SS201μ Introduction to the Internal Design of Operating Systems

Session 2

Algorithms to Manage Operating System Abstractions

Sébastien Combéfis

Winter 2020

This work is licensed under a Creative Commons Attribution – NonCommercial – NoDerivatives 4.0 International License.

Objectives

- Discover several process scheduling algorithms

  How the OS allocates the CPU resource to the processes

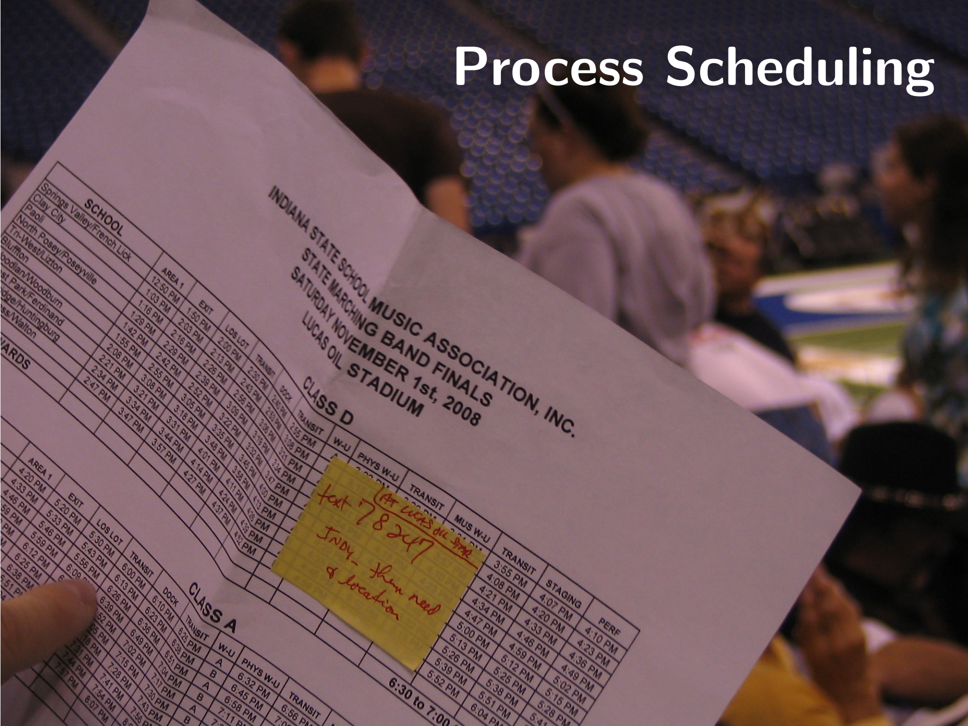
- Discover several page replacement algorithms

  How the OS frees space in the memory to execute processes

- Discover several disk scheduling algorithms

  How the disk controller optimises I/O wait time

Process Scheduling



Multiprogramming (1)

- Multiprogramming makes the most of the CPU

  The CPU is shared between several processes

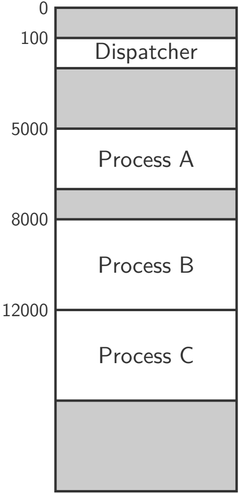
- A process is executed until it has to wait

  For exemple, for an input/output operation

- Several processes are maintained in memory

  The scheduler alternates these processes on the CPU

Multiprogramming (2)



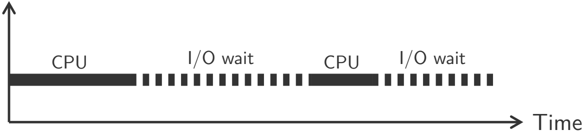
Execution example:

01	5000	
02	5001	
03	5002	
04	5003	
05	5004	I/O Request
06	100	
07	101	
08	102	
09	103	
10	8001	
11	8002	
12	8003	Timeout
13	100	
14	101	
15	102	
16	103	
17	5005	
18	5006	
19	5007	

CPU and I/O Burst

- The life of a process is a cycle between CPU and I/O waiting

  Alternating between CPU and I/O bursts



- A lot of short CPU bursts, and a few long

  Based on numerous measurements

CPU Scheduler

- Choosing a process in the ready queue for the CPU

  We take a process that is ready to start directly

- Choice made by the short-term scheduler

  Among all the processes which are already in memory

- The scheduler takes a decision whenever a process...

  1. ...goes from *Running* to *Waiting* (I/O request, `wait` call...)
  2. ...goes from *Running* to *Ready* (interrupt...)
  3. ...goes from *Waiting* to *Ready* (I/O response...)
  4. ...finishes

Preemption

- Non-preemptive scheduler (only 1 and 4)
  - Also called cooperative
  - No choice in terms of scheduling, a new one is needed
  - A process keeps the CPU until it releases it
- Preemptive scheduler
  - Requires a hardware timer
  - A process can be removed from the CPU at any time

Scheduling Criteria (1)

- Usage of the CPU
  - Pourcentage of time the CPU is occupied
  - From 40% for light load to 90% for heavy load
- Process throughput
  - Number of processes terminated by unit of time
  - From $1/h$ for long processes to $10/s$ for short transactions

Scheduling Criteria (2)

- Rotation time
  - Total elapsed time for the execution of a process
  - Memory loading, ready queue, CPU execution, I/O
- Wait time

  Sum of wait times in ready queue
- Response time
  - Time between process submission and first response
  - The output begins to arrive, while the sequel is computed

First-Come First-Served (1)

- Processes executed in order of arrival (FIFO)

$\Rightarrow$ Wait time $P_1 : 0$, $P_2 : 24$, $P_3 : 27$, average wait time: **17**

$\Rightarrow$ Wait time $P_1 : 6$, $P_2 : 0$, $P_3 : 3$, average wait time: **3**

First-Come First-Served (2)

- Non-preemptive scheduling
  - A long process can keep shorter ones from finishing
  - Not suitable for a timeshare system
- Average wait time depends on the scheduling choice

  Bursts of different lengths are penalising
- Induces a convoy effect
  - A CPU-attached process, small I/O-attached processes
  - Small processes get stuck behind big ones

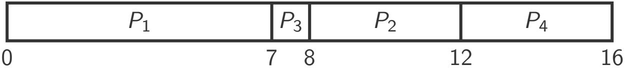
Shortest-Job-First

- Duration of the next shortest CPU burst

  Proven optimal when using average wait time

- FCFS is used in case of a tie

Processus	Arrival time	Burst duration
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



$\Rightarrow$ Wait times $P_1 : 0$, $P_2 : 8-2$, $P_3 : 7-4$, $P_4 : 12-5$
Average wait time: **4**

Shortest-Remaining-Time-First

- Preemptive scheduling as a variant of SJF

  When a new process arrive, possible change

Process	**Arrival time**	**Burst duration**
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

P_1	P_2	P_3	P_2	P_4	P_1

0 2 4 5 7 11 16

$\Rightarrow$ Wait times $P_1 : 11 - 2$, $P_2 : 5 - 4$, $P_3 : 0$, $P_4 : 7 - 5$
Average wait time: **3**

Priority (1)

- Each process is assigned a priority number
  - Highest priority process chosen first
  - FCFS is used during ties
- SJF is a particular case of priority
  - Priority is equal to 1 / CPU burst duration
  - Lower priority for longest bursts

Priority (2)

- Can be non-preemptive or preemptive

Process	**Arrival time**	**Burst duration**	**Priority**
P_1	2	10	3
P_2	0	1	1
P_3	7	2	4
P_4	0	1	5
P_5	0	5	2

P_4	P_5	P_1	P_3	P_2

0 1 6 16 18 19

$\Rightarrow$ Wait times $P_1 : 6 - 2$, $P_2 : 18$, $P_3 : 16 - 7$, $P_4 : 0$, $P_5 : 1$
Average wait time: **6.4**

Priority (3)

- Can be non-preemptive or preemptive

Process	Arrival time	Burst duration	Priority
P_1	2	10	3
P_2	0	1	1
P_3	7	2	4
P_4	0	1	5
P_5	0	5	2

P_4	P_5	P_1	P_3	P_1	P_5	P_2

0 1 2 7 9 14 18 19

$\Rightarrow$ Wait times $P_1 : 9 - 7$, $P_2 : 18$, $P_3 : 0$, $P_4 : 0$, $P_5 : 1 + (14 - 2)$
Average wait time: **6.6**

Choosing Priorities

- Priorities can be defined internally or externally
  - **Internally**, priorities based on measurable quantities

    Time limit, memory requirement, number of open files...
  - **Externally**, priority based on criteria outside the OS

    Importance of the process, payments...
- Low priority processes are never executed

  Aging: increasing priority over time

Round-Robin (1)

- Short unit of CPU time (*time quantum* or *time slice*)

  FCFS with preemption, units from 10 to 100 ms

- The ready queue is a circular queue

  If the queue contains n processes, and the time quantum is q each process receives $1/n$ time by chunks of q time

- Time quantum high $=$ FCFS

  But q must be larger than the switching time ($\sim 10\mu s$)

Round-Robin (2)

- Example with a time quantum of 4

Process	Burst duration
P_1	24
P_2	3
P_3	3

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1
0–4	4–7	7–10	10–14	14–18	18–22	22–26	26–30

$\Rightarrow$ Wait times $P_1 : 10 - 4$, $P_2 : 4$, $P_3 : 7$
Average wait time: **5.6**

Multi-Level Queue (1)

- Multi-level queue if the processes are classified into categories

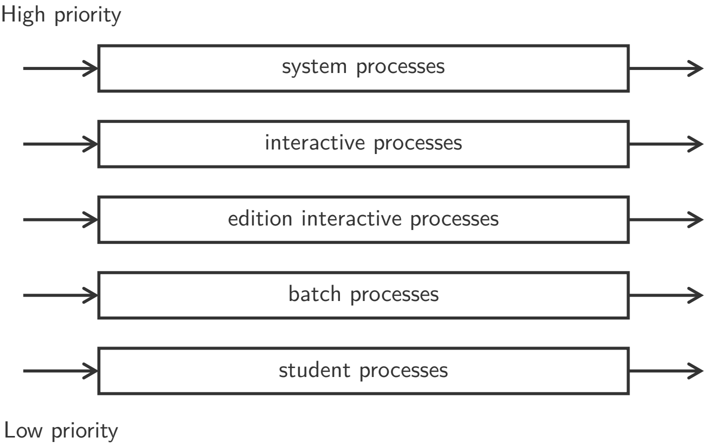
  Foreground processes (interactive) / background (batch)

- Each queue has its own scheduling algorithm

  Typically RR for foreground and FCFS for background

- Scheduling algorithm between queues
  - Absolute priority fixed between queues
  - Time slice (*e.g. 80%/20% for foreground/background*)

Multi-Level Queue (2)



Retroaction Multi-Level Queue (1)

- Queue change according to CPU burst duration
  - To **low priority** if too much CPU usage

    Priority to interactive and I/O-attached processes
  - To **high prority** if too long wait time

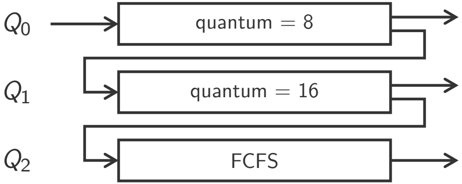
    Kind of aging to prevent degeneration
- Several possible parameters
  - The total number of queues
  - The scheduling algorithm for each queue
  - Rule that {promote/retrograde/select initial queue} of process

Retroaction Multi-Level Queue (2)

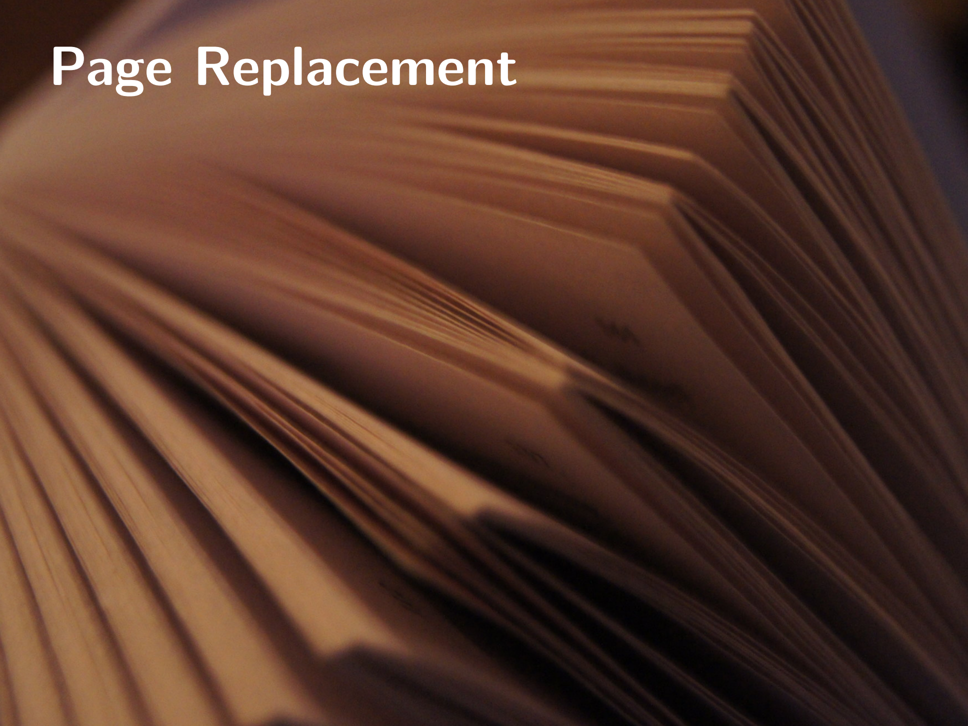
- Example with three queues

  RR with $q = 8$ *and* $q = 16$ *for* Q_0 *and* Q_1 *and FCFS for* Q_2

- Choice of three rules for the processes
  - New process enters in Q_0
  - Process transit $Q_0 \rightarrow Q_1 \rightarrow Q_2$
  - Process which enters in Q_i preempt processes from Q_{i-1}



Page Replacement

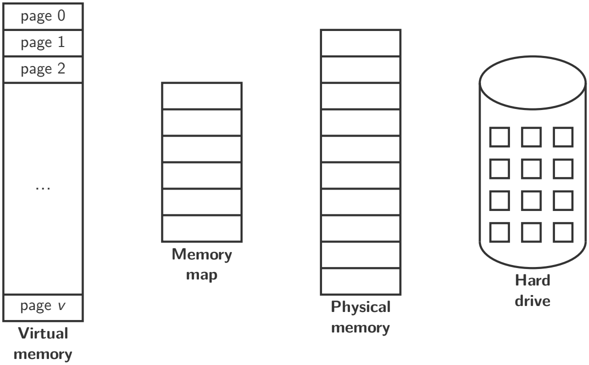


Virtual Memory

- Separation of logical and physical memories

  Logical memory as seen by the user

- A process uses its own virtual address space during its lifetime

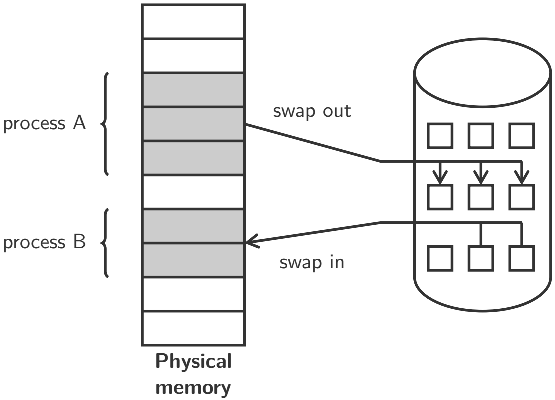


On-Demand Paging (1)

- Pages loaded on demand when they are necessary
  - Less input/output operations
  - Less physical memory required
  - Faster response, more users
  - A page never used will never be loaded in memory
- Similar to a paging system with swapping

  Swapper moves an entire process, pager only moves pages

On-Demand Paging (2)



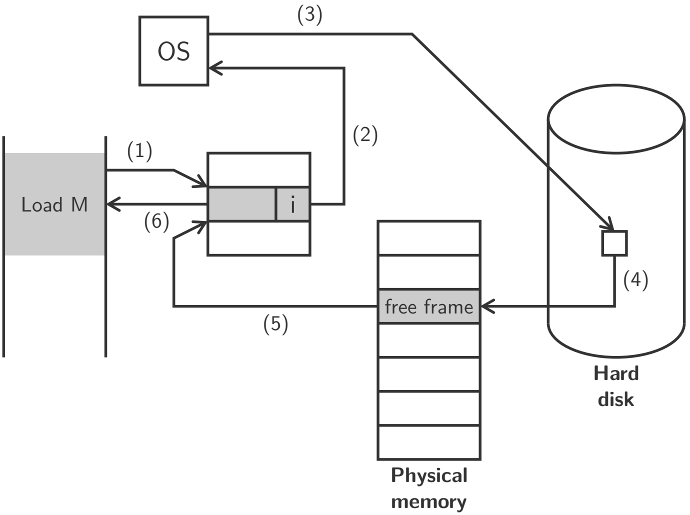
Page Fault (1)

- (In)valid bit associated to each page in the page table
  - Valid means that the page is legal and in memory
  - Invalid means that it is either invalid or not in memory
- Accessing to an invalid page results in a page fault

  Hand returned to the OS by the hardware page handler
- Execution of a handler code to make the page accessible

  Six big steps to bring the page back into memory

Page Fault (2)



Page Fault (3)

1. Check of the address validity in an internal table
2. If invalid, end of the process; otherwise loading the page
3. Choice of a free frame
4. Disk operation request to read the page in the frame
5. Modification of the validity in an internal table and page table
6. Re-execution of instruction which caused the page fault

Page Replacement

- Fault page, but no free frame

  Need to find room in memory to be able to execute the process

- Selecting an unused victim frame and free it

  1. Writing the content of the frame in swap
  2. Updating the page table (invalid)
  3. Moving the requested page in the freed frame

- Dirty bit on each page and read only page

  Only a modified page will be written to disk during the swap

Algorithm Evaluation (1)

- **Objective**: smallest possible number of page fault
- Two problems to solve
  - How to allocate the frame to a given process?
  - How to select the pages to replace in memory?
- Evaluation of a given algorithm
  - Executing the algorithm in a reference sequence
  - Counting the total number of page fault

Algorithm Evaluation (2)

- Saved adresses sequence

  0100 0432 0101 0612 0102 0103 0104 0101 0611 0102 0103 0104 0101 0610 0102 0103 0104 0101 0609 0102 0105

- Reference sequence for pages of 100 bytes

  1 4 1 6 1 6 1 6 1 6 1

- #Frame $\uparrow$ implies #Page fault $\downarrow$

  For the example, with three frames, three page faults

First-In First-Out

- Victim is oldest page with First-In First-Out (FIFO)

  The page that has been placed in a frame the least recently

- A page actively used can be replaced...

  A page fault will immediately follow

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
	0	0	0		3	3	3	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	3	3			3	2			2	2	1

$\Rightarrow$ **15 page faults**

Belady Anomaly (1)

- Increase in page faults with more frames

  While this number should intuitively decrease

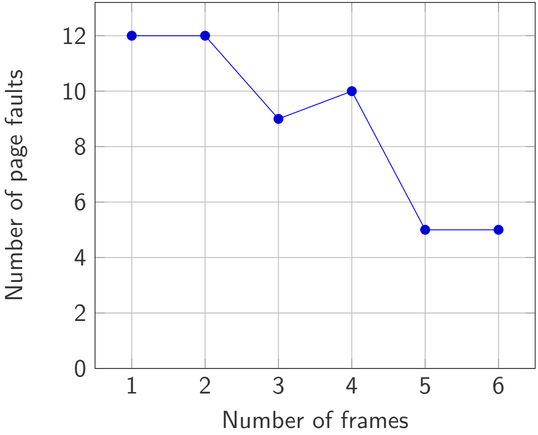
1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5			5	5	
	2	2	2	1	1	1			3	3	
		3	3	3	2	2			2	4	

$\Rightarrow$ **9 page faults**

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1			5	5	5	5	4	4
	2	2	2			2	1	1	1	1	5
		3	3			3	3	2	2	2	2
			4			4	4	4	3	3	3

$\Rightarrow$ **10 page faults**

Belady Anomaly (2)



Optimal Algorithm

- Victim is the one that will not be used before longest time

  How to predict this time?

- Proven guarantee of the lowest number of page faults

  As with the SJF process scheduling algorithm

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		2			2			2				7		
	0	0	0		0		4			0			0				0		
		1	1		3		3			3			1				1		

$\Rightarrow$ **9 page faults**

Least Recently Used (1)

- Victim is the page that is not used for the longest time
  - Approximation of the optimal algorithm
  - Time of last use associated with each page
- Does not suffer from the Belady anomaly

  As is the case with the optimal algorithm

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		

$\Rightarrow$ **12 page faults**

Least Recently Used (2)

- **Counter**
  - Counter added to the CPU
  - Usage timestamp associated with each page
  - Search in all pages of the smallest timestamp
- **Stack**
  - Maintaining a page number stack
  - Referenced page removed and put back on top of the stack
  - The least recently used is always at the bottom of the stack

LRU Approximation

- LRU requires hardware support

  Operations to perform at each memory access (clock or stack)

- A reference bit set to 1 when a page is accessed

  All the reference bits are initialised to zero

- Information associated with the page table entries

  Information on the access, but not on the access order

Additional Reference Bit

- History of the reference bit values

  Regular recording of the reference bits

- For example, an 8-bit byte for each page
  - Updated by an interrupt every 100ms, for example
  - Addition of the reference bit to the right and offset
- Choosing the largest value among all pages
  - 11000100 has just been used twice (196)
  - 01110111 has not been used during the last round (119)

Second Chance

- Variant of the FIFO algorithm

  Size of the history is reduced to a single bit

- Using the reference bit
  - If 0, the page is replaced
  - If 1, change to 0, updating time of arrival of the page, and moving on to find the next victim (second chance)

- A page that received a second change wins a whole round

  It will not be replaced until everyone has passed

Improved Second Chance

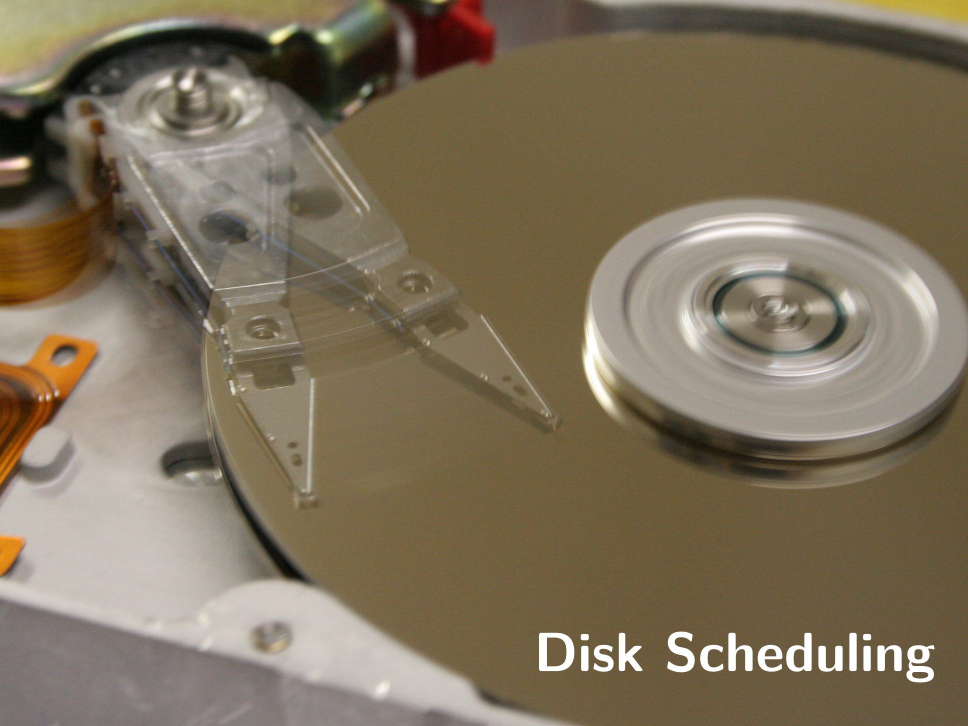
- Using the pair (*reference bit*, *dirty bit*)
- Four values are possible for the pair of bits
  1. $(0, 0)$ neither used, nor modified recently

     Best replacement candidate
  2. $(0, 1)$ not recently used, but modified

     Must be written to disk when replaced
  3. $(1, 0)$ recently used, but unchanged

     Will probably be used soon
  4. $(1, 1)$ recently used and modified

     Surely used soon and will have to be written to disk


Disk Scheduling

Disk Scheduling (1)

- The operating system must use the disk efficiently

  Fast access time and high bandwidth

- Two major components of disk access time
  - **Seek time** to move on the right cylinder
  - **Latency time** to reach the desired sector

- Disk bandwidth measures transfer capacities
  - Total time between first request and transfer completion
  - Total number of bits transferred over total time

Disk Scheduling (2)

- A process makes a system call to access the disk
  - Input or output operation
  - Address on the disk for the transfer
  - Memory address for the transfer
  - Numbers of the sectors to transfer
- Request either satisfied immediately or placed in a queue

  Scheduling algorithm to choose the request to serve

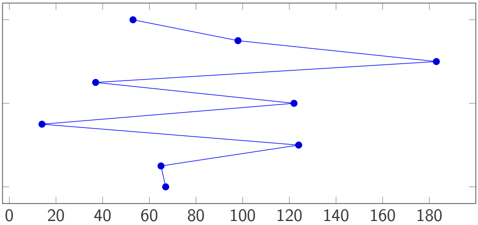
First-Come First-Served

- First-Come First-Served (FCFS) is a fair algorithm

  But does not result in the fastest service

- Blocks request on the following cylinders (head on 53)

  $98, 183, 37, 122, 14, 124, 65, 67$ *(640 cylinders displacement)*



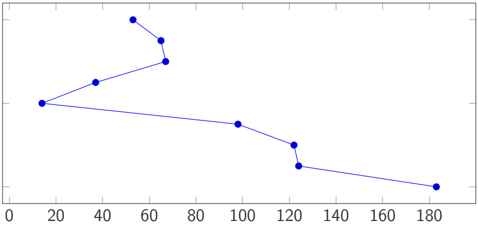
Shortest-Seek-Time-First (1)

- Shortest-Seek-Time-First (SSTF)

  Chooses the block closest to the head

- Blocks request on the following cylinders (head on 53)

  98, 183, 37, 122, 14, 124, 65, 67 *(236 cylinders displacement)*



Shortest-Seek-Time-First (2)

- Perform way better than the FCFS algorithm

  Overall decrease of the head movement

- Similar to the SJF algorithm
  - Same risk of starvation as with SSTF
  - Some requests are not served quickly

- No guarantee of optimality with the SSTF algorithm

  $53 \rightarrow 37 \rightarrow 14$ *then 65, 67... (208 cylinders displacement)*

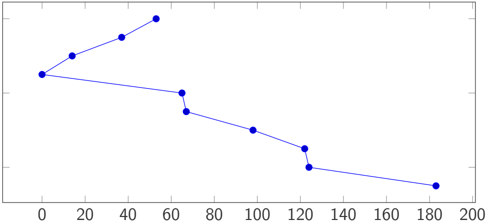
SCAN (1)

- The head goes back and forth between beginning and end

  Blocks are served when the head passes over (escalator)

- Blocks request on the following cylinders (head on 53 $\searrow$)

  $98, 183, 37, 122, 14, 124, 65, 67$ *(236 cylinders displacement)*



SCAN (2)

- A new request can be served quickly or not
  - In front of the head, will be served almost immediately
  - Behind the head, will have to wait for his return
- Given a uniform distribution of the requests on the cylinders
  - When returning, low to high request density
  - Requests near the head have been served recently

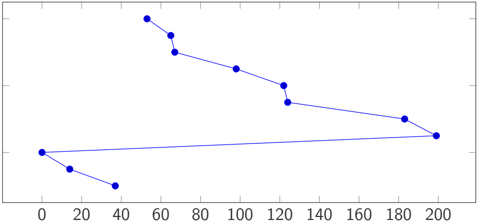
C-SCAN

- Circular SCAN does not serve any request when returning

  Because the waiting queue is often on the other side

- Blocks request on the following cylinders (head on 53 $\nearrow$)

  $98, 183, 37, 122, 14, 124, 65, 67$ *(382 cylinders displacement)*



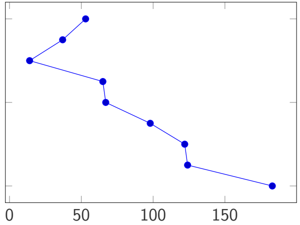
LOOK and C-LOOK

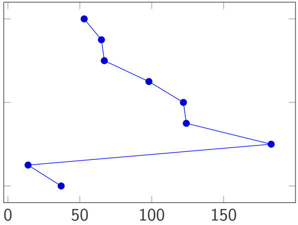
- SCAN and C-SCAN move head across the width of the disk

  No algorithm does this in practice

- LOOK and C-LOOK variants do not go all the way

  Respectively 208 *and* 322 *cylinders displacement*





Choosing an Algorithm

- SSTF is the most common one

  Result in better performance than with FCFS

- SCAN and C-SCAN reduce the risk of starvation

  Widely used on systems that are very dependent of the disk

- Performance depends on the number and type of request

  No difference if only one request in queue on average

- Also highly depends on the method used to allocate files

  SSTF and LOOK are the two algorithms by default

References

- Abraham Silberschatz, Peter B. Galvin, & Greg Gagne (2013). *Operating System Concepts*, John Wiley & Sons, ISBN: 978-1-11809-375-7.
- William Stallings (2017). *Operating Systems: Internals and Design Principles*, Pearson, ISBN: 978-1-29221-429-0.

Credits

- Michael Thom, November 1, 2008, https://www.flickr.com/photos/michaeljthom/3059568153.
- Kimberly Koppen, January 2, 2010, https://www.flickr.com/photos/kimberlykoppen/5858521608.
- Alpha six, June 2, 2006, https://www.flickr.com/photos/alphasix/158829630.