Модели ввода-вывода.

Flashback с л/р №2 (см. конспект)

Когда идёт речь о прерываниях, в первую очередь имеются в виду прерывания от внеш. устр-в (устр-в ввода-вывода)
Ввод-вывод - работа с внешними устройствами;

Выполнение операций ввода-вывода сопровождается обращениями к оперативной памяти;

С оперативной памятью мы работаем при помощи команды mov; {в системе есть локальная шина памяти}

А с внеш устр-вами - через порты ввода-вывода командами in и out

{Memory mapping, I/O mapping}

3-шинная концептуальная архитектура
↳ концепция, а не аппаратная реализация

В системе сигналы делятся на 3 типа;
Данные, адреса и сигналы управления - качественно разная информация;

{Архитектура Фон Неймана: хранимая программа}

Процессор собственной памяти не имеет и постоянно обменивается данными с оперативной памятью;
Данные - команды и собственно данные (операнды);

Команды (данные для системы) передаются по шине данных;

В системе всё адресуется;

60-е – начало 70-х – появление интегр. микросхем, позволивших значительно увеличить мощности выч. систем, уменьшить габариты;

Это было реализовано IBM 360 и IBM 370;
(не одна машина, а серия машин)

В этих машинах была реализована концепция распараллеливания функций;

Т.к. процессор всегда был самым быстродействующим устр-вом системы и это стало отличительной особенностью IBM 360 и IBM 370;

Внешние устр-ва – механика, медленные устр-ва;

Чтобы более эффективно исп-ть проц. время, была реализована архитектура с распараллеливанием ф-ций: в состав IBM 360 были включены специальные процессоры (каналы), которые взяли на себя ф-цию упр-я внеш. устр-вами.

Т.о. процессор был освобождён от упр-я внеш. устр-вами, но это потребовало реализации в системе полноценной системы прерываний: процессор инициализирует операцию ввода-вывода, а управляет ей канал;

По завершении операции ввода-вывода прерывание информирует процессор о завершении данной операции – это основа функционирования, в общем, любой системы;

В ПК на базе проц. Intel-
– шинная архитектура.
(А архитектура мейнфреймов IBM наз. канальной)

В шинной архитектуре внеш. устр-ва упр-ляют специальные устройства – контроллеры;

Спрашивать на экзе будут в комплексе

5 Моделей ввода-вывода (с т.з. программиста): (американский материал (Стивен?))

Файлы предназначены для долговременного хранения данных, а массивы (структуры) – для оперативного доступа к этим данным

Модели предназначены для целостного представления о том, как м.б. организован ввод-вывод;

1) Блокирующий ввод-вывод (запрос на получение данных (read))
Используется ключевое слово receive from-
– общее средство передачи данных в системе

Передача данных с помощью сообщений – самый общий

способ, который может рассматриваться как на отдельно стоящей машине, так и в распределённой системе;

receive from = запрос на получение данных = read;

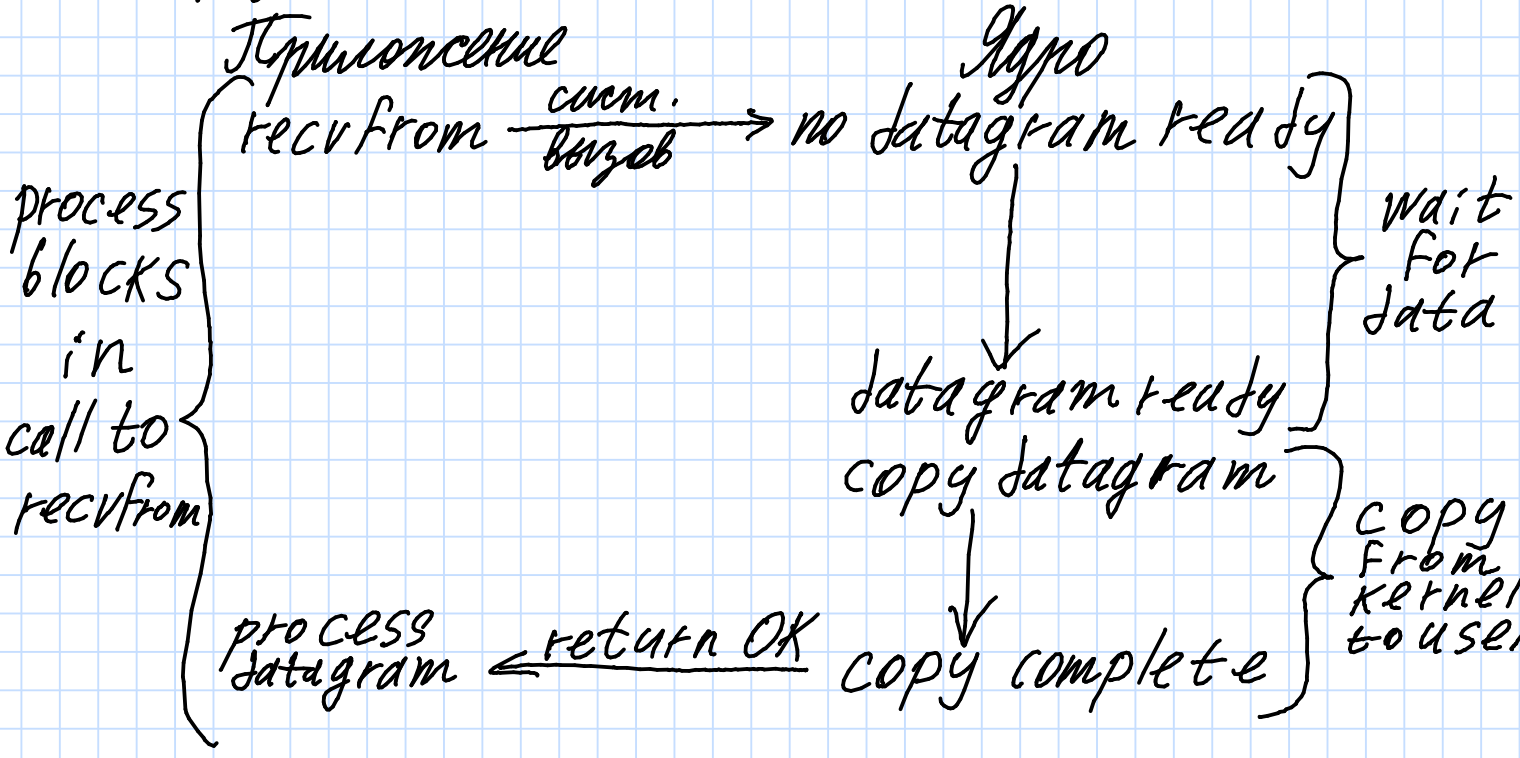
(сист. вызовы read/write - к ним всё сводится)

send to / receive from, как и read/write, переводят систему в режим ядра, т.к. это системные вызовы;

Процесс, выполнивший сист. вызов ввода-вывода (не важно, read или write), будет ожидать получения данных (система вернёт либо код ошибки, либо значение, соответствующее успешному выполнению сист. вызова);

Т.е. процесс в любом случае получит данные (в случае read/receive from это особенно явно);

Блокирующий ввод-вывод (blocking):



Когда приложение выполняет запрос ввода-вывода (в примере ввод - receive From), процесс блокируется до тех пор, пока не получит данные;

Что значит datagram ready (данные готовы)? Если мы вводим символ с клавиатуры, то он „готов" тогда, когда он введён в буфер клавиатуры;

Далее он копируется в буфер ядра, и уже оттуда в буфер приложения, после чего он сможет продолжить своё вып-е;

Диаграмма из книги Шоу (очень старая: 60-е-70-е) „Логическое проектирование операционных систем":

Запрос приложения на ввод-вывод:

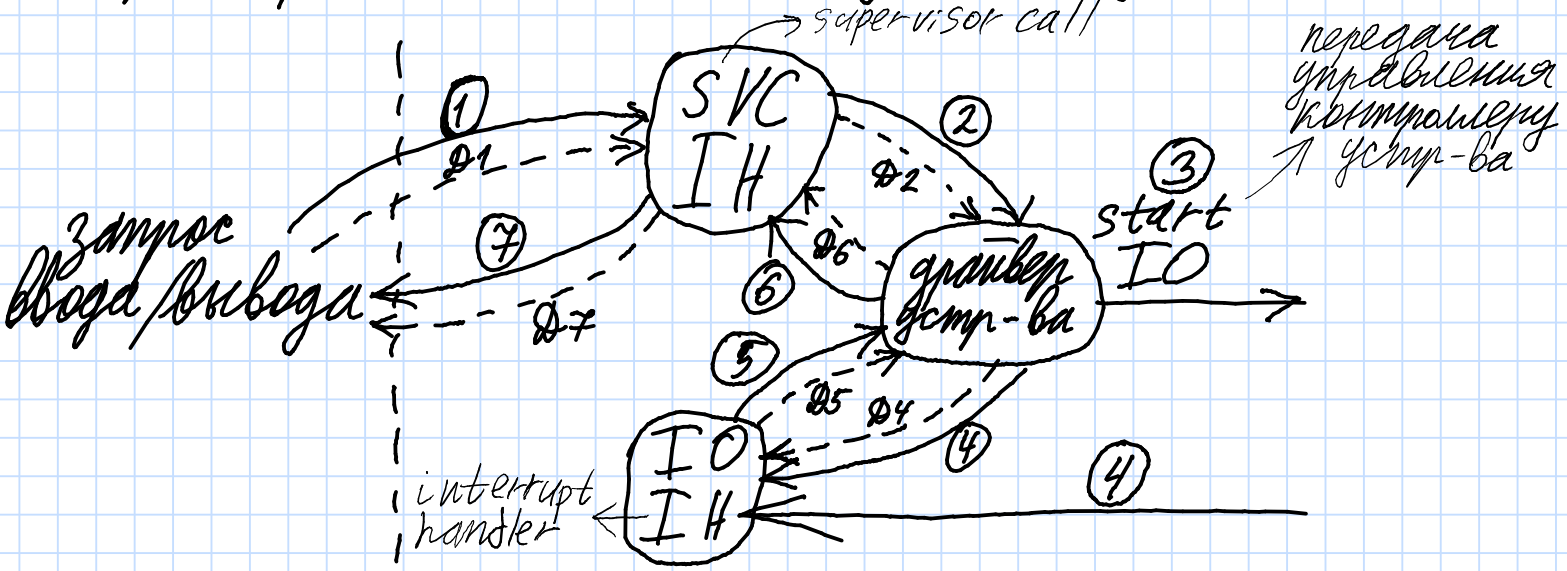


Диаграмма демонстрирует посл-ть действий системы, которые выполняются для обслуживания запросов ввода-вывода;

supervisor - „супернадсмотрщик", имеющий суперправа;

ОС, когда она выполняется, наз. супервизором;

supervisor call interrupt handler

||

В рез-те сист. вызова система переходит в режим ядра

В расшир. пол-ти вызовов есть ф-ции, составляющие интерфейс ядра. Эти ф-ции вызываются пос-тельно в зав-ти от флагов, установленных при вызове, и типа данных;

В рез-те пос-тельных вызовов ф-ций ядра ОС будет вызвана точка входа драйвера устройства;

Драйвер устройства - ПО, предназначенное для упр-я внеш. устр-вами;

Драйвер - многовходовая программа, т.е. имеет несколько точек входа;

Одной из точек входа драйвера явл. обработчик прерываний;

Когда вызывается драйвер, вызывается его точка входа;

В рез-те выполнения кода драйвера по шине данных контроллеру устройства будет послана команда;

Операцией ввода/вывода управляет драйвер устройства;

По завершении операции ввода-вывода

данные записываются в буфер устройства;
(например, когда мы отпустили клавишу,
в буфер клавиатуры запишутся данные)

После того, как данные поступили в буфер устройства, контроллер сформирует сигнал прерывания, который поступит на контроллер прерываний

По схеме трёхшинной архитектуры из л/р 2:
В рез-те будет сформирован сигнал INT,
процессор пошлёт в ответ сигнал INTA,
и контроллер выставит на шину данных
вектор прерывания, который в реальном режиме
будет являться смещением в таблице векторов прерываний.

Далее в обратном порядке (т.к. обработчик прерываний явл. одной из точек входа драйвера)
(в драйвере имеются callback-функции)
в рез-те посл-ти вызовов данные будут помещены
в буфер устройства.

Т.о. весь ввод-вывод, который мы реализовывали
в своих программах, явл. блокирующим.
(пока процесс не получит данные, он будет заблокирован

В л/р 3 (exec) мы видели именно
такие блокировки (если программы
затрагивали ввод/вывод);
Мы видели, что процесс, запрашивающий

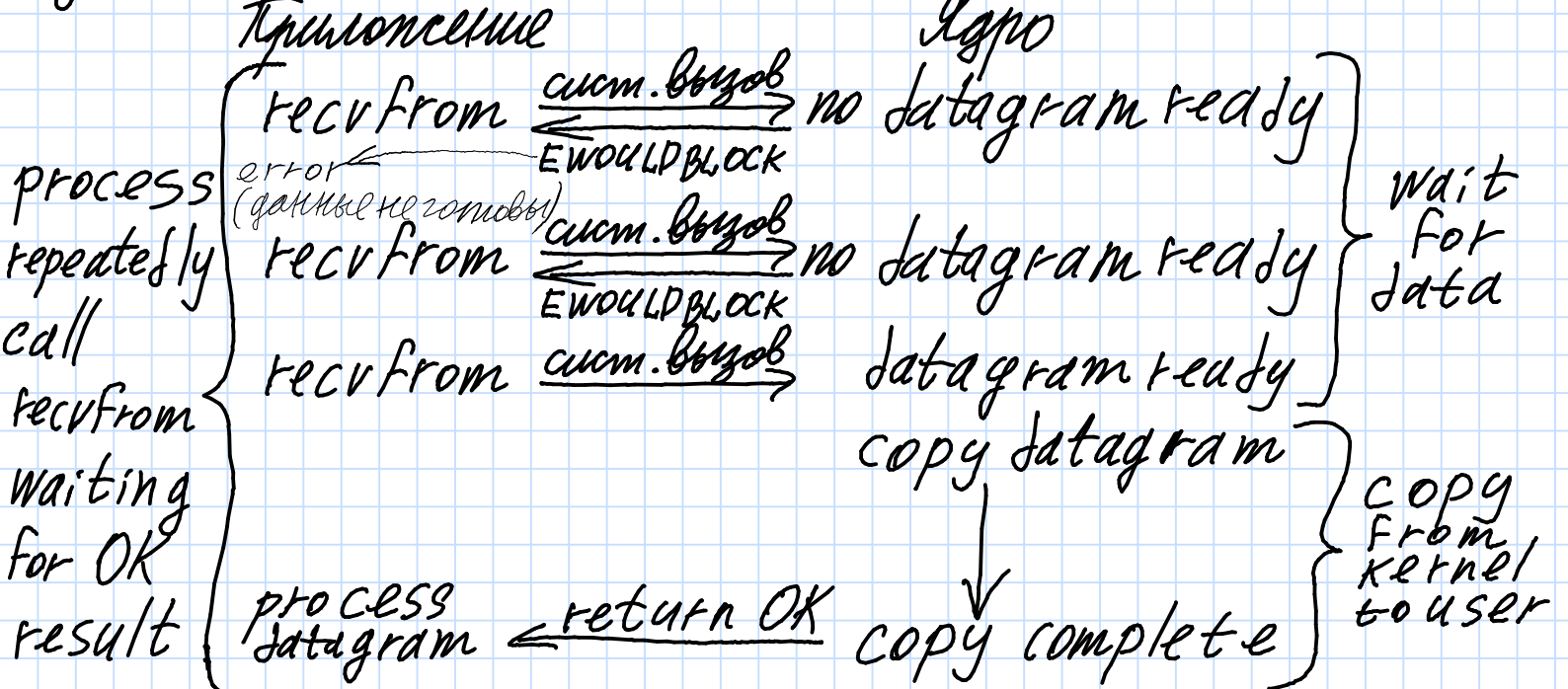
ввод-вывод, блокируется, и квант передаётся другому процессу;

Такой ввод-вывод возможен только если в системе реализованы аппаратные прерывания, т.е. прерывания от внеш. устройств;

② Неблокирующий ввод/вывод;

Исторически был реализован раньше блокирующего;

Управление операцией ввода-вывода управляет процессор: он опрашивал флаг устройства, который информировал процессор о готовности данных;



Опрос (polling) - периодическое действие: опрос выполняется до тех пор, пока данные не будут готовы, т.е. пока они не поступили в буфер устр-ва. Оттуда они поступают в буфер ядра и из него

уже в буфер приложения;
При этом процесс не блокируется,
но процессорное время тратится на опрос;

E-Error: ошибка обращения, данные не готовы;

Для этого определены спец. возвращаемые значения для каждой конкретной модели ввода-вывода.
Это учитывается разработчиками системы

Здесь нет аналогии с активным ожиданием на процессоре: это концептуально разные вещи.
В данном случае идёт опрос флага готовности контроллера прерывания, т.е. это обращение к внеш. устройству;

В системе должны быть спец. команды для работы в режиме poll;ing, однако это устаревший способ взаимодействия с внеш. устр-вами. Он затратный, но исключать опрос для любой системы мы не можем. В каких-то системах допускается возможность реализации такого опроса, а в каких-нет, т.к. у нас есть прерывания от внеш. устр-в (блокирующий ввод-вывод);

③ Мультиплексирование ввода/вывода
Мультиплексирование всегда рассм. для модели „клиент-сервер" на сокетах;

Этот способ выделяется как самост. модель ввода/вывода без привязки к сокетам;

Мультиплексирование - очень хорошая альтернатива многопоточности;

Многопоточность очень затратна, особенно на UNIX/Linux&Windows заточен на потоки, а не на процессы)

Вирт. ф. с. proc предоставляет информацию о потоках, которые запускаются в процессе;

В UNIX потоки равносильны процессам;

Владелец ресурсов - процесс;

Ещё про затратность многопоточности: для каждого потока в куче выделяется адр. пр-во (не зависимо от того, нужно оно или нет), а выделение физ. памяти осуществляется на основе виртуальной;

Всё строится на виртуальной памяти;

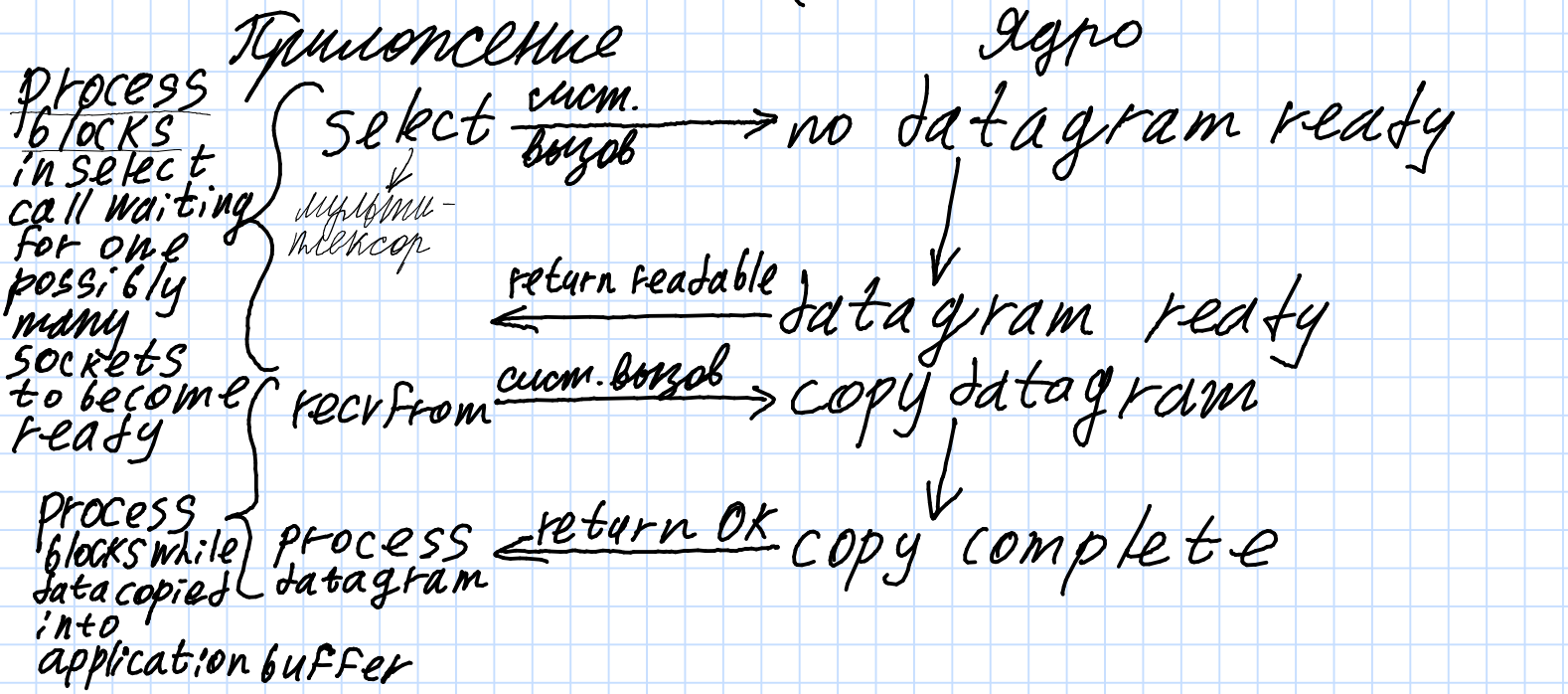
Для реализации мультиплексирования система предоставляет спец. сист. вызовы (мультиплексеры): select и poll;

RPC и сокеты - тоже альтернатива многопоточ-

ности. И то, и другое взаимодействие осуществляется по модели клиент-сервер. Но RPC предполагает наличие системного средства rpcgen – генератора кода. В распоряжение программисту поступают скелетоны, которые он должен переписать в соответствии с поставленной задачей;

В л/р на RPC сервер надо писать многопоточным, чтобы возникали проблемы взаимоисключения (но если что примут и однопоточный сервер);

Мультиплексирование ввода/вывода (multiplexing I/O) (выкинули receive from, иначе диаграмма была бы слишком большая)



Взаимодействие также выполняется путём передачи сообщений, но receive from выкинули, иначе диаграмма была бы слишком большой. Поэтому указывается только select;

Модель клиент-сервер предполагает, что есть

один сервер и много клиентов. Клиенты обращаются к серверу за обслуживанием.

При этом взаимодействие клиента и сервера осуществл. через сокеты;

Сокет – абстракция точки соединения;

Сокеты появились в UNIX BSD как универсальное (сокеты позволяют взаимодействовать процессам на отдельно стоящей машине, так и для взаимодействия параллельных процессов в распр-ных системах);

Мультиплексирование исп. только для сетей (распр-ных систем)

Мультиплексирование позволяет сократить время блокировки (блокировки - зло, но зло неизбежное) за счёт того, что мультиплексор фактически отражает готовность точек соединения (на своей стороне, т.к. клиенты устанавливают соединение с серверам);

Время ожидания готовности одной точки соединения из пула точек соединения будет меньше времени ожидания соединения в определённом порядке;

Пример: имеется 5 точек соед-я (клиентов); Если ждать в строгом порядке (сначала ждать соед-я с 1-м клиентом, потом строго со 2 и т.д., то,

т.к. это асинхр. паралл. процессы, может возникнуть след. ситуация:
3-й клиент сработал быстрее и быстрее остальных установил соединение с сервером. Т.е. если не ждать посл-ти соединений в опр. порядке, а ждать соед-я с любым клиентом (готовности любого сокета), то такое время ожидания на мультиплексоре будет значительно меньше;

После того, как select определил, что соединение установлено, вызывается receive From;

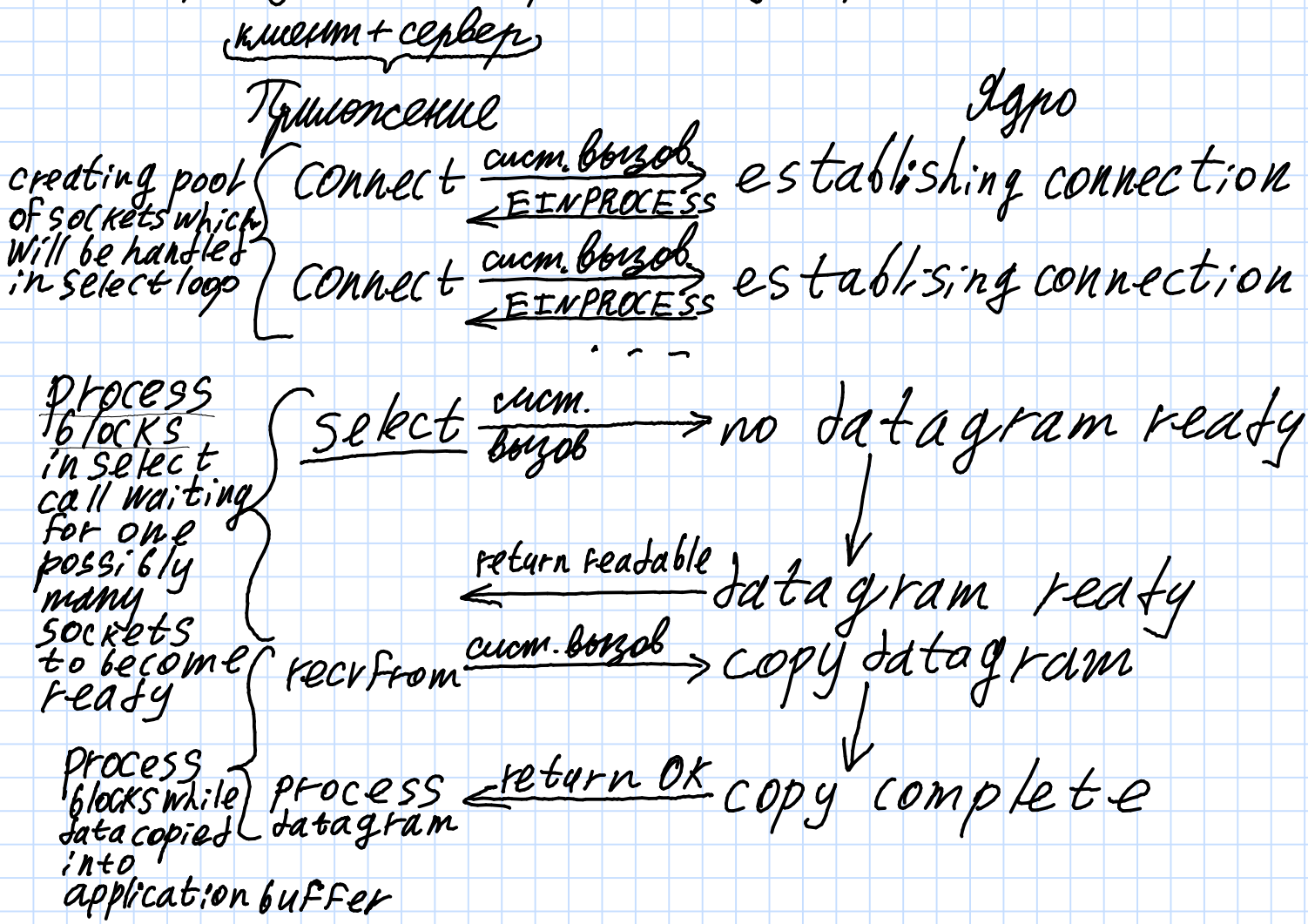
когда данные готовы, возвращается readable (данные доступны), вызывается receive From и производится копирование из буфера ядра в буфер приложения;

process datagram – обработка данных;

select вызывается на стороне сервера, сервер однопоточный, т.к. мультиплексирование – альтернатива многопоточности;

Пример мультиплексера – переключатель режимов работы стиральной машины;

Развёрнутый вариант диаграммы:



Формируется пул сокетов, которые будут обрабатываться в цикле (цикл) select;

На select сервер будет блокирован, но время блокировки будет меньше, т.к. отдаётся сразу пул сокетов. Вер-ть готовности какого-либо сокета из кол-ва выше вер-ти готовности конкретного сокета;

Блокировки увеличивают время выполнения приложения, время блокировки – случайная величина;
Разработчики стремятся избегать блокировок, и системы предоставляют соотв. сист. вызовы, обеспечивающие асинхронный ввод-вывод ... см. след. семинар