P2L1 Processes and Process Management

Goal

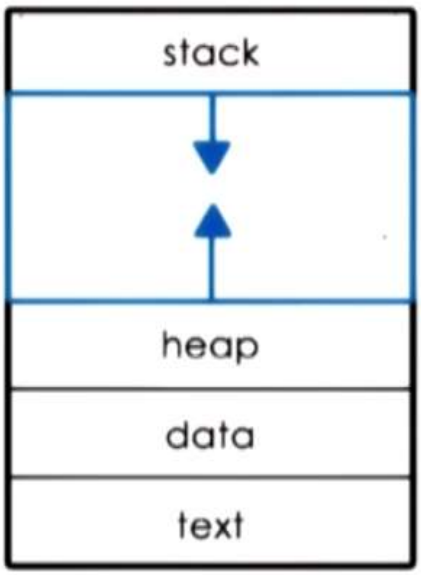
1. What is a process
2. How are processes represented by Os?
3. How are multiple concurrent processes managed by OS?

Simple Process Definition

> **One-liner:** A process is an instance of an executing program.

Synonym: task/ job

1. What does a process look like?



A process encapsulates all the data for running application.
Every element of the process statate has to be uniquely identified by its address.

It includes:

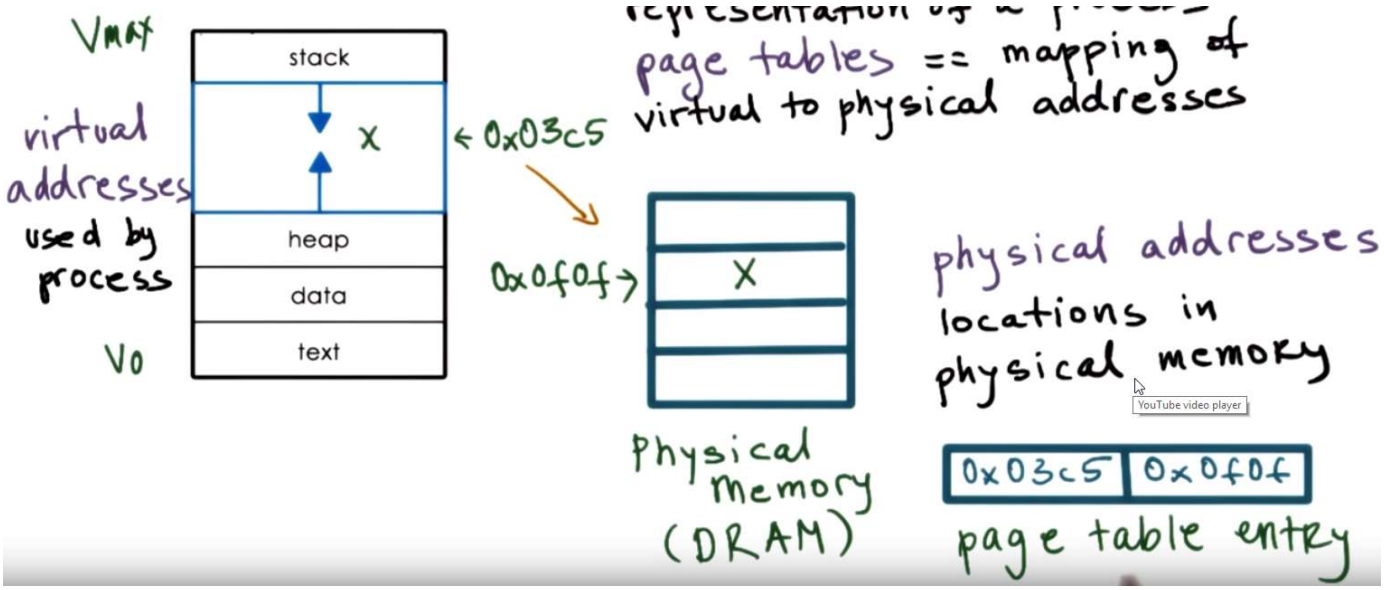
- code (text)
- data (statical state when the process first loads)
- heap (dynamically allocated during execution)
- stack (dynamic LIFO state)

1.1 Virtual Address Space

Address Space is in memory representation of a process.
The address space a process uses is **virtual addresses**.

Physical addresses: Location in physical memory (DRAM).
Page Tables: Keeps mapping of virtual to physical addresses.

When an executing process tries to access an address in its virtual address space, it's actually accessing the physical address in memory that the virual address maps to.



However, there are problems:
Parts of virtual address space may not be allocated.
Solution: Only the allocated virtual addresses are mapped onto physical memory.
There may not be enough physical memory for all state in all processes
Solution: When there is not enough physical memory, some of the allocated data will be moved to disk. The memory stores different process' allocation.

In both cases, the OS must keep the information about where each piece of data are located.

To each process running at the same time, they both have access to the entire virtual address space while the OS makes sure their address spaces are mapped to different places in the underlying real memory.

2. How is a process executed?

2.1 How does the OS know what a process is doing?

Souce Code --> Assembly Code -->binary code -->
Program Counter (PC):

- Points to the current binary code the process is executing.
- Maintains on the CPU while the process is executing.

Other registers on the CPU:

- Hold values necessary during the execution
- They have information like addresses for data
- Other status information

Process Stack Pointer (SP):

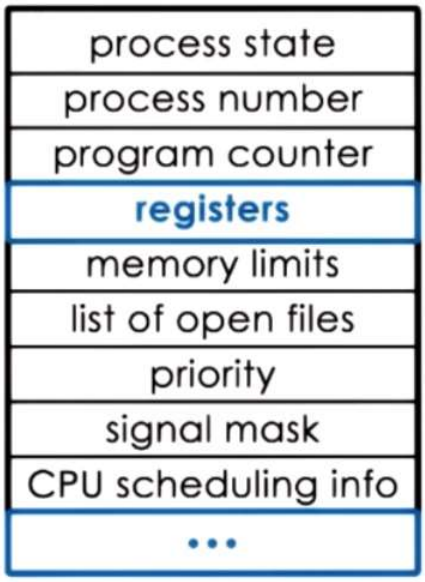
- Points to the top of a process's stack.

...

2.2 PCB (Process Control Block)

A data structure that the operating system maintains for every one of the processes that it manages.

What does it include?



Properties of PCB

- created when the process is initially created itself
- initialized at creation --e.g. PC will point to the first instruction
- Certain fields are updated when process state changes
  - e.g. When the process requests more memory the memory limits changes as well as the mapping between virtual and real address space.
- Other fields change too frequently
  - e.g. Program counter changes after execution of every instruction. We don't want to spend time writing it so the PC register gets automatically updated by CPU on every new instructions.

The OS makes sure to collect and save all info the CPU maintains for a process --> store in PCB when the process is no longer running on the CPU.

Example, the CPU is executing P1 and P2 at the same time. But they are switched on and off interchangeably. When P1 is switched off CPU, P1's CPU data will be stored in its PCB and CPU will load P2's execution data from P2's PCB.

Context Switch: CPU's swapping between processes is called context switch.

2.3 Context Switch

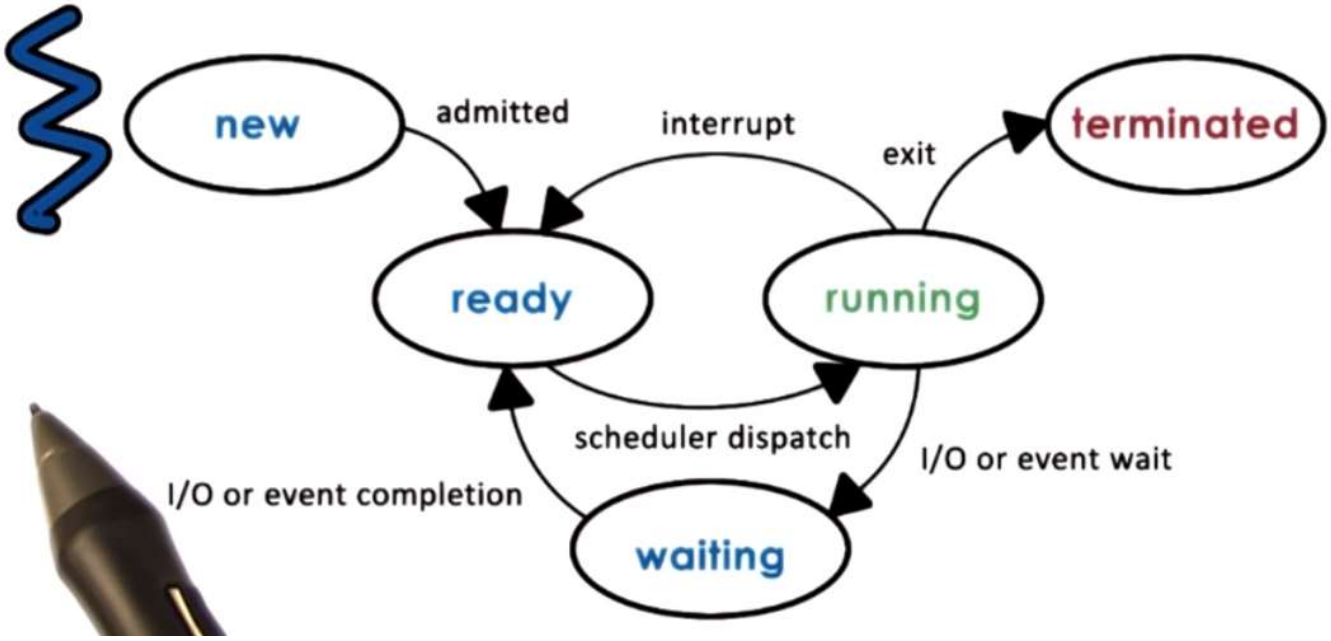
Formal definition: Switching the CPU from the context of one process to the context of another.

Context Switch is expensive

- Direct Costs:
  - Number of cycles to load and store instructions.
- Indirect Costs:
  - Cold Cache & cache miss
    - When p1 is running a lot of its data are in the cache
    - If the CPU switch to P2, the P1 data in chache will be replaced to make place for P2 data
    - When P1 is switched back, it will have a cold cache
    - P1 then have to retrieve data from memory, which could be 100 times slower.

We want to limit the frequency of context switchin.

3. Process Life Cycle

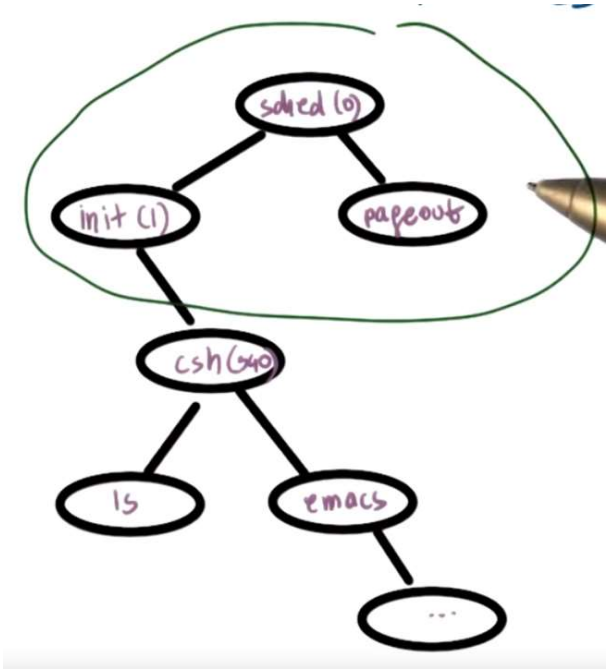


3.1 Possible Process States

- new:
  - OS perform admission control, if ok then proceed
  - allocated & initiate PCB and initial resources
  - Ready to start execution when minimum resources are available
- Ready:
  - wait until the scheduler is ready to move it into running state on the CPU.
- Running:
  - The process gets executed on the CPU
  - It could be interupted from there and go back to running state
  - If it needs to initiate some longer operation like I/O or wait for an event, it'll be moved to the waiting state.
- Waiting:
  - When the longer operation is completed it'll become ready again.
- Terminated:
  - Execution is completed.
  - Exit with approapriate exit code.

3.2 How are processes created?

In OS, a process can create child processes.
All processes in an OS comes from a single root and they form tree structure:



Some processes in the tree are privileged processes — root processes.

boot processes spawned --> OS is loaded on the machine -->create some number of initial processes

Then, when a user logs in its shell process is created --> user types in commands --> new processes gets spawned from shell parent process

Basic mechanisms for process creation

fork:

- create a copy of the parent PCB into the chid PCB.
- child continues execution at instruction after fork.
- Parent and child processes have the identical virtual memory space

Exec:

- Take a child process created by fork
- replace child process' entire PCB
- Load new program and start from new program's first instruction.

Usual workflow: fork() to create a child process -> call exec() to replace the PCB of child process

On unix based OS, init is the parent of all processes
On Android OS parent of all app processes is ZYGOTE

3.3 Role of CPU Scheduler

Determines which ready state process should be dispatched onto the CPU for execution and for how long should the process run.

Preemption: Interrupt the executing process and save its current context.

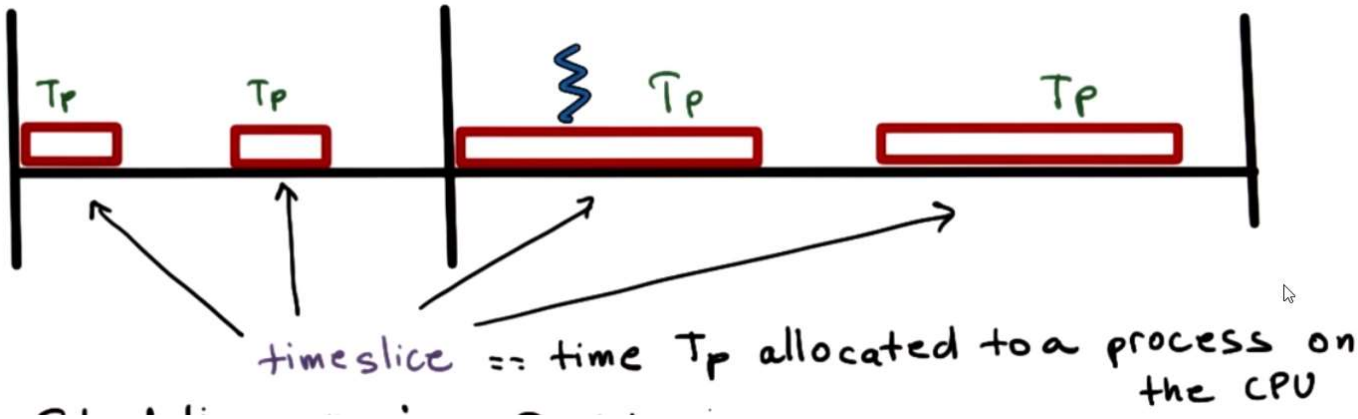
Current running process time's up --> preempt --> choose next process --> dispatch process onto CPU + save and load context.

The OS should minimize the time it takes to perform scheduling.

Efficiency is crucial in CPU scheduler.

3.4 Length of Process

How long should the process run for? The longer each runs the less frequently CPU time wasted on scheduler.

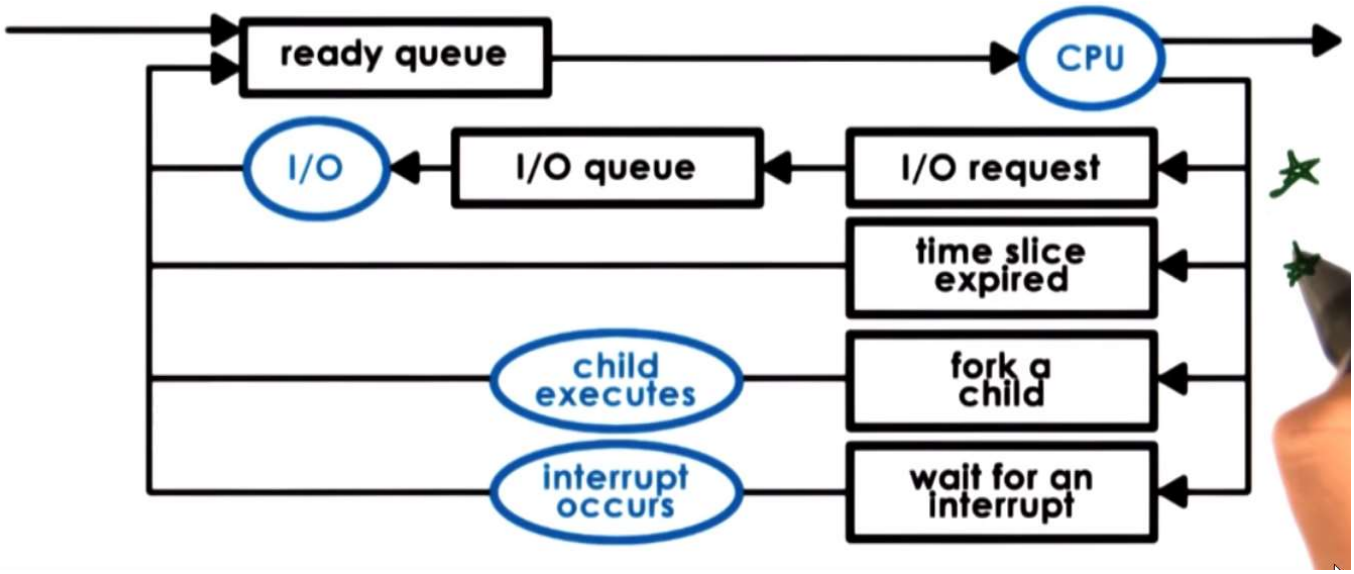


timeslice: time Tp allocated to a process on the CPU
useful CPU work = timeslice / total time

Scheduling Design Decisions

- Appropriate timeslice value?
- Metrics to choose the next process to run?

3.5 How I/O Operations affect scehduling?



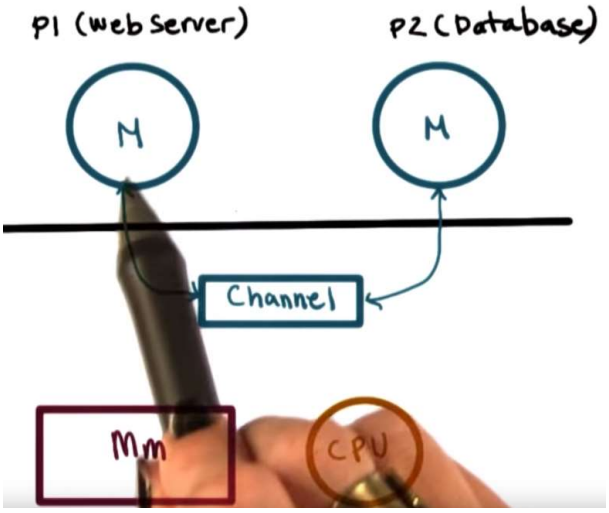
4. Inter Process Communication (IPC)

An OS must provide mechanisms for processes to interact with one another.

- transfer data and info between address spaces
- maintain protection and isolation
- provide flexibility and performance

4.1 Message-passing IPC:

OS provides communication channel like a shared buffer
processes write(send) / read(recv) messages to/from channel(buffer.



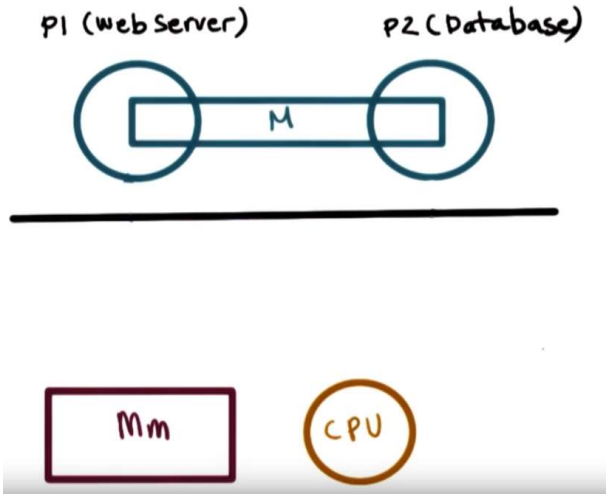
The good:

- The OS manages the channel/buffer
- Maintains isolation and protection

The bad

- Overhead message passing time
  - copy and load

4.2 Shared Memory IPC:



- OS establishes a shared channel and maps it into each process address space
- Processes directly read/write from shared channel
- OS is out of the way, so no overhead

The good

- OS is out of the way, no overhead

The bad

- No fixed and well-defined API how the shared channel should be used.
- Error prone, complexity in development.
- Mapping between memory/ opening the shared memory channel itself is still pretty expensive.

Trade-offs
It only makes sense to do shared memory-based communication if that cost, the setup cost, can be amortized across a sufficiently large number of messages.