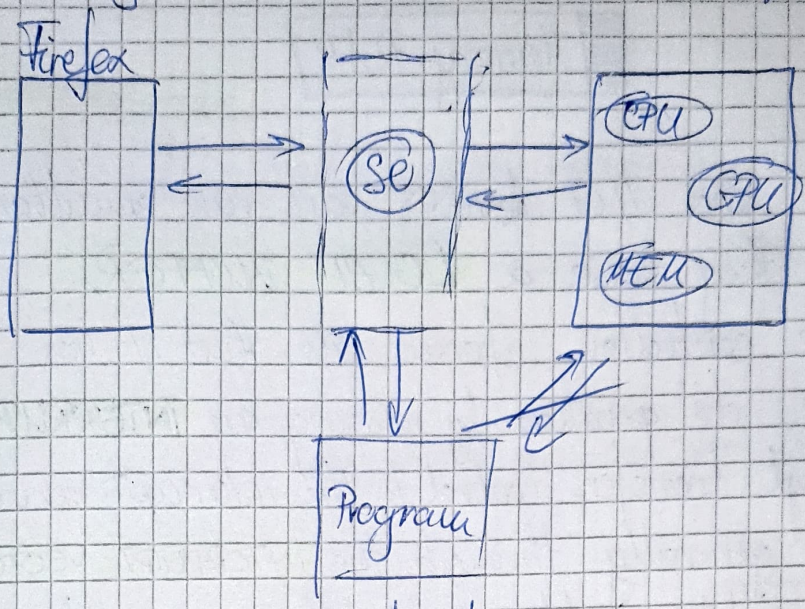
# CURS 1

Sistem de operare = program that acts as an intermediary between a user and the computer hardware



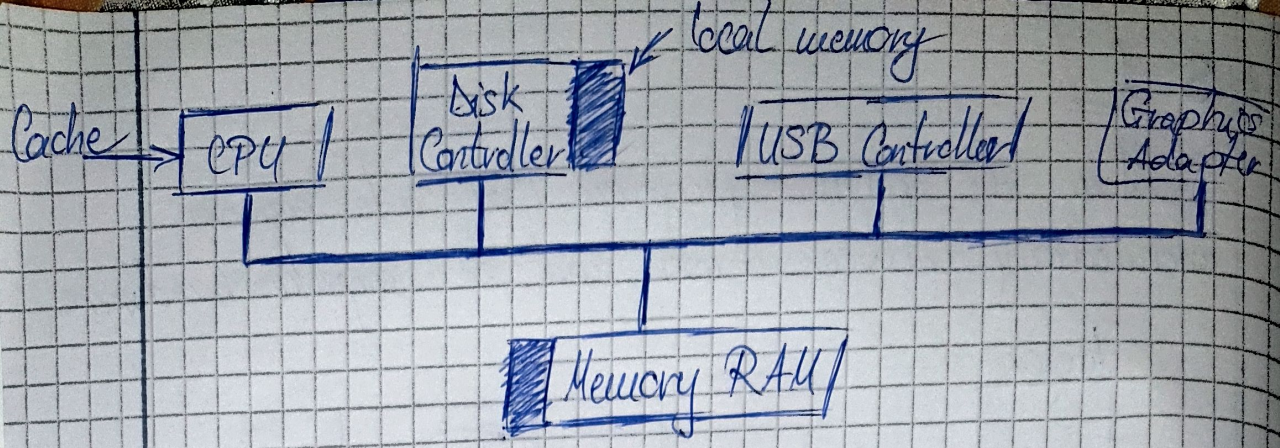

Computer System
- hardware
- SO → allocates resources
- Apps
- Users (can be other machines too)

On start-up:
- **BOOTSTRAP** (Firmware) → loaded on power-up or reboot
  - → typically stored in ROM
  - → initializes all aspects of system
  - → loads kernel
- **BOOTLOADER** checks which OS is installed.

One or more CPU's, device controllers connect through **BUS** providing access to shared memory.

există un controller în memorie care se ocupă de management-ul memoriei

BUS-ul e la viteza procesorului




- CPU and I/O devices can run simultaneously because they have a LOCAL BUFFER
- Device controller informs CPU that it has finished its operation by causing an INTERRUPT

Interrupt transfers control to the interrupt service routine generally through the INTERRUPT VECTOR which contains addresses of all service routines

A TRAP/EXCEPTION is a software generated interrupt caused either by an ERROR or a USER REQUEST

An ~~opr~~ OS is INTERRUPT DRIVEN

type of interruptions → POLLING inefficient dpdv tiwp
→ VECTORED INTERRUPT SYS

# CURS 2

Fisierele sunt stocate in SSD si' devin procese in RAM

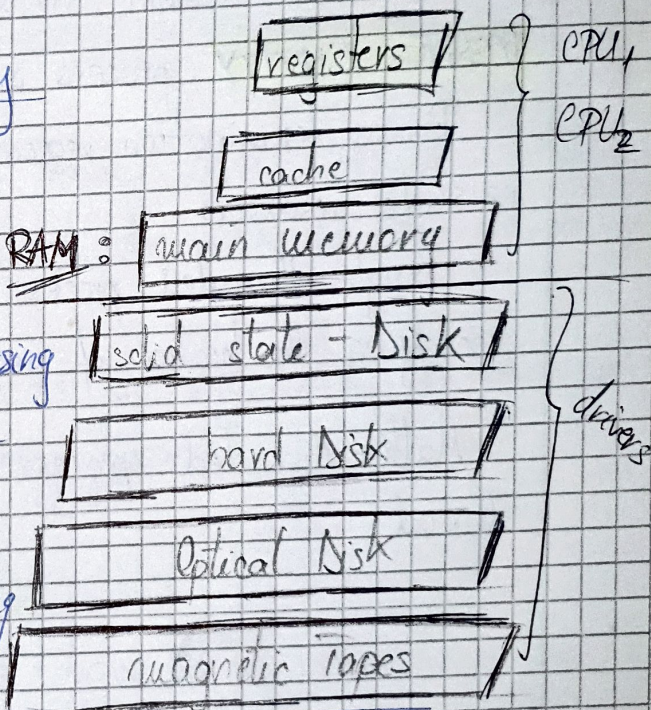
## I/O Structure

After I/O starts, control returns to user program only upon I/O completion

OR

——— 4 ——— without waiting for I/O completion

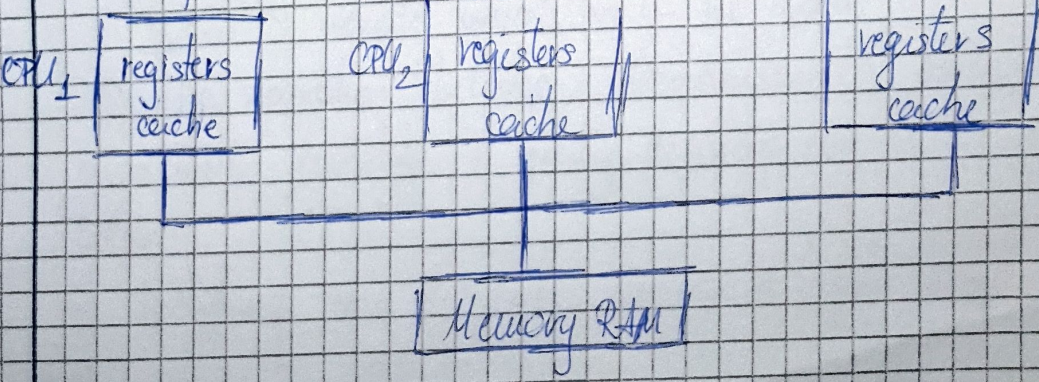
## Storage Device Hierarchy




## Multiprocessors

↳ Asymetric Multiprocessing
= each processor is assigned a specific task

↳ Symetric Multiprocessing
= each processor all tasks



# FROM USER TO KERNEL

Timer to prevent infinite loop

- timer is set to interrupt the computer after some time
- timer is set by OS
- when counter = 0 ⇒ causes an interrupt
- set up before scheduling process to regain or terminate program that exceed allotted time

A PROCESS is a program in execution. It is a unit of work within the system. Program is a PASSIVE ENTITY, process is an ACTIVE ENTITY

Process termination requires reclaim of any reusable resources

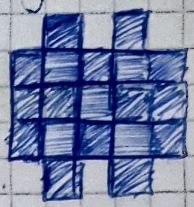
Single threaded process has one PROGRAM COUNTER specifying the location of next instruction to execute

Multi-threaded process has one program counter per thread

OERS 3

Each system call is associated with a number
- system-call interface maintains a table indexed according to these numbers

The system call interface invokes the intended system call in OS kernel => returns the status of the system call



## System Call Parameters Passing

- pass parameters in registers
- parameters stored in a block or table in memory and address of block passed as a parameter in a register
- parameters placed, or PUSHED ONTO THE STACK by the program and POPPED off the stack by the operating system

Block and stack methods do not limit the number or length of parameters being passed

## OS Design & Implementation

MECHANISMS determine how to do something

POLICIES decide what will be done

The separation between Mechanism & Policies allows maximum flexibility if policy decisions are to be changed later

Specifying and designing an OS is highly creative task of software engineering.

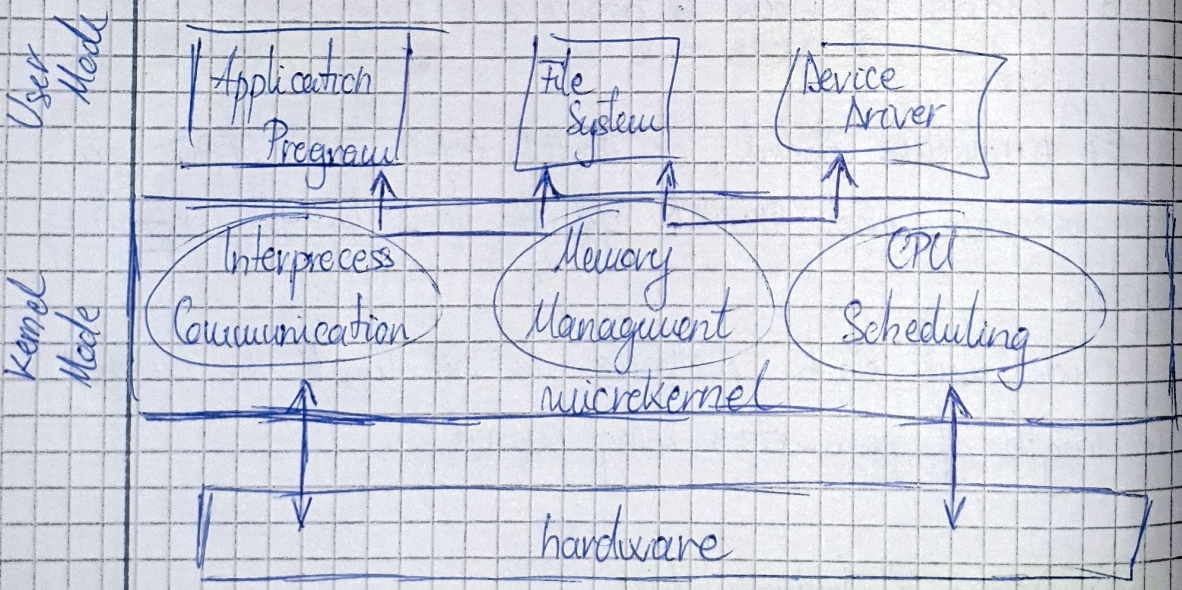
EMULATION can allow an OS to run on non-native hardware

The OS is divided into a number of layers. With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

## MICROKERNEL

Moves as much from the Kernel into user space

Communication takes place between user modules using MESSAGE PASSING




## HYBRID SYSTEMS

They combine multiple approaches to address performance, security, usability needs

SYSGEN program obtains information concerning the specific configuration of the hardware system
- Used to build system-specific compiled kernel or system-tuned

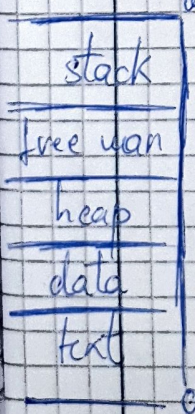
SYSTEM BOOT
- Firmware ROM used to hold initial boot code
- OS must be made available to hardware so hardware can start it

CURS 4

## PROCESSES

An OS executes → batch systems = jobs
→ time-shared systems = user prog.

PROCESS = a program in execution; process must progress in sequential ~~action~~ fashion

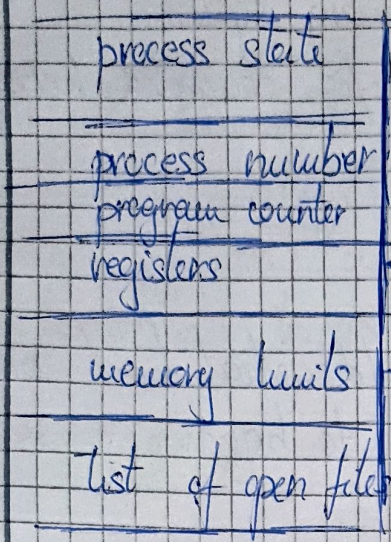



text section = code
program counter = current activity
stack = temporary data
data section = global variables
heap = memory dynamically allocated

process states
- terminated
- new
- ready
- running
- waiting

PROCESS CONTROL BLOCK

TASK CONTROL BLOCK




accounting information = CPU used, clock time elapsed since start, time limits

THREADS

A process has a single thread of execution

multiple program counters/process

⇒ multiple locations can execute at once

⇒ multiple threads of control → THREADS

PROCESS SCHEDULING

Maximize CPU use, quickly switch processes onto CPU time sharing

PROCESS scheduler selects among available processes for next execution on CPU

Maintains SCHEDULING QUEUES of processes

- job queue → set of all processes in the system
- ready queue → set of all processes residing in main memory, ready and waiting to execute

- device queues → set of processes waiting for an I/O device
- Processes migrate among the various queues

## CURS 5

A process cannot access the memory address of another process

- Sistemul de operare da impresia procesului ca toata memoria este pentru el

### PROCESS CREATION

Parent process creates CHILDREN PROCESSES which in turn create other processes, forming a tree of processes

Processes identified and managed via PID (process identifier)

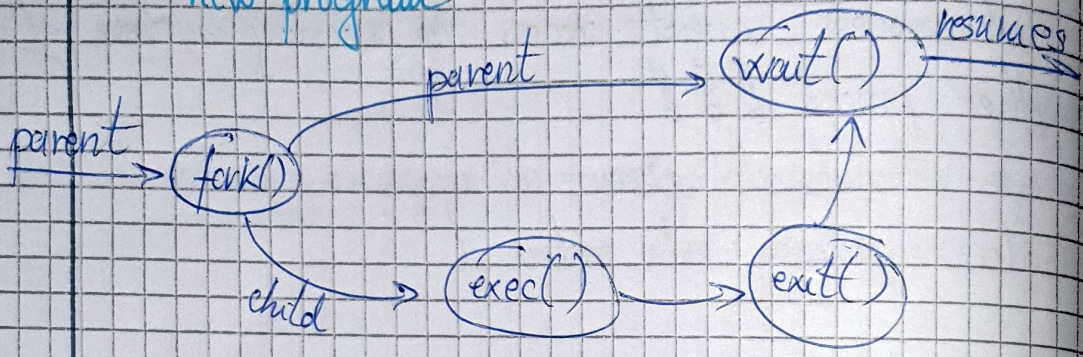
RESOURCE SHARING OPTIONS
- Parent and Children share all resources
- Children share subset of parent's resources
- Parents & children share no resources

EXECUTION OPTIONS
- Parents & children execute concurrently
- Parent wait until children terminated

ADDRESS SPACE

- Child duplicate of parent
- Child has a program loaded onto it

fork() → system call creates new process
exec() → system call used after fork()
to replace the process' memory space with a
new program




PROCESS TERMINATION

Process executes last statement and then asks the OS to delete it using exit()

- Returns status data from child to parent (via wait())
- Process' resources are deallocated by OS

Parent may terminate the execution of children processes using the abort() sys call

- child has exceeded allocated resources
- Task assigned to child is no longer required
- The parent is exiting and the OS does not allow a child to continue if its parent terminates

Some SO dont allow child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.

- cascading termination: All children, grandchildren are terminated
- the termination is initiated by the OS

The parent process may wait for the termination of child process by using the wait() sys call. The call returns status info and the PIDs of the terminated process

→ pid = wait(&status)

If no parent waiting (didn't invoke wait())
⇒ zombie process

If parent terminated without invoking wait()
⇒ orphan process

If one web tab causes trouble ⇒ entire browser can crash

## INTERPROCESS COMMUNICATION

Process within a system may be → INDEPENDENT / → COOPERATING

Cooperating process can affect or be affected by other processes including SHARING DATA

Cooperating processes need IPC (interprocesses communication)

IPC → shared memory / → message passing

~~Advantages~~: modularity

PRODUCER PROCESS produces info that is consumed by a CONSUMER PROCESS

- Unbounded-buffer = places NO practical limit on the size of the buffer
- bounded-buffer = assumes that there is a fixed buffer
  - → shared memory solution

~~SHARED MEMORY~~ MESSAGE PASSING

An area of process to communicate and to sync their actions

Message System = processes communicate with each other without reserting to share variables

IPC facility provides 2 operations → SEND, RECEIVE } mess

The message size is either fixed or variable

If process P & Q wish to communicate

steps → establish a communication link
→ exchange messages (send/receive)

Implementation of Communication Link
→ physical → shared memory
→ hardware bus
network
→ logical → direct or indirect

- communication links are established automatically
  → can be associated with exactly one pair of communication processes
- each pair of processes may share multiple communication links

SYNCHRONIZATION

Message passing may be either blocking OR non-blocking

BLOCKING → synchronous
- blocking send → the sender is blocked until the message is received
- blocking receive → the receiver is blocked until a message is received

NON-BLOCKING → asynchronous
- non-blocking send → the sender sends the message and continues
- non-blocking receive → the receiver receives NULL or a valid message

If blocking and receive we have randez-vous

# CURS 6

Message passing centric via ADVANCED LOCAL PROCEDURE CALL (LPC) facility
- Only links between processes of the same system
- Uses ports to establish and maintain communication channels

## SOCKETS

A SOCKET is defined as an endpoint for communication
- Communication consists between a pair of sockets
- host:port

## REMOTE PROCEDURE CALLS

Data representation handled via External Data Representation (XDL) format to account for different architectures
- big-endian : MS bit
- little-endian : LS bit

Remote communication has more failure scenarios than local
- Messages can be ~~believed~~ delivered exactly once rather than at most once

Os typically provides matchmakers service to connect client and server

## PIPES

Acts as conduit allowing 2 processes to communicate

ORDINARY PIPES - cannot be ~~stressed~~ accessed from outside the process that created it. A parent process creates a pipe and uses it to communicate with a child process that it created

NAMED PIPES - can be accessed without a parent - child relationship

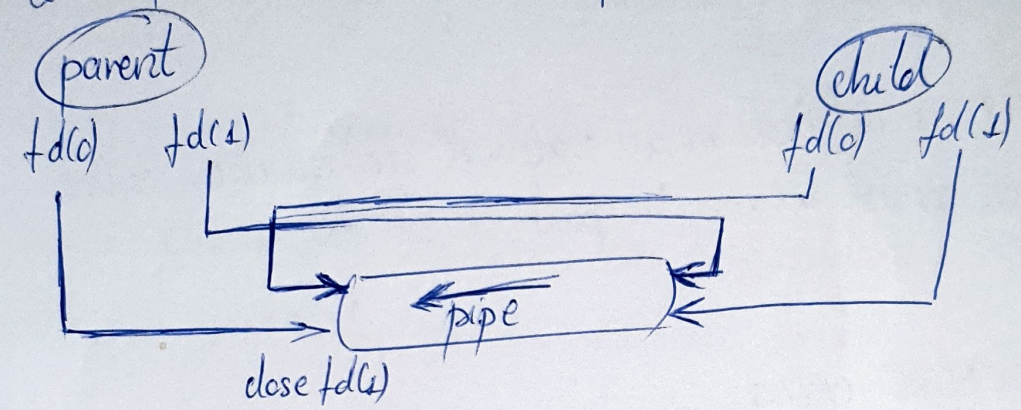
## ORDINARY PIPES

allow communication in standard - ~~procedure~~ producer consumer style

Producer writes to one end and consumer reads from the other end.

Ordinary pipes are therefore unidirectional

Require parent-child relationship between communicating processes




## NAMED PIPES

Are ~~the~~ more powerful than ordinary pipes

Communication is bidirectional

NO parent-child relationship is required

Several processes can use the named pipe for communication

## MULTICORE PROGRAMMING

types of parallelism
- DATA PARALLELISM - distributes subsets of the same data across multiple cores, same operation on each
- TASK PARALLELISM - distributing threads across cores, each thread perform unique operation

As the number of threads grows, so does architectural support for threading

# CURS 7

## AMDAHL'S LAW

Identifies performance gains from adding cores to an app that has both serial and parallel components

$$\text{speedup} \leq \frac{1}{S + \frac{1-S}{N}} \quad ; \quad \begin{matrix} S - \text{serial portion} \\ N - \text{processing cores} \end{matrix}$$

As $N \to \infty \Rightarrow$ speedup $\to \frac{1}{S}$

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

MANY-TO-ONE threads mapped to a single Kernel thread.
- one thread blocking causes all to block
- multiple threads may not run in parallel on multicore sys because only one may be in kernel at a time

CPU → kernel → user → thread$_1$, thread$_2$, thread$_3$

ONE-TO-ONE
- each user-level thread maps to Kernel thread
- creating a user-level thread creates a kernel thread
- more concurrency than many-to-one
- ~~not~~ number of threads/process sometimes restricted due to overhead

MANY-TO-MANY allows many user level threads to be mapped to many kernel threads
- allows the OS to create a sufficient number of kernel threads

THREAD LIBRARY → for API

PTHREADS may be provided either as user-level or kernel-level

CRITICAL SECTION PROBLEM
Each process has a CRITICAL SECTION segment of code
- process may be changing common variables, updating tables etc
- when one process is in critical section, no other may be in its critical section

Critical section problem is to design protocal to solve this
Each process must ask permission to enter critical section in ENTRY SECTION, may follow critical section with EXIT SECTION then REMAINDER SECTION

```
do {
    entry section
        critical section
    exit section
        remainder section } while (true);
```

## SYNCHRONIZATION HARDWARE

Many systems provide hardware support for implementing the critical section code.

idea of locking: protecting critical regions via locks

Uniprocessors - could disable interrupts
- currently running code would execute without preemption
- generally too inefficient on multiprocessor sys

lock => actions become ATOMIC operations (-> cannot be interrupted)
↳ check if there is access to the variable which shows if the operation is locked or not

MUTEX LOCKS = protect a critical section by first acquire() a lock and then release() the lock
- boolean variable to indicate if the lock is available or not
- acquire() & release() are atomic
- this solution requires BUSY WAITING => lock is called SPINLOCK

```
do{ acquire lock()
      critical section
  release
      remainder
```
} HARDWARE SOLUTION

SEMAPHORE = sync tool that provides more sophisticated ways for processes to sync their activities
- can only be accessed via two indivisible (atomic) operations wait(); signal()

# DEADLOCK & STARVATION

DEADLOCK = 2 or more processes are waiting indefinitely for an event that can be caused by only one of the waiting process

STARVATION - INDEFINITE BLOCKING

A process may never be removed from the semaphor queue in which it is suspended

PRIORITY INVERSION = scheduling problem when lower-priority process hold a lock needed by higher-priority process

solution: priority-inheritance protocol

↑

↳ PETERSON

- syncs 2 processes
- the 2 processes have 2 variables in common
  - → turn: index which describes which process is in the critical condition
  - → flag: shows which process waits in line ~~for~~ to enter the critical condition

```
do
{ flag[i] = TRUE;
turn = j;
while (flag[j] && turn == j)
    // critical section
flag[i] = FALSE
    // remainder section } while(TRUE);
```

} SOFTWARE SOLUTION

# CURS 9

## MONITORS - SOLUTION FOR DEADLOCKS

↳ a high-level abstraction that provides a convenient & effective mechanism for process sync

- ABSTRACT DATA TYPE → internal variables only accessible by code within the procedure
- only 1 process may be active within the monitor at a time
- monitor is not powerful enough to model some sync schemes

condition x, y;

x.wait(); → a process that invokes the operation is suspended until x.signal()

x.signal() → resumes one of the processes that called x.signal()

options → SIGNAL & WAIT = P waits until Q leaves the monitor or for another condition

SIGNAL & CONTINUE = Q waits until ——— 〃 ———

```
wait(mutex)
    ||
  body of F
    ||
if(next_count > 0)
  signal(next)
else signal(mutex);
```

# CURS 10

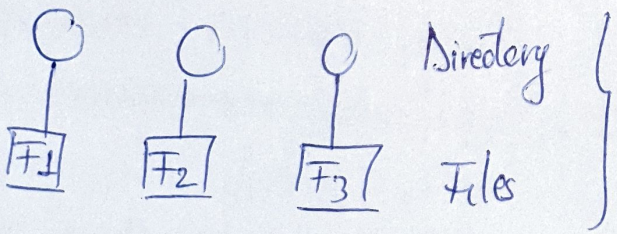
OPEN FILES → file-open count = counter of number of times a file is open to allow removal of data from open-file table when last process closes it

OPEN FILE LOCKING

- SHARED LOCKS ↔ several processes can acquire concurrently; similar to reader lock
- EXCLUSIVE LOCK → similar to writer lock

- MANDATORY → access is denied depending on locks held and requested
- ADVISORY → processes can find status of locks and decide what to do.

DIRECTORY STRUCTURE




Both the directory & files reside on disk

TREE STRUCTURED DIRECTIONS

path
- ABSOLUTE: from source
- RELATIVE: from current folder

## FILE SYSTEM MOUNTING

A file system must be MOUNTED before it can be accessed. An UNMOUNTED file system is mounted at MOUNT POINT

## REMOTE FILE SYSTEMS

Uses networking to allow file system access between systems
- → manually via programs
- → automatically via DISTRIBUTED FILE SYSTEMS
- → semi-automatically via WORLD WIDE WEB

CLIENT SERVER model allows clients to mount remote file systems from servers

DISTRIBUTED INFORMATION SYSTEMS (D. NAMING SERVICES) such as Active directory implement unified access to info needed for remote computing

## FILE SYSTEMS - FAILURE MODELS

~~METADATA corruption of directory structures or non-user data~~

Recovery from failure can involve STATE INFORMATION about status of each remote request

STATELESS PROTOCOLS such as NFS v3 include all info in each request, allowing easy recovery but less security

# CURS 11

## PARTITIONS & MOUNTING

Partition can be a volume containing a file system or RAW - just a sequence of blocks with no file system

BOOT BLOCK can point to a boot volume/loader set of blocks that contain enough code to know how to load the kernel from the file system

ROOT PARTITION contains the OS, other partitions can hold other OSs
- mounted at boot time
- other partitions can mount automatically or manually

## DIRECTORY IMPLEMENTATION

LINEAR LIST of file names with pointer to data blocks
- simple to code
- time-consuming

HASH TABLE - linear list with hash data structure
- decreases directory search time
- collisions - situations where 2 file names hash to the same location
- good if entries are fixed size or use chainsaw-overflow method

## ALLOCATION METHODS - how disk blocks are allocated to files

CONTIGUOUS ALLOCATION - each file occupies set of contiguous blocks

LINKED ALLOCATION - each file a linked list of blocks
- file ends at NIL POINTER
- NO external fragmentation
- improve efficiency by clustering blocks into groups but increases internal fragmentation
- → FAT (File Allocation Table)

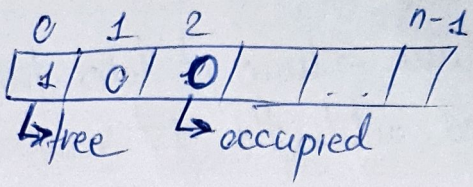
INDEXED ALLOCATION - each file has its own index of blocks of pointers to its data blocks

PERFORMANCE

FREE SPACE MANAGEMENT

File system maintains FREE SPACE LIST to track available blocks/clusters

BIT VECTOR/MAP →




## CURS 12

CPU BURST distribution is of main concern

↳ load > store
add
read from file

CPU SCHEDULER

SHORT-TERM SCHEDULER selects from among the processes in ready queue and allocates CPU to one of them

states: 1 running → ready
2
3 waiting → ready
4. terminates
running → waiting

} NONPREEMPTIVE
all other scheduling is PREEMPTIVE

DISPATCHER module gives control of the CPU to the process selected by short term scheduler

→ switch context
→ switch to user mode
→ jumping to the program location in the user program to restart the program

DISPATCH LATENCY = time for the dispatcher to stop a process and start another one (more like "freeze"

SCHEDULING CRITERIA

MAX
- CPU usage
- throughput → #processes that complete their exec/time unit

MIN
- turnaround time → time to execute a particular process
- waiting time →
- response time → time it takes from when a request was submitted until the 1st response is produced (UI, UX)

FCFS SCHEDULING – First Come First Served

CONVOY ~~EFFECT~~ – short process behind long process.

SJF SCHEDULING – Shortest Job First

Gives the minimum average waiting time for a set of processes.
- associate with each process the length of its next CPU burst
→ is priority scheduling where priority is the inversed predicted next CPU burst time

DETERMINING LENGTH OF NEXT CPU BURST

$t_n$ – actual length of $n^{th}$ CPU burst

$t_{n+1}$ – predictive value

$0 \leq \alpha \leq 1 \Rightarrow \boxed{t_{n+1} = \alpha t_n + (1-\alpha) t_n}$

↳ usually set to $\frac{1}{2}$

Preemptive version called SHORTEST REMAINING TIME FIRST

problem = STARVATION – low priority processes may never execute
solution = AGING – as time progresses increases the priority of the process

## ROUND ROBIN (RR)

Each process gets a small unit of CPU time (TIME QUANTUM $q$) usually 10-100 ms. After this time has elapsed, the process is preempted and added to the end of the ready queue (RQ).

If there are $n$ processes in the RQ and $q$ => each process gets $1/n$ of the CPU time in chunks of at most $q$ time units at once. No process waits more than $(n-1)q$ time units

$q$ large => FIFO

$q$ small => $q$ must be large to respect the context switch

$q >$ CONTEXT SWITCH

## MULTILEVEL QUEUE

ready queue → FOREGROUND → RR (interactive)
→ BACKGROUND → FCFS (batch)

each queue has its own scheduling ~~program~~ alg.

highest priority
- system processes
- interactive processes
- interactive editing processes
- batch processes
- student processes

lowest priority

# THREAD SCHEDULING

- distinction between user-level and kernel level threads
- many-to-one
- many-to-many
- ~~thread library~~

§ run on LWP = process-contention scope (PCS) = competition among all threads

# MULTI-PROCESSOR SCHEDULING

HOMOGENEOUS PROCESSORS

ASYMETRIC MULTIPROCESSING = only 1 processor accesses the system data structures

SYMMETRIC PROCESSING (SMP) = each processor is self-scheduling all processes in common ready queue or each one has its own private ready queue

PROCESSOR ~~AFFINITY~~ - process has affinity for processor on it is currently running
- SOFT ~~AFFINITY~~
- HARD ~~AFFINITY~~
- PROCESSOR SETS

## SMP

LOAD BALANCING - attempts to keep workload evenly distributed

PUSH MIGRATION - periodic task checks load on each processor

PULL MIGRATION - idle processors pulls waiting task from busy processor

# CURS 13

## LOGICAL vs PHYSICAL ADDRESS SPACE

Logical address space that is bound to a separate PHYSICAL ADDRESS SPACE is central to proper memory management.

- LOGICAL ADDRESS - generated by the CPU (VIRTUAL ADDRESS)
- PHYSICAL ADDRESS - address seen by the memory unit

→ they are the same in compile/load time address-binding schemes

→ ≠ execution-time

LOGICAL ADDRESS SPACE = set of all logical addresses generated by the program

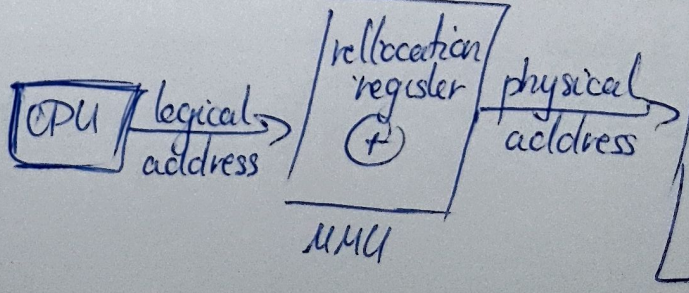
PHYSICAL ADDRESS SPACE = — " — physical — " —

## MEMORY-MANAGEMENT UNIT (MMU)

- hardware device that at run time maps virtual to physical address.
- the user program deals with logical addresses
  - ↳ exec. time binding occurs when reference is made to location in memory
    - ↳ logical address bound to physical addresses

## DYNAMIC RELOCATION USING A RELOCATION REGISTER

- routine is not loaded until is called
- better memory space utilization
- all routines kept on disk in relocatable load format



## DYNAMIC LINKING

STATIC LINKING = sys libraries & program code combined by the loader into the binary program image

DYNAMIC LINKING = linking postponed until execution time

STUB = small piece of code used to locate the appropriate memory resident library routine (↳ useful for libraries)

↳ replaces itself with the address of the routine & executes the routines

system also known as SHARED LIBRARIES

## MULTIPLE PARTITION ALLOCATION

- degree of multiprogramming limited by num. of partitions
- VARIABLE PARTITIONS sizes of efficiency
- HOLE - block of available memory

→ when a process arrives, it is allocated memory from a hole large enough to accomodate it

→ existing process frees its partition, adj free partitions combined

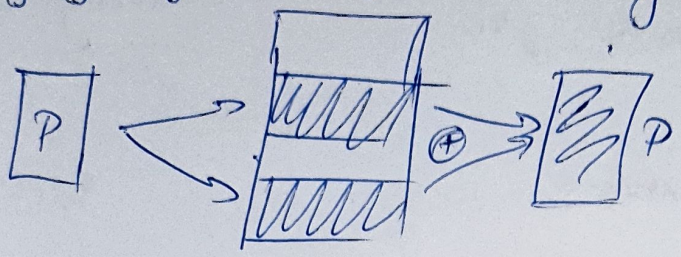
→ OS maintains ↗ allocated partitions
↘ free —"—

## DYNAMIC STORAGE-ALLOCATION PROBLEM

- FIRST-FIT : first hole big enough
- BEST-FIT : allocate the SMALLEST hole that is big enough
- WORST-FIT : allocate the LARGEST hole

FRAGMENTATION

- EXTERNAL FRAGMENTATION = total memory space to satisfy a request
- INTERNAL FRAGMENTATION = allocated memory may be slightly larger than requested memory




SEGMENTATION ARCHITECTURE

SEGMENT TABLE = maps 2-dimensional physical addresses (PA)
- BASE = contains the starting PA
- LIMIT = length of the segment.

SEGMENT-TABLE BASE REGISTERS (STBR) = points to the segment table's location in memory.

— " — LENGTH — " — (STLR) = indicates the number of segments from the program

segment_num < STLR !

PAGING

Process is allocated physical memory whenever the latter is available
Divide { physical memory into fixed-sized blocks called FRAMES
logical — " — blocks of the same size called PAGES
Set up a PAGE TABLE to translate logical → physical address
- still have internal fragmentation

ADDRESS TRANSLATION SCHEME

PAGE NUMBER (p) = used as an index into PAGE TABLE which contains base address of each page in PHYSICAL MEMORY

PAGE OFFSET (d) = combined with base address to define the physical memory address that is sent to the memory unit

LOGICAL ADDRESS SPACE - $2^m$

PAGE SIZE $2^n$

IMPLEMENTING OF PAGE TABLES

Page table is kept in main memory

PAGE-TABLE BASE REGISTER (PTBR) - points to the page table

— " — LENGTH — " — (PTLR) - size of the page table

Every data access requires 2 memory accesses

The two memory access problem can be solved by the use of a special fast-lookup hardware cache called ASSOCIATIVE MEMORY OR TRANSLATION LOOK-ASIDE BUFFERS (TLBs)

EFFECTIVE ACCESS TIME (EAT)

$EAT = (1+\varepsilon)\alpha + (2+\varepsilon)(1-\alpha) = 2+\varepsilon-\alpha$

$\varepsilon$ = time unit

$\alpha$ = hit ratio = % of times that a page number is found in the associative registers

MEMORY PROTECTION - associate protection bit with each frame to indicate if RO or RW access is allowed

VALID-INVALID BIT → VALID ⇒ the associated page is in the process logical Address space

↓

INVALID = — u — is not — u —

↓

OR use

↓

PTLR