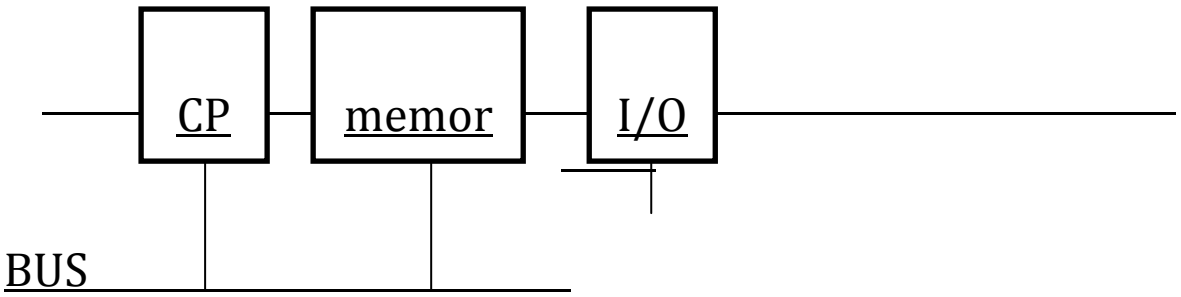
Операционни системи

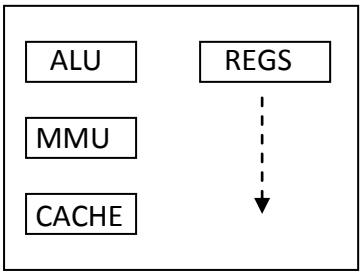
Въведение



Шината може да се счита за 4-ти ресурс.

2^{10}= 1KB =1024B
2^{20}= 1MB =1024KB = 1048576B
2^{30}= 1GB =1014MB = 1024.1024KB
2^{40}= 1

CPU

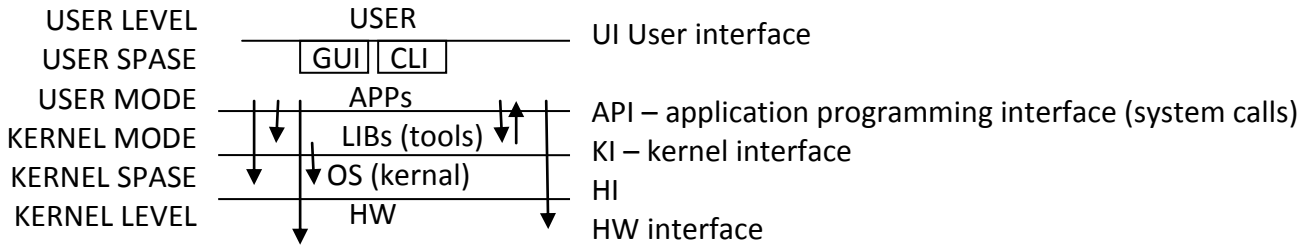


Aritmethic Logic Unit
Memory Menagment Unit
REGS – Регистри

Всеки CPU Има поне 2 режима на работа:

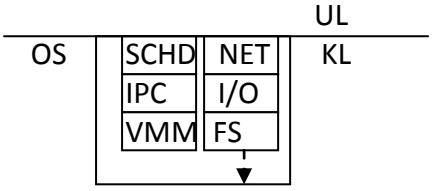
- USER MODE (режим потребител)
- KERNAL MODE (режим ядро) – OS

Софтуерни слоеве и интерфейси в КС



APPs -> LIBs - memory
APPs <- LIBs - memory
APPs -> HW - printf
LIBs – OS - KERNEL sys calls

Структури на OS, модули в OS



SCHD – SCHEDUER – планировчик
IPC – Inter Process Communication
VMM – virtual Memory Menagment
FS – File Sistem

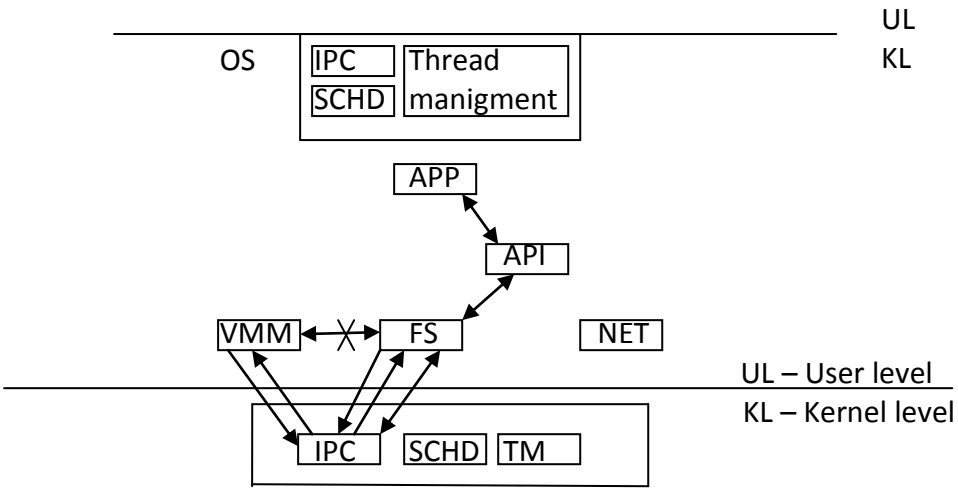
Фиг 1

OS с макро ядра

- Monolithic kernels (монолитни ядра OS) –фиг 1
- Module – based Monolithic kernals (модулно – базирани монолитни ядра)

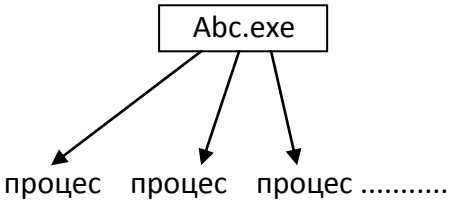
Съвременни OS

-Micro kernel OS
НЕ!!!!



Процеси и нишки

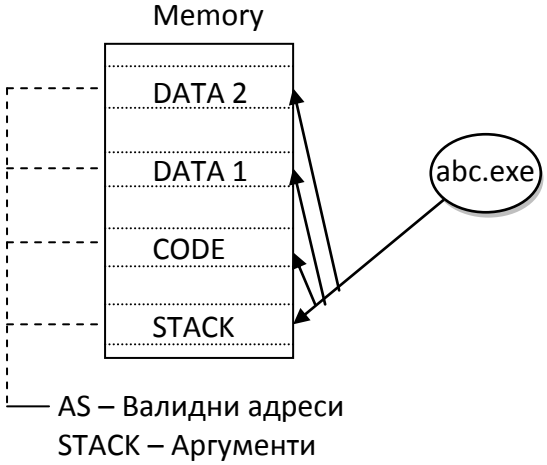
Процеси



ПЕОЦЕС – Това е единица в OS, която има собствено адресно пространство и е основната единица за притежаване на ресурси.
Единствената единица, която има своето собствено адресно пространство AS (adress pace) и основна едицина за притежание на ресурси.

Забележка: Цялата тема е в контекст за съвременните ОС!

AS (adress space) – Това е множеството от валидни логически адреси за даден процес

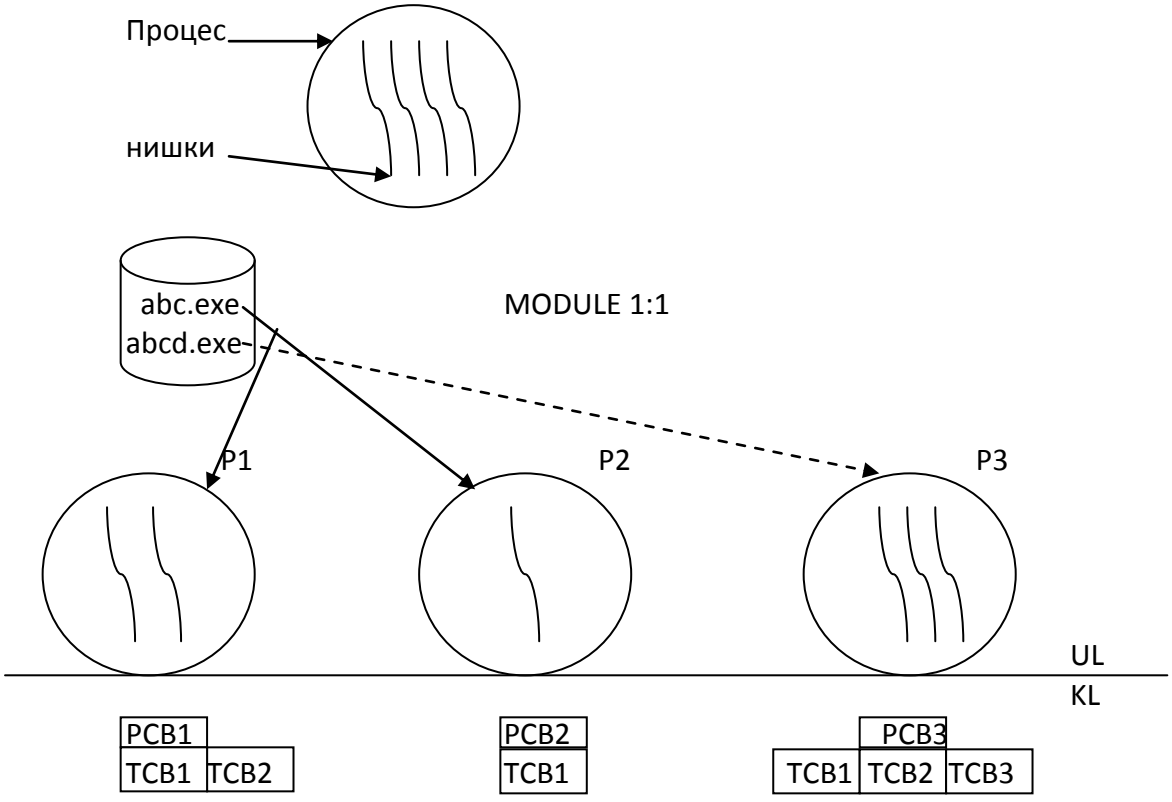


Основна функция на AS е предаване на процесите един на друг
Процес:
Може да има множество сегменти като DATA и CODE, но винаги само 1 – STACK.

- Всеки процес си има Proces ID – PID
- Всеки процес има PCB – Proces Control Block (данни, описващи процеса)
    - PID
    - Owner(USER)
    - Кога е създаден
    - Памет

Нишки

НИШКА е единствена единица в OS за планиране и изпулнение на програмен код.
Нишката не съществува самостоятелно, тя е винаги част от един процес.
В един процес може да има множество нишки, но минимум 1.
Когато се създаде 1 процес, автоматично се създава 1 нишка. Това е т.нар. главна нишка – MAIN THREAD.
Всяка нишка е част от процес, която има собствен набор от значения на машинните регистри и собствен потребителски стек. Всички останали ресурси на процеса са общи за всички негови нишки. Това означава, че нишките ще са по-ефективни за създаване и поддържане от процесите и превключването между тях ще се осъществява по-лесно.
Нишката не притежава ресурси. Един от малкото ресурси, които притежават нишката това е т.нар регистров контекст, всяка нишка има свой собствен регистров контекст.
Всяка нишка има свой собствен стек и стеков указател:
Всяка нишка има: TCB, TID;



WINSOWS

WIN32 API функция:
Create Process(…….)

HANDLE – тип в WIN32 API
Създаване на нишка:

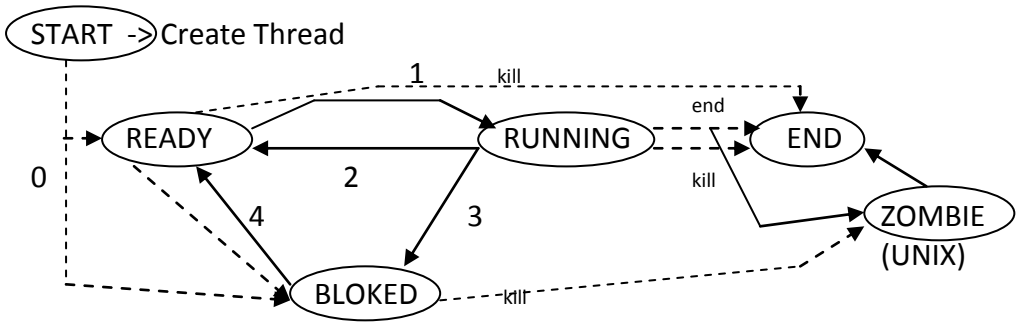
```
HANDLE = CreateThread(
  __in_opt   LPSECURITY_ATTRIBUTES lpThreadAttributes,  // NULL
  __in       SIZE_T dwStackSize,                        // 0
  __in       LPTHREAD_START_ROUTINE lpStartAddress,
  __in_opt   LPVOID lpParameter,
  __in       DWORD dwCreationFlags,
  __out_opt  LPDWORD lpThreadId
);
```

```
Struct thread info
{
        HANDLE h;
        DWORD t;
};
```

Thread function – Код на нишката
Когато Tread function на една нишка приключи, тогава нишката се води приключена.

DWORD WINAPI име(void*p)
{
……..
Return x;
}

Състояния на процеси и нишки



Основните състояния на нишката са 3: Ready, Running, Blocked.

- Ready – това е, когато нишката е готова за изпълнение, но не е получила процесор и чака.
- Running – означава, че нишката е присвоена на процесора и си работи.
- Blocked – Нишката чака за приклйчване на някаква бавна операция. Може да се получи и при явно блокиране на др. Нишка;
  Чака за вх/изх; изчакване за синхронизационен примитив (ресурс).

Причини:

- За 1 и 2 е планировчика – SHEDULER
- 3 – вх/изх; явно блокиране; синхр. Примитив;
- 4 – 3
- READY – BLOCKED – явно блокиране;

Връзката между процеса и нишките

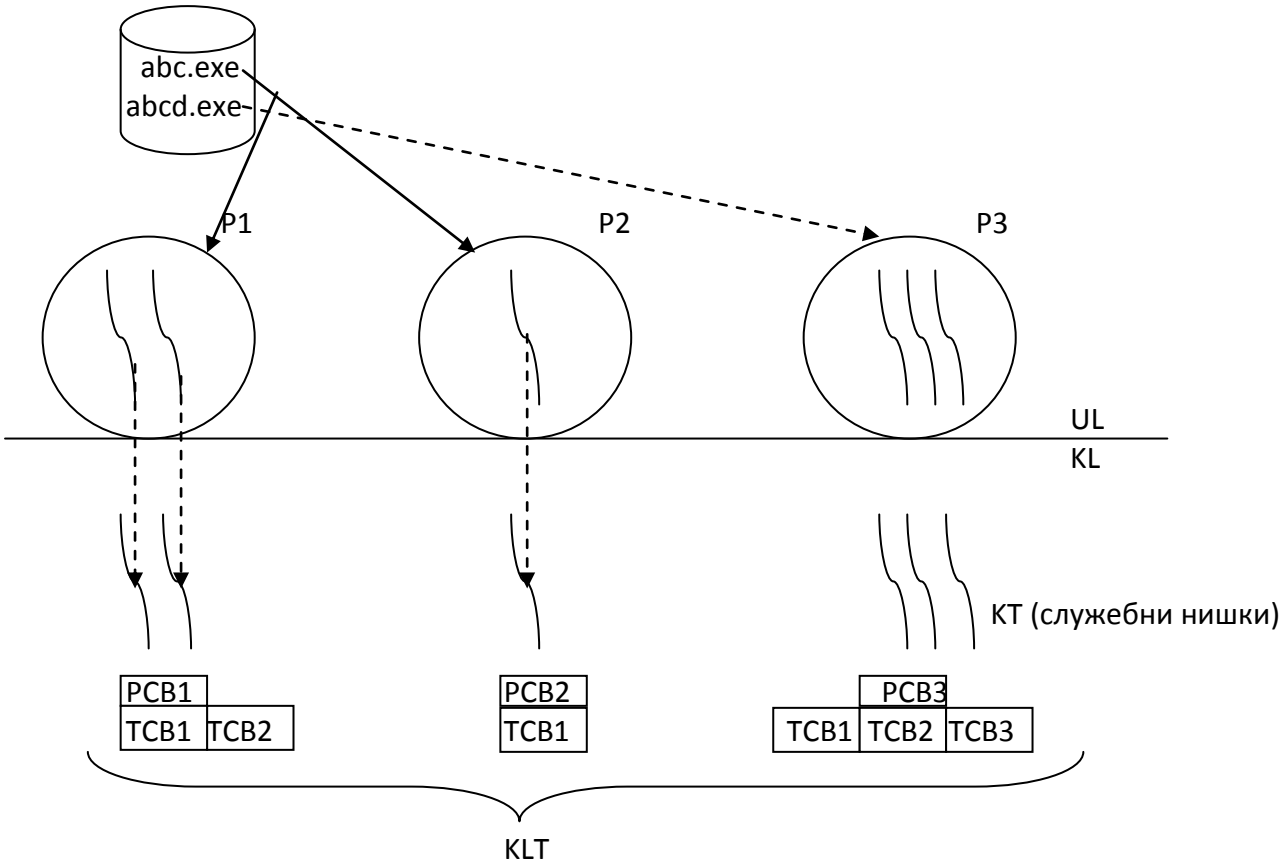
Ако поне 1 нишка в 1 процес е Running или Ready, то съответно целият процес е Running или Ready.
Един процес ще бъде блокиран, ако всички нишки са блокирани.
Rn>Re>Bl

Многонишкови модели (Multithreading models)

3 модела:

1/ „1:1“ KLT – най-разпространен (Kurnel Level Thread)

OS прилага само KLT - нишки
KLT != KT



Предимства:

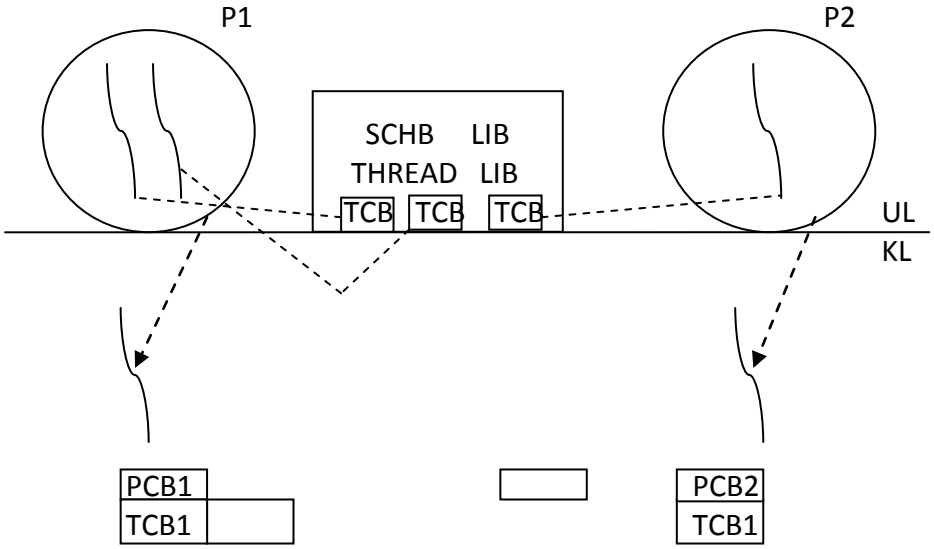
- Може да се получи малък паралелизъм, тъй като OS планира KLT нишките, а съответствието между нишките е 1:1
- Блокиране на 1 нишка НЕ ВОДИ до блокиране на процеса.

Недостатъци:

- Превключване между нишки предизвиква преход UL->KL, contact swich

Всички съвременни OS работят с модел 1:1

2/ „m;1“ PULT, Pure User Level Thread



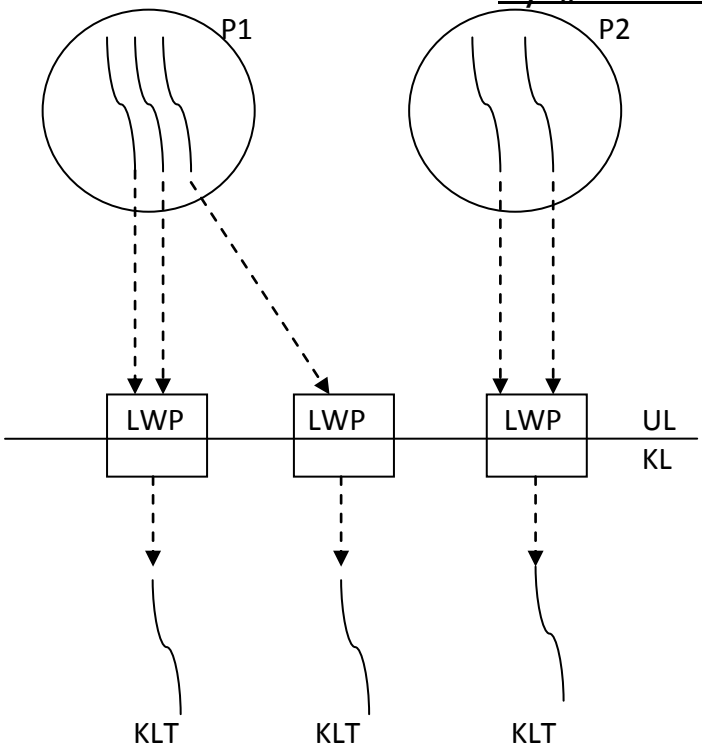
Предимства:

- Възможно е планиране на алгоритъма на работа с нишките.
- Превключването между нишките е бързо, без contact swich.

Недостатъци:

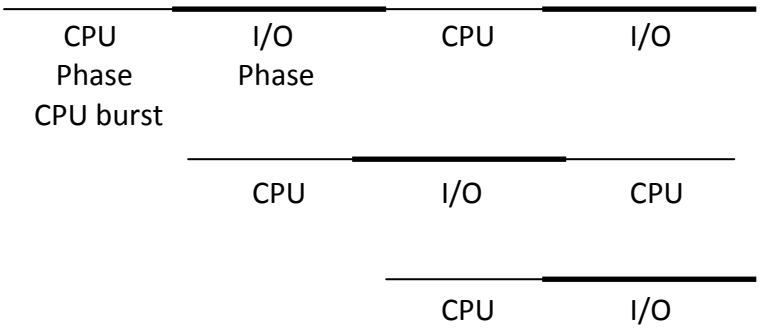
- При блокиране на 1 нишка, блокира и процеса.
- Невъзможност за истински паралелизъм.

3/ „m:n“ hybrid modul

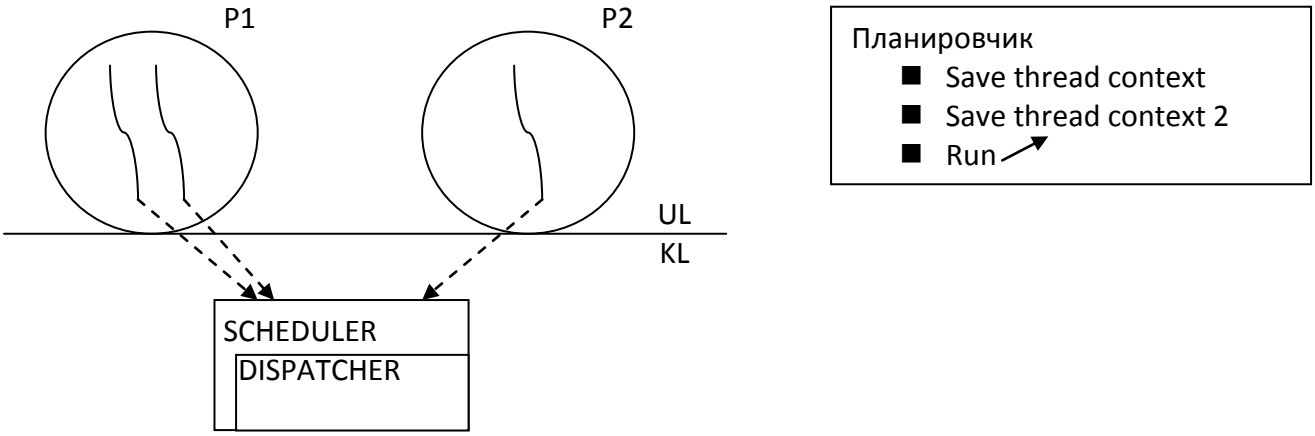


LWP – Light Weight Process
Не ни трябва!

Планиране (Scheduling)



Псевдопаралелни изпълнения:



Алгоритми за планиране (Scheduling policies)

- Приоритетно базирани – priority based
- Неприоритетно базирани – non priority based

Прекъсваеми – preemptive
Непрекъсваеми – non preemptive

Основни параметри:
Throughput – пропускаща скорост – брой задачи, приключили за единица време
Turnaround time – Време за изпълнение – време, необходимо за извършване нач->край
Waiting time – време за изчакване

Оценка на производителността Производителността на различните дисциплини на обслужване е съществен фактор при съответна дисциплина. Не е възможно да се направи окончателно
сравнение, тъй като относителната производителност ще зависи от следните

фактори вероятностното разпределение на времето за обслужване на различните процеси; ефективността на диспечирането в контекста на механизмите на превключване; естеството на загубите от В/И;
производителността на В/И система.
Дисциплините на обслужване, чрез които се избира следващия елемент за обработка, са независими от времето за обслужване, представено чрез следната връзка: $T_r / T_s = 1 / 1 - \rho$, където T_r - оборотното време -времето на пребиваване, общото време в системата, turnaround time, времето между приемането на процеса и неговото завършване.
Включва действителното време за изпълнение, плюс времето изразходвано за изчакване на ресурси, включително и процесора). Процесор Освободени процеси RQ0 Допуснати процеси Процесор RQ1 Процесор RQn
6 - 9 T_s - средното време за обслужване на процеса. Средното време в състояние готов. ρ - коефицент на използване на процесора. Един диспечер, базиран на приоритети, в който приоритета на всеки процес се присвоява независимо от очакваното време за обслужване, осигурява същото
средно оборотно време и средно нормализирано оборотно време, както в простата FCFS дисциплина на обслужване. Следователно, наличието или липсата на превключване няма да доведе до разлики в тези средни стойности.
С излючение на RR и FCFS, различните дисциплини на обслужване (обсъдени по-горе) извършват избора на базата на очакваното време на обслужване. За съжаление, това води до големи затруднения при разработването на аналитичните модели на тези дисциплини.

Таблица на процесите:

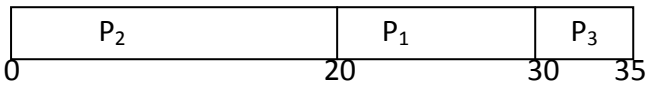
P (process)	T време за изпълнение	A кога е пристигнала
P_1	10	1
P_2	20	0
P_3	5	3

Неприоритетно базирани

FCFS (first come first served)

Първи дошъл – първи обслужен (First-Come-First-Served, FCFS,FIFO) Това е най-простата дисциплина на обслужване. Известна е като FIFO (first- come-first-out). Когато един процес стане готов, той се включва в опашката на готовите процеси. Когато текущо изпълняваният процес престане да се изпълнява, избира се за изпълнение процес, който е бил най-дълго в опашката на готовите процеси.
FCFS е по-благоприятна за процеси, които повече са обвързани с процесора (процес, който повече използва процесора), отколкото тези, които са обвързани с В/И (процес, който има повече В/И).
FCFS е дисциплина без превключване. Тя не се използва самостоятелно в едно-процесорните системи. По-ефективна е при комбинирането й с приоритетна схема на обслужване.

Непрекъсваем алгоритаъм.



Att – average turnaround time
Awt – average waiting time

awt = $\frac{Wtp_1+Wtp_2+Wtp_3}{3} = \frac{19+0+27}{3} = 15,33$

wt+T

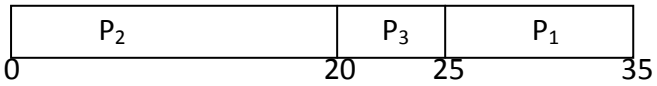
19+10

att = $\frac{29+20+32}{3} = \frac{81}{3} = 27$

SJF(shortest job first)

Непрекъсваем алгоритъм

Определяме подредбата като извадим от turnaround waiting



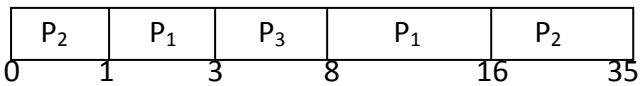
awt = $\frac{24+0+17}{3} = \frac{41}{3} = 13,67....$

att = $\frac{34+20+22}{3} = 25,...$

SRTF (shortest remaining time first)

Прекъсваем алгоритъм.

Най-краткото оставащо време (Shortest Remaining Time, SRT) SRT е версия на дисциплината SPN, но с превключване. Това е приемлив вариант за системите с времеделене. Диспечерът избира най-краткото очаквано оставащо време за обработка. Когато един нов процес се присъедини към опашката на готовите процеси, възможно е той да има по-кратко оставащо време за обработка, отколкото текущо изпълнявания процес. Следователно, диспечерът може да превключи всеки път когато нов процес стане готов. Както при SPN, SRT трябва да разполага с изчисленото време за обработка, за да се изпълни функцията за избор. Тук отново има риск за увисване на по-дългите процеси.



$$awt = \frac{\overbrace{8-3}^{5} + \overbrace{16-1}^{15} + 0}{3} = 6{,}6\ldots. \qquad att = \frac{\overbrace{5+10}^{15} + 35 + 4}{3} = 18{,}\ldots..$$

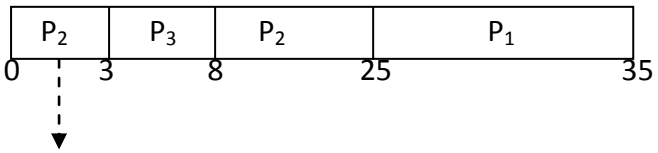
Приоритетно базирани

P	T	A	Pr приоритет
P_1	10	1	8
P_2	20	0	9
P_3	5	3	10

По-голямо число – по-голям приоритет.

Задължително прекъсваем.

Priority – based



Приема P_1, но е с по-голям приоритет и нищо не става

$$awt = \frac{24+5+0}{3} = 9{,}6\ldots.$$

$$att = \frac{34+25+5}{3} = 21{,}\ldots.$$

Dinamic priority

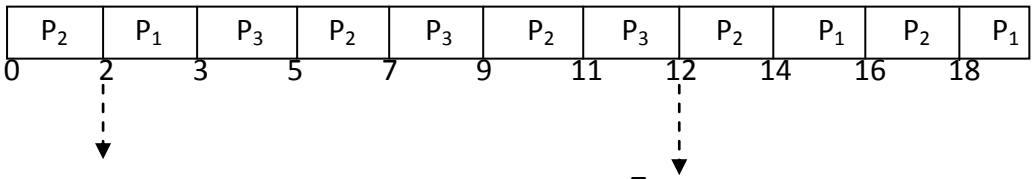
DP = f(SP,……,……,……)

Round – Robiu (въртелешка)

Кръгова (циклична) дисциплина (Round Robin, RR) Един прост начин да се намали престоя на кратките процеси, който се получава при FCFS, е да се използва превключване, базирано на системния таймер.
Най-простата дисциплина от този вид е кръговата (RR). Прекъсване по таймер се генерира през определен интервал от време. Когато се появи прекъсването, текущо изпълняваният процес се поставя в опашката на готовите процеси и се избира следващото готово задание на базата на FCFS. Тази техника е известна като насичане (квантоване) на времето, защото на всеки процес е предоставен отрязък от времето преди да бъде превключен.

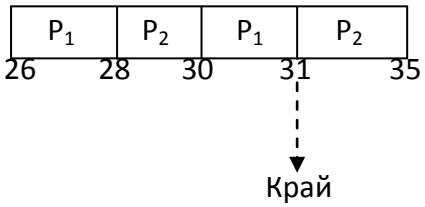
RR може да бъде усъвършенствана, за да се отстрани несправедливото получаване на порции процесорно време.

Пример:
q=2 кванта започва да се брой от момента на стартиране на процеса.



P_3 още не е пристигнал

Приключва

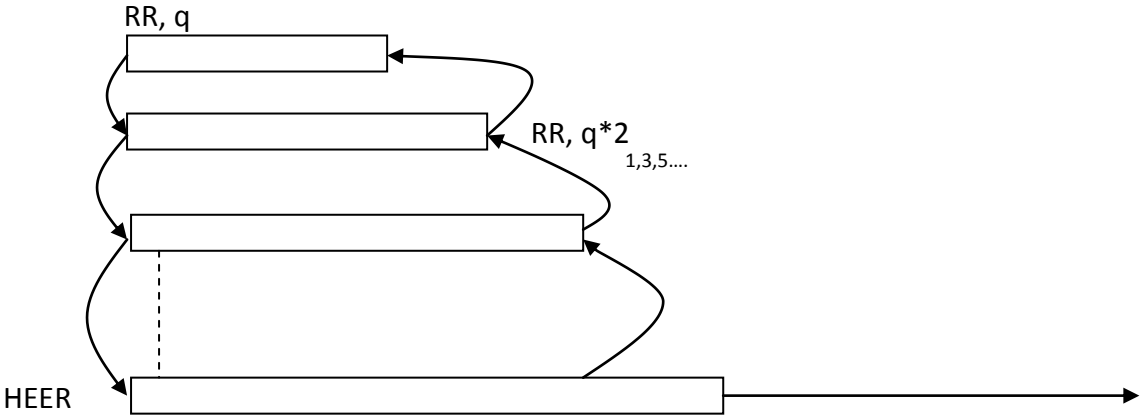


Край

awt = $\frac{20+17+4}{3} = 41 = 13,6....$

att = $\frac{30+37+9}{3} = 25,....$

Multilevel Feedback Quake
Multilevel Priority Quake



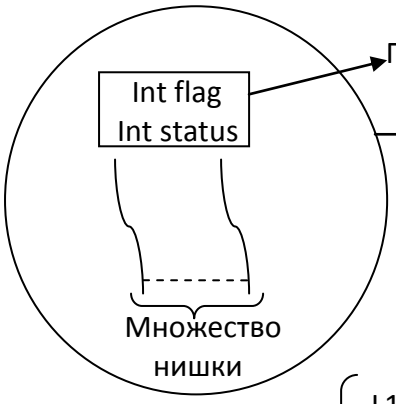
Няма да има диаграми на Гант!!!

Синхронизация на нишки

T[n].h = Create Thread

Генерира 1
Генерира 2
Генерира 3
Генерира n

Една програма се свежда до няколко инструкции. Дори 1 обикновен оператор, може да генерира 10-ки функции. Една нишка може да бъде прекъсната от планировчика, Между 2 инструкции, т.е след като едната приключи и другата да започне.



Глобална променлива

процес

```
while(flag);
flag=1;
if(status==100)
{
        …….
        …….
}
flag=0;
```

Може да има точка на прекъсване.

```
L1;
CMP  flag 0;
JNE  L1
MOV  flag 1
```

```
L1;
CMP flag 0
MOV flag 1
JNE L1
```

TAS (Test And Set) За да се реши пробл. с синхронизацията в процесорите съществува инструкцията TAS – хардуерно решение с критичната секция.

Teterson's solution

```
L1;
Tas flag
JNZ L1
```

bts – bit test and set
Btr – bit test and reset

В интелските процесори

```
Int tas(int*flag)
{
……
}
```

```
while(tas(&flag));
if(status==100)
{
……..
}
Flag=0;
```

Критична секция

Правила за валидно решение на критичната точка

- Mutual Exclusion - Взаимно изключване – критичната точка може да има само 1 нишка.
- Ограничено чакане – Bonuded waiting
- Progress – Прогрес – нишката да напредва към изхода
- Без времеви предложения – No timing assumption

MUTEX

MUTEX – (MUTual EXclusin) – Синхр. примитив и в даден момент само 1 нишка може да го притежава или никой да не го притежав. Ако някоя нишка опита да „вземе“ MUTEX, който е притежаван, тя блокира – „нишката е блокирана на MUTEX'а“. На 1 MUTEX може да са блокирани неограничен брои нишки. Когато една нишка реши да напусне MUTEX'а, тогава 1 нишка от всички, които са блокирании го чакат се взима.

Създаване на MUTEX в WIN32 API:
HANDLE h= Creat Mutex
WaitForSingleObject(h, timeout)
INFINITE
Release Mutex(h)

SEMAPHOR

СЕМАФОР – синхр. Примитив, който има брояч, освен това има и 2 финкции, които се дефинират в семафора:

- P(s) – lock(entry section) – тя получава семафора
- V(s) – exit section – освобождава функция

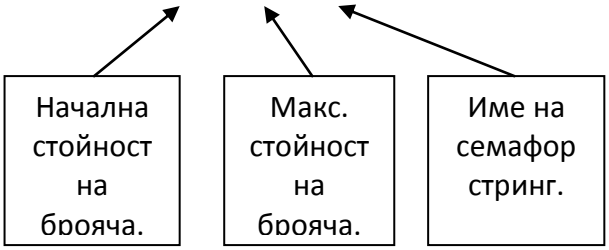
P(s) прави следното – проверява дали брояча на семафора е 0, ако е 0, нишката, която получава P(s) блокира, ако брояча е !=0 (обикновенно по-голям от 0, някои OS допускат отрицателен брояч.) брояча се намаля с 1 и си продължава изпълнението.

V(s) увеличава брояча на семафора с 1 и ако има блокирана нишка автоматично се извършва P(s) за нея, т.е нишката се разблокирва.

Брояча на семафора винаги има max стойност и начална стойност. Двете са зададени при създаване на семафора.
Създаване на семафор:

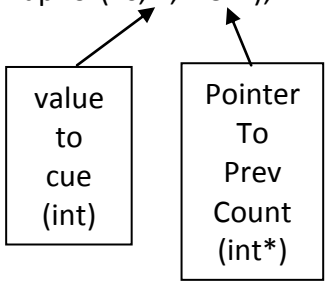
Create Semaphore()
WaitForSingleObject – P(s)
Release Semaphore – V(s)

HANDLE hs = CreateSemaphor (NULL,init,max,NULL);

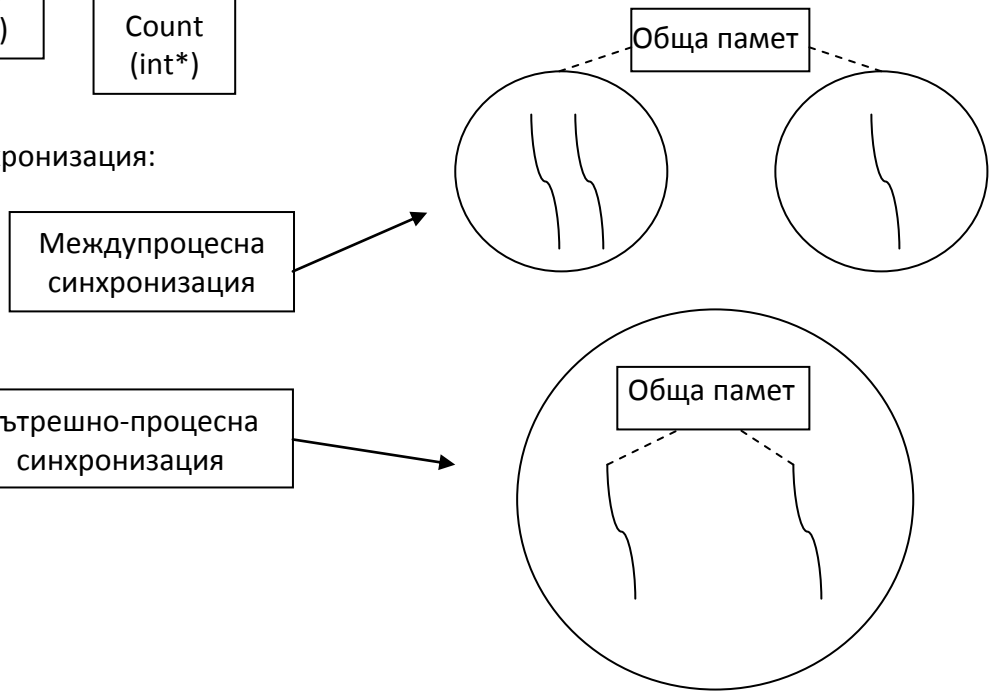


WaitForSingleObject(hs, INFINITE)

ReleaseSemaphor(hs, 1, NULL);

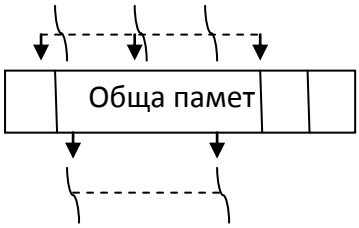


Има 2 вида синхронизация:



Пример за вътрешнопроцесна синхронизация:

Producer – Consumer



Producer thread

Consumer thread

```
S1 arr[1000];
Struct S1
{
        Char empty;
        Int data;
}
```

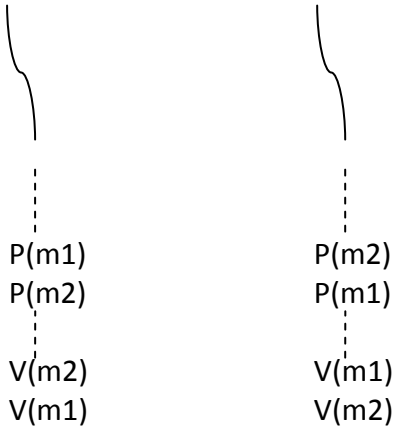
mw – mutex за write
mr – mutex за read
Se – семафор за броене за празини ел/ти
Sf – семафор за броене на пълни ел/ти

```
void write(int data)
{
        P(Se)
        For(int n=0; !arr[n].empty; n++);
        {
                Arr[n].data = data;
                Arr[n].empty = 0;
        }
        V(Sf);
}
```

```
Int Read()
{
        int tmp;
        P(sf);
        For(int n=0; arr[n].empty; n++)
        {
                tmp = arr[n].data;
                arr[n].empty = 1;
                V(Se);
                Return tmp;
        }
```

DEADLOCK

Мъртва хватка



*Когато 2 или повече нишк борят за повече от 1 процес, който всички нишки искат да вземат.

Има 4 условия за възникване на дедлок:

- Ако всички условия са изпълнени, 1 система е податлива към деадлок.
- Дори 1 от тези условия да е нарушен няма деадлок (deadlock free).

- Mutual exclusion (взаимно изпълнение на нисхки)
- Hold and wait (държи се ресурс и се чака за друг)
- No preemption (неотнемаемост на ресурс)
- Circular wait (циклично чакане)

Методи за борба с деадлок

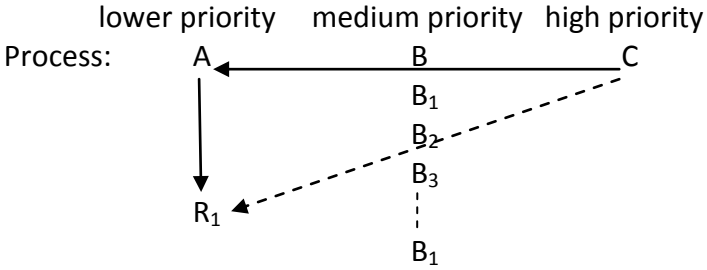
- Deadlock avoidance – избягване на деадлок.
  - Матрица с процеси и техните ресурси:

	R_1	R_2	R_3	....
P_1				
P_2				
P_3				
....				

- Deadlock prevention
  - Нарушава някои от 4-те условия, системата да стане deadlock free(no preemption).

- Deadlock detection and recovery – установяване и възстановяване на/от деадлок.

Priority invertation



WATCHING TIMER
Mars pathfinder – сонда

Priority ceiling
Priority in puritance

IPC
Inter Process Communication

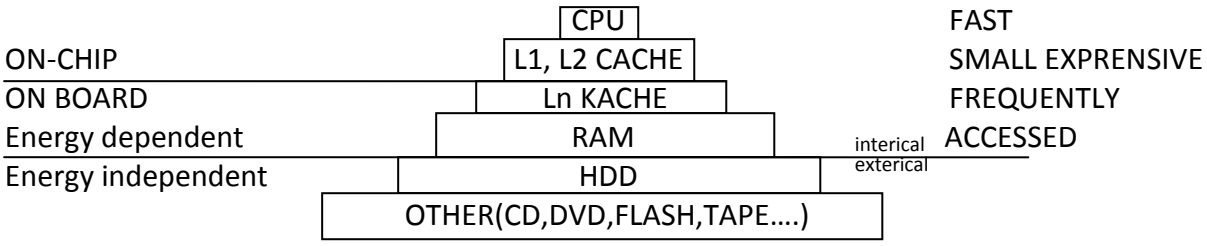
- Shared memory НЕ!
- Pipe
- Sorest
- Synchronization primitives

Памет

- Cache memory
- Memory management
- Virtual memory

Cache memory

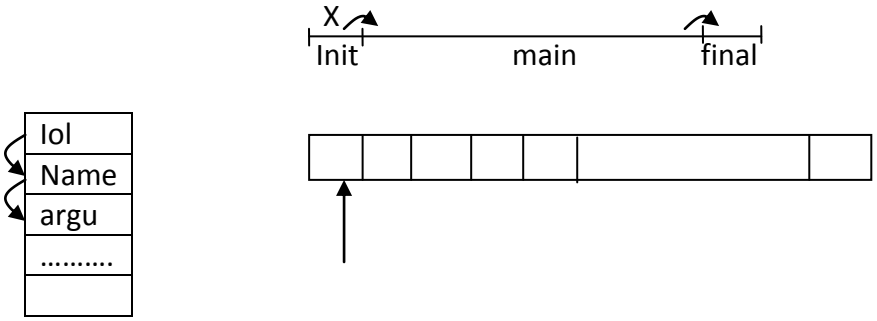
Memory hierarchy:



Принципи на локалността:

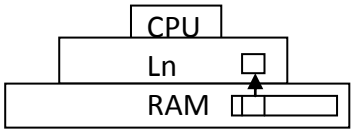
- Пространствена локалност (spatial locality)
- Времева локалност (temporal locality)

Програма:



Пространствена локалност имаме, когато се използват данни от съседни или много близки адреси.
Времева локалност има, когато се извършва чест достъп до 1 и същ адрас.

Кеш паметта се възползва от локализацията на данните.



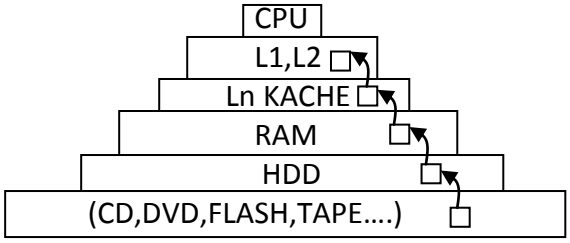
Кеш памет
Механизма на кеширането се състои в това, да се възползваме от двата вида локалност на данните и части от по-долните, по-бавни нива да се копират в по-горните, по-бързи нива.

Характеристики на една кеш памет:

1/ Ниво
2/ размер
3/ структура
4/ mapping function (функция за съпоставяне)
5/ латентност
6/ read/write policy
7/ remplacement policy (политика на закасняване)
8/ hit ratio (степен на попадение)

Характеристики на кеш паметта

1/ LEVEL(ниво) – на всяко едно ниво, от по-ниско към по-високо ниво.

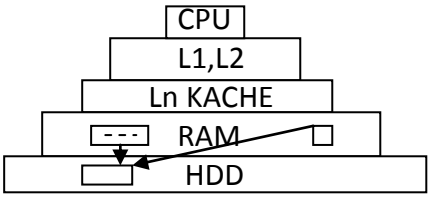


2/ SIZE (размер)
3/ STRUCTORS – типа на данните, различни структури зависещи от нивото
- ACCESS PATTERN – модул на достъп

4/ mapping function – „този адрес има ли го в кеша?“

5/ latency – връща отговора на въпроса 4/(времето за отговора на въпроса)
6/ Read/write policy
- PRE-CACHING
- PRE-PAGING – за виртуална памет

7/ REPLACEMENT POLICY – алгоритъм, по-който се определя кой елемент от кеша да бъде премахнат



7/ Два основни вида(Least Recently Used)->LRU – фактор – време като статистика за използване на ел-та нишка (посл. Време за използране) кога е използна.
(Least frequintly Used)->LFU based – най-рядко изп. бр. Обръщения към елемент.

!6/ *WRITE
- WRITE – THROUGH -> запис през, директен запис. Директно се копира на по-долното ниво пред консистентни данни. Недост: забавен запис
- WRITE – BACK – отложен запис (по-често използван). Предимство: бърза операция. Недост: не консистентност на данните.

8/ HIT RATIO(степен на попадение) – HIT/MISS RATIO
HIT RATIO HR = $\frac{\text{бр. HIT . 100}}{\text{Общ бр. Обрщ.}}$ [%]

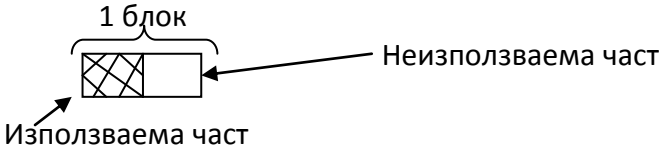
Memory management (MM)

P_1		P_2		P_1	……

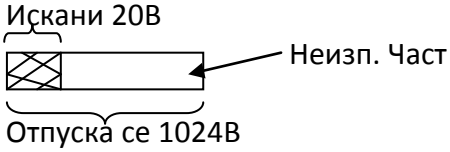
Характеристики на MM
- Вид на метаданните
  - Bitmap – работи с фиксиран размер на блока. Паметта се разделя на 2^n байта блока.
  - Non-bitmap – списъци, дървета – не е задължително паметта да е разделена на блокове.

- Размер на блока
- Allocation policy (политика на отпускане на памет)

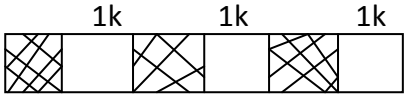
Паметта е разделена на блокове



***Intenral fragmentation – вътрешна фрагментация** – появява се когато паметта се управлява на блокове и от 1 блок се отпусне част от него, другата част е неизползваема.



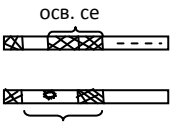
***external fragmentation – външна фрагментация** – Общ размер на свободната памет е по-голяма от заявката, но заявката не може да бъде изпълнена поради разкъсваност на свободната памет:



=> заявка за 2k

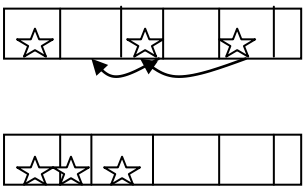
Начини за предотвратяване на външна фрегментация

- Reunification – обединяване
  - Lazy(мързелива)
  - eager(незабавна)



Да стане на 2-ти собствени блока

- Compaction
  - Преместване в края са заетите а в другия свободните



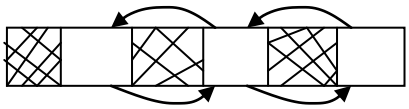
При *bitmap – няма нужда от reunifivation

Allocation policies(методи за отпускане на памет)

- First fit – започва от началото и отпуска при съответствие на размера на желаната памет.
- Next fit – с помоща на 1 указател, когото помни каде е било отпускато последния път и заделя следващия свободен блок, удовлетворяващ желаната памет.
- Best fit – по бавен метод(остават дупки в паметта) взима възможно най-малката дупка.
- Worst fit – първата най-голяма дупка (остават множество дупки, външна фрагментация)

Boundary tag

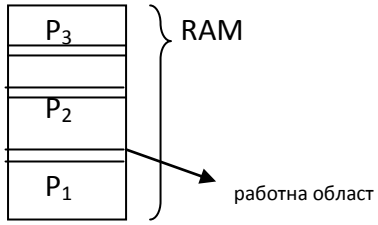
RAM



Virtual memory

50-60те години възниква

VM- HW възможности при SW реализация



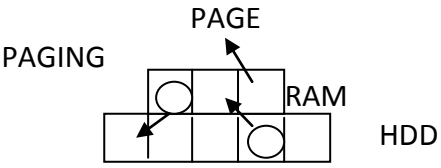
VM е кеш система на ниво RAM

1/ level -> cache RAM – външна памет (HDD)
2/ Size -> определя се от HW дадености на процесора
3/ Data structurs

- TRANSFER UNTT (PAGE)

*VM

- PAGING ()
- SEGMENTATION ()
- P+S



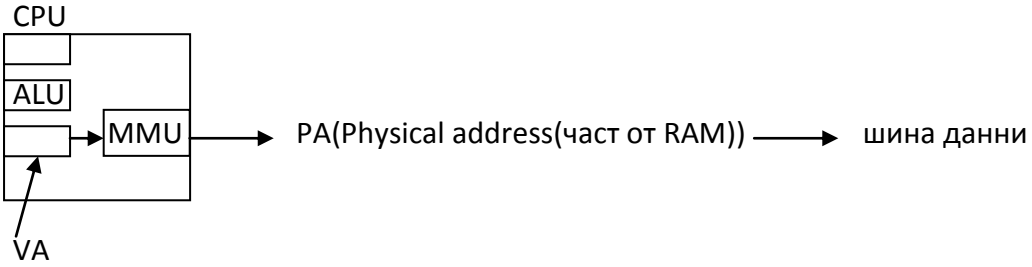
Паметта се разделя на страници

2^n страници пулно страницириране ако големината им е точно 2^n, а при непълно са по-малко.

При страницириране на виртуална памет, единицата за обмен е страница. Всяко прехвърляне на данни между нивата се извършва на страници. Размера на страницата се задава от HW.

Virtual address (VA)

MMU – memory management unit

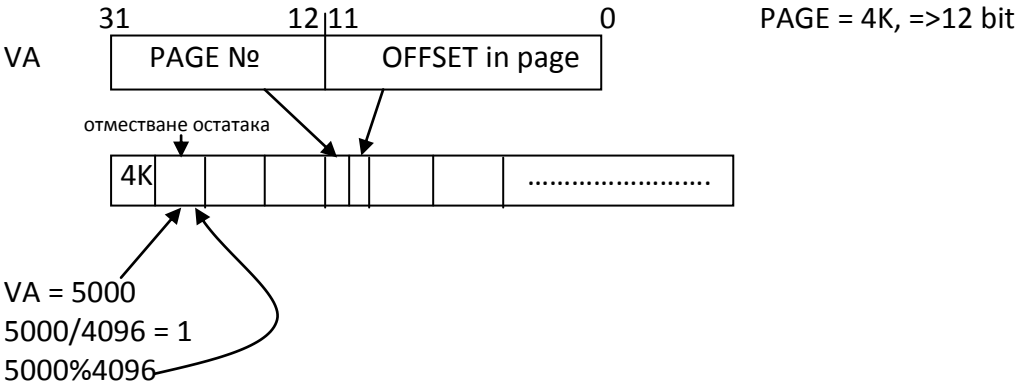


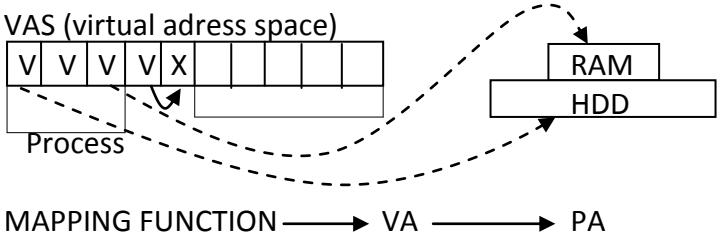
При страницирането VA се определя от максималния брой байтове, с които работят регистрите на процесора.
Абсолютно всичко - адрес на програмен код, адрес на данни, всичко е виртуален адрес

VA -> PA
MMU
прилага->MF

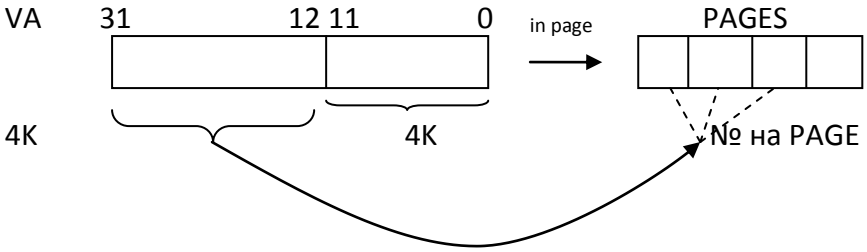
mapping function (рат. Е виртуален адрес)

32bit архитектура



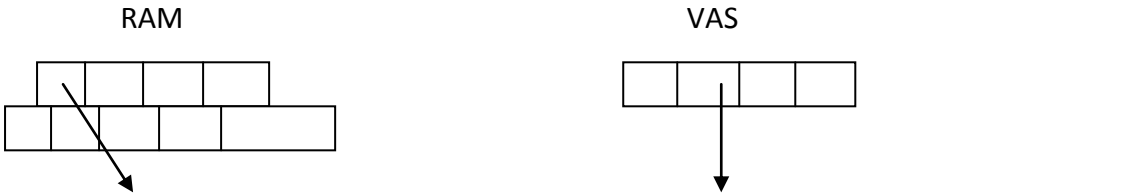


MAPPING FUNCTION → VA → PA



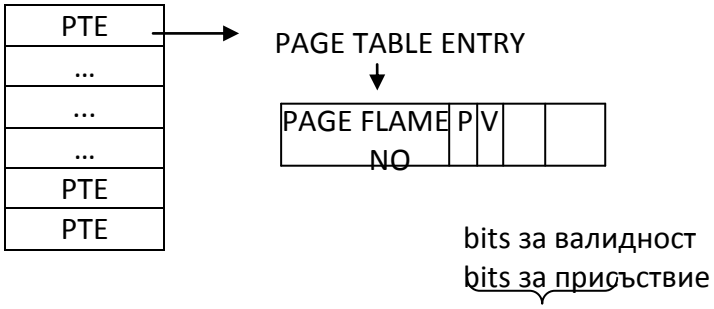
*За всяка страница трябжа да знаем:

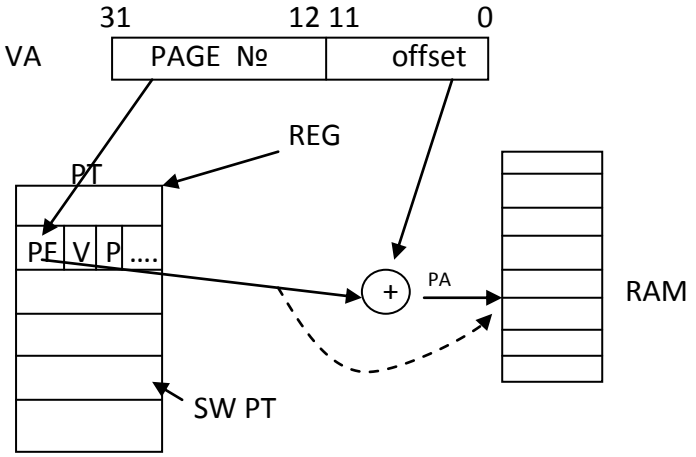
- Дали е валидна? (VALID?)
- Дали присъства в паметта? (PRESENT?)
- Дали е слопната? (SWAPPED)
  Дали е прехвърляно от SWAP ADRESS IN SWAP FILE RAM в диска?
  PEGEFILE.SYS ->SWAP файла на WINDOWS
- Дали една страница е физически адрес? PA



MAPPING FUNCTION
PAGE-> ? PAGE FRAME

PAGE TABLE





1/ Старшите 20bit са номер на страница в VAS те са индекс в Page Table
2/ След като направиме PTE, се проверява датата за валидност
3/ Проверява се бита за присъствие

VAS = PT
Връзката между VAS и PT
Всеки процес има PT

VAS = PT
Process

HW PT -> MMU
- по-бърз
- не е гъвкав

Multi-level PT

64bit
IPT (inverted page table)

TMB
TLB (translation lookside buffer)

TLB

PID	VA	PA	bits

TLB entry = 64,128,256
(ограничен по байтове)

Един VA може да е абсолютно различен физически адрес и същ за различните процеси.
Пример:
97% - 99% hit (попадения)

VA
Адрес 1000

PA1

VA
Адрес 1000

PA2

READ / WRITE
-pre-paging | (не се прилага при виртуалната памет)
SWAP – OUT
(запис на страница в SWAP файл(без отложен запис))

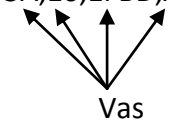
REPLACEMENT POLICY
(политика на записване във виртуалната памет)

RESIDENT SET (of pages)(пост. множество, отделено за процес)(краен бр. страници)

Как се определя Res Set?
- започва от някакъв брой страници, стигащ до макс.
- ако ни са нужни се намаля броя на старта.

За да се стигнем до пълни страници (не остане памет) се премахва ненужна (използвана) страница, за да се използва за друг процес.

REFERENCE STARING – послед. От виртуалния адрес
5A,10,1FBD,A8…..

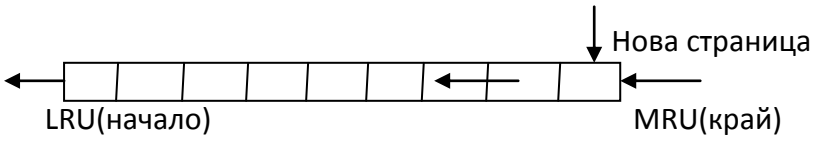


VIRTUAL TITE UNIT (единица за виртуално време) (времето, с което работи OS)

Алгоритми за запосване

1/ Optimal algorithm
Премахва страницата, към която ще се обърнем най-далечен момент
Статистика за :
LRU (фактор време) – по- добре работят (не се изисква ....)
LFU (фактор - честота)

LRU модификация (теоретична ..... на заместване)
LRU stack (не се използва в OS)



-Масив от страници

-сортирани

Алгоритъм
- При поредния виртуален адрес се проверява дали страницата е в RS(резидентното множество)
    - ако има масто отива в RS
    - ако е пълно:
        *PAGE FAULT(miss)(валидна страница, но неприсъстваща в RS)
    - новата страница се записва в MRU(най-вдясно)
    - винаги се изхвърля най-лявата(LRU)

FIFO-алгоритъм (*First In First Out*) и още

- **LRU-**алгоритъм (*Least Recently Used*) ;
- **LFU**-алгоритъм (*Least Frequently Used*) ;
- **MFU**-алгоритъм (*Most Frequently Used*) .

“Логиката” на алгоритъм FIFO е най-елементарна – на изхвърляне подлежи онази страница, която най-дълго време е останала в оперативната памет. В същност, времето не се измерва фактически. При зареждане страниците се регистрират в един последователен списък, който поддържа системата. Списъкът е от тип FIFO-опашка (от тук идва и наименованието на алгоритъма). В десния край на списъка се намира номерът на първата страница, а в левия – на последната. Когато възникне необходимост от изхвърляне на страници, кандидатите за това са онези страници, чиито номера са десния край на списъка или с други думи в дъното на FIFO-списъка. За съжаление тази логика, или още стратегия, не оценява фактическата активност на страниците и вероятността да бъдат изхвърляни активни страници е напълно реална. Като типичен пример в литературата често се посочва работата на процес, при който работи програма текстов редактор с файл с текстови данни.

За пълнота на картината ще спомена една модификация на алгоритъм FIFO, която се нарича **алгоритъм *Second-Chance***. В този алгоритъм загубата на често използваните страници се избягва с помощта на анализ на признака за обръщение. Ако признакът на разглежданата страница е установен, то тя, за разлика от алгоритъм FIFO, не се изхвърля – признакът й за обръщение се нулира, а нейният номер се премества в левия край на списъка от страници, т.е. в края на FIFO-опашката. Ако при анализа и преподреждането на опашката се намери страница, чийто признак не е установен (към нея не имало обръщение), то тя подлежи на изхвърляне. Това периодично прочистване на FIFO-опашката дава възможност съдържанието на оперативната памет да бъде поддържано актуално. Ако признаците на всички страници в списъка са установени, то алгоритъмът *Second-Chance* се превръща в алгоритъм FIFO – изхвърля се първата в опашката страница, тъй като изхвърляне трябва да има задължително. Алгоритъмът на втория шанс е също недостатъчно ефектовен, тъй като причинява непрекъснато разместване на страниците отметнати в списъка. Известно подобрение той получава ако опашката се зацикли. В така зацикленатa опашка указателят към кандидата за изхвърляне се интерпретира като часовникова стрелка, откъдето и произтича наименованието на тази модификация на FIFO алгоритъма –**часовников алгоритъм**. Логическата реализация на това зацикляне вече многократно сме разглеждали, например вижте поясненията в §4.3. В този вариант не се премества елемент в списъка, а само указателят, като той се модифицира (инкрементира) за да сочи следващата страница. Съществува вариант на този алгоритъм с две “часовникови” стрелки.Като основно достойнство на алгоритмите от тип FIFO ще отбележим тяхната лесна и икономична реализация.

“Логиката” на алгоритъм LRU предполага, че следва да се изхвърля онази страница, към която най-отдавна не е имало обръщение. Тази логика може да се изкаже и с други думи – *близкото минало е добър ориентир за прогнозиране на близкото бъдеще*. Тук интуицията подсказва, че “логиката” на този алгоритъм като че ли е по-правилна в сравнение с тази на FIFO-алгоритъма. Интуицията обаче не е доказателство и за съжаление тя се опровергава от ситуации, в които алгоритъм FIFO се справя по-добре.

LRU е добър, но доста скъп за реализация алгоритъм. Необходимо е да се поддържа свързан списък на всички страници в паметта, в началото на който се съдържат номерата на последно използваните страници. Този списък трябва да се обновява след всяко обръщение към паметта, за което се губи много време. Съществува вариант на LRU алгоритъма, в който се използва специален 64-битов указател, който автоматично се увеличава с единица след изпълнение на всяка машинна команда, а в таблицата на страниците съществува отделно поле, в което този указател се записва, при всяко обръщение към дадената страница. Кандидатът за

изхвърляне се сочи от указателят с най-малката стойност. Апаратната поддръжка на логиката на LRU алгоритъма оскъпява техническото му реализиране.

Алгоритъмът на **най-отдавна неизползваната страница** (LFU) изисква поддържането на брояч на обръщенията към всяка отделна страница, намираща се в паметта. При прекъсване се изхвърля онази, чийто брояч съдържа минимална стойност в сравнение с останалите броячи. Псевдологиката на този алгоритъм се основава на предположението, че тази страница вече е загубила своята активност и вероятно вече няма да бъде необходима.

Алгоритъмът на **най-често използваната страница** (MFU) също изисква поддържането на брояч на обръщенията към всяка отделна страница, намираща се в паметта. В противовест на горната псевдологика тук се приема за твърде вероятно страницата, към която са направени най-малко обръщения, е вероятно токущо въведена в паметта, ето защо тя има правото да остане, а за изхвърляне да бъде избрана онази, която има най-висока честота на използване до момента.

Съществуват и други алгоритми, които тук няма да разгледаме подробно, тъй като приемаме, че принципната същност на проблематиката в този пункт е достатъчно добре изяснена. Ще спомена само две наименования, колкото да събудим интереса на читателя – NFU-алгоритъм (*Not Frequently Used*) и *стекови* алгоритми. Конкретиката на всеки алгоритъм следва да се търси в архитектурата на съответните процесори. В бъдеще предстои да се запознаем и с други алгоритми за заместване.

FIFO (опашка)

- при попадение буфера не се променя
- при ...:
    - новата се записва най-вдясно
    - изхвърля се най-лявата

В ПЕРДИШНАТА СТАНИЦА

FIFO SECOND CHANCE

Битове за всяка страница

- R – referenced (OV1)
- M – modified (OV1)

Тези 2 бита се управляват хардуерно

- (D-Dirty) - при всяко обръщение хардуерно се взима R bit
    - при всияко записване в страница се взима M bit
    - нулиране на M и R става софтуерно с OS

Как работи?

Алгоритъма е подобен на FIFO

1/ При попадение е същото като FIFO(R бита се вдига)

2/ При непопадение се премахва най-левия елемент.

Ако R=1 елемента не се премахва, неговия R бит се нулираи се премества най-вдясно.

Ако е R=0 тогава се премества този най-ляв елемент

FIFO SECOUND CHANCE CLOCK

Циклично изпълнение

R базиран алгоритъм

NRU (Not Recently used)(ненаскоро използвана страница)

	R	M
Най-добрия вариант за изхвърляне	0	0
Добър кандидат за премахване	0	1
Добър кандидат за премахване	1	0
Не е добър варянт за премахване	1	1

* ако M = 1 трябва да се запише на диск за да се премахне елемент

F базиран алгоритъм

NFU (Not Frequently Used) (sledi èstotata)

Aging (използва се при съвременните OS)

R

1 →	1	обръща се внимание за скорошно обръщение
1 →	11	
1 →	111	
0 →	0111	R е толкова бита, колкото CPU-то