Очереди сообщений.

Для ОС разд. времени характерно квантование проц. времени, т. к. они обязаны обеспечить гарантированное время ответа;

Message queue – очередь, куда кладутся сообщения.

Основа многопроцессных систем – защита адр. пр-в;

Мы уже изучили 2 средства передачи данных: программные каналы и сегменты разделяемой памяти;

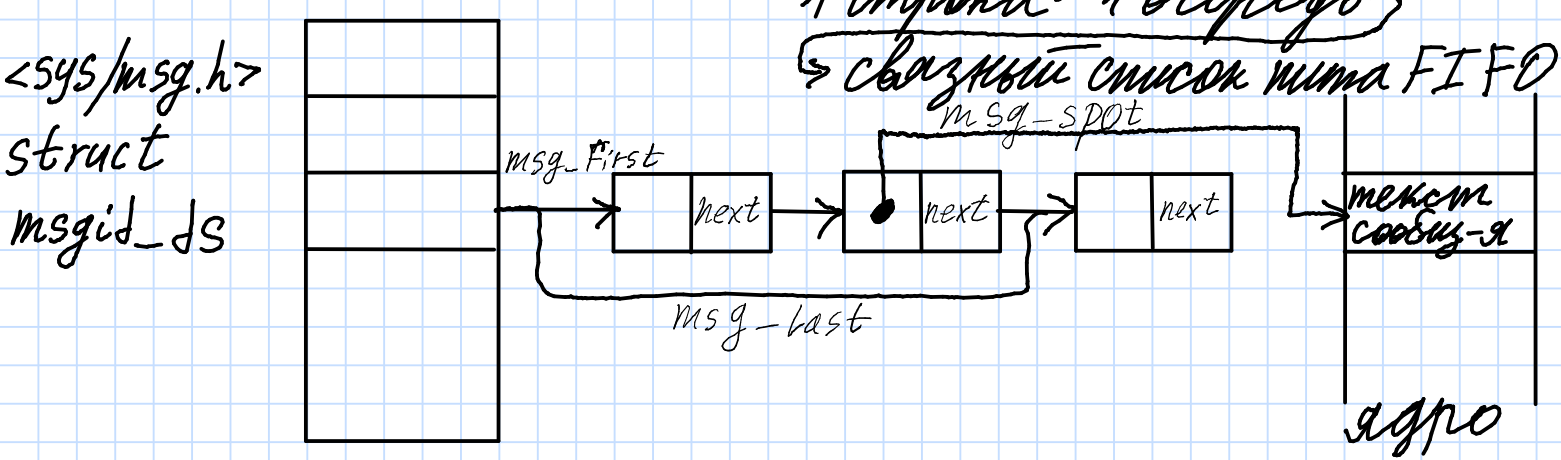
Прогр. каналы поддерживаются файловой подсистемой
Размер прогр. канала устанавливается в 1 стр. (мы не указываем)

Сегм. разд. памяти поддерживаются таблицей в обл-ти данных ядра ОС („ядрёной структурой");

Нет таблицы, описывающей прогр. каналы, они описываются в системе как файлы, у них есть inode.

Несмотря на то, что в UNIX всё файлы, не у всего есть inode;

В ядре имеется таблица очередей сообщ-й:



Это качественно новое средство взаимодействия параллельных процессов;
Можно предположить, что это шаг к распр. системам;

Диаграмма состояний процесса при передаче сообщений из лекции 8

Блокировки процессов предсказать невозможно — основная проблема такого взаимодействия;

Когда речь идёт о состояниях блокировки процесса, речь идёт о надёжной передаче сообщений, т.е. когда приём сообщения подтверждается (сообщением о получении) — надёжное сообщение

TCP — протокол надёжной передачи сообщений: в нём сделано всё для того, чтобы подтвердить установление соединения, и только после подтверждения начать передачу сообщений с контролем передачи данных;
UDP — „широковещательная рассылка"

3 сост-я блокировки возникают тогда, когда нужна надёжная передача данных;

Такая схема, в которой приём сообщений подтверждается, связана с непредсказуемым временем блокировки;
Блокировка — зло, но это необходимое зло.
От блокировок постоянно хотят избавиться;

На очередях сообщ-й определены след. систем. вызовы:

1) msgget(); - создать очередь сообщений;
2) msgctl();
3) msgsnd();
4) msgrcv();

(достаточный набор)

Здесь нет reply (хотя на лекциях обсуждался), но он есть, например, в QNX для того, чтобы подчеркнуть, что сообщение дву. ответом, т.к. для ОСРВ очень важно получить ответ на запрос (вся работа ОСРВ строится по принципу запрос-ответ);

На сообщениях определена структура

```
struct msgbuf
{
    long msgtype; //тип сообщ-я
    char msgtext[MSGMAX]; //текст сообщения
};
```

Сжатое изложение мануала:

Когда процесс передаёт сообщение очереду ядро создаёт для него новую запись и помещает её в конец связного списка записей данной очереди (очереди, которую указал процесс);

В каждой такой записи указывается тип сообщ-я, число байт в сообщении (длина) и указатель на область данных ядра, в которой фактически находится сообщение;

При этом ядро <u>копирует</u> сообщение из адр. пр-ва процесса-отправителя в своё адр. пр-во;

В рез-те процесс, отправивший сообщение в очередь сообщений может завершиться (по диаграмме сост-й), может не блокироваться при посылке;
Отправленное в очередь сообщ-е становится доступным другим процессам;

Когда какой-то процесс выбирает сообщение из очереди, ядро копирует его в адр. пр-во этого процесса. После этого запись удаляется.

Сообщение - потребляемый ресурс
(см. про тупики в лекции 11)

Процесс может взять сообщение из очереди 3 способами:

1) Процесс может взять из очереди (идентификатор которой он указал) самое старое сообщение (из головы);

Ни в коем случае не говорить здесь про время. В очередях сообщений время не указывается, оно тут ни при чём;

2) Процесс может взять сообщение, если его идентификатор совпадает с идентификатором, который указал процесс. Если существует несколько сообщений с таким идентификатором, берётся самое старое. При этом придётся сканировать очередь с головы.

В struct msgbuf идентификатор - msgtype;

3) Процесс может выбрать сообщение, числовое значение типа которого есть наименьшее из

меньших или равное значению типа, который указал процесс. Если этому условию удовлетворяет несколько сообщений, берётся самое старое.

Пример:

```
#include <string.h>
#include <sys/msg.h>
#ifndef MSGMAX
#define MSGMAX 1024
#endif
struct mbuf
{
        long mtype;
        char mtext[MSGMAX];
} mobj = { 15, "aaa" };
int main()
{
        int fd = msgget(100, IPC_CREATE|IPC_EXCL|
                        0642);
        if (fd==-1 ||msgsnd(fd, &mobj, strlen(mobj.mtext)+1,
                            IPC_NOWAIT);
                perror("msg");
        return 0;
}
```

процесс отправил сообщение в очередь и не ждёт блокировки (по диаграмме состояний), указывает об этом ядру и завершается. Сообщение отправлено в очередь и другой процесс сможет его получить.

Чем обусловлена популярность Windows?

Наличием „убойных" приложений: Word, Excel, Access.

Гейтс понял, что огромное число пользователей заинтересованы в одних и тех же средствах: текстовом редакторе, инструмент для составления

БД и электронные таблицы. И Гейтс их хорошо сделал. Adobe PhotoShop и т.д. – продолжение этой коммерческой идеи.
UNIX: они упустили эту тенденцию и продолжали ориентироваться на программистов, а не на обычных пользователей.

IBM поддерживает Linux – пишет ПО;

Многие аудио- и видеокарты не имеют драйверов под Linux, и самостоятельно их не написать, т.к. драйвер может написать только разработчик соответствующей системы, т.к. только он знает форматы передаваемых данных и особенности её работы.
Монополизм!

Функции обработчика прерывания от системного таймера в системах разделения времени.
2 части: ф-ции обработчика прерывания от системного таймера и пересчёт динамических приоритетов.
Книги:
1) Сальмон, Вустинович – Windows;
2) Вахалия – UNIX;

Функции обработчика прерывания от системного таймера и пересчёт динамических приоритетов.
в каждой ОС имеют особенности, поэтому пишем отдельно про Windows и UNIX;
Функции:

Если основной особенностью систем разделения времени явл. квантование, то главная задача обработчика прерывания от системного таймера –
– декремент кванта;

I) по тику:
– декремент кванта (чтобы определить время до его истечения, а затем диспетчер (модуль ядра) передаёт квант другому процессу по приоритетам); см. тему „планирование и диспетчеризация" в лекциях
планирование процессов – постановка процессов в очередь (в первую очередь к процессору за получением проц. времени, а вообще в системе много очередей);
Когда истекает квант, поздно планировать.
В таком случае система должна как можно быстрее его передать, поэтому

II) по главному тику:
– вызов планировщика – это в UNIX, а в Windows нет отдельного модуля планировщика, а ф-ции планирования время от времени выполняют разные участки кода ядра;

III) по кванту;

Обработчик прерывания от сист. таймера выполняется на высочайшем уровне приоритета, никакой код не может его прервать, он будет выполняться от начала до конца ⟹ он должен завершаться как можно быстрее, иначе никакая другая работа в системе будет невозможна;

(выше только power – сбои питания, аварийное завершение)

Код, вызывающий подпрограмму, не завершается. Вызов - переход на другой участок кода, запоминающийся в стеке, после чего происходит возврат на следующую команду (точка возврата);

Обработчик прерывания от системного таймера только инициализирует отложенные (разными способами) действия. (обработчик int 8h не отключает мотор дисковода, а только инициализирует его, а отключением занимается уже контроллер)

Инициализация - отправка сигнала или постановка в очередь (в Windows это DPC);

В Windows есть один уровень привилегий ядра - нулевой. Дальше в ядре все действия выполняются по IRQL (Interrupt Request Level);

(см. таблицу с IRQL из книги Соломона и Руссиновича)

В UNIX существует система приоритетов, в которой все приоритеты поделены на 2 части: приоритеты ядра и приоритеты user'а, т.е. в ядре действия выполняются по приоритетам: чем меньше значение, тем выше приоритет;

Пересчёт динамических приоритетов.

Пересчитываться могут только приоритеты пользовательских процессов, только они м.б. динамическими;

Это делается для того, чтобы исключить бесконечное откладывание процессов: в Windows это делает-

ся сканированием очереди готовых процессов („на коленке"), а в UNIX приоритеты пересчитываются по формуле (см. вахалию, формула старая, но полезная, т.к. учитывает время ожидания процесса и полученное процессорное время);

В Linux совсем другой алгоритм планирования (к/ч дерево, самый справедливый алгоритм планирования)

Приоритет процесса также зависит от того, на чём был блокирован процесс (в Соломоне, Русиновиче есть таблица с повышениями приоритетов при различных блокировках в Windows); (в книге Робачевского есть аналогичная таблица с приоритетами сна в UNIX);

Таблица и её анализ должны быть в отчёте + вывод (3-4 основные фразы по анализу)

Л/р: Читатели - писатели под Windows. Пишем по Дейкстре Хехтеру на потоках (create_thread);
3+ писателей
4+ читателей
Потоки разделяют адресное пр-во процесса, а конкретно одну переменную, которую писатели инкрементируют, а читатели читают;
4 ф-ции по монитору Хоара;

Прочитать в Рихтере введение к главе про потоки, там есть удачный пример распараллеливания процесса (на примере текстового редактора + еще чего-то);

Задача полностью решается на объектах ядра (event);

2 типа event:

1) с автосбросом;
2) со сбросом вручную;

Уметь объяснить, почему в конкретном месте исп. тот или иной тип события.

Притянуть за уши в код mutex (он не нужен, но надо с ним познакомиться);

В Windows есть 2 сист. вызова: wait и release;

2 типа wait:

- wait for single object;
- wait for multiple object;

wait for single object mutex — release mutex;
wait for multiple object mutex;

Mutex захватывается и освобождается в одной и той же ф-ции!

С помощью Mutex мы можем сделать часть кода ф-ции или всю функцию неделимой;
Это будет означать, что другой процесс не сможет выполнить эту же часть кода, пока 1-й процесс не завершит её выполнение;

Exclusive

Л/Р – RPC („high class", {распр. системы})

↳ это не посл-ть сист. вызовов, а композиция;

см. методичку

RPC выполняется вне протоколов

RPC можно сравнить с сокетами, т.к. и то, и другое – средство взаимодействия процессов в распр-ных системах, но они совершенно разные;

По опыту сдачи лабы:

Библиотека rpcgen (системное средство) генерирует пол-ть файлов:

- скелеты для клиента и для сервера (код, который надо модифицировать по заданию);
- заглушки (stub) для клиента и для сервера;
- заголовочный файл, описывающий функционал сервера;
- Makefile;

Генерация кода осуществляется по файлу конфигурации *.x;