# 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.

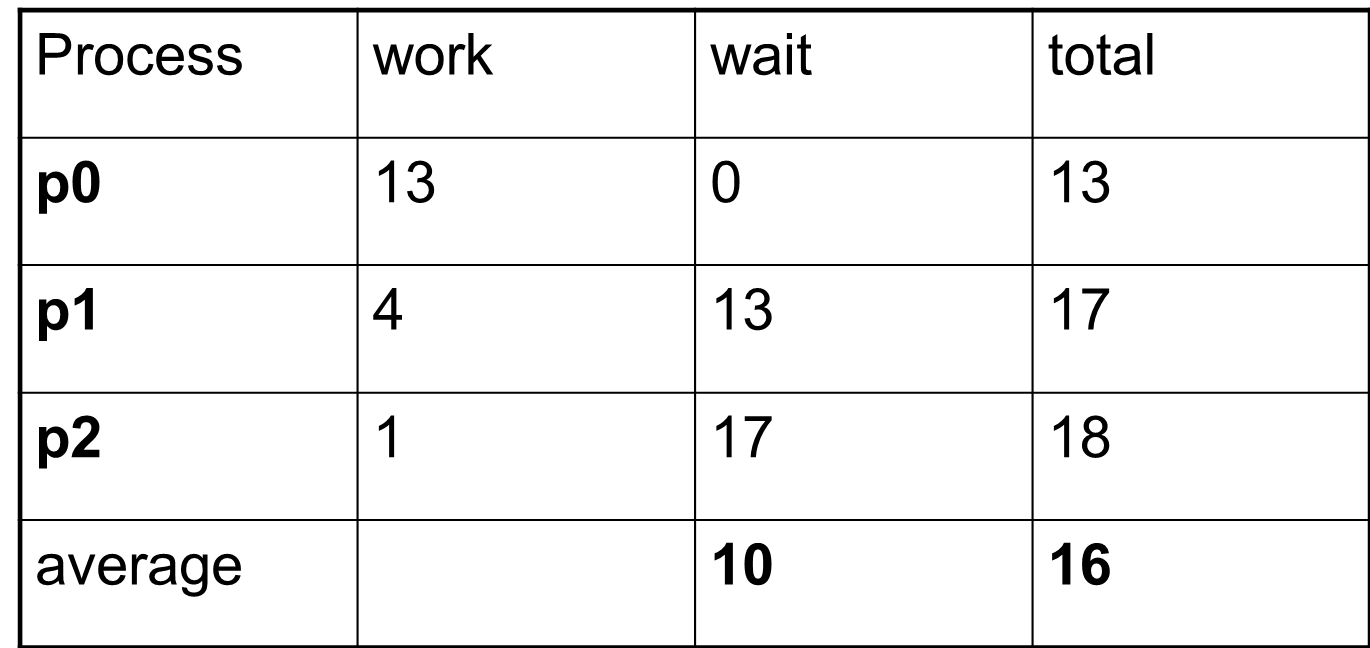
Параметры планирования

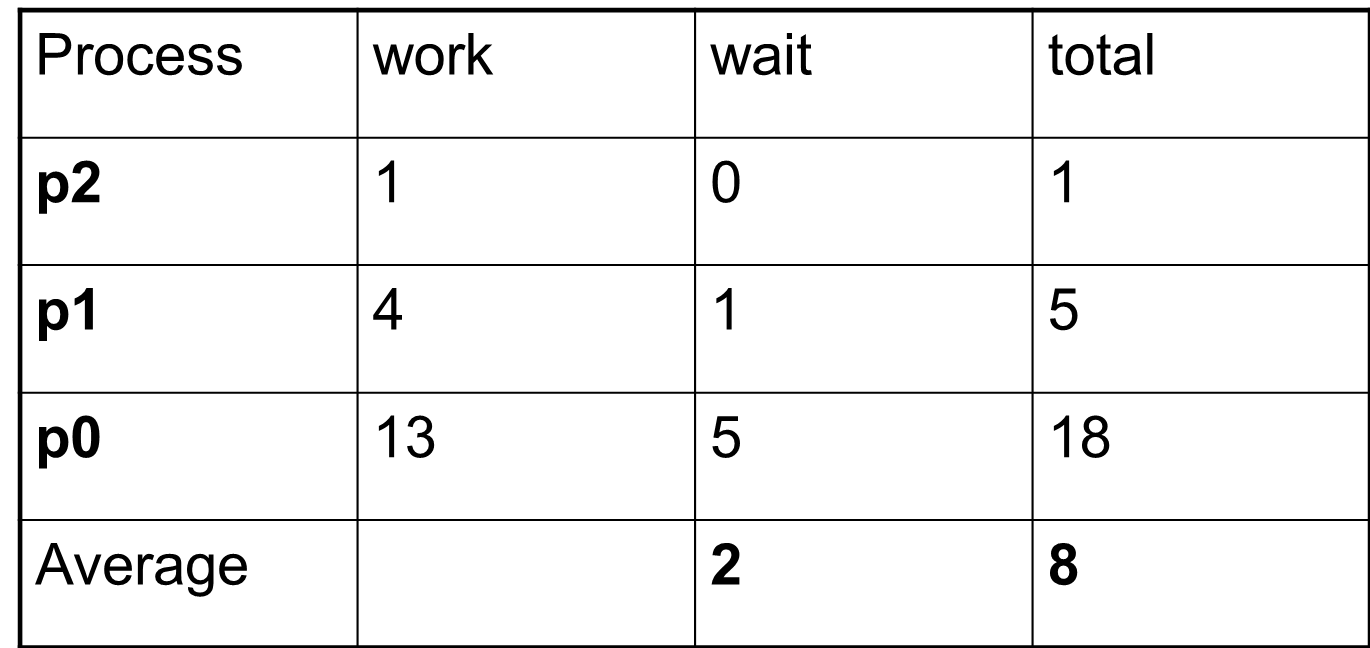
* Динамические параметры системы – количество свободных ресурсов на данный момент времени.
* Статические параметры системы – предельные значения ресурсов:
  + объем оперативной памяти
  + количество подключенных устройств ввода-вывода
  + объем своп-раздела

## 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)



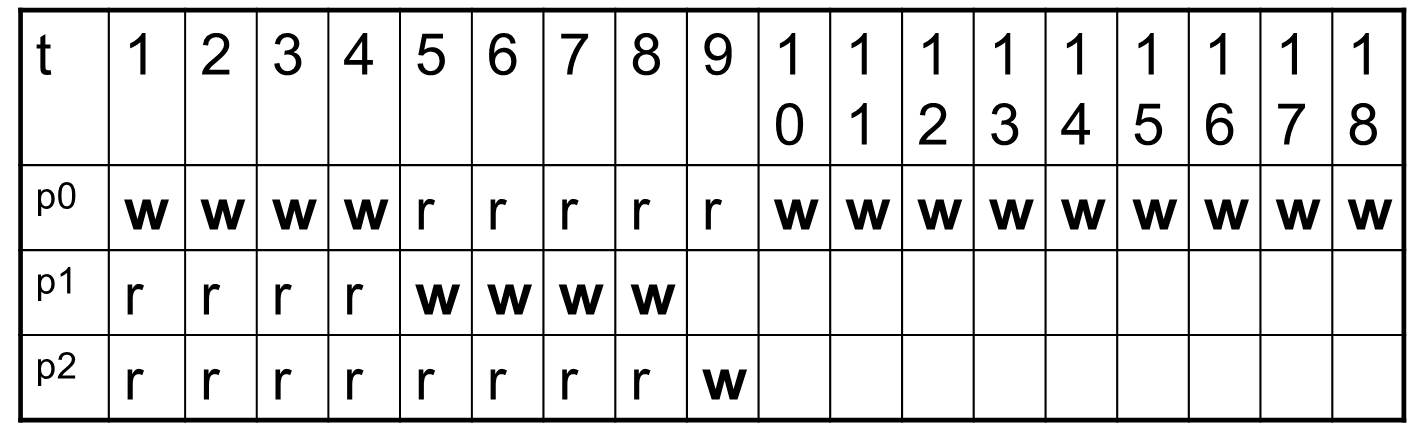


### Round Robin (RR)

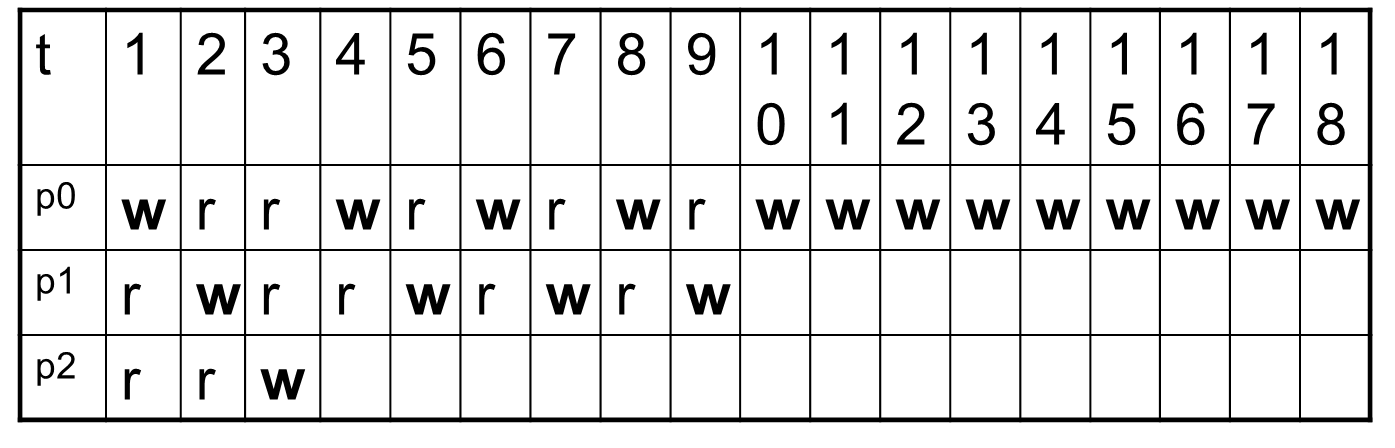
* Вытесняющее планирование
* Циклический список процессов, находящихся в состоянии готовности
* Каждому процессу выделяется определенный квант времени
* Производительность зависит от величины кванта времени

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**;

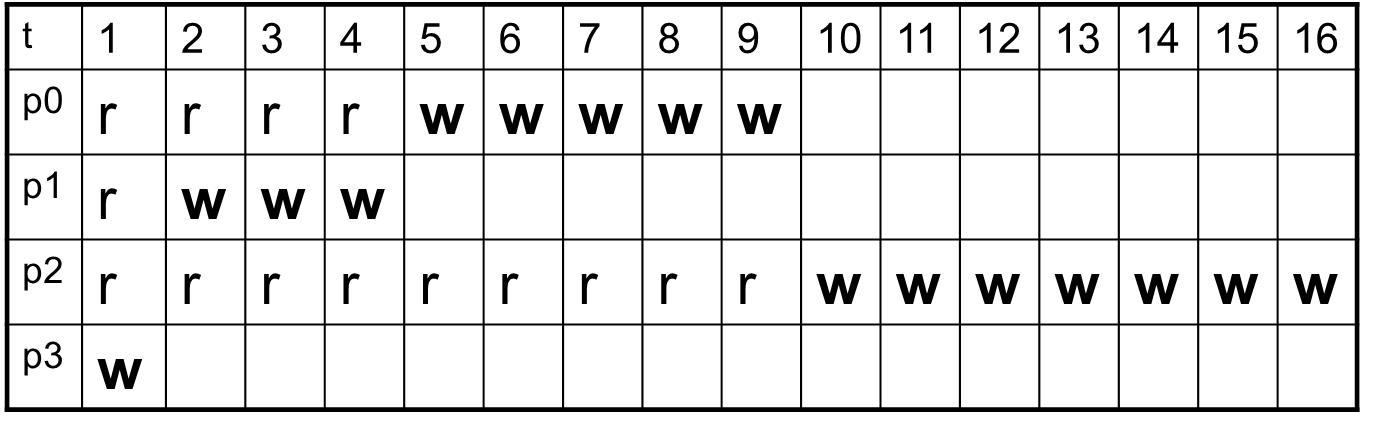


### Shortest Job(Task) First (SJF)

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

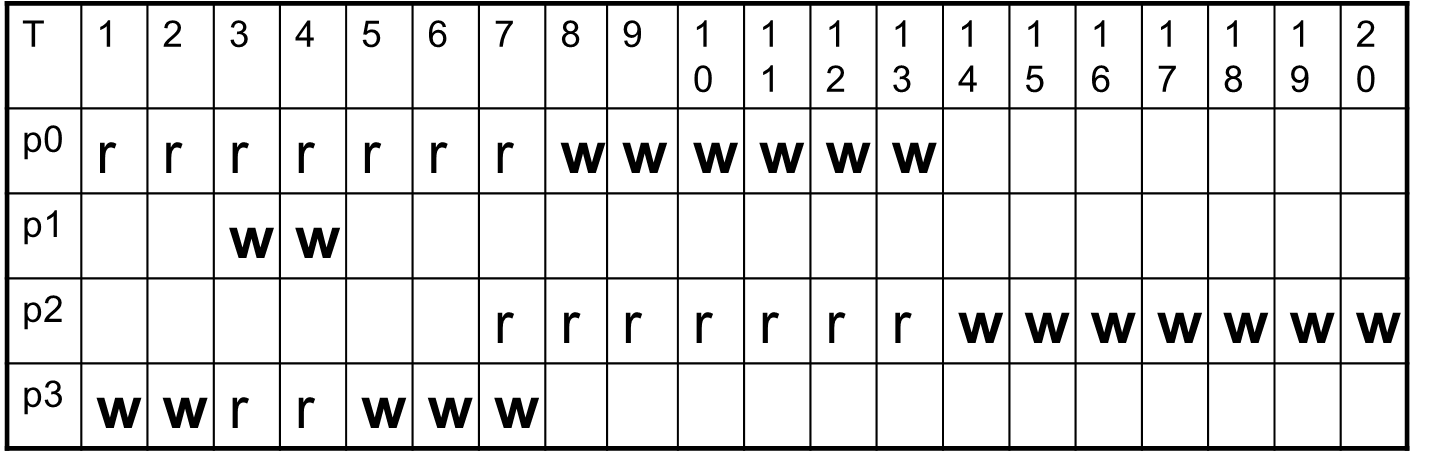
Невытесняющий SJF

* 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

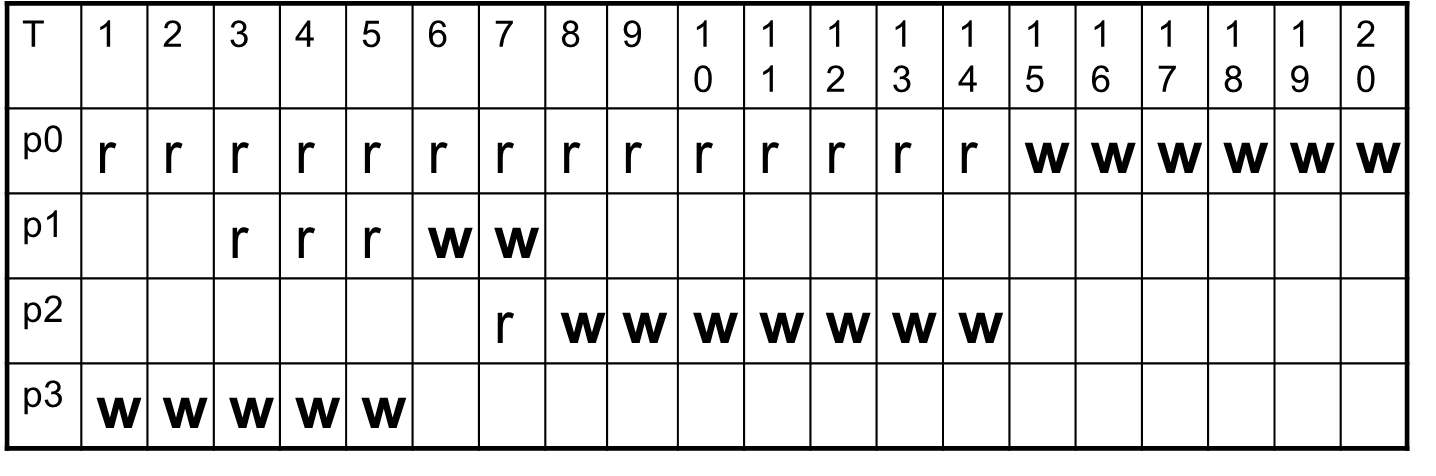
### Приоритетное планирование

Гарантированное планирование и SJF – частные случаи приоритетного планирования

Вытесняющее и невытесняющее планирование.

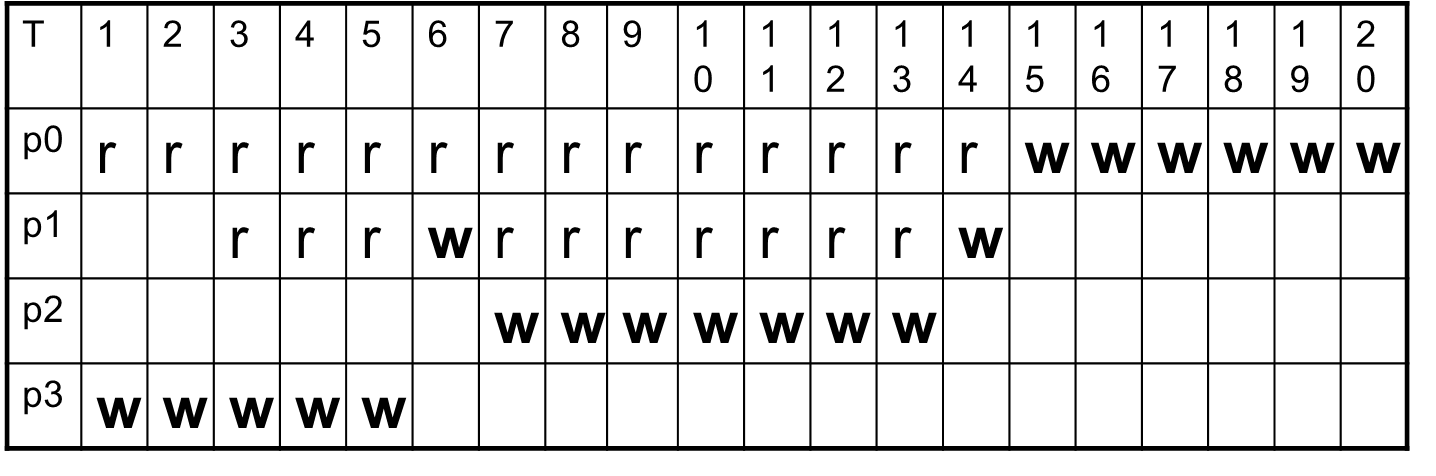
Приоритетное планирование невытесняющее

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



Приоритетное планирование вытесняющее

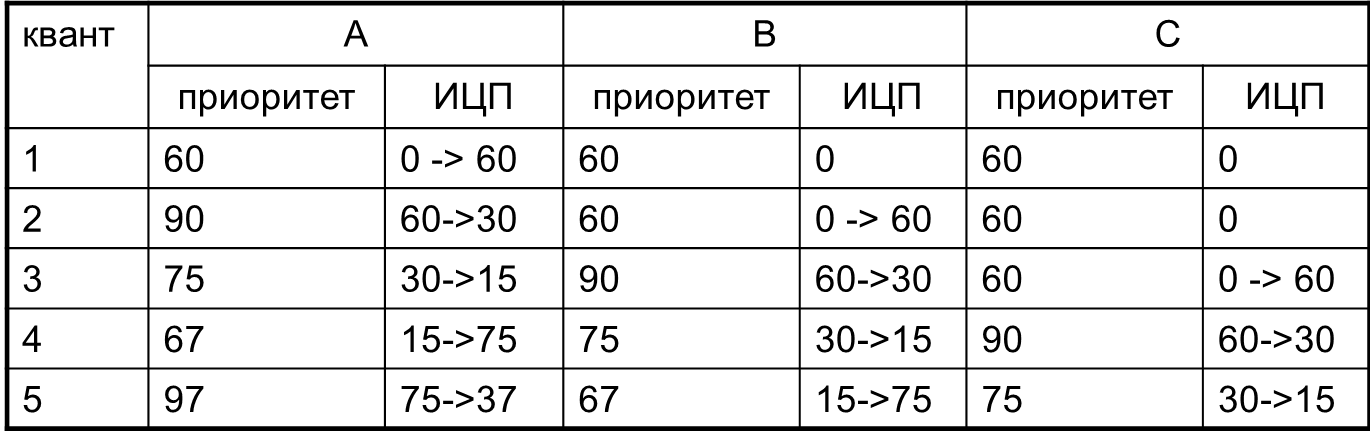
* CPU burst = {6,2,7,5};
* T\_start = {0,2,6,0};
* Priority = {4,3,2,1};



### Диспетчеризация

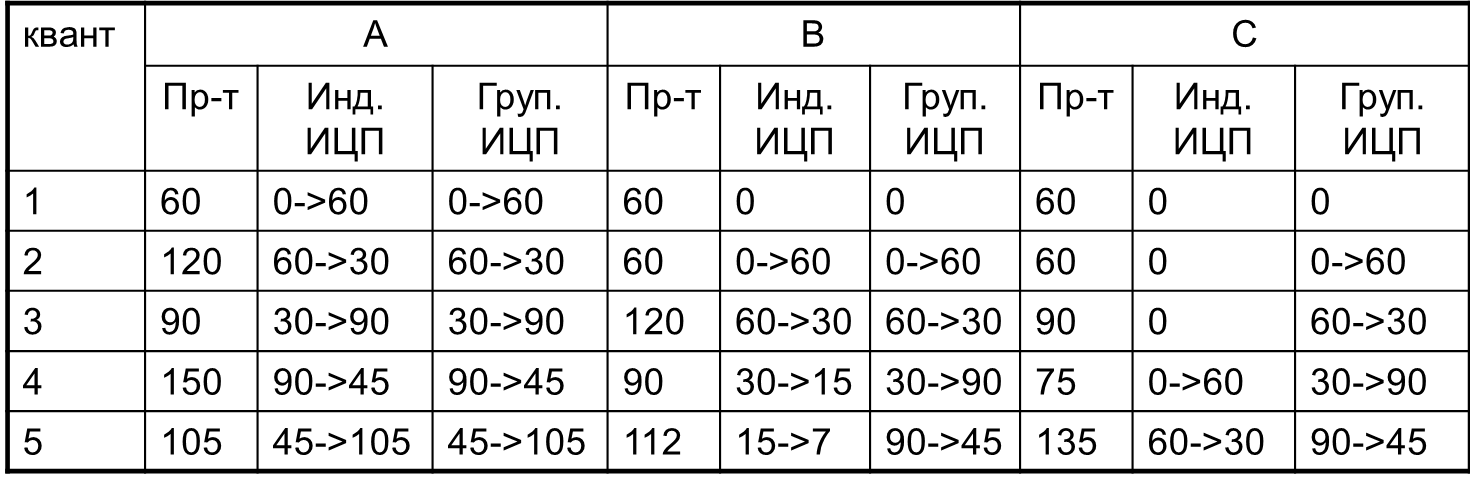
1)

* 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

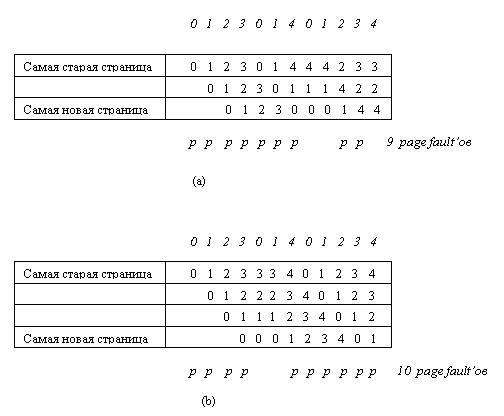
* Замещается самая «старая» страница

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.

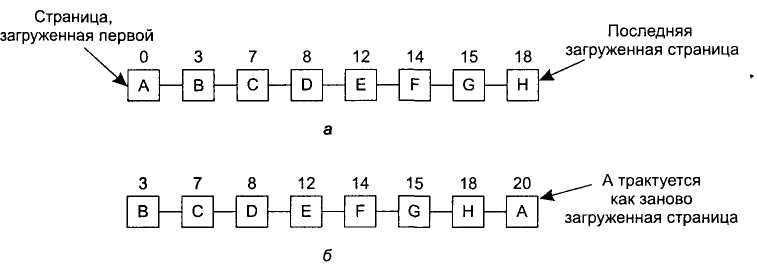


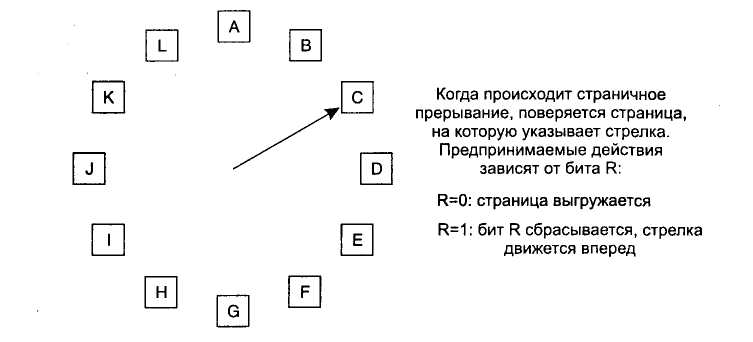
Рис. 3.1. Действие алгоритма «вторая попытка»: страницы, отсортированные в порядке очереди (FIFO) (а); список страниц, если страничное прерывание произошло во время 20, а страница *А* имеет бит R, равный 0 (б)

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