Important terms in OS

Processes and CPU scheduling

Processes

Process is the execution of a program.
Create a process: **fork()**
Execute a process: **execve()**
Let a process wait: **wait()**
Terminate a process: **kill()** or **exit()**

Interrupts and Exceptions

Interrupts = Force quit on a running proccess, happens because of asynchronically event (done disk operation)
Exceptions = Force quit on a running proccess, happens because of syncronically event (system call)

Context Switches

- Switching context between two processes
- Take care of the first process before making the second process to work
- Expensive, overhead
- Interrupts
- Transition between kernel-mode and user-mode

Process vs. Threads

Threads often called "light-weight" process. But threads switching is much cheaper than process switching. In general is threads and process very similar.

CPU Scheduling

Scheduling has deciding as their job. They get to decide which task that may use the resource, i.e., determines order by which requests are serviced, using different algorithms.

Scheduling algorithms

FIFO

This algorithm can be described as fair. The first task/process that gets in is the one that gets to finish their task first.

- Fair
- Simple

- Can have long finish and waiting time

Round Robin

This algorithm splits all the tasks up in same size (time based), then they do all the task in FIFO order, but with the splits.

- Better for interactivity
- Fair, but no task is luckier than others

Shortest Job First

This algorithm choose the shortes task (time based) and do their task first, before doing the other tasks.

- Simple
- Better average times compared to FIFO
- Hard to determine processing requirements
- Potentially huge finishing times and starvation (new shorter jobs arrive)

Priority & Preemption illustrated

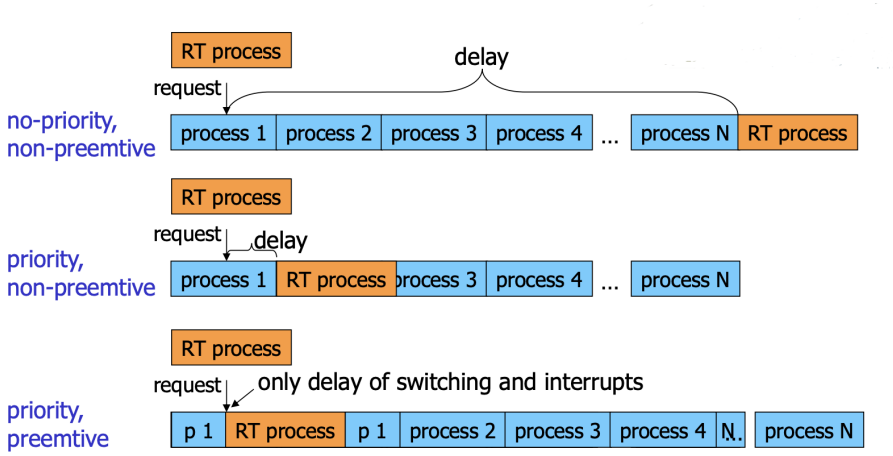

Figure 1: Here is an illustration on how priority and preemption works with a running process

More algorithms

Rate Monotonic (RM) Scheduling

- Preemtive
- Static priority

Earliest Deadline First (EDF)

- Preemtive
- Dynamic priority

Memory

Memory Management

Goal: managing the systems memory resources. - Allocate space to process - Protect the memory regions - Provide a virtual view of memory giving the impression of having more bytes than availability - Controll different levels of memory based on a hierarchy

Memory Hierarchies

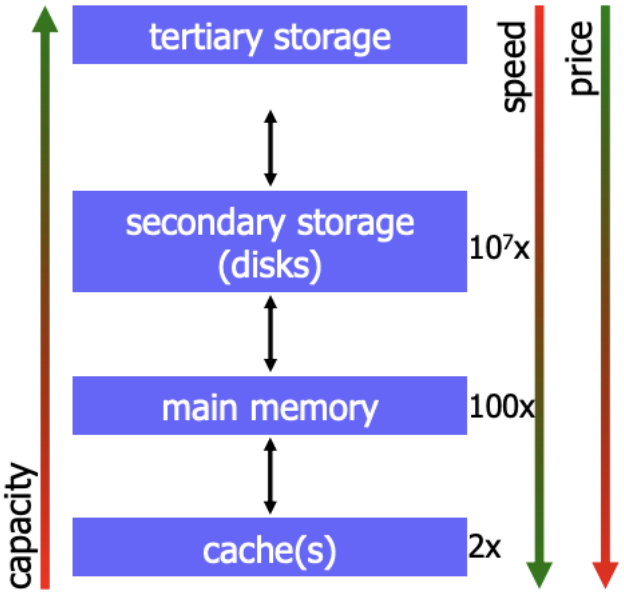

Figure 2: A illustration of a memory hierarchy

Addressing

Every posistion in memory has their own address, just like all houses has their own address.

Absolute addressing

- Often used by hardware

- Have reserved memory regions
- Read data by referencing the byte numbers in memory

We cant predict which addresses a programs gets and we cant use addresses directly.

Relative addressing

- Addressing dynamic
- Base addresses
- Relative addresses
- Dynamic address translation

Memory Layout

Code segment

- Reads from the program, for example execve()
- Usually read-only
- Can be shared

Data segment

- Global variables
- Here is the heap
- Dynamic allocated memory (malloc)
- Heap grows from low to high address

Stack segment

- Data used during a active program
- Stack grows from hight to low address

Memory Management for Multiprogramming

Swapping

- We remove the running process from memory and place it on the disk, then swap with a process from the disk that now is going to run in memory.
- Takes a lot of time!

Overlays

- User decides whats in memory and load overlays during the running task

Segmentation and paging

- Memory splits up in small pieces
- Pieces in same size (usually)

- Swapping the pieces after own needs between the disk and memory

Partitioning

Fixed Partitioning

- Easy to implement
- Support swapping of processes

The memory is divided into static partitions at the systems initialization time, the partitions size is made before letting programs use the memory.

Equal-sized partitions

- Large programs cannot be executed
- Small programs dont use the entire partition

Unequal-sized partition

- Large programs can be loaded at once
- Less internal fragmentation
- One queue or one queue per partition
- Require assignment of jobs to partitions

Dynamic Partitioning

The memory is divided at run-time

- Partitions are created dynamically
- And gets removed after doing their job

The partiotions is divided while the programs work.

Algorithms

- First fit
- Next fit
- Best fit

The Buddy System

A mix of dynamic partitioning and fixed partitioning.
Keyword: The power of 2

Segmentation

The size for memory is determined by a programmer. A process get dynamic allocated place in the memory. Different lenghts.

Pros:

- Dynamic
- No intert fragmentation
- Less external fragmentation

Cons:

- Adds a step to the adress translation

Paging

A process gets its place in one or more memoryframes, where the size is decided by processes.
Pros:

- No extern fragmentation

Cons:

- Some intern fragmentation (based on framesize)

Virtual Memory

Uses pagning to load parts of processes in memory, which makes it seems that we have more memory than we actually have. A virtual memory...

Memory Lookup

Looking for a specific bit in the memory, we have to use a page table. A page table represent binary numbers and a present bit. It its not any data in the page table (present bit = 0), there will be a page fault.

Page fault

Page fault can be described as data that isnt saved in the main memory, but has place in secondary memory (disk)

Page fault handeling

1. Hardware traps to the kernel savning program counter and process state information
2. Save general registers and other volatile information
3. OS discovers the page fault and determines which virtual page is requested
4. OS checks if the virtual page is valid
5. Select a page to replace
6. Checks if page frame is "dirty", updated. If so, write back to the disk, otherwise, just overwrite
7. When selected page frame is ready, the OS finds the disk address where the needed data is located and schedules a disk operation to bring into the memory

8. A disk interrupt is executed indicating that the disk I/O operation is finished, the page table are updated, and the page frame is marked "normal state".
9. Faulting instructions is backed up and the program counter is reset
10. Faulting process is scheduled, and OS returns to the routine that made the trap to the kernel
11. The register and other information is restored, and control is return to user space to continue execution as no page fault had occured

Page Replacement Algorithms

When a page fault is happening, OS has to select a page for replacement. We have several algorithms to replace a page.

FIFO (First In First Out)

- Easy to implement
- Fair
- Low overhead
- Cost a lot to access to the disk (unnecessary replacing happens to often)

Second Chance

- A r-bit that knows if a page is repeating itself
- In the name
- Less disk operation then FIFO

Clock

- More efficient that Second Chance
- R-bit

Least Recently Used (LRU)

- If a page is already in memory is about to access, put it back in the queue again (most recently used)
- Saves disk access
- Expencive

Interval Caching

- Caches all data between users.
- Prorities to cache the shortes intervalls between users first, for less disk access.

Storage/Disk and File Systems

Fact About Disk

- Secondary storage (disk)
- Cheaper than main memory
- More capacity
- Slow

Mechanics of Disk

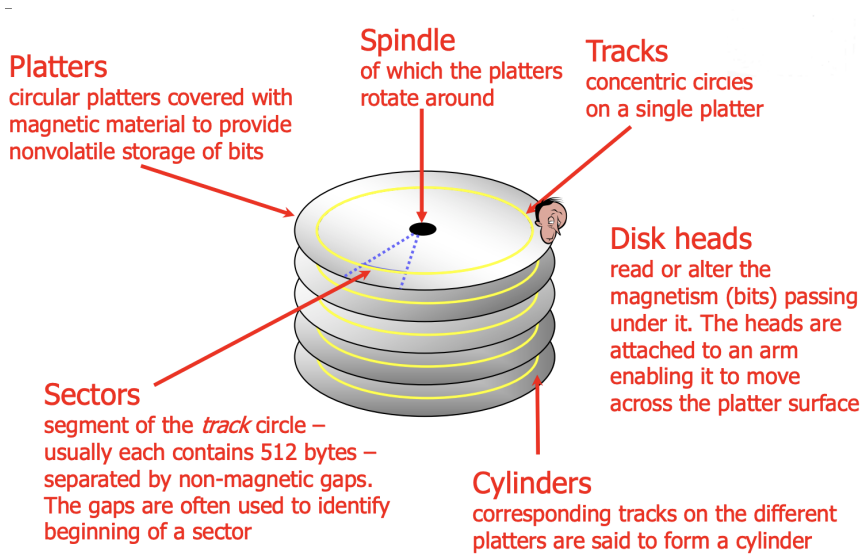

Figure 3: Picture of a disk and their mechanics

Disk Capacity

The size (storage space) depends on (multiply all):

- The number of platters
- Wheter the platters uses both side or not
- Numbers of tracks per surface
- Numbers of sectores per track
- Number of bytes per sector

Disk Access Time

Access time = seek time + rotational delay + transfer time + other delays

Seek time

Fixed Overhead + Seek Time Constant $\sqrt{Numberoftracks}$

Rotational Delay

Average delay is 1/2 revolution

Transfer time

Transfer rate: $\frac{amountofdatapertrack}{timeperrotation}$

Transfer time: $\frac{amountofdatatoread*timeperrotation}{amountofdatapertrack}$

Other delays

- CPU time
- Contention for controller, bus, memory
- Verifying block correctness with checksums (retransmissions)
- Waiting in scheduling queue

Disk Scheduling

Several algortithms when scheduling disks

First Come First Serve

- Just like FIFO, the first block that is requested is reading first
- Fair
- No starvation
- Lot of movement from disk head
- Unnecceccery intervalls between block

Shortest Seek Time First

- Goes to the block closest to the disk head at the moment
- Does move back and forth that much
- Starvation

SCAN

- Goes in one direction to the edge
- Doesnt change direction that often
- Takes long time if its only one block on the right side, and lots of block in the left side
- Unneccecery to go to the edge if it doesnt anything to read there

C-SCAN

- A lot like SCAN, but more of an optimization of SCAN
- Shorter response time
- Fair

LOOK

- Just like SCAN, but doesnt go to the edge

C-LOOK

- Just like C-SCAN, but doesnt go to the edge

Modern Disk Scheduling

Modern Disk Scheduling are more complex

- Hides their true layout
- Transparently move blocks to spare cylinders

- Have different zones
- Head accelerates – most algorithms assume linear movement overhead
- On device buffer caches may use read-ahead prefetching
- Black Box

Data Placement

Interleaved Placement

- Saves blocks from the same file in same sizes and space between files
- Minimal disk head movement

Contigous Placement

- Saves block from the same file right after each other
- Less disk head movement
- Reads more then one file at the time

Organ-Pipe

- Place files that often is access closest to where the disk head is the most (usually in the middle).
- Fast if we read files that are accessable often
- Slow if we have to read files that are far away

IPC

Communication between two processes. Or communication across address spaces and protection domains

Direct Communication

- Has to define sender/reciver
- Must have a buffer for sender/reciver

Indirect Communication

- Sender and reciver shares a queue (buffer) between each other

Mailboxes

- Messagetype: Sends messages with different types
- Buffer: Mailboxes are implemented as message queues sorting messages according to FIFO
- More than two processes: Many senders and recivers
- Two file descriptors

Pipes

- Messagetype: Sends messege with only one type
- Buffer: One or more pages saves the messages continuous
- More than two processes: Only one sender and reciver for Linux before
- Two file descriptors

Shared Memory

Shared Memory is an efficient and fast way for processes to communicate

- Read and write to the same memory address (virtual memory)
- Mapping into shared memory

Signals

A software generated interrupt that sends to a process.
Kill(pid, signal)
Raise(getpid(), signal)

Signal handling

A signal handler can be invoked when a specific signal is received. Can happen default, blocked or catches.