10.12.20 12:24

-Я вам дам свой телефон, вы мне позвоните?

-Там еще Силантьева будет.

Неименованный прогрманные каналы - базовое средство взаимодействия процессов, потоковая передача данных. ПК обеспечивают симпликсную (одностороннюю) связь.

Двусторонняя свзяь - дуплексная.

02:31

**Очереди сообщений.**

В распределенных системах вся передача данных - с помощью сообщений. Очереди сообщений - взаимодействие на отдельной машине, характерны и для распределенных системах.

Таблица очередей сообщений - системная таблица - содержит дескрипторы всех очередей сообщений в системе.

Связанный список типа очередь, каждый элемент указывает на следующий.

Дескриптор очереди сообщений - struct msgid\_ds. Элемент очереди не содержит текста сообщения, содержит ссылку на текст сообщения в области данных ядра системы.

Сообщение имеет тип и текст.

Когда процесс передает сообщение в очередь, ядро создает для него новую запись и помещает ее в конец связного списка соотв записей.

В каждой такой записи указывается тип сообщения, число байт, указатель на другую область данных

Ядро копирует сообщение из пространства отправителя  в область данных ядра системы, чтобы процесс отправитель мог завершиться, а его сообщение осталось доступным для чтения другим процессам.

Когда процесс выбирает сообщение из очереди, ядро копирует это сообщение в адресное пространство процесса получателя сообщения, после этого сообщение удаляется.

Процесс может выбрать сообщение несколькими способами:

    1) Взять самое старое сообщение, независимо от его типа

    2) Взять сообщение, если идентификатор сообщения совпадает с идентификатором, который указал процесс. Если сущ несколько сообщений с данным идентификатором, то процесс возьмет самое старое сообщение.

    3) Выбрать из очереди сообщение, числовое значение которого является наименьшим из меньших или равным значению типу указанного в процессе?Если этому условию соотв несколько сообщений в очереди, то процесс возьмет самое старое.

    КОгда сообщение выбрано из очереди, сообщение перестает существовать.

    На сообщениях определены следующиесистемные вызовы:

        1) msgget()

        2) msgctl()

        3) msgsnd()

        4) msgrev()

        Для сообщений характерно копирование. При посылке - сообщ копируется из адресного пространствапроцесса в адр пространство ядра. При получении, наоборот - сообщ копируется из адр простр ядра в адр простр процесса. В отличие от разделяемой памяти, так как разделяемй сегмент подключается в адресное пространтсво сегмента.

        Поэтому разделяемая память обеспечивает более быстрый способ взаимодействие - за счет отсутствия копирования.

        Когда процесс отправляет сообщение в очередь, сообщение копируется из адресного пространства процесса отправителя в адресное пространство ядра - процесс отправитель не должен блокироваться, дожидаясь получениясообщения другим процессом.

        Очереди сообщений позволяют устранить блокировку при посылки.

        Процесс не блокируется в ожидании получения сообщения - не блокируется при отправке, не блокируется при получении.

        16:00

**Пример работы с очередью сообщений.**

#include <stdio.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// определяем максимальный размер текста сообщения

#ifndef MSGMAX

#define MSGMAX 1024

#endif

struct mbuf

{

    long mtype; // тип сообщения

    char mtext[MSGMAX]  // сообщение

} mobj = {15, "Hello"};

int main()

{

    // file descriptor

    // очередь сообщений с id = 100

    int fd = msgget(100, IPC\_CREATE | IPC\_EXCL | 0642);  // 0642 - права доступа

* if (fd == -1 || msgsnd(fd, &mobj, strlen(mobj.mtext) + 1, IPC\_NOWAIT));
* perror("message");
* return 0;
* // IPC\_NOWAIT -- процесс не блокируется при посылке

}

 Очереди сообщений обеспечивают дуплексную связь одной очередью - в очередь можно писать, из очереди можно читать.  Кроме этого, имеется возможно извлекать сообщения из очереди не только по FIFO, но и используя систему приоритетов.

 28:43

**Тупики.**

 тупики тупики тупики

 Тупик - deadlock - тупик становится проблемой, когда он возникает(?).

 Одно из трех негативных состояний в системе.

 Рассмотрим простейший случай. Пусть два процесса монопольно владеют ресурсами r1 и r2. В процессе выполнения каждому из этих процессов требуетсяполучить дополнительно:

 \* первому r2

 \* второму r1

 Это можно записать следующей последовательностью запросов для каждого процесса (31:24)

 P1: ...

     запросил и получил R1

     запросил R2

  P2: ...

     запросил и получил R2

     запросил R1

    Процессы удерживают полученные ими ресурсы и запрашивают доп ресурс, который занят другим процессом. Процесс P1 не может получить R2, P2 не может получить R1 - оба процесса не могут продолжить своего выполнения, находятся в тупике.

    33:37

    Графовая модель Холта - двудольныйнаправленный граф.

    (петля запросов/замкнутая цепь запросов) -

    На рисунке граф тупика на единичных ресурсах.

   Если ресурсов n единиц, граф будет выглядеть - (35:53) , маленькие кружочки - единица ресурса.

   P1 может сформировать сообщение R3, получив сообщение R1 от процесса P3.

Аналогично для других процессов.

Такая схема расспронена в распределённых системах.

    43:08

**Типы ресурсов и тупики**

   В теории тупиков принято различать два типа ресурсов (эти типы обладают набором характерных особенностей, которые потребовали такой классификации):

       1) повторно используемые ресурсы

       2) потребляемые ресурсы

Наиболее подробно теория тупиков изложена в книге Шоу  "Логическое проектирования ОС"

  Глава тупики этой книги - самое полное описание утройства тупиков.

  Повторно используемые ресурсы - многократно используется, использование ресурса не изменяет качества и характеристик ресурса.

  К потребляемым ресурсам относится аппаратная часть - память, каналы, внешние устройства, процессор, сообщения.

  (Попробуйте прочитать из программного канала сообщение более одного раза, на экзе расскажете)

  К повторно использованным - реинтерабельный код системы, системные таблицы, разделяемая память, семафоры, программные каналы.

  Реинтерабельные коды - коды чистых процедур, использование процедуры не изменяет ее, такая процедура может быть использована несколькими процессамиодновременно.

  К потребляемым ресурсам относятся сообщения (получено -> перестало существовать).

  Сигналы не являются в чистом виде сообщениями - обладают своими свойствами, могут быть посланы целому ряду процессов.

  55:05

  4 условия возникновения тупика:

  1)  Взаимоисключение.

Возникает, когда процессы монопольно используют ресурсы. (mutual exclusion).

  2) Ожидание.

  Когда процессы удерживают занятые ими ресурсы, ожидая предоставления им дополнительных ресурсов. (Hold and wait).

  3) Неперераспределяемость.

  Когда ресурсы нельзя отобрать у процесса до их завершения, или как говорят,  добровольного особождения этих ресурсов. (надпись: ресурсы у процесса нельзя принудительно отобрать)

  (no preemtion?preemption)

  4) Круговое ожидание.

  Возникает замкнутая цепь запросов процессов на дополнительные ресурсы.

Опредление.

Тупиковая ситуация или тупик - это ситуация, которая возникает в результате монопольного использования разделяемых ресурсов, когда процесс,владея ресурсом, запрашивает другой ресурс, занятый непосредственно или через цепочку запросов другим процессом, ожидающим освобожденияресурса, занятого первым процессом.

  69:00

**Методы борьбы с тупиками.**

  Три основных подхода к тому, чтобы тупики в системе не возникали, или их можно было распознать и ликвидировать.

  1) Предотвращение или недопущение,или исключение (самой возможности возникновения тупика)

  2) Обход или недопущение

  3) Обнаружение и восстановление

  Стратегия Холендера.

он показал, что возникновение тупика невозможно, если нарушено хотя бы одно из 4 условий возникновения тупика.

  Способы исключения тупиков:

  1) Опережающее требование

  которое означает, что процесс до начала своего выполнения должен получить все необходимые и потенциально необходимые  (в процессе выполненияпроцесса есть вероятность, что ресурсможет понадобиться) ресурсы. Для этого процесс должен знать свою необходимость в ресурсах, запросить все ресурсы одновремнно и получить их до начала выполнения, если система обладает всеми требуемыми ресурсами.

**Вопрос**: какое из условий возникновения тупика исключает данный подход? (исключаем условие тупика номер 4(?))

  Недостатки - неэффективное использование реусурсов, так процесс получает ресурсы задолго до их использования (или вообще может их не использовать). Процесс должен знать свою максимальную потребность в запрашиваемых ресурсах (что в современных системах практически невозможно). Возможно бесконечное откладывание. Для повышения эффективности процесс может быть разделён на несколько задач, тогда ресурсы запрашивает каждая задача. Это позволяет сократить простой ресуров, но требует .... менеджера ресурсов.

  2) упорядочивание ресурсов. (Иерархическое распределение)

  Ресурсы делятся на классы каждому из которых присваивается номер.

        Процесс может запрашивать ресурсы только в порядке возрастания их номеров.

**Вопрос**: какое из условий возникновения тупика исключает данный подход?

(исключаем условие тупика номер 4(?))

        Если процесс пытается запросить ресурс с меньшим номером, чем номера процессов, которые он удерживает, то процесс должен освободить занимаемые им ресурсы, и запросить их в правильном порядке. Если процесс запрашивает ресурсы в порядке, обратном их нумерации, то это будет выглядет как опережающее требование. Но до того, как процесс запросит все необходимые ему ресурсы, он сделает n операция запроса/освобождения, пока не запросит все необходимые ресурсы.

        Несмотря на то, что запрашивание ресурсов в порядке их нумерации решает проблему тупиков, часто невозможно определить справедливый порядок нумерации. Справедливый = удовлетворяющий все выполняемые процессы. Это связано с тем, что в системе большое количество ресурсов, в том числе, системные таблицы, объекты ядра: семафоры, мьютексы, разделяемая память. Число ресурсов, которые потенциально может затребовать процесс и число вариантов их использования огромно - это основаная проблема для иерархического распределения ресурсов. Но исключать такую возможность нельзя - в хорошо изученных системах, с хорошо изученными процессами, к которым относятся системы разделения времени, такой подход может использоваться.

        3) Устранение  условия неперераспределяемости. Если запрос процесса на нек ресурс не может быть сразу удовлетворен, то процесс возвращает системе занимаемые им ресурсы (может быть часть). И затем опять пытается их запросить.

        В результате процесс запрашивает и освобождает одни и те же ресурсы - это увеличивает нагрузку на систему, ресурсы используются неэффективно .

        Обход (или недопущение)

        тупики возможны, но система анализа системы работает таким образом, чтобы

недопустить возникновение тупика.

         Алгоритм Банкира (97:00).

Правила Дейкстры выделения ресурсов.

Система удовлетворяет запрос процесса  на доп ресурс только тогда, когда её состояние после выделения этого ресурса останется надёжным.

Задача банкира решить кому выделять суду, а кому нет. Система делает примерно тоже самое.

        При выделении ресурсов процессам состояние системы останется надежным. В качестве банкира выступает менеджер ресурсов, заемщиками являются процессы, которые делают заявки на ресурсы. Основная идея алгоритма: сначала процесс делает заявку на необходимые ему ресурсы, в этой заявке процесс указывает свою максимальную потребность в каждом типе ресурсов.

        Сформировав такую заявку и начав выполняться, процесс не может затребовать ресурсов больше, чем указано в заявлке.

        Для успешного выполнения алгоритма необходимо выполнение следующих условий:

            1) Процесс не может требовать ресурсов больше, чем имеется в системе

            2) Процесс не модет получить ресурсов больше, чем указано в его заявке.

            3) Сумма всех удерживаемых единиц ресурса одного класса отражает количество распределенных единиц ресурса данного класса. И не может превышать общего число единиц ресура данного класса, имеющегося в системе.

Менеджер ресурсов работает следующим образом: каждый запрос проверяется по отношению к количеству свободных единиц ресурса в системе. При этом сравнивается количество свободных единиц ресурса и максимальная потребность процесса в единицах ресурса данного класса.

107:57

Пример (таблица)

\* таблица отражает рапределение одного типа ресурса (надёжное состояние)

проц | тек. распр-е | заявка |  свобод. ед.

P1       |            1              |      4         |

P2       |            3              |      5         |            2

P3       |            5              |      9         |

--------------------------------------------------------

Имея такую информацию менеджер ресурсов может найти такую последовательность процессов, что каждый процесс в последовательности может гарантированно завершиться. Такая ситуация в системе называется безопасной.

Процесс P2 не может запросить больше 2 единиц ресурса, потому что в заявке указал максимальную потребность 5, 3 единицы он удерживает - может гарантированно завершиться.

Завершившись, P2 вернет системе ресурсы, суммарно, они смогут удовлетворить макс потребность процесса P1. Завершившись, P1 вернет ресурсы, суммарно они смогут удв макс потребность процесса P3. Следовательно, существует последовательность процессов, в которой все процессы могут гарантированно завершиться.

\* таблица отражает рапределение одного типа ресурса (ненадёжное состояние)

проц | тек. распр-е | заявка |  свобод. ед.

P1       |            2              |      4         |

P2       |            3              |      5         |            1

P3       |            5              |      9         |

--------------------------------------------------------

         Ненадежная ситуация в системе

        Мы не можем найтипоследовательность процессов, которая может гарантированно завершиться. Если текущее состояние ненадежное, то оно не обязательно приведет к тупику, так как процессы могут не запросить максимального количества ресурсов, указанного в заявке.

Формальное определение безопасного состояния системы

Состояние системы явл безоасным, если существует последовательность процессов такая, что

1) Первый процесс последовательности обязательно завершиться, так как даже если он запросит максимальнозаявленное кол-во единиц ресурса, у системы имеется необходимое кол-во единиц ресурса для удовлетворения запроса этого процесса

2) Второй процесс может завершиться, если завершить первый процесс и вернет системе все занимаемые им единицы ресурсов, что в сумме со свободными единицами ресурса позволит удовлетворить максимальную потребнотсь в ресурсе второго процесса

...

i-тый процесс в последовательности может завершиться, если все предыдущие процессы в последовательности завершатся, вернут системе занимаемые ими ресурсы, и в результате суммарное количество свободных единиц ресурса может удовлетворить потребность i-того процесса в ресурсе.

Чтобы найти оптимальную последовательность, необходимо исследовать n! последовательностей(т.е.он скорее теоритический, а не практический)

возможно 17 будет не последняя лекция