаем параметры в канальную программу
-передача адресов ВУ(Лукас абонентская система)
-выдвча команды в/в
-проверка канального слова
-прога порашивает словосостояние канала на подтверждение нормального старта.
-возвращаем в текущее PSW старое

-по завершению это фиксируется в CSW и идет физ. Прерывание по в/в.
-проверка CSW и переход на след. Команду.