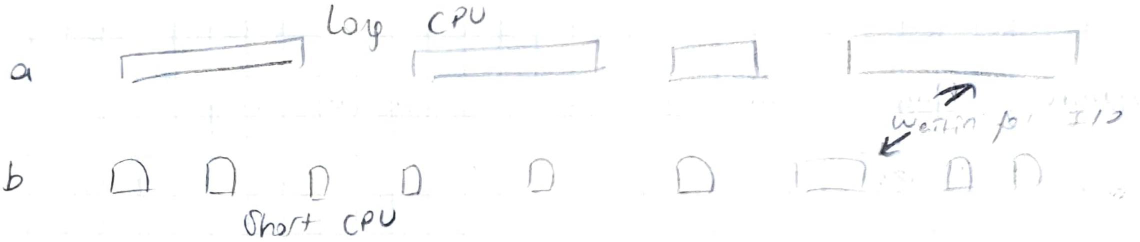
Scheduling

Suppose several processes runnable
which one is next

Old batch systems didn't have a scheduler, they just read next on input
Time-sharing systems tend to give priority to short timesharing requests

06. 04. 2021



We have to be fair.

When To Make Scheduling Decisions

- After fork
- On process exit
- When a process block
- When I/O complete
- Sometimes, after timer interrupts
  - ↳ Mesela CPU 'yu çok kullanan processi ready queue'ya alip baska processi ekler.

Preemptive & Nonpreemptive Schedulers

- Nonpreemptive: Lets a process run as long as it wants. Only switches when it blocks
- Preemptive: Switches after a time quantum, (Ex: At most 50 ms)

There should be a system clock. If not, just we have to use nonpre..

H	10	10	20
L	20	10	10
	↓ Choose		

Lower priority jobs get the higher quantum.

Categories of Scheduling Algorithms

- Batch: Responsiveness isn't important, preemption moderately important
- Interactive: Must satisfy a human; preemption important
- Real Time: Often / nonpreemptive

Goals - All Systems

Fairness → Give each process its share of the CPU

Policy and enforcement → Give preference to work that is administratively favored

Balance → Keep all parts of the system busy

Goal - Batch Systems (Backing App, Airline reservation etc.)

Throughput → Maximize jobs/hour

Turnaround time → Return jobs quickly, Often want to finish short jobs quickly

CPU Utilization

Goal - Interactive System

Response time - Respond Quickly to user requests

Meet user expectations - Psychological

→ click button etc.

- Users have a sense of "cheap" and "expensive" requests
- Users are happier if "cheap" request finish quickly

"Cheap" and "expensive" don't always correspond to reality

Goals - Real Time Systems

- Meet deadlines → avoid losing data (or worse!)
- Predictability → Users must know when their request will finish
- Requires careful engineering to match priorities to actual completion times and available resources.

Batch Schedulers

First come first served

Shortest first

Shortest remaining time first

When the running process blocks, the first process on the queue is run next

Never preempt based on timer

Seems simple, waiting in line

Not very fair

A	B	C
20	10	30

Throughput = 3 jobs / h

Turaround → for ABC $\frac{20+30+60}{3} = \frac{110}{3}$

First run shortest !

for BAC $\frac{10+30+60}{3} = \frac{100}{3}$

Jobs don't all arrive at the same time

	A	B	C	D	E
Job times	2	4	1	1	1
Arrive	0	0	3	3	3

A B C D E 4.6

B C D E A avg. 4.4

While B running, more jobs arrive allow better decision

Shortest Remaining Time Next

Preemptive variant of FCFS

Helps short jobs to get good service

May have a problem with indefinite overtaking

B'yi queue'ya alip digerlerini islerken B uzun zaman queue'da kalabilir.

Interactive Schedulers

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

1- Round-Robin (Dönme Dolap)

Current ↘ Next ↘

B → F → D → G → A

Current ↘

F → D → G → A – B

Oldest, simplest, fairest, most widely used

Each process is assigned a time interval, called quantum

Quantum Length

The shorter the quantum, the more responsive the system

For longer, this is bad for interactivity

2- Priority Scheduling

Not all processes are equally important

Assign priority

Simplest version Always run the highest priority

Not a good idea: What if it's CPU bound

Priority Inversion

H → P1 P1 running and needs a mutex so it blocks
M → P2 P3 holds mutex. After P3 finish, p2 runs.
L → P3

So P1 has to wait P2 which is lower priority tha

Priority Adjustments

H □ □ □ If we add some process to H,

M □ □ then M and L starve.

L □

Periodically So reduce the priority of the running processes

or increase priority of M or L

12.04.2021

Dynamic Priority Adjustment

Adjust priority according to its recent history

If process used 1/f of its last quantum, boost its priority proportional to f

Use priority class: H M L

Run Queues

Each queue is a linked list

To raise or lower a process's priority, move it to different list

In 1 second, there is 50 - 100 interrupt

Varying Quanta

Many processes need just a little bit of CPU time

Process switches can be expensive

Solution to both: Give lower priority queues longer quanta

Top priority queue: One quantum

Second " : Two quanta CPU allocation

Third Four quanta etc.

Alternate solution "Short" (initial) quantum at high priority and "long" quanta at low priority

Process Priorities

System Performance

- Kernel Processes (up to a point)
- Interactive Services Processes

Users can lower priority of own processes, to avoid competing

Unix Priorities

Lower numbers indicate higher priorities

"Nice" value is a user-specified modifier

A nice value of +20 specifies a very low priority

Only root can set negative niceness

Default is 0

4- Shortest Process Next

- Like batch process algo.

But we don't have good estimates

Let $T_0 = 25$ ms $T_1 = 10$ ms $\qquad 0 < \alpha < 1$

$T_2 = \alpha T_0 + (1-\alpha) T_1 \qquad \alpha = 0.5$ lets say

$= 17.5$

$$T_3 = \alpha^2 T_0 + \alpha(1-\alpha) T_1 + (1-\alpha) T_2$$

5- Guaranteed Scheduling

For n users or processes, give each $1/n$ of the CPU

6- Lottery Scheduling

P1	5 ticket
P2	[illegible]
P3	2

At scheduling time, pick random ticket

Tickets can be exchanged b/w processes

7 Fair - Share Scheduling

Some systems, process don't compete, user do

Solution: Make decisions based on user CPU consumption instead of process CPU consumption

U1 P1
P2
P3
P4] %80 we don't want that

U2 P5

Q1

1- How do you prevent priority inversion using priority lists

A: Change the priority class of low priority task

2- Why can't we implement the word processor application with 3 processes instead of 3 threads

All of them share same address space

3 - Which one is better, lower quanta or higher quanta

A: None of them for single purpose (Lower quata good for interactivity
Higher " " " CPU usage)

Real Time Systems

1- Hard Real Time   Absolute deadlines that must be met

2- Soft Real Time   Missing an occasional deadline is undesirable, but nevertheless tolerable

Periodic Events

	P_1	P_2	P_3
occuring in	200	100	150
needing	50	10	30

$$\frac{50}{200} + \frac{10}{100} + \frac{30}{150}$$

if result < 1.0 ✓

Aperiodic Events

Total load must be less than total capacity

Thread Scheculing

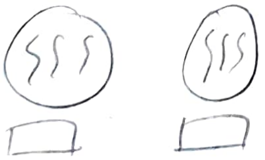
User level

How process a1 pass to a2
- Maybe load some data from disk and thread scheduling knows that
- Maybe thread yield called

✓ A1, A2, A3, A1, A2, A3

✗ A1, B1, A2, B2, A3, B3

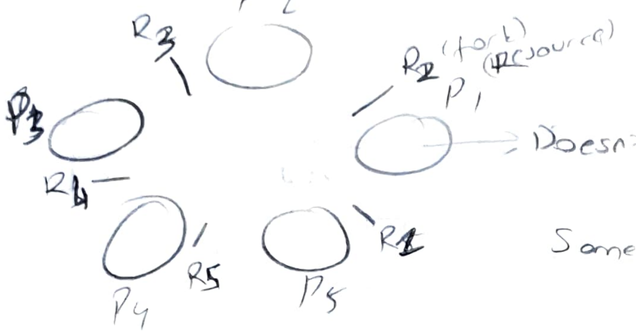
Kernel level



✓ A1, A2, A3, B1, B2, B3 → Better for CPU usage
✓ A1, B1, A2, B2, A3, B3

13.04.2021

Dining Philosophers Problem
Processes



Doesn't eat his food without R1 and R5

Some time just 2 process run

```
N=5
void   (int i)
  while (TRUE)
     think()
     take fork (i)
     take fork((i+1) %N)
     eat()
     put_fork(i)
     put_fork((i+1) %N)
```

critical

If switch in there, It causes problems deadlock

If I do that, there is no race cond but just one process run at some time

```
        Thinking=1  Hungry =2  eating=3
      mutex    =1
     semaphore   s[N]
      int state [N]
   while
      think
      take forks(i)
      eat
      put forks (i)
```

IInd

```
take forks(int i)
   down (&mutex)
   state(i) = HUNGRY
   test (i)
   up(mutex)
   down (&s[i]))
```

critical

```
void test (i)
     if (state[i] == HUNGRY && state [LEFT] != EATING &&
                                 state [RIGHT] != EATING)) {

           state[i] = EATING)
           up (&s [i])
     }
)

void put forks(i)
     down (mutex)
     state[i] = THINKING
     test (LEFT) ; test (RIGHT) ;
     up (mutex)
```

critical

Reader - Writer

Reader can be many at same time.
But writer can be just one or zero

```
semaphore mutex = 1
   "      db    = 1
          rc    = 0

reader
  while (TRUE)
      down (&mutex)
critical [ rc = rc + 1
         [ if (rc==1)  down (&db)
      up ( &mutex )
      read_db()
critical [ down (&mutex)
         [ rc = rc - 1
         [ if (rc==0)  up(&db)
      up (&mutex)
      use_data_read()

writer

  while TRUE
      think_up_data()
      down (&db)
      write_db()
      up( &db )
```

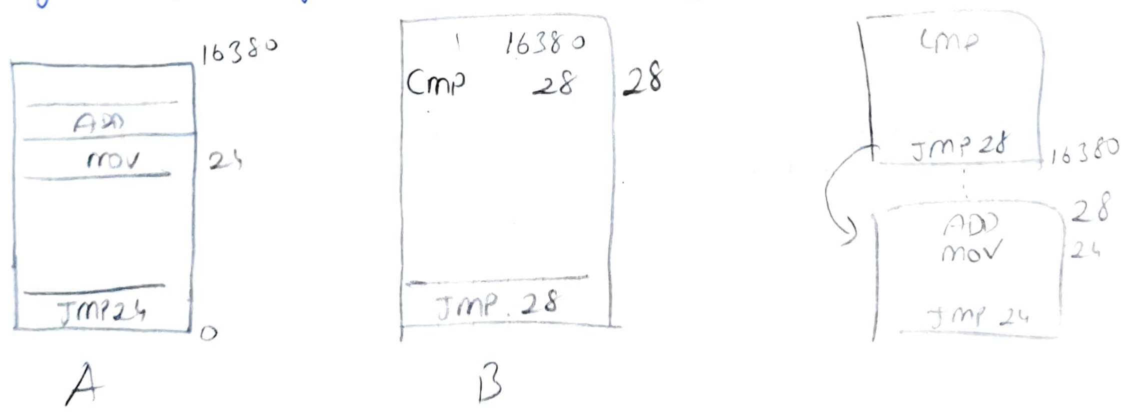
CHAPTER 3: MEMORY MANAGEMENT

1960's (32 K words)

Main memory is a resource

OS have to manage it

Running Multiple Programs Without a Memory Abstraction



Static Relocation

At program load time, execution address is known

Add the actual offset to the presumed location specified by compiler and linker

Modify address

A memory Abstraction: Address Spaces

Add. Space is the set of addresses that a process can use to add. mem

Each process has its own add. space

Process concept creates a kind of abstract CPU to run, the add. space creates a kind of " memory for programs to live in

Problems has to be solved: Protection and Relocation

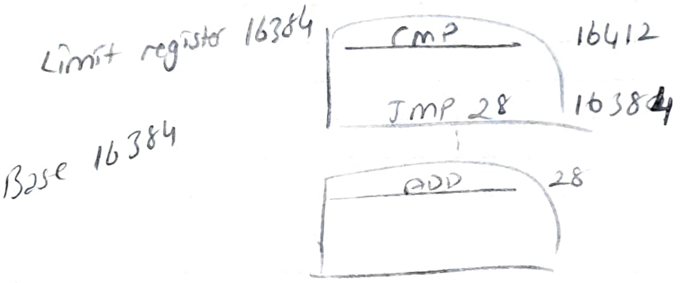
Protection ↓ Each program has its own add

Relocation ↓ They should not know where they are actually in memory

Base and Limit Registers

Simple Version of Dynamic Relocation

Whenever I try to reach memory, I always add my memory address with base register. The result < limit register



16384+28=16412

Disadv: Addition expensive ! (Serial carry)

Large programs that do not fit the memory!

Solution: Swapping and Virtual Memory

Swapping

Physical memory is not large enough to hold all processes

Swapping : Write the entire prog. out to disk when it's blocked

Read it all back again

Context switching is inefficient ✓

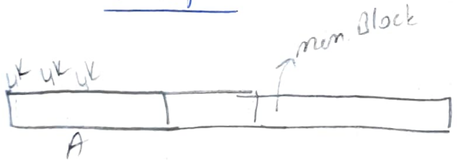
It's good Idea to allocate a little extra memory than needed

When memory is assigned dynamically, OS must manage it

There are 2 ways to keep track of memory usage:

1 - Bitmaps
2 - Free Lists

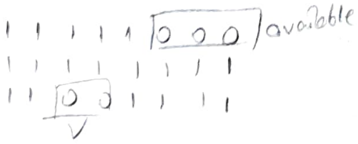
1- Bitmap



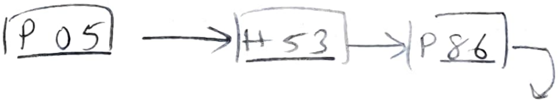
2^{20} = 1 MB memory size

2^{12} = 4 KB block size

2^{8} = 256 block



2nd approach
Linked List

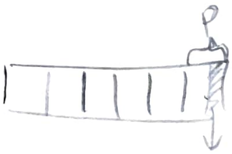


For a few process, linked list ✓

Memory is divided into units

Small block size, bitmap will be large

Big block size, small table but wasted



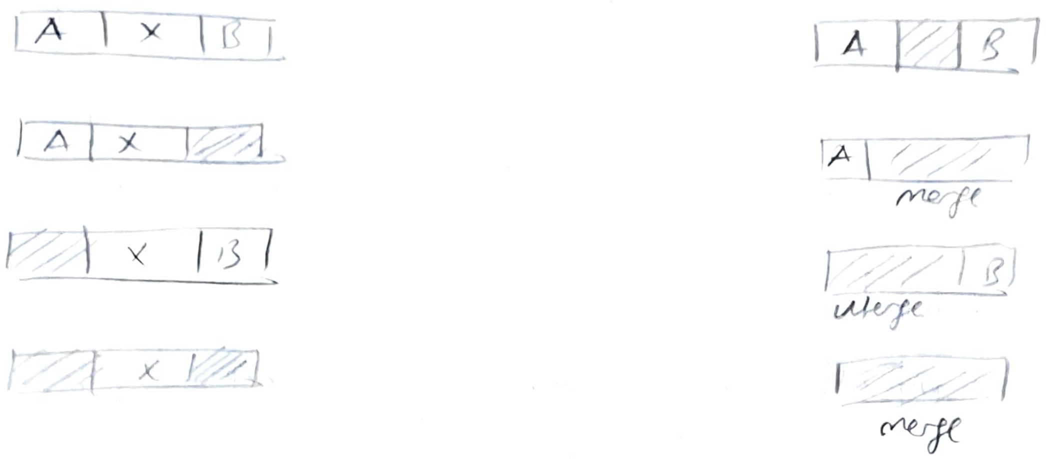
If didn't use
called
Internal
fragmentation

19.04.2021

2- Linked Lists

Before X terminates

After X terminates



Memory Management Algorithms

- First Fit

Traverse list until large enough block is found

Allocate as much of it as needed

Return rest to the free list

When freeing memory, must merge adjacent blocks.

If Size might be alternative. It would be very easy to find the first fit first node is answer. Remaining part inserted to the back to the list

I need to go back and scan my list and insert the remaining parts of that available block (Sinavda kağıda give an example yazdı)

- Next Fit

Next search starts from the result of prev search

- Best fit

First fit seems wasteful. Maybe exact match further down the list.

Order linked list by size

Makes merging less efficient

Not so efficient. More wasted memory

External fragmentation
P
P

-Worst fit

Largest block is picked and sub-divide it

First and best better and close

For first fit, 1/3 may be unusable

Compaction is solution for external fragmentation

Move the processes and merge the rest of blocks.

Better Implementation

Keep the hole list and process list seperate.

Hole list can directly use the free memory

Overlays

There is a need to run programs that are too large to fit in memory.

In 1960s, programmers split programs into little pieces, called overlays

Kept on disk, swapped in and out of memory by the program itself

VIRTUAL MEMORY

Each prog. has its own add. space

Convert a virtual address to a physical add.

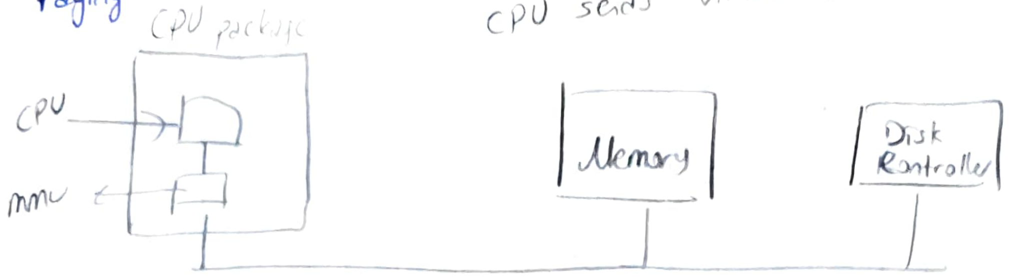
Mapping is done by pages. (perhaps) 4K bytes

Programs need not to be on contiguous areas of physical memory

bitişik

Paging

CPU sends virtual add. to MMU



Page replacement: We need to load the page from disk

Page fault: If the data is not available on main memory.

Virtual add. consists of fixed size units called pages.
Corresponding units in physical memory are called page frames.

Virtual page Table keeps in mmu

We need a page table for each process.

VMemory eliminates, allocation inneffiency (no external frag. anymore)
Because no physical memory

" " memory protection
compaction
relocation

Implementing VMemory

Divide a virtual add. A into <V,O>   V: Virtual page number
O: Offset

MMU maps V into P, physical page number and produces <P,O>

Faster than base register, NO ADDITION

Ex.

2^{30} memory space

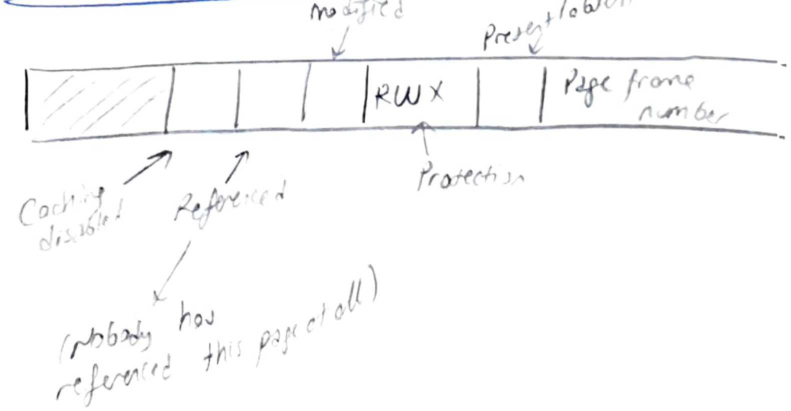
2^{12} page site

2^{18} page frame

Page table size 2^{18}

$2^{30}/2^{12}$

Structure of Page Table



Translation Lookaside Buffer (TLB)

MMU is not large
Main Memory is slow

So Mappings from virtual address spaces are cached in the TLB

20.04.2021

~~Page Tables in RAM~~

Software TLB Management

Many RISC machines do nearly all of this page management in software.

↳ keep your CPU simple. Instead give me lots of registers

TLB

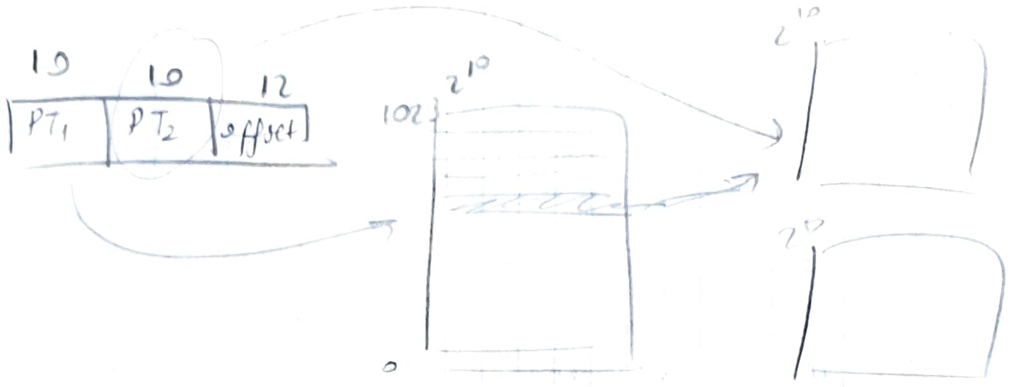
Valid	Virtual Page	Modified	Protection	Page Frame
1	140	1	RW	31

If I don't have in TLB, I have to load from disk

- Soft Miss: Page referenced is not in TLB, but is in memory.
- Hard Miss: It occurs when the page itself is not in memory
- Minor Page Fault: The page may actually be in memory, but not in process' page table, the page may have been brought in from disk by another process
- Major Page Fault: The page needs to be brought in from disk
- Segmentation Fault: Program simply accessed an invalid address

Multilevel Page Tables

Deal with very large virtual address spaces



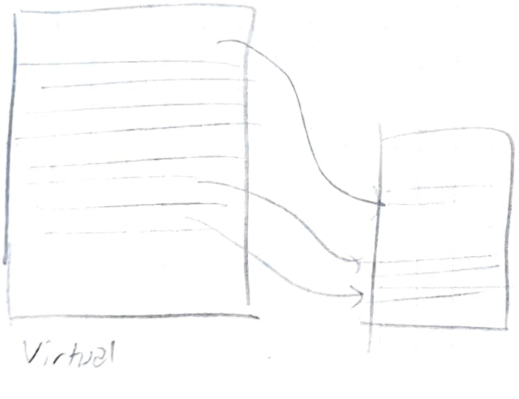
Solved: Page tables are being too large

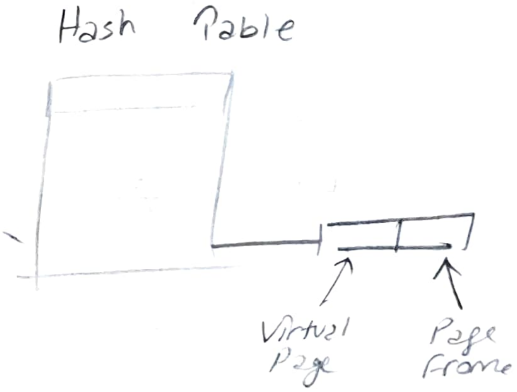
Adv: Don't have to keep all pages in memory.

Inverted Page Tables

64 bit PC 2^{64} 2^{12} $= 2^{52}$ virtual pages

8Gb 2^{33} 2^{212} $= 2^{21}$ actual size





Ask the page table, do you have any virtual address

Adv: We keep 2^{21} elements instead of 2^{52}

Page Replacement Algorithms

- Optimal
- Not Recently Used
- FIFO
- Second-Chance
- Clock
- Least Recently Used (LRU)
- Working Set
- WSClock

Constraints

Should be efficient
Must approximate correct answer
Must be implementable on real hardware
Must work well on multitasking systems

Tools

OS has a few tools available to it
The referenced bit - This page has been used recently
The modified bit - Discarding this page will be more expensive
Clock interrupts
Page fault "
Advice from the app → I'm done with this data, so OS will know

	R	M
1	0	0
2	0	1
3	1	0
4	1	1

How could a page be not referenced or modifi?
Every clock interrupt, they are all made 0

Replace order : 1, 3, 2, 4

- Optimal Algorithm

It says that the page will not be referenced longest should be replace

Theorethical algo.

- Not Recently Used Algorithm

At page fault, system inspects pages

Categories of R and M

Class 0	: Not referenced, not modified
1	" " modified
2	Referenced, not modified
3	" modified

Üstteki gibi
1, 2, 3, 4
ama 2-3 swap

On clock interrupts, clear R bits
On page fault, discard a random page from lowest class

Resetting R and M

Why do we reset R on clock interrupts?

We want to know if a page has been used recently

Why not reset M?

Until page has been written out to disk, it cannot be reset

Properties of NRU

* Primarily useful for teaching

What's Interesting About NRU

It has the essential properties of any page replacement algo.

It looks for a (relatively) idle page

It handles modified pages, but is biased against using them

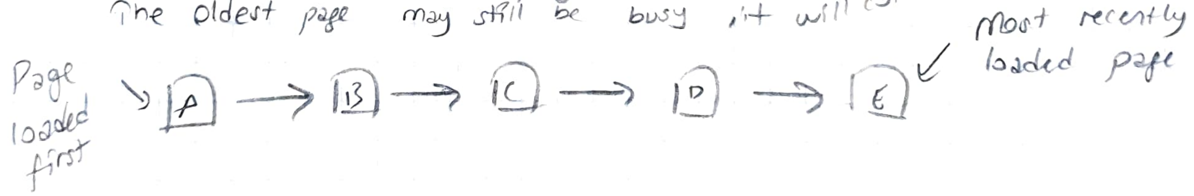
It is reasonably efficient

- FIFO Algorithm

Forget about R and M

Discard the oldest page

The oldest page may still be busy, it will come right back in


Page loaded first
A
B
C
D
E
Most recently loaded page