# Операційні системи

## Поняття операційної системи, її призначення

Операцио́нная систе́ма, сокр. ОС (англ. operating system, OS) — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

## Види операційних систем

Это список известных операционных систем. Операционные системы могут быть классифицированы по базовой технологии (UNIX-подобные, пост-UNIX/потомки UΝΙΧ), типу лицензии (проприетарная или открытая), развивается ли в настоящее время (устаревшие или современные), по назначению (универсальные, ОС встроенных систем, ОС PDA, ОС реального времени, для рабочих станций или серверов), а также по множеству других признаков.

## Структура операційної системи

ОС состоит из:

– загрузчика

– ядра,

– базовой системы ввода-вывода,

– оболочки,

– сервисных программ.

Загрузчик операционной системы

Загрузчик операционной системы — программа, обеспечивающая запуск операционной системы из начального состояния (после включения питания, сброса), инициализацию основных её компонентов.

Ядро операционной системы

Ядро операционной системы — часть ОС, выполняющаяся при максимальном уровне привилегий. Как правило, в ядро помещаются процедуры, выполняющие манипуляции с основными ресурсами системы и уровнями привилегий процессов, а также критичные процедуры, выполнение которых с пользовательскими привилегиями невыгодно. Ядро считается необходимым компонентом ОС, по крайней мере в системах с распределением привилегий, впрочем заявлено создание и безъядерных ОС[1].

Базовая система ввода-вывода

Базовая система ввода-вывода (БСВВ, BIOS) — набор программных средств, обеспечивающих взаимодействие ОС и приложений с аппаратными средствами. Обычно БСВВ представляет набор компонент — драйверов. Также в БСВВ входит уровень аппаратных абстракций, минимальный набор аппаратно-зависимых процедур ввода-вывода, необходимый для запуска и функционирования ОС. Нередко функции БСВВ включаются в ядро.

ОболочкаОболочка операционной системы — часть ОС, задачей которой является взаимодействия ОС с пользователем, заключающееся в предоставлении ему управления и контроля за работой вычислительной системе. Это может быть командный интерпретатор — обеспечивающий управление системой посредством ввода текстовых команд (с клавиатуры, через порт или сеть), графическая оболочка — обеспечивающая работу при помощи координатных устройств ввода (принцип «укажи и нажми»). Операционные системы, не предназначенные для интерактивной работы часто его не имеют.

Служебные программы

Сервисное программное обеспечение (утилиты) – пакет программ, включённый в состав ОС, реализующий действия по управлению и обслуживанию ОС.

## Системні виклики

Систе́мный вы́зов (англ. system call) в программировании и вычислительной технике — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

Современные операционные системы (ОС) предусматривают разделение времени между выполняющимися вычислительными процессами (многозадачность) и разделение полномочий, препятствующее обращению исполняемых программ к данным других программ и оборудованию. Ядро ОС исполняется в привилегированном режиме работы процессора. Для выполнения межпроцессной операции или операции, требующей доступа к оборудованию, программа обращается к ядру, которое, в зависимости от полномочий вызывающего процесса, исполняет либо отказывает в исполнении такого вызова.

С точки зрения программиста, системный вызов обычно выглядит как вызов подпрограммы или функции из системной библиотеки. Однако системный вызов, как частный случай вызова такой функции или подпрограммы, следует отличать от более общего обращения к системной библиотеке, поскольку последнее может и не требовать выполнения привилегированных операций.

## Процеси, потоки та методи їх синхронізації. Критичні області, семафори, м’ютекси, події.

* Процесс – **исполняемая** программа, с которой связаны:
  + Системные информационные ресурсы;
  + Область основной памяти (адресное пространство);
  + Файлы, устройства ввода-вывода.
* Задачи подсистемы управления процессами:
  + Создание и уничтожение процессов;
  + Планирование выполнения процессов;
  + Выделение ресурсов процессам;
  + Защита ресурсов процесса;
  + Синхронизация и взаимодействие процессов.
* Потоки – последовательности команд, совместно использующих данные
* Процесс – совокупность
  + потоков (не менее одного) и
  + защищенного адресного пространства

Средства обмена информацией:

Сигнальные (сигналы UNIX)

минимальный объем передаваемой информации, минимальное влияние. Cпособ информирования процесса со стороны ядра о происшествии некоторого события:

Канальные (FIFO, pipes, sockets)

объем передаваемой информации ограничен пропускной способностью линий связи.

Разделяемая память (shmem)

наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

main(){

int var = 5;

if(fork()) wait(&var);

else exit(var);

var++;

printf("%d\n",var);

return var;

}

В среде, позволяющей исполнять несколько процессов (потоков) одновременно, очень важно синхронизировать их деятельность. Для этого операционные системы, предлагают несколько синхронизирующих объектов: критические разделы (critical sections), объекты mutex (сокращение от mutual exclusion), семафоры и события.

Критические разделы (только WIN32)

*Критический раздел* (critical section) – объект, к которому должен обратиться поток перед получением эксклюзивного доступа к каким-либо общим данным. Среди синхронизирующих объектов критические разделы наиболее просты, но применимы для синхронизации потоков лишь в пределах одного процесса. Они дают возможность сделать так, чтобы единовременно только один поток получил доступ к определенному региону данных.

При необходимости синхронизировать приложения со специфическими событиями, возникающими в системе, или с операциями, выполняемыми в других процессах, критические разделы непригодны. Например, при создании дочернего процесса, возможно, придется сделать так, чтобы родительский процесс ожидал его завершения и только потом продолжал свою работу.

Каждый объект синхронизации может находиться в одном из двух состояний: *свободном* (signaled) или *занятом* (nonsignaled). Потоки могут остановиться и ждать, пока какой-либо объект не освободится. Если поток родительского процесса должен ждать завершения дочернего, его можно отправить "в спячку" до освобождения объекта ядра, идентифицирующего дочерний процесс. Объекты "процесс" получают статус свободных в момент завершения соответствующих процессов. Это относится и к объектам "поток". Когда поток создан и начинает исполнение своего кода, сопоставленный с ним объект ядра "поток" получает статус занятого. При завершении потока соответствующий объект ядра освобождается.

Объекты Mutex

Эти объекты аналогичны критическим разделам, но с их помощью можно синхронизировать доступ к данным со стороны не только нескольких потоков одного процесса, но и нескольких процессов.

Семафоры

Объекты ядра "семафор" используются для учета ресурсов. При запросе у семафора ресурса операционная система проверяет, свободен ли данный ресурс, и — если свободен — уменьшает счетчик доступных ресурсов, не давая вмешиваться другому потоку. Только после этого система разрешает другому потоку запрашивать какой-либо ресурс.

Для мониторинга занятости ресурсов необходимо создать семафор со счетчиком. При этом нужно учитывать, что семафор считается свободным, если его счетчик ресурсов больше нуля, и занятым, если счетчик равен нулю. При каждом вызове из потока специальной функции с передачей ей описателя семафора система проверяет: больше ли нуля счетчик ресурсов у данного семафора. Если да, уменьшает счетчик на единицу и "будит" поток. Если при вызове функции счетчик семафора оказался обнулен, система оставляет поток неактивным до того, как другой поток освободит семафор (т. е. увеличит его счетчик ресурсов).

Поскольку на счетчик ресурсов семафора могут влиять несколько потоков (или процессов), семафоры — в отличие от критических разделов и объектов mutex — не передаются во владение какому-либо потоку. А значит, один поток может ждать объект "семафор" (уменьшив его счетчик ресурсов), а другой поток освободить семафор (и тем самым увеличить его счетчик ресурсов).

События

События — самая примитивная разновидность синхронизирующих объектов, отличающаяся от семафоров и объектов mutex. Последние обычно применяются для контроля за доступом к данным, а события просто оповещают об окончании какой-либо операции. Существуют два разных типа объектов "событие": со *сбросом вручную* (manual-reset events) и *с автоматическим сбросом* (auto-reset events). Первые используются для оповещения об окончании операции сразу нескольких потоков, вторые — для оповещения единственного потока.

К событиям обычно прибегают в том случае, когда один поток выполняет какую-либо инициализацию, а затем сигнализирует другому потоку, что тот может работать дальше. Инициализирующий поток переводит объект "событие" в занятое (non-signaled) состояние и приступает к своим операциям. По окончании инициализации поток возвращает событие в свободное (signaled) состояние. В то же время рабочий поток приостанавливает свое исполнение и ждет перехода события в свободное состояние. Как только инициализирующий поток просигнализирует событие (т.е. освободит его), рабочий поток "проснется" и продолжит работу.

Например, в процессе исполняются два потока. Первый считывает данные из файла в буферную память и оповещает второй поток, что можно заняться обработкой данных. Закончив обработку, второй поток сигнализирует первому, чтобы тот загрузил новый блок данных из файла и т.д.

## Класичні проблеми синхронізації та їх вирішення

Проблема производителя и потребителя (проблема ограниченного буфера)

Два процесса совместно используют буфер ограниченного размера. Один из них, производитель, помещает данные в этот буфер, а другой, потребитель, считывает их оттуда. Задача может быть обобщена на случай ***т*** производителей и ***п***потребителей.

Проблема обедающих философов

Сформулирована в 1965г. Дейкстрой.

Условия задачи:

Имеется N философов, сидящих за круглым столом. Перед каждым из них находится тарелка со спагетти. Спагетти настолько скользкие, что каждому философу необходимо две вилки , чтобы с ними управиться. Между каждыми двумя тарелками лежит одна вилка.

Жизнь философа состоит из чередующихся периодов поглощения пищи и размышлений (остальные процессы жизнедеятельности в данном случае несущественны). Когда философ голоден, он пытается получить две вилки, левую и правую, в любом порядке. Если ему удалось получить две вилки, он некоторое время ест, затем кладет вилки и продолжает размышления.

Данная проблема полезна для моделирования процессов, соревнующихся за монопольный доступ к ограниченному количеству ресурсов (например, к устройствам ввода-вывода).

Проблема читателей и писателей

Проблема моделирует доступ к базе данных.

Условие задачи: необходимо обеспечить монопольный доступ к базе данных процессу, выполняющему запись данных, и множественный доступ процессам, выполняющим чтение данных. Задача может быть сформулирована с предоставлением приоритета как процессу-писателю, так и процессам-читателям.

Проблема спящего брадобрея

Условие задачи: имеется один процесс-«брадобрей», его кресло и N стульев для процессов-«клиентов». Если «клиентов нет», «брадобрей» спит в своем кресле. Если в парикмахерскую приходит «клиент», он должен разбудить «брадобрея». Если клиент приходит, и видит, что брадобрей занят, он либо занимает один из стульев (если есть место), либо уходит (если места нет). Необходимо запрограммировать брадобрея и клиентов так, чтобы избежать состояния состязания.

## Планування

Задача планирования возникает при наличии нескольких потребителей ограниченного набора ресурсов. Выбор алгоритма определяется классом решаемых задач и поставленными целями.

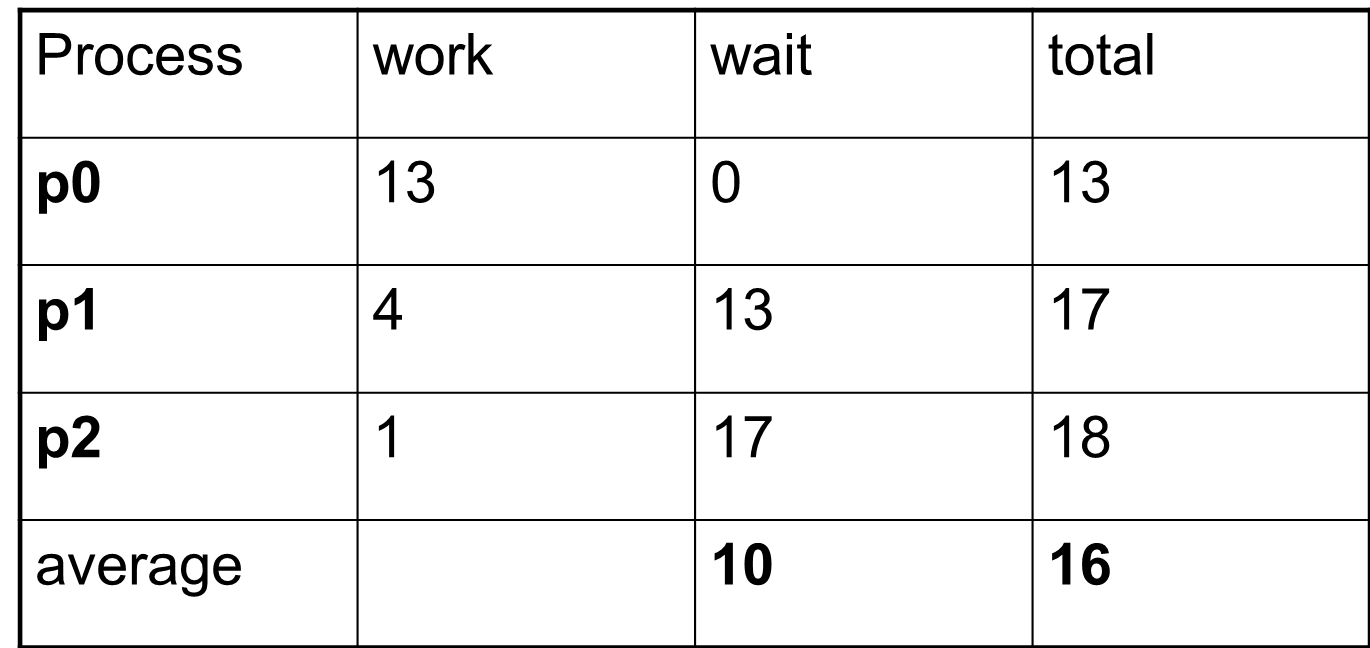
Параметры планирования

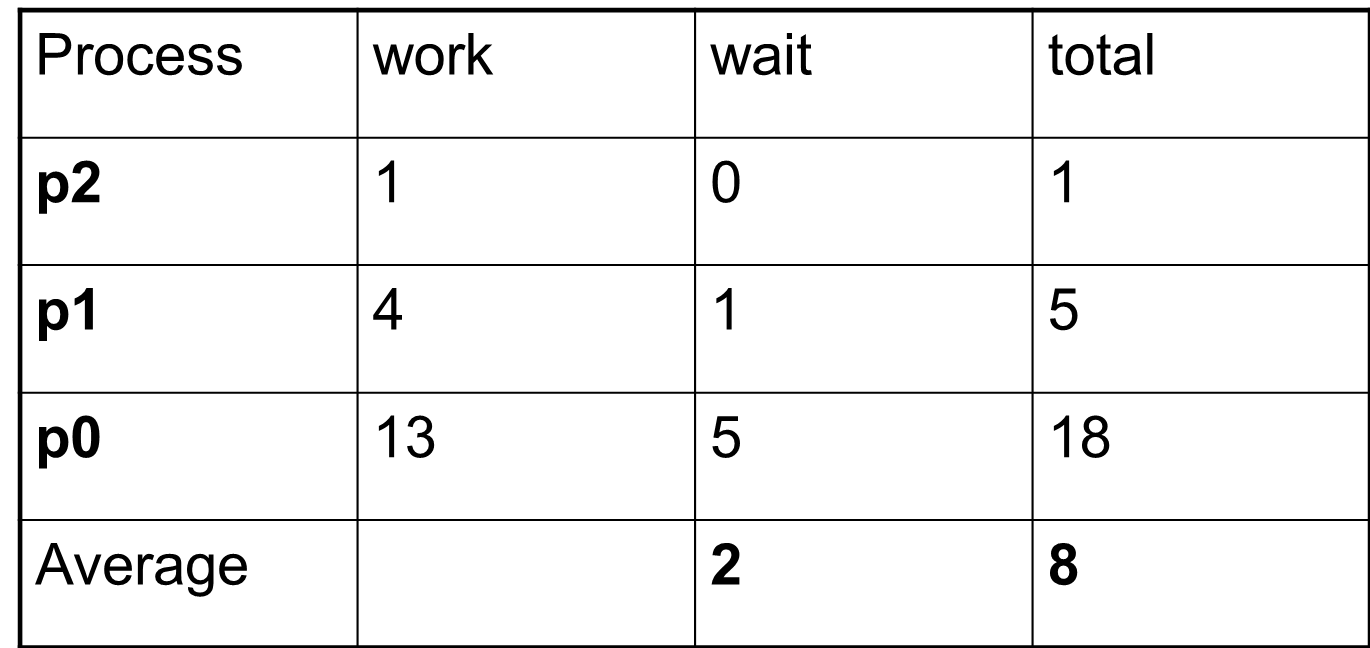
* Динамические параметры системы – количество свободных ресурсов на данный момент времени.
* Статические параметры системы – предельные значения ресурсов:
  + объем оперативной памяти
  + количество подключенных устройств ввода-вывода
  + объем своп-раздела

## Алгоритми планування. Політика та механізм планування

### First-Come, First-Served (FCFS)

* Невытесняющее планирование
* Процессы, находящиеся в состоянии готовности, помещаются в очередь
* Простота реализации
* Большое среднее время отклика в интерактивных процессах (неприменим для систем разделения времени)



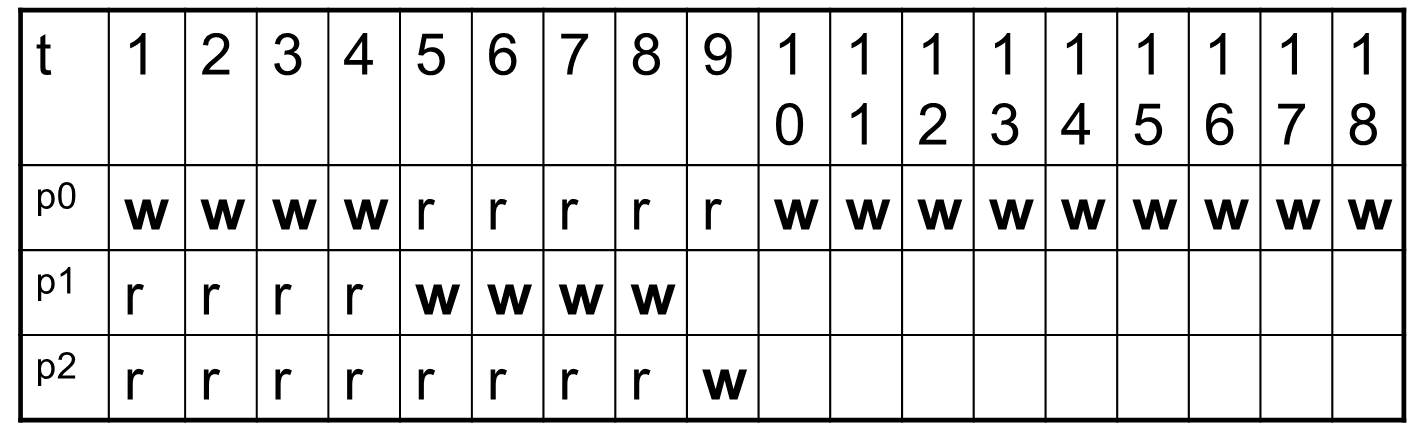


### Round Robin (RR)

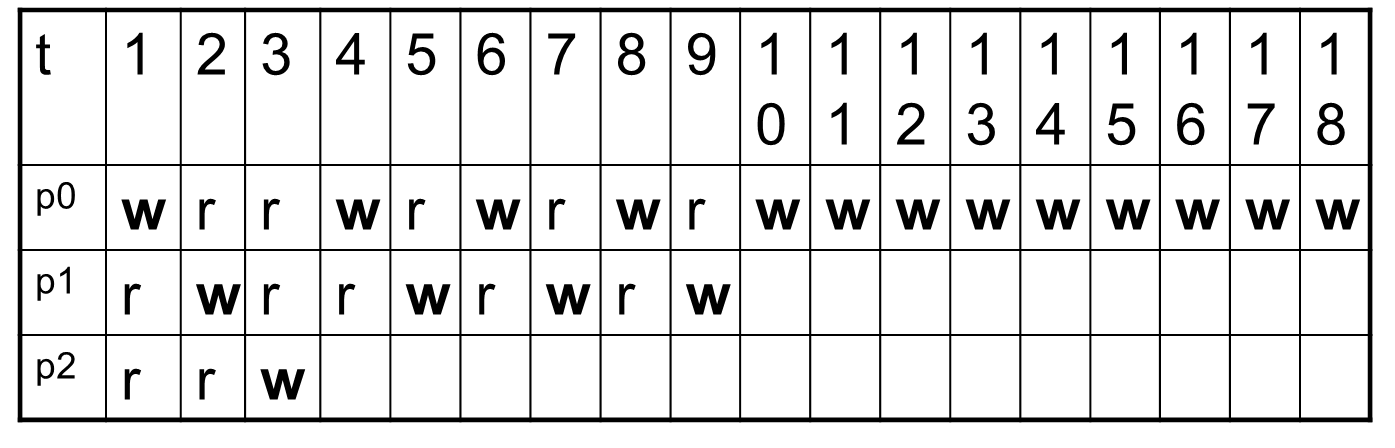
* Вытесняющее планирование
* Циклический список процессов, находящихся в состоянии готовности
* Каждому процессу выделяется определенный квант времени
* Производительность зависит от величины кванта времени

1)

* Длительность кванта = 4;
* wait = {5,4,8}, wait\_a = **5,7**;
* work = {18,8,9}, work\_a = **11,7**;



* 2)
* Длительность кванта = 1;
* wait = {5,5,2}, wait\_a = **4**;
* work = {18,9,3}, work\_a = **10**;

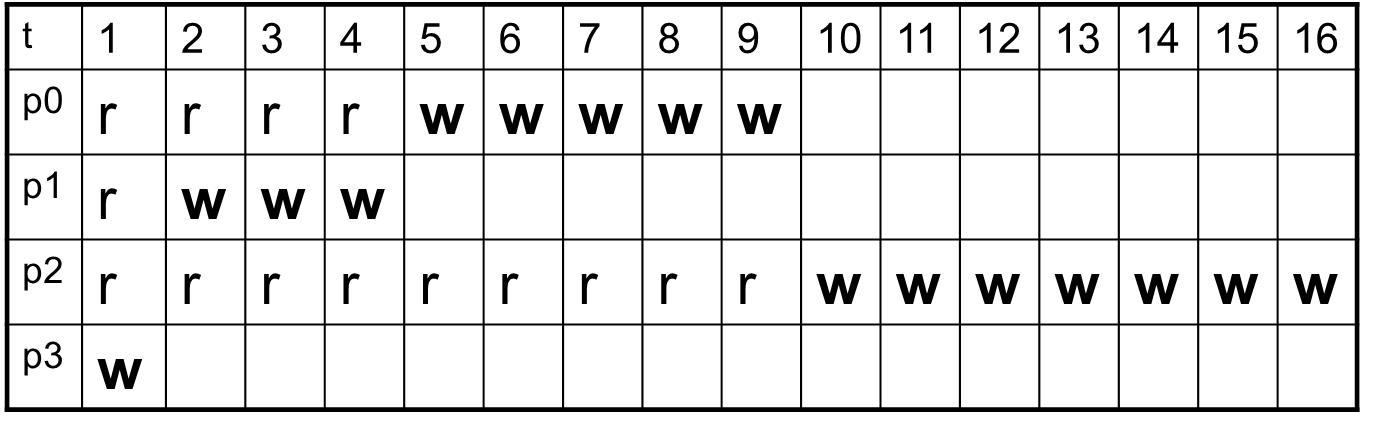


### Shortest Job(Task) First (SJF)

* Выбирает из очереди процесс с минимальным CPU burst
* Квантование не применяется
* Может быть как невытесняющим, так и вытесняющим (учитывается появление новых процессов)

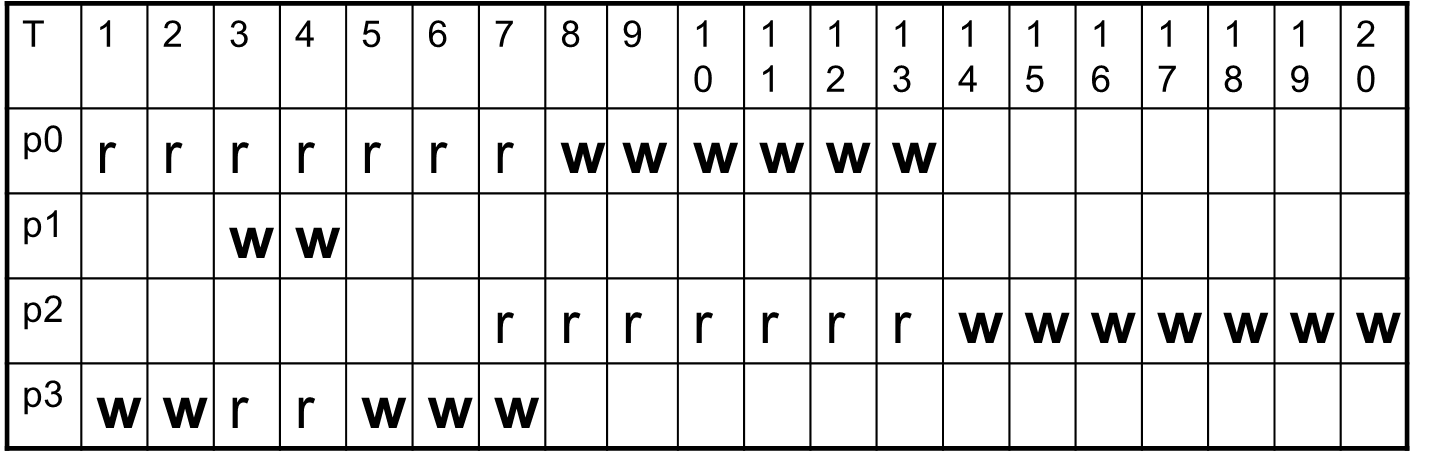
Невытесняющий SJF

* CPU burst = {5,3,7,1}
* wait = {4,1,9,0}, wait\_a = **3,5**;
* work = {9,16,4,1}; work\_a = **15**;



Вытесняющий SJF

* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* wait = {7,0,7,2}, wait\_a = **4**;
* work = {13,2,14,7}; work\_a = **9**;



* Оценка CPU burst:
  + Задается пользователем (пакетные системы);
  + Вычисление с учетом предыстории:

sr3-f1

### Гарантированное планирование

* Очередной квант времени предоставляется процессу с минимальным значением коэффициента справедливости
* Невозможность прогнозирования поведения пользователей
* N – число пользователей, T – время нахождения пользователя в системе

sr3-f7

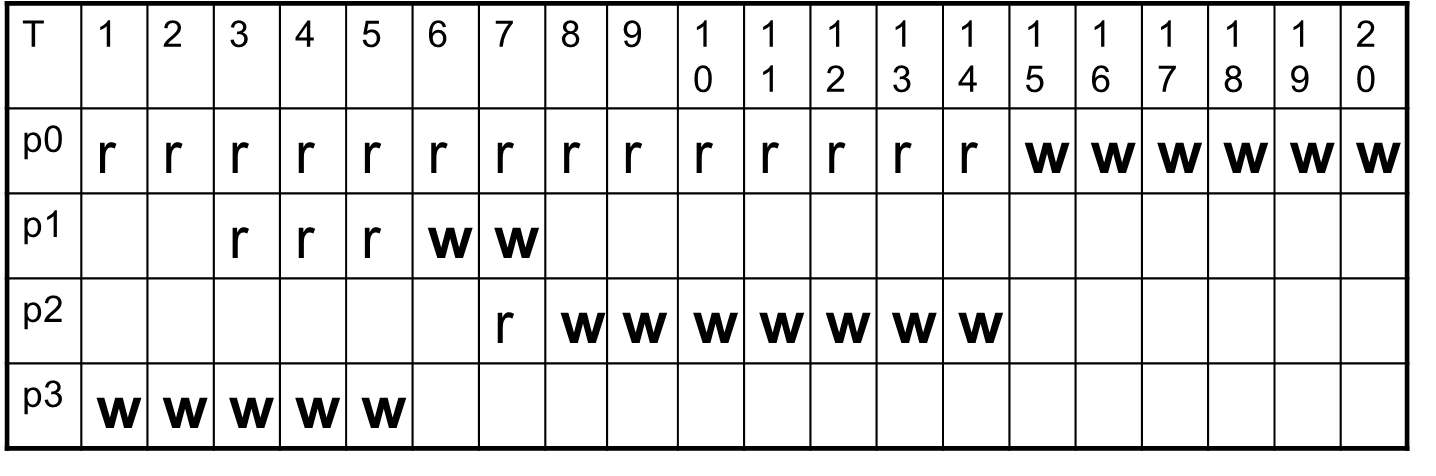
### Приоритетное планирование

Гарантированное планирование и SJF – частные случаи приоритетного планирования

Вытесняющее и невытесняющее планирование.

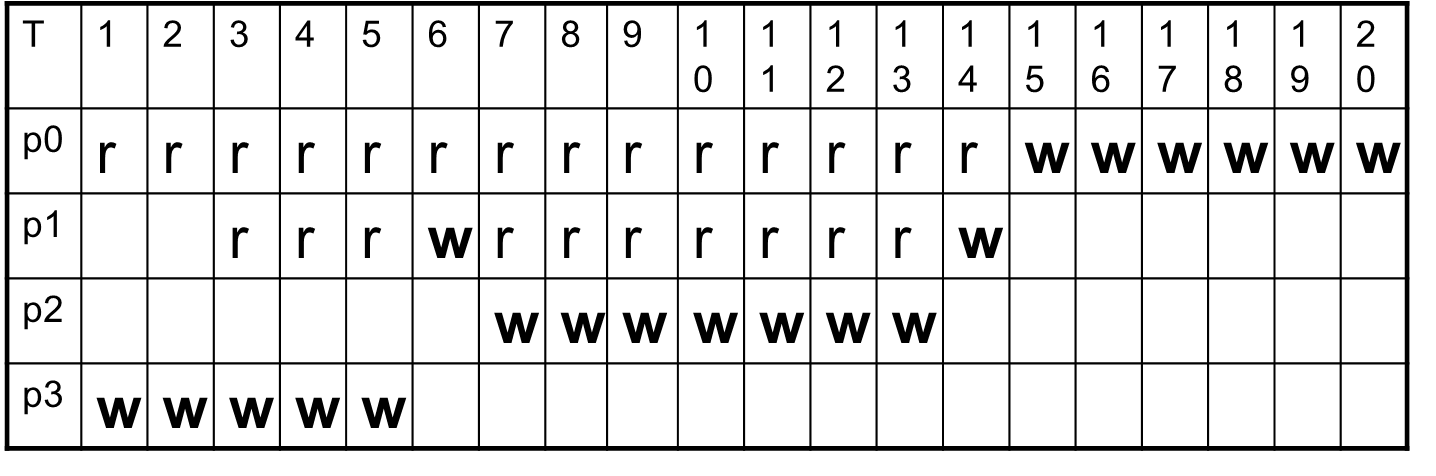
Приоритетное планирование невытесняющее

* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* Priority = {4,3,2,1};



Приоритетное планирование вытесняющее

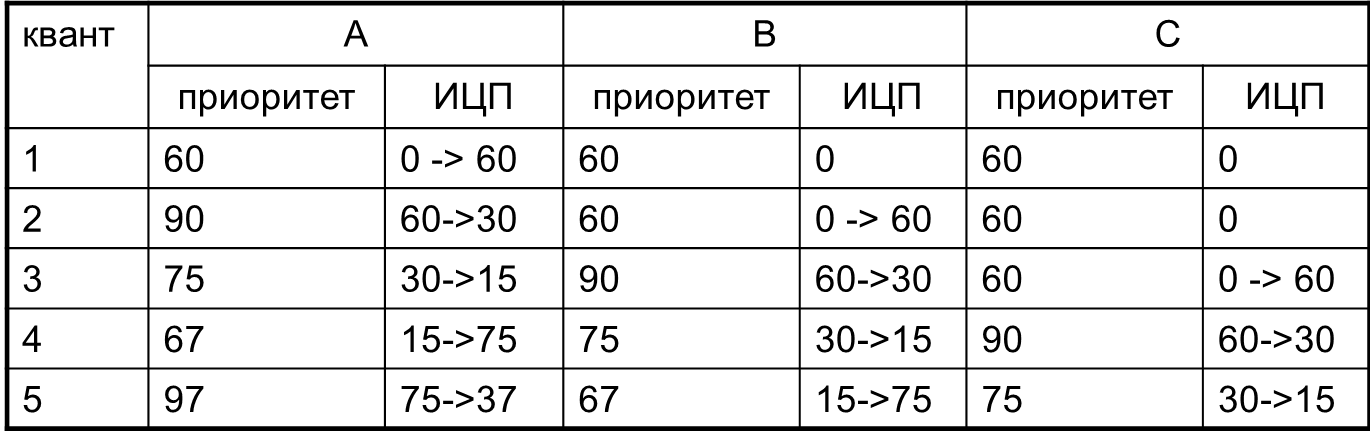
* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* Priority = {4,3,2,1};



### Диспетчеризация

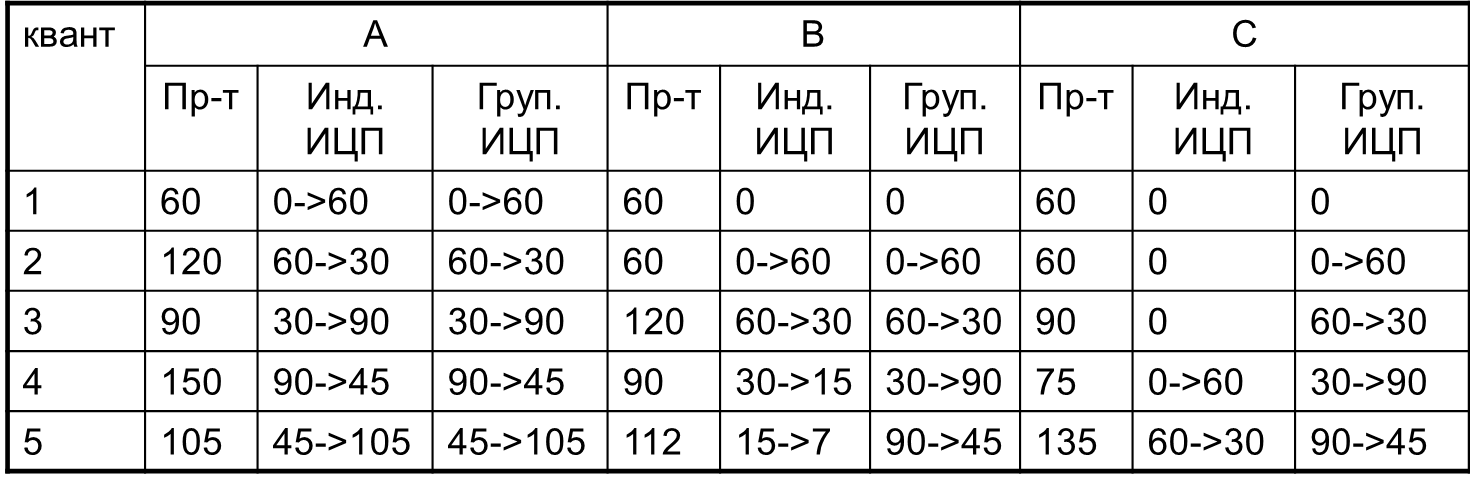
1)

* Три процесса (A, B, C);
* base = 60 - базовый приоритет;
* ИЦП = ИЦП/2;
* Приоритет = ИЦП/2 + base;
* Наращивание ИЦП – 60 раз в сек.



2)

* Три процесса (A, B, C);
* base = 60 - базовый приоритет;
* ИЦП = ИЦП/2; ИЦПГ = ИЦПГ/2;
* Приоритет = ИЦП/2 + ИЦПГ/2 + base;
* Наращивание ИЦП – 60 раз в сек.



## Системи керування пам’яттю

Память – ресурс вычислительной системы, предназначенный для хранения программного кода и даннях. Модуль управления памятью или менеджер памяти – часть операционной системы, отвечающая за управление памятью.

## Підкачка та віртуальна пам’ять

Виртуальная память

* **схема адресации памяти**, при которой память представляется программному обеспечению **непрерывной** и однородной,
* в то время как в реальности для фактического хранения данных используются **отдельные** (разрывные) **области** **различных** **видов** памяти, включая кратковременную (оперативную) и долговременную (жесткие диски, твердотельные накопители)
* **упрощает** адресацию памяти в программном обеспечении
* позволяет **рационально** управлять основной памятью компьютера (хранить в ней только активно используемые области памяти)
* позволяет **изолировать** процессы друг от друга

## Алгоритми заміщення сторінок. Моделювання алгоритмів заміщення сторінок.

Алгоритмы замещения страниц^

* Бит модификации
* Бит обращения
* Локальные алгоритмы - распределяют определенное количество страниц для каждого процесса
* Глобальные алгоритмы - распределяют все страницы между всеми процессами

### FIFO

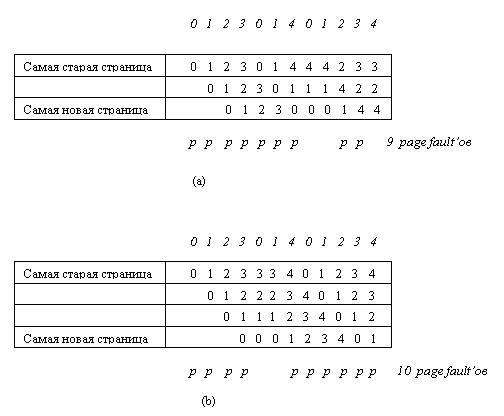
* Замещается самая «старая» страница

Операционная система поддерживает список всех страниц, находящихся в данный момент в памяти, в котором первая страница является старейшей, а страницы в хвосте списка попали в него совсем недавно. Когда происходит страничное прерывание, выгружается из памяти страница в голове списка, а новая страница добавляется в его конец.

Этот алгоритм не требует аппаратной поддержки, однако он не учитывает информацию об использовании страницы.

* Аномалия FIFO – c увеличением числа фреймов растет число страничных прерываний

(“Самая старая страница” <-> “Самая новая страница” -- reverse)



### NRU (Not Recently Used)

Использует R(eferenced) и M(odified) биты для классификации страниц по 4-м классам:

R=0, M=0 – не было обращений и изменений

R=0, M=1 – не было обращений, страница изменена

R=1, M=0 – было обращение, страница не изменена

R=1, M=1 – было обращение и изменение

The not recently used (NRU) page replacement algorithm is an algorithm that favours keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set. The setting of the bits is usually done by the hardware, although it is possible to do so on the software level as well.

At a certain fixed time interval, a timer interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current timer interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:

3. referenced, modified

2. referenced, not modified

1. not referenced, modified

0. not referenced, not modified

Although it does not seem possible for a page to be not referenced yet modified, this happens when a class 3 page has its referenced bit cleared by the timer interrupt. The NRU algorithm picks a random page from the lowest category for removal. So out of the above four pages, the NRU algorithm will replace the not referenced, not modified. Note that this algorithm implies that a modified but not referenced (within last timer interval) page is less important than a not modified page that is intensely referenced.

### LRU (The Least Recently Used)

Замещает страницу, которая не использовалась в течение долгого времени

Для реализации необходим список всех страниц в памяти, отсортированный по частоте обращений и обновляемый при каждом обращении

Требует аппаратной реализации

### NFU (Not Frequently Used)

The least recently used (LRU) page replacement algorithm, though similar in name to NRU, differs in the fact that LRU keeps track of page usage over a short period of time, while NRU just looks at the usage in the last clock interval. LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too.

### Second Chance

Данный алгоритм является версией алгоритма FIFO, который позволяет избежать проблемы вытеснения из памяти часто используемых страниц. У самой старейшей страницы изучается бит *R.* Если он равен 0, страница находится в памяти долго и не используется, следовательно, она может быть заменена новой. Если же бит *R* равен 1, то ему присваивается значение 0, страница переносится в конец списка, а время ее загрузки обновляется, то есть считается, что страница только что попала в память. Затем процедура продолжается.

Работа данного алгоритма показана на рис. 3.1, *а.* Здесь изображены страницы от Адо H, хранящиеся в связанном списке и отсортированные по времени их поступления в память. Числа над страницами обозначают их время загрузки в память.

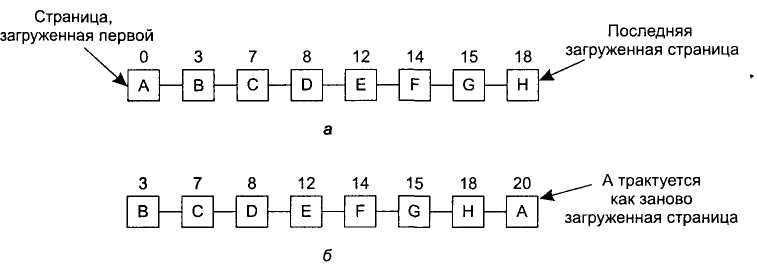


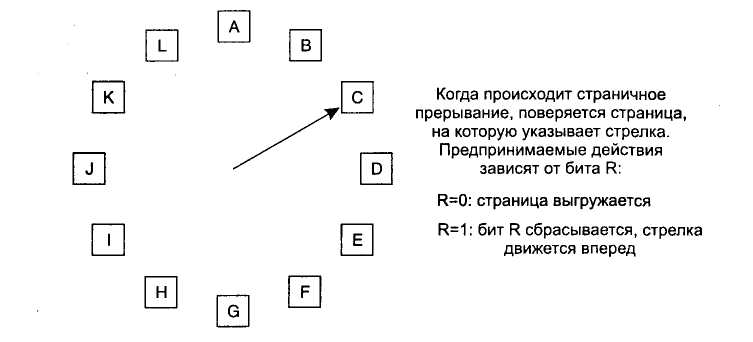
Рис. 3.1. Действие алгоритма «вторая попытка»: страницы, отсортированные в порядке очереди (FIFO) (а); список страниц, если страничное прерывание произошло во время 20, а страница *А* имеет бит R, равный 0 (б)

Алгоритм «вторая попытка» ищет в списке самую старую страницу, к которой не было обращений в предыдущем временном интервале. Если же происходили ссылки на все страницы, то «вторая попытка» превращается в обычный алгоритм FIFO.

### Часы

Данный алгоритм отличается от алгоритма «вторая попытка» только своей реализацией. Алгоритм «вторая попытка» является корректным, но не эффективным, поскольку постоянно передвигает страницы по списку. Поэтому лучше хранить все страничные блоки в кольцевом списке, как показано рис. 3.2. Стрелка указывает на старейшую страницу.

Когда происходит страничное прерывание, проверяется та страница, на которую направлена стрелка. Если ее бит R равен 0, страница выгружается, на ее место в круг встает новая страница, а стрелка сдвигается вперед на одну позицию. Если бит R равен 1, то он сбрасывается, стрелка перемещается к следующей странице. Этот процесс повторяется до тех пор, пока не находится та страница, у которой бит R = 0.



### Рабочий набор

Идея алгоритма рабочих наборов – ОС в каждый момент времени должна обеспечивать наличие в основной памяти текущих рабочих наборов всех процессов, которым разрешена конкуренция за доступ к процесору.

«Сборщик страниц» – выполняет откачку страниц, не входящих в рабочие наборы процессов. Начинает работать при достижении некоторого минимального порога числа свободных страниц. Аналог в Windows -- менеджер балансного набора (Working set manager)

Свопинг – выполняет выгрузку всех страниц процесса в ситуации, если свободных страниц нет

## Принципи роботи апаратури введення-виводу

* Устройство ввода-вывода
  + Механическая часть (в самом устройстве)
  + Электронная часть (контроллер, адаптер)
* Низкоуровневый интерфейс между устройством и контроллером (битовый поток)
* Контроллер формирует из потока битов блоки (с проверкой контрольной суммы) и генерирует команды устройству
* Контроллер содержит управляющие регистры и буферы данных

## Принципи побудови програмного забезпечення введення-виводу

Принципы организации ввода-вывода

* Программируемый ввод-вывод Процессор посылает необходимые команды контроллеру ввода-вывода. Процесс, запросивший ввод-вывод, находится в состоянии ожидания завершения операции ввода-вывода
* Управляемый прерываниями ввод-вывод. Для реализации необходима дополнительная линия – линия (шина) прерываний. По завершении выполнения операции устройство выставляет на линию прерываний сигнал. Для реализации необходима дополнительная линия – линия (шина) прерываний. По завершении выполнения операции устройство выставляет на линию прерываний сигнал.
* Ввод-вывод с использованием DMA (Direct Memory Access). Режим обмена данными без участия центрального процесора. DMA-контроллер получает доступ к системной шине независимо от ЦП. Содержит регистры, доступные ЦП для чтения и записи. Регистры задают порт, направление переноса данных (чтение/запись), единицу переноса (побайтно/пословно), число байтов, которое следует перенести. ЦП программирует контроллер DMA, устанавливая его регистры. По окончании записи устройство посылает сигнал подтверждения контроллеру DMA.

Программные уровни ввода-вывода:

## Переривання. Обробники переривань

При возникновении прерывания инициируется его обработка, включающая следующие шаги:

* Сохранение всех регистров (включая PSW), не сохраненных аппаратно
* Установка контекста для процедуры обработки прерываний
* Установка указателя стека для процедуры обработки прерываний
* Выдача подтверждения контроллеру прерываний. Если централизованного контроллера прерываний нет, разрешение прерываний
* Копирование содержимого регистров в таблицу процессов
* Запуск процедуры обработки прерываний, которая извлечет информацию из регистров контроллера устройства, инициировавшего прерывание
* Выбор процесса, которому передается управление
* Установка контекста для выбранного процесса
* Загрузка регистров нового процесса, включая его PSW
* Выполнение нового процесса

## Драйвери пристроїв

Программа для управления устройствами определенного вида (класса), принимающая стандартные запросы от ОС и преобразующая их в команды устройства. Для получения доступа к аппаратной части устройства драйвер должен быть частью ядра Классификация драйверов – по типам устройств (символьные, блочные).

## Типи пристроїв введення-виводу. Диски, таймери, алфавітно-цифрові термінали, графічні інтерфейси користувача, мережні термінали.

* Диски, дорожки, сектора
* *Время поиска* (seek time) – время перемещения манипулятора для позиционирования на дорожке (5-10 мс)
* *Задержка вращения* (rotational delay) – время ожидания поворота пластины так, чтобы сектор оказался под головкой (3 мс)
* *Пропускная способность передачи данных* (200 Мб/сек).

Время передачи даннях^

* T = b/(r\*N)
* b – количество передаваемых данных
* r – скорость вращения (об/с)
* N – число байтов на дорожке

RAID – набор физических дисков, «видимый» ОС как единый логический диск

Таймеры управляют устройствами, передающими в систему информацию о времени (текущее время суток, учет процессорного времени). Используют отдельный набор системных вызовов.

Таймеры отложенного выполнения используют для оповещения процесса об окончании некоторого промежутка времени Хранят число прерываний, оставшееся до сработки таймера (данное число уменьшается при каждом прерывании).

* Терминальный ввод-вывод
* Аппаратный терминал (дисплей + клавиатура) используется для организации работы множества пользователей
* Подключается через последовательный интерфейс
* Работает в символьном режиме (обмен данными и их отображение выполняется посимвольно)
* Символы делятся на отображаемые и управляющие (ESC-последовательности)

Каждому терминалу соответствует файл символьного устройства.

## Файли

Правила именования

* ограничения на длину имени (поддержка длинных имен и генерация коротких синонимов)
* ограничения на используемые символы
* чувствительность к регистру символов
* понятие расширения файла
* уникальность имени и символьные ссылки

Типы файлов

* Обычные (регулярные)
  + Текстовые
  + Двоичные
* Специальные (связанные с устройствами ввода-вывода)
  + Устройства
  + Сокеты
  + Каналы

Выделение дискового пространства файлу:

* Непрерывная последовательность блоков
* Связный список
* Связный список с использованием индекса (file allocation table – FAT)

## Каталоги

* Специальный файл, предназначенный для организации многоуровневой структуры хранения файлов
* Содержит список хранящихся в нем файлов
* Каждая запись имеет фиксированную структуру и длину
* Полное и относительное имя файла
* Монтируемые файловые системы
* Текущий каталог (.)
* Родительский каталог (..)

## Реалізації та приклади файлових систем

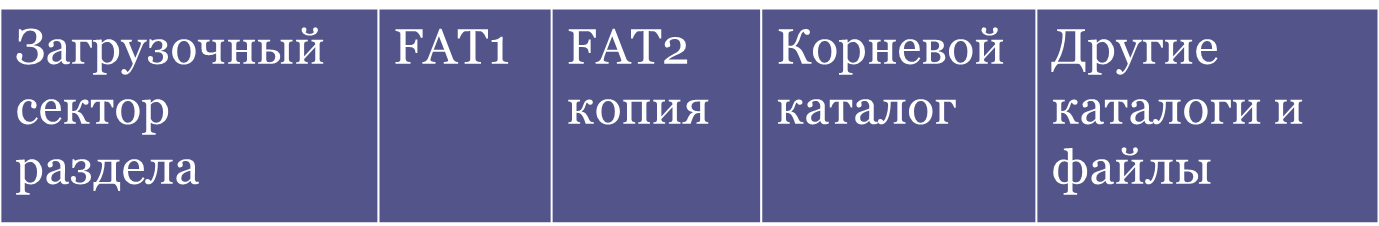
Файловые системы семейства Windows:

* FAT (FAT-12, FAT-16, FAT-32) – ориентирована на небольшие диски и простые структуры каталогов
* exFAT (FAT-64)
* NTFS

Термины

* **Сектор** – наименьшая единица физического хранения на диске (степень 2 и почти всегда 512 байт)
* **Кластер** – один или несколько (степень 2) последовательных секторов на одной дорожке
* **Том** – пространство, занимаемое файловой системой. Может занимать весь диск, его часть или несколько дисков

FAT – структура тома:



FAT32:

Позволяет создавать разделы более 2Гб (до 2 Тб). Загрузочная запись содержит резервные копии наиболее важных структур данных.

exFAT – extended FAT:

Ориентирована на мобильные устройства хранения. Предельный размер файла 264 байт.

NTFS:

Возможность установки прав доступа к отдельным файлам и каталогам. Журналируемая файловая система – восстановление после сбоев. Поддержка жестких и символьных ссылок.

## Определить время загрузки с диска программы определенного размера, если известно его среднее время поиска, время вращения, вместимость одной дорожки диска и размер страницы.

* *Время поиска* (seek time) – время перемещения манипулятора для позиционирования на дорожке (5-10 мс)
* *Задержка вращения* (rotational delay) – время ожидания поворота пластины так, чтобы сектор оказался под головкой (3 мс)
* *Пропускная способность передачи данных* (200 Мб/сек).

Время передачи данных: T = b/(r\*N)

* b – количество передаваемых данных
* r – скорость вращения (об/с)
* N – число байтов на дорожке

Размер программы на диске – 64 Кб. Каждая дорожка – 32 Кб. Среднее время поиска – 10 мс. Время вращения – 10 мс. Размер страницы 2 Кб. =>

=> 32 Кб 10 мс (вращ.)

2 Кб (1 страница) x мс.

x= (2\*10)/32 = 0.625 (мс)

Пусть страницы – на разных дорожках. Тогда общее время T=Nстр. \* (Tпоиск. + Tсчит.)= (64/2)\* (10+0,625).

## Вычислить время считывания файла, если известны параметры размещения файла на диске и параметры доступа и считывания данных с диска

Есть гибкий диск с шагом чередования 2. На каждой дорожке по 8 секторов по 512 б.Скорость вращения диска 300 об/мин. На первоначальное позиционирование необходимо ½ оборота. =>

=> на считывание одной дорожки – (0,5 (нач. поз.) + 1 + 1 (шаг чередования – 2))/(300/60)=0,5 с. Скорость считывания 1 дорожки – (8\*512)/0,5.

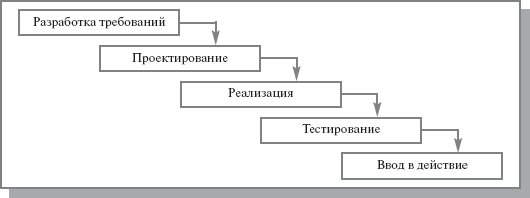
# Технологія створення програмних продуктів

## Життєві цикли програмного забезпечення систем обробки інформації

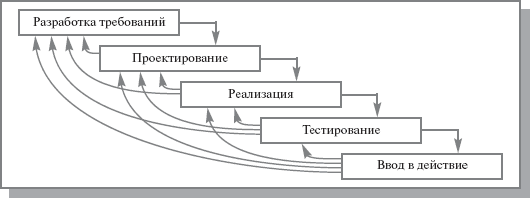
Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Извесны и используются следующие модели жизненного цикла:

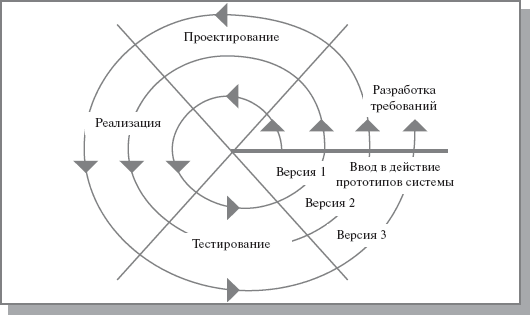
*Каскадная* модель предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.



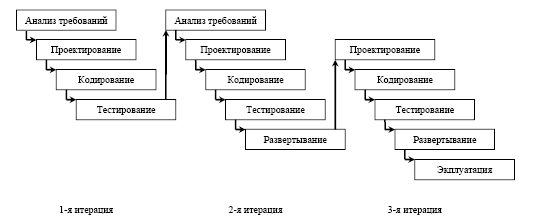
*Поэтапная* модель с промежуточным контролем. Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки.



*Спиральная* модель. На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка. Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

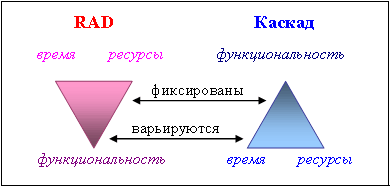


*Ітеративні* або *инкрементальные* моделі припускають розбиття що розробляється ПО на набір частин, які розробляються послідовно, один за одним так, щоб на самому початку — на першій ітерації — можна було розробити частину, не залежну від інших, пройшовши повний цикл робіт на ній, потім оцінити результати і на наступній ітерації або переробити першу частину, або розробити наступну частину, яка може залежати від першої, або якось сумістити доопрацювання першої частини з додаванням нових функцій.

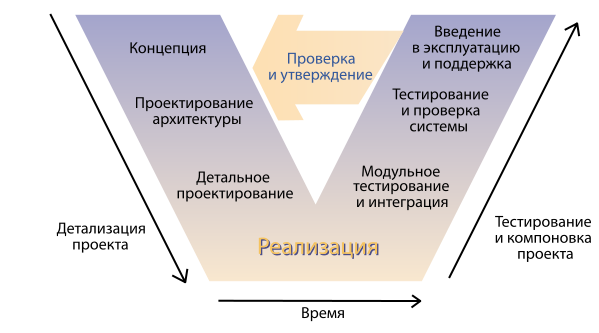


*Rapid Application Development* (RAD) – это жизненный цикл процесса проектирования, созданный для достижения более высоких скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию.

RAD предполагает, что разработка ПО осуществляется небольшой командой разработчиков за срок порядка трех-четырех месяцев путем использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки.



Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять.



## Мери і метрики програмного забезпечення

При планировании программного проекта надо оценить людские ресурсы (в человеко-месяцах), продолжительность (в календарных датах), стоимость (в тысячах долларов). Обычно исходят из прошлого опыта. Если новый проект по размеру и функциям похож на предыдущий проект, вполне вероятно, что потребуются такие же ресурсы, время и деньги.

### Размерно-ориентированные метрики

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Основываются размерно-ориентированные метрики на LOC-оценках (Lines Of Code). LOC-оценка — это количество строк в программном продукте.

;

;

;

.

*Достоинства размерно-ориентированных метрик:*

1) широко распространены;

2) просты и легко вычисляются.

*Недостатки размерно-ориентированных метрик:*

1) зависимы от языка программирования;

2) требуют исходных данных, которые трудно получить на начальной стадии проекта;

3) не приспособлены к непроцедурным языкам программирования.

### Функционально-ориентированные метрики

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC-оценки при этом рассматривается не размер, а функциональность или полезность продукта.

Используется 5 информационных характеристик.

1. *Количество внешних вводов.* Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.

2. *Количество внешних выводов.* Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных внутри отчета отдельно не подсчитываются.

3. *Количество внешних запросов.* Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений. Подсчитываются все запросы — каждый учитывается отдельно.

4. *Количество внутренних логических файлов.* Подсчитываются все логические файлы (то есть логические группы данных, которые могут быть частью базы данных или отдельным файлом).

5. *Количество внешних интерфейсных файлов.* Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

После сбора всей необходимой информации приступают к расчету метрики — *количества функциональных указателей FP* (Function Points).

Количество функциональных указателей вычисляется по формуле

FP = Общее количество х (0,65+ 0,01 x),

где *Fi* — коэффициенты регулировки сложности.

Каждый коэффициент может принимать следующие значения: 0 — нет влияния, 1 — случайное, 2 — небольшое, 3 — среднее, 4 — важное, 5 — основное.

Значения выбираются эмпирически

После вычисления FP на его основе формируются метрики производительности, качества и т. д.:

;

;

;

.

Область применения метода функциональных указателей — коммерческие информационные системы. Для продуктов с высокой алгоритмической сложностью используются метрики *указателей свойств* (Features Points). Они применимы к системному и инженерному ПО, ПО реального времени и встроенному ПО.

Для вычисления указателя свойств добавляется одна характеристика — *количество алгоритмов.*

*Достоинства функционально-ориентированных метрик:*

1. Не зависят от языка программирования.

2. Легко вычисляются на любой стадии проекта.

*Недостаток функционально-ориентированных метрик:* результаты основаны на субъективных данных, используются не прямые, а косвенные измерения. FP-оценки легко пересчитать в LOC-оценки.

## Інженерія вимог до програмних систем

Разделим требования на две большие группы – функциональные и нефункциональные.

*Функциональные* требования являются детальным описанием поведения и *сервисов* системы, ее функционала. Они определяют то, что система должна уметь делать.

*Нефункциональные* требования не являются описанием функций системы. Этот вид требований описывает такие характеристики системы, как *надежность*, особенности поставки (наличие инсталлятора, документации), определенный уровень качества (например, для новой Java-машины это будет означать, что она удовлетворяет набору тестов, поддерживаемому компанией *Sun*). Сюда же могут относиться требования на средства и процесс разработки системы, требования к *переносимости*, соответствию стандартам и т.д. Требования этого вида часто относятся ко всей системе в целом. На практике, особенно начинающие специалисты, часто забывают про некоторые важные нефункциональные требования.

Сформулируем ряд важных свойств требований.

– ясность, недвусмысленность — однозначность понимания требований заказчиком и разработчиками;

– полнота и непротиворечивость;

– необходимый уровень детализации;

– прослеживаемость — важно видеть то или иное требование в различных моделях, документах, наконец, в коде системы;

– тестируемость и проверяемость — необходимо, чтобы существовали способы оттестировать и проверить данное требование;

– модифицируемость.

Определяют следующие виды *деятельности* при работе с требованиями.

* *Выделение требований (requirements elicitation)*, нацеленное на выявление всех возможных источников требований и ограничений на работу системы и *извлечение тренований* из этих источников.
* *Анализ требований (requirements analysis)*, целью которого является обнаружение и устранение противоречий и неоднозначностей в требованиях, их уточнение и систематизация.
* *Описание требований (requirements specification)*. В результате этой *деятельности* требования должны быть оформлены в виде структурированного набора документов и моделей, который может *систематически* анализироваться.
* *Валидация требований (requirements validation)*, которая решает задачу оценки понятности сформулированных требований и их характеристик, необходимых, чтобы разрабатывать ПО на их основе, в первую очередь, непротиворечивости и полноты, а также соответствия корпоративным стандартам на техническую документацию.

## Функціональне тестування програмних систем

Функціональне тестування, воно ж тестування чорного ящика — тести для нього, а також використовувані критерії повноти проведеного тестування визначають на основі вимог до функціональності.

**Функціональне тестування** – один з основних видів незалежного тестування програмного забезпечення, спрямований на перевірку реалізованості функціональних вимог.  Інакше кажучи, фахівці з функціонального тестування визначають, чи вирішує розроблене ПЗ завдання, заради якого воно було створено, чи задовольняє воно потреби замовника / користувача.

Основні критерії функціонального тестування:

* Функціональна придатність продукту.
* Точність виконання поставлених завдань.
* Відповідність стандартам розробки.
* Відповідність кінцевої реалізації продукту пред’явленим функціональним вимогам.
* Захищеність від несанкціонованого використання і некоректних дій користувачів.

Рівні тестування:

* Модульне (компонентне) (Unit testing)
* Інтеграційне (Integration Testing)
* Системне (System testing)
* Регресійне (Regression testing)
* Приймальне (Acceptance testing)

**Переваги функціонального тестування:**

* імітує фактичне використання системи.

**Недоліки функціонального тестування:**

* можливість упущення логічних помилок в програмному забезпеченні;
* ймовірність надмірного тестування.

## Структурне тестування програмних систем

Структурне тестування, воно ж тестування білого ящика — тести створюються на основі знань про структуру самої системи і про те, як вона працює. Критерії повноти засновані на відсотку елементів коду, які відпрацювали в ході виконання тестів (покриття коду тестами). Для оцінки ступеня відповідності вимогам можуть притягуватися додаткові знання з дослідження вимог.

Структурне тестування програмного забезпечення може бути реалізоване такими методами:

* тестуванням маршрутів;
* тестуванням циклів;
* тестуванням обробки даних.

**Тестування маршрутів**

Тестування структури маршрутів програмних модулів виконують шляхом перевірки коректності виділених маршрутів виконання програм і виявлення логічних помилок формування маршрутів. На практиці за відсутності впорядкованого аналізу потоків управління деякі маршрути у програмі (до 50%) виявляються пропущеними під час тестування.

**Тестування циклів**

Наявність циклів у програмних модулях здатне різко збільшувати складність їхнього тестування. На складність тестування циклу впливає його структура та два параметри: число маршрутів у тілі циклу та число ітерацій циклу. За умови зростання кожного з цих параметрів пропорційно зростає їхній добуток, а отже, і складність тестування. Тому вичерпне тестування реальних складних програм із циклами практично неможливе.

**Тестування обробки даних**

Функціонування будь-якої програми можна розглядати як обробку потоку даних, переданих від входу у програму до її виходу. Вхідні дані послідовно використовуються для визначення ряду проміжних результатів аж до одержання необхідного набору вихідних даних.

## Види архітектури програмних систем. Межі використання різних архітектур.

Архитектура ПО обычно содержит несколько видов, которые аналогичны различным типам чертежей в строительстве зданий.

Архитектурный вид состоит из 2 компонентов:

* Элементы
* Отношения между элементами

Архитектурные виды можно поделить на 3 основных типа:

1. Модульные виды (англ. *module views*) — показывают систему как структуру из различных программных блоков.
2. Компоненты-и-коннекторы (англ. *component-and-connector views*) — показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).
3. Размещение (англ. *allocation views*) — показывает размещение элементов системы во внешних средах.

Примеры модульных видов:

* Декомпозиция (англ. *decomposition view*) — состоит из модулей в контексте отношения «является подмодулем»
* Использование (англ. *uses view*) — состоит из модулей в контексте отношения «использует» (т.е. один модуль использует сервисы другого модуля)
* Вид уровней (англ. *layered view*) — показывает структуру, в которой связанные по функциональности модули объединены в группы (уровни)
* Вид классов/обобщений (англ. c*lass/generalization view*) — состоит из классов, связанные через отношения «наследуется от» и «является экземпляром»

Примеры видов компонентов-и-коннекторов:

* Процессный вид (англ. *process view*) — состоит из процессов, соединённых операциями коммуникации, синхронизации и/или исключения
* Параллельный вид (англ. *concurrency view*) — состоит из компонентов и коннекторов, где коннекторы представляют собой «логические потоки»
* Вид обмена данными (англ. *shared-data (repository) view*) — состоит из компонентов и коннекторов, которые создают, сохраняют и получают постоянные данные
* Вид клиент-сервер (англ. *client-server view*) — состоит из взаимодействующих клиентов и серверов и коннектором между ними (например, протоколов и общих сообщений)

Примеры видов размещения:

* Развертывание (англ. *deployment view*) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов
* Внедрение (англ. *implementation view*) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т.д.)
* Распределение работы (англ. *work assignment view*) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

## Моделі управління програмних систем

### Адміністративна модель (теорія X)

Традиційна ієрархічна модель.

Характерні риси моделі:

* Владна піраміда.
* Чіткий розподіл ролей і обов'язків.
* Чіткий розподіл відповідальності.
* Проходження інструкціям, процедурам, технологіям.
* Роль менеджера: планування, контроль, ухвалення основних рішень.

Переваги моделі: ясність, простота, прогнозованість. Модель добре поєднується з каскадною моделлю життєвого циклу і застосовна в тих же випадках, що і каскадна модель. Модель ефективна у разі сталого процесу.

Недоліки моделі пов'язані з тим, що адміністративна система прагне самозбереженню (стабільності) і погано сприйнятлива до зміни ситуації – нові типи проектів, застосування нових технологій, оперативна реакція на зміну ринку. Крім того, в адміністративній моделі погано уживаються індивідуалісти і генератори ідей.

### Модель хаосу (теорія Y)

У основі моделі хаосу лежить Теорія Y, яка є повною протилежністю Теорії X. Основна теза Теорії Y: робота — природна і приємна діяльність і більшість людей, насправді, дуже відповідальні і не ухиляються від роботи.

Характерними рисами моделі хаосу є:

* Відсутність явно виражених ознак влади
* Роль менеджера – поставити завдання, забезпечити ресурсами, не заважати і стежити, щоб не заважали інші
* Відсутність інструкцій і регламентованих процедур
* Індивідуальна ініціатива – рішення з проблеми приймається там, де проблема виявлена
* Процес нагадує творчу гру учасників.

Переваги такої моделі в тому, що творча ініціатива учасників нічим не зв'язана і потенціал учасників розкривається повною мірою. Це буває особливо ефективно у разі, коли для вирішення проблеми потрібний пошук нових підходів, методів, ідей і засобів.

### Відкрита архітектура (теорія Z)

Адміністративна і хаотична моделі є двома «крайнощами», між якими знаходяться безліч моделей, що поєднують переваги «крайніх» моделей. Однією з таких моделей є модель відкритої архітектури, заснована на Теорії Z.

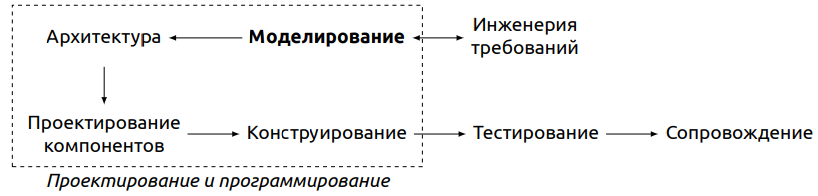
Особливостями цієї моделі є:

* Адаптація до умов роботи – якщо робимо незалежні модулі, то розходимося і робимо, якщо потрібна архітектура бази даних, то збираємося разом і обговорюємо ідеї.
* Колективне обговорення проблем, вироблення консенсусу і ухвалення рішення – не всі можуть погодиться, але ухвалене рішення є колективним і через це – обов'язковим для всіх.
* Розподілена відповідальність – відповідають всі, хто обговорював, виробляв, приймав.
* Динаміка складу робочих груп залежно від поточних завдань.
* Відсутність спеціалізації – учасники міняються ролями і функціями і можуть при необхідності замінити один одного.
* Завдання менеджера – активна участь в процесі, контроль конструктивності обговорень, забезпечення можливості активної участі всіх.

Відкрита архітектура є гнучкішою, такою, що адаптується, настроюється на ситуацію. Вона дає можливість проявити себе всім членам команди – в ній можуть уживатися і індивідуалісти і колективісти. Колективне обговорення висловлених ідей дозволяє залишати тільки прагматичні ідеї.

## Моделювання програмних систем. Типи і призначення моделей структурного моделювання.

https://softandware.org.ua/wp-content/uploads/2014/11/07-modeling.pdf



Представление – абстрактная модель системы, выделяющая ее характеристики, соответствующие определенному аспекту ее функционирования.

Основные представления:

▶ контекстное представление модель окружения, в котором выполняется система;

▶ взаимодействия связи системы с окружением, а также элементов в системе;

▶ структурное представление организация системы и данных для обработки;

▶ поведение модель реагирования системы в ответ на внешние события.

### Контекстные модели

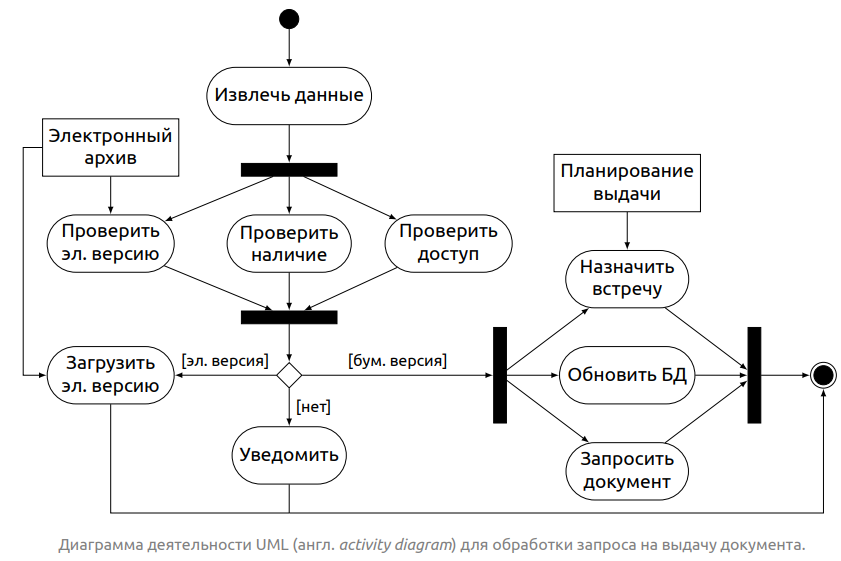
Цель:

▶ разграничение функций системы и ее окружения;

▶ определение компонентов, которые надо имплементировать, и используемых интерфейсов.



Модель процесса:



### Модели взаимодействия

Цели:

▶ отладка взаимодействия с пользователями;

▶ выработка пользовательских требований;

▶ определение возможных «узких мест» и проблем коммуникации;

▶ отработка производительности (англ. performance) и надежности (англ. dependability).

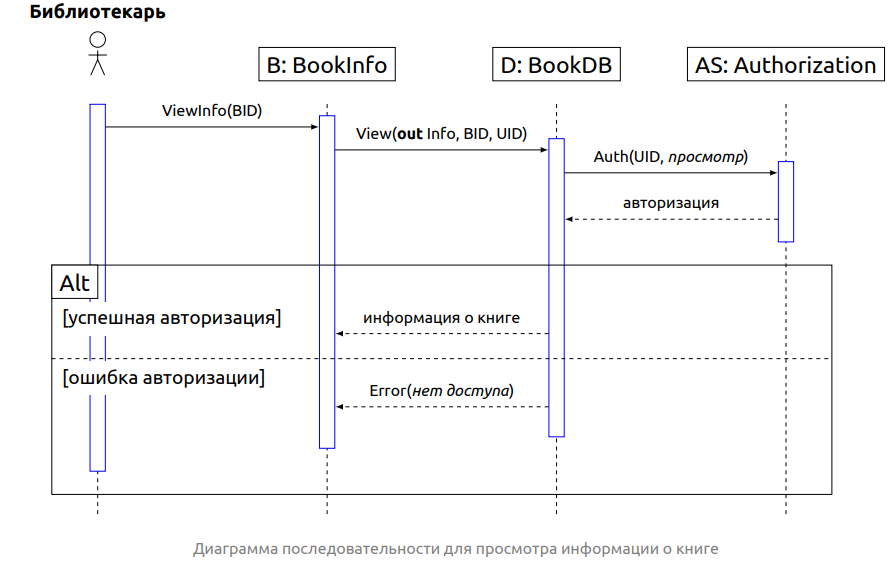
Типы моделей:

▶ варианты использования моделирование взаимодействия системы с актерами

(пользователями или другими системами); (use case diagram, діаграма прецидентів)

▶ диаграммы последовательностей моделирование взаимодействия компонентов

системы. 



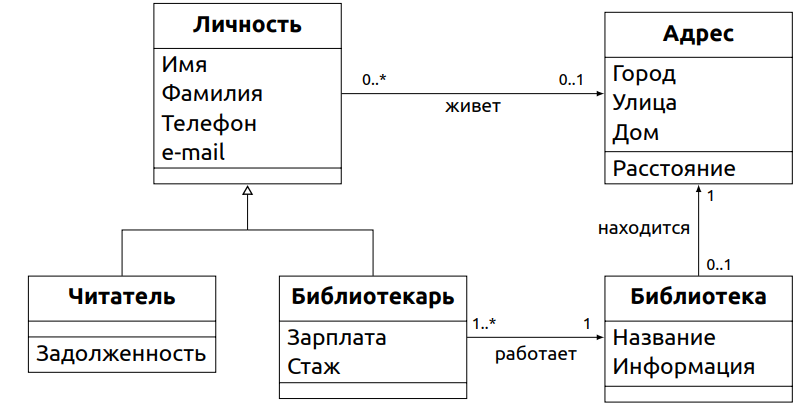
### Структурные модели

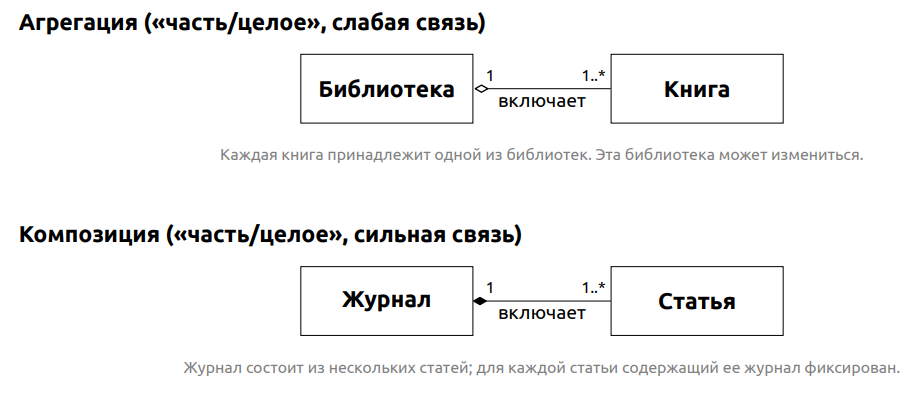
Цели:

▶ определение архитектуры системы;

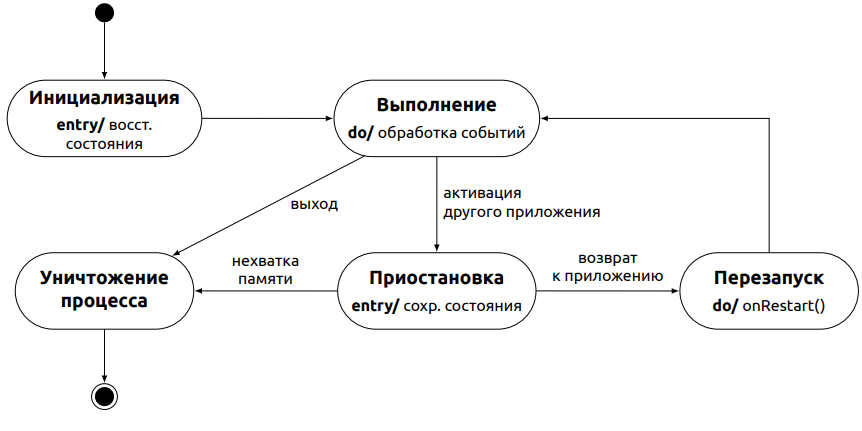
▶ определение структур хранения данных.





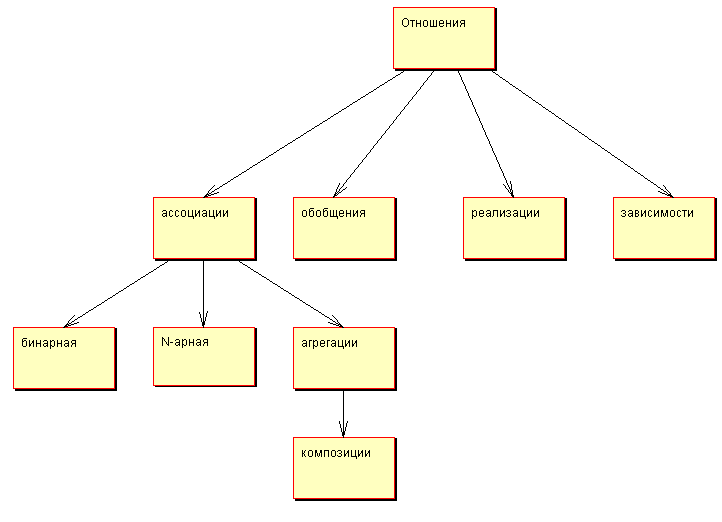


### Модели поведения

Цель: определение реакции системы на внешние и внутренние входные сигналы. 

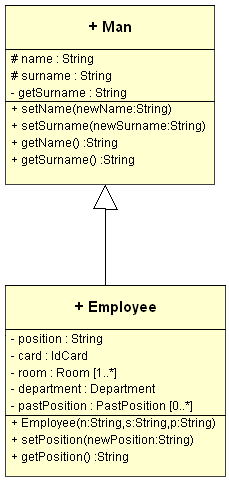
## Уніфікована мова моделювання (UML), значення UML в процесі проектування програмного забезпечення.

Диаграмма классов UML позволяет обозначать отношения между классами и их экземплярами.



### Обобщения

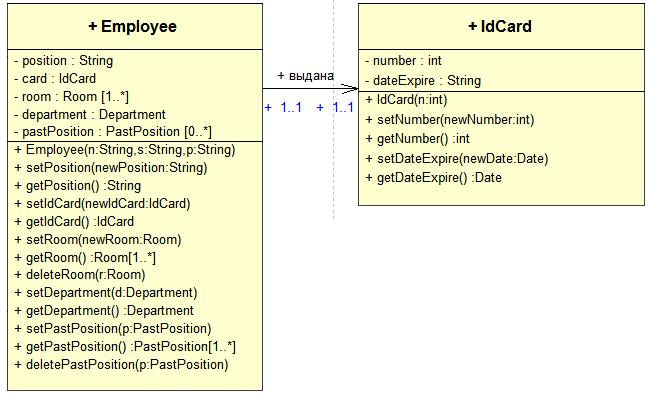
Отношение обобщения — это наследование



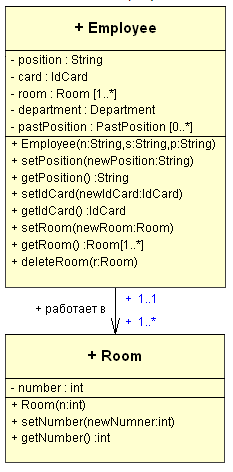
### Ассоциация

Ассоциация показывает отношения между объектами-экземплярами класса.

Каждому сотруднику может соответствовать только одна идентификационная карточка, мощность связи 1 к 1.

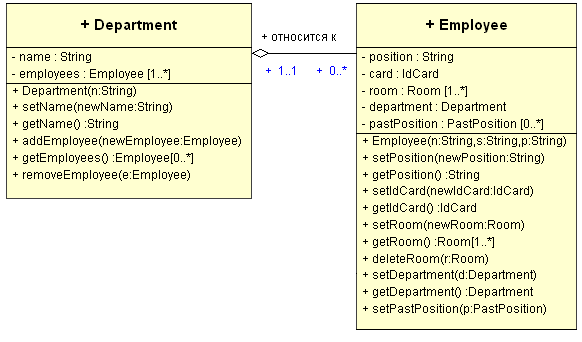


Каждому объекты работник(Employee) может соответствовать несколько рабочих помещений.



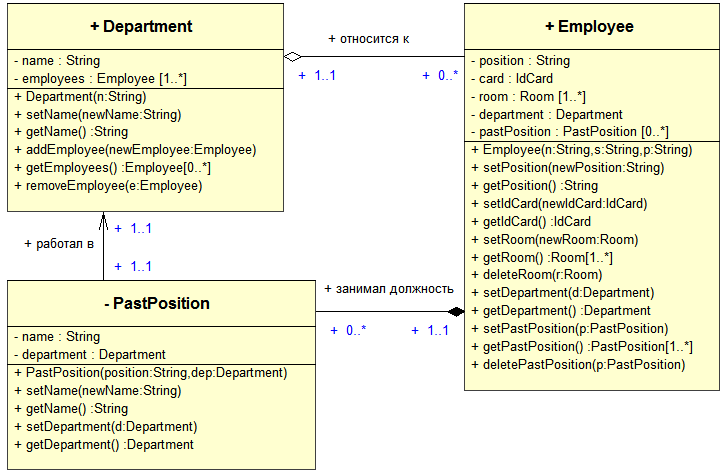
### Агрегация

В каждом отделе может работать один или более человек. Можно сказать, что отдел включает в себя одного или более сотрудников и таким образом их агрегирует. На предприятии могут быть сотрудники, которые не принадлежат ни одному отделу, например, директор предприятия.



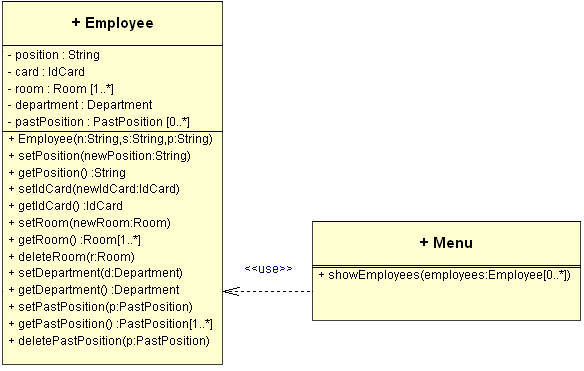
### Композиция

Данные о прошлых занимаемых должностях являются частью данных о сотруднике, таким образом между ними связь целое-часть и в то же время, данные о прошлых должностях не могут существовать без объекта типа «Employee». Уничтожение объекта «Employee» должно привести к уничтожению объектов «pastPosition».



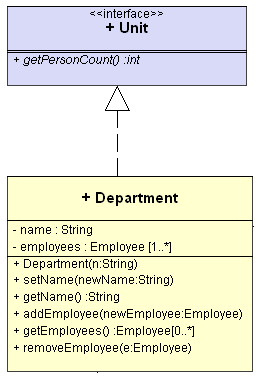
### Зависимость

Параметром для метода является массив объектов «Employee». Таким образом, изменения внесенные в класс «Employee» могут потребовать и изменения класса «Menu».



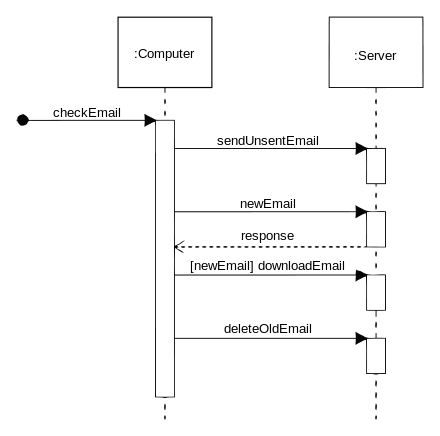
### Реализация

Если представить, что организация может делиться не только на отделы, а например, на цеха, филиалы и т.д. Интерфейс «Unit» представляет собой самую абстрактную единицу деления.



## Диаграмма последовательности

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. lifeline), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.





## Діаграма прецедентів

Расширение (англ. Extend) — разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.

Включение (англ. Include) — определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого всегда задействуется базовым вариантом использования.

