Processes and Process Management

Key Questions

- what is a process?
- How are processes represented by OS's?
- How are multiple concurrent processes managed by OS's (esp on a single system?

Process Definition (simple)

- instance of an executing program
- synonymous with "task" or "job"

Visual Metaphor

A process is like an order of toys

State of execution
- completed toys, waiting to be built

Parts & temporary holding area
- plastic pieces, containers

May require special hardware
- sewing machine
- glue gun

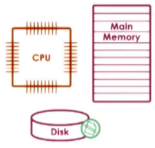
State of execution
- program counter, stack

Parts & temporary holding area
- data, register state occupies state in memory

May require special hardware
- I/O devices

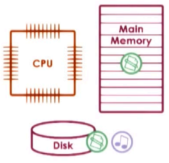
What is a Process?

OS manages hardware on behalf of applications

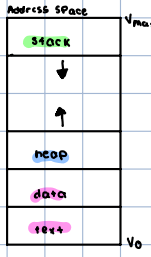


application == program on disk, flash memory... (static entity)

process == state of a program when executing loaded in memory (active entity)



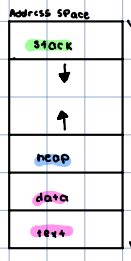
What does a process look like?



- a process encapsulates all of the state of a running application. This includes:
  - the code of the application
  - the data of the application
  - all the variables that the application needs to allocate
- every single element of the process state has to be uniquely identified by its address
- different types of processed state will appear in different regions in the address space

Types of State

- text and data
  - code
  - data available when a process is first initialized
  - static state when the process first loads
- heap
  - dynamically created during execution
- stack
  - grows and shrinks
    - ~LIFO queue



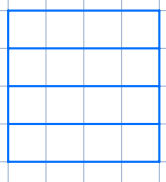
address space == "in memory" representation of a process

$V_{max} > x > V_0$ == virtual addresses used by process to reference some of the processes state

page tables == mapping of virtual to physical addresses

- we call the addresses virtual because they don't have to correspond to actual locations in the physical memory

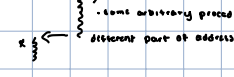
- by using this mapping technique we decouple the layout of the data in the virtual address space (complex, dependencies of application, tools we use) to physical memory



Physical Addresses
locations in memory

Physical (DRAM) memory

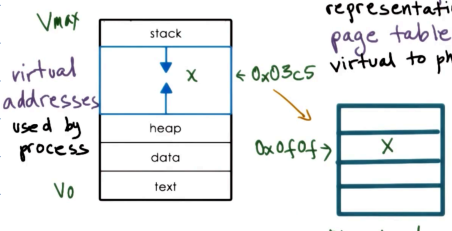
Consider we are executing a part of the process.



- now we need to call one procedure to jump to a different part of the address space
- place the original state before calling the other procedure on the stack memory, and then it is restored once we come back for the execution

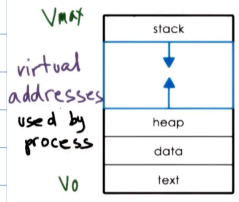
What does a Process look like?

address space == "in memory" representation of a process
page tables == mapping of virtual to physical addresses

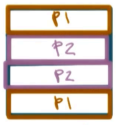


What does a Process look like?

- parts of virtual address space may not be allocated
- may not be enough physical memory for all state



32bits => 4GB!



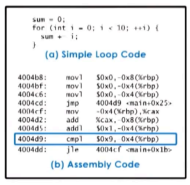
Physical Memory (DRAM)

physical addresses locations in physical memory

How does the OS know what a process is doing?

- Program Counter
- CPU registers
- Stack pointer
- ...

PC →



```
sum = 0;
for (int i = 0; i < 10; ++i) {
  sum += i;
}
```

(a) Simple Loop Code

```
4004b8:   movl   $0x0,-0x8(%rbp)
4004bf:   movl   $0x0,-0x4(%rbp)
4004c6:   movl   $0x0,-0x4(%rbp)
4004cd:   jmp    4004d9 <main+0x25>
4004cf:   mov    -0x4(%rbp),%eax
4004d2:   add    %eax,-0x8(%rbp)
4004d5:   addl   $0x1,-0x4(%rbp)
4004d9:   cmpl   $0x9,-0x4(%rbp)
4004dd:   jle    4004cf <main+0x1b>
```

(b) Assembly Code

⇒ Process Control Block (PCB)

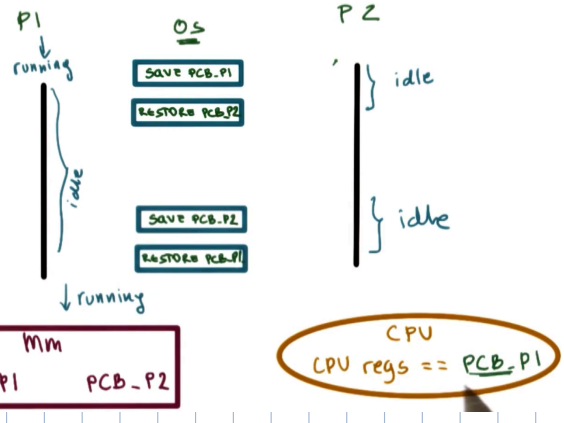
What is a Process Control Block (PCB)?

- dynamically updated

process state
process number
program counter
registers
memory limits
list of open files
priority
signal mask
CPU scheduling info
...

- PCB created when process is created
- Certain fields are updated when process state changes
- Other fields change too frequently

How is a PCB used?



- This Slide explains how an operating system(OS) keeps track of what a process is doing by maintaining Key information about the process in a data structure called the Process Control Block
- To manage processes effectively, the OS needs to know their current state. The following components are crucial:

1. Program Counter
   - the program holds the memory address of the next instruction that the CPU will execute
2. CPU Registers
   - Registers are small storage locations within the CPU that store temporary data needed for computation
   - This includes general-purpose registers, special-purpose registers, and others used during execution. The OS saves their values when switching between processes to ensure no data loss
3. Stack Pointer
   - The stack pointer tracks the top of the process's stack in memory. The stack is used for function calls, local variables, and return addresses
   - This allows processes to resume execution after an interrupt or context switch
4. Other Information
   - Additional details about a process, such as memory management information, open file descriptors, and scheduling priorities are also stored in the PCB

What is a Process Control Block(PCB)?

The PCB is a data structure maintained by the operating system for each process. It acts as a repository for all the information needed to manage and execute a process. When a process is running, its current state(e.g. program counter, register values, stack pointer) is stored in its PCB.

- When a context switch occurs(e.g. switching from one process to another), the OS saves the current current processes's state into its PCB and loads another from its PCB into CPU

- This image illustrates how the process control block is used during context switching in an OS

1. two processes
   - P1 is initially running while P2 is idle.
   - The CPU's registers match the state of PCB_1, meaning it is executing P1.
2. Context Switching
   - When the OS needs to switch from P1 to P2:
     - it saves the state of P1(e.g. program counter, CPU registers) into PCB_P1
     - it then restores the state of P2 from PCB_P2, loading its program counter, stack pointer, and registers into the CPU.
3. Result
   - After the switch, P2 is running, and its state is active in the CPU registers. Meanwhile, P1 becomes idle, with its state safely stored in PCB_P1.

This mechanism ensures that processes can pause and resume execution seamlessly.

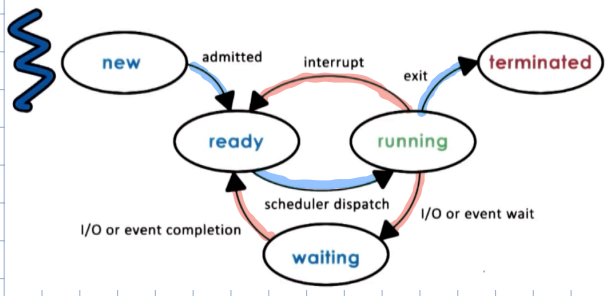
What are hot caches?

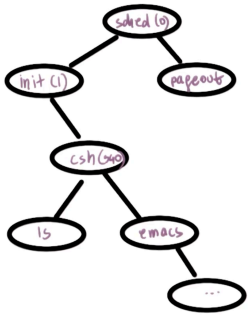
- a hot cache is where the cache has(small blocks of data stored in the cache) are populated with data or instructions that have been used frequently and are likely to be used again in the near future
- **Benefit**: when a threads frequently accessed data remains in the cache, it benefits from fast memory access, resulting in higher performance
- **Hot cache**: a cache populated with data that a thread has been actively using

Impact on context switching: when a context switch occurs, the cache might still be "hot" for the previous thread. If the new thread doesn't use the same data, it starts with a "cold" cache, leading to performance penalties until its own data is loaded

Process life cycle

Process Lifecycle





Process Creation

Mechanisms for process creation

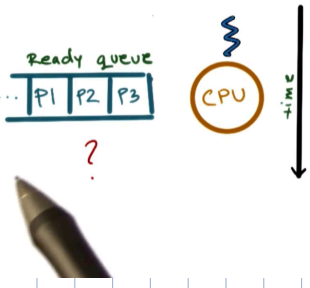
fork ==
- copies the parent PCB into new child PCB
- child continues execution at instruction after fork

EXEC ==
- replace child image
- load new program and start from first instr.

- fork() = system call to create a new process by duplicating the current one
- exec() = replaces the current process's code and data with a new program
- Every user-space process is a child of another process in this hierarchy

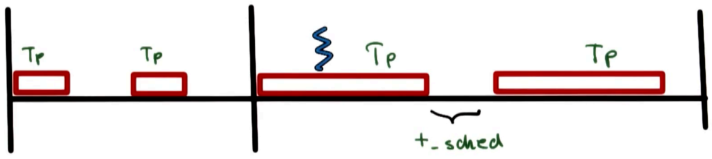
What is the role of the CPU scheduler?

A CPU scheduler determines which one of the currently ready processes will be dispatched to the CPU to start running, and how long it should run for.



OS must... BE EFFICIENT!
- preempt == interrupt and save current context
- schedule == run scheduler to choose next process
- dispatch == dispatch process & switch into its context

How long should a process run for? How frequently should we run the scheduler?



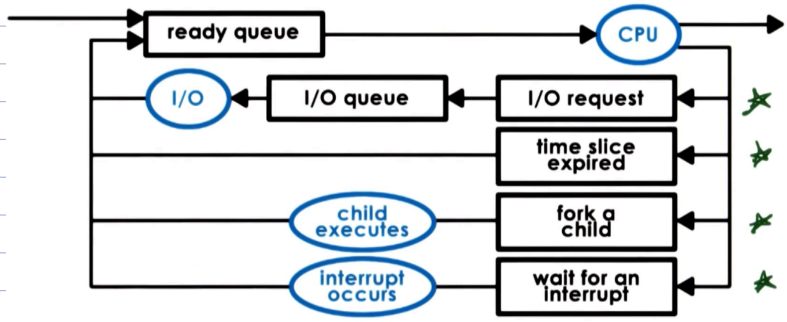
Useful CPU work:
== Total processing time / Total time =
(2 * Tp) / (2 * Tp + 2 * t_sched)

if Tp == 10 * t_sched => ~91% of CPU time spent on useful work!

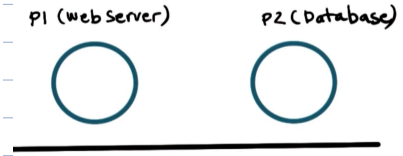
- time slice = time to allocated to a process on the CPU

Scheduling design decisions:
- what are appropriate timeslice values?
- metrics to choose next process to run?

What about I/O?

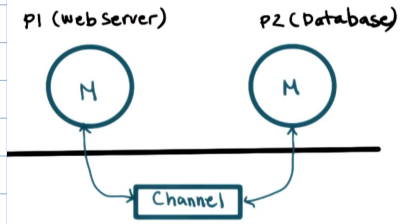


Can Processes Interact? (1)



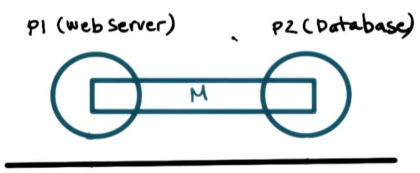
Inter-Process Communication IPC mechanisms:
- transfer data/info between address spaces
- maintain protection and isolation
- provide flexibility and performance

Can Processes Interact? (2)



Message-passing IPC:
- OS provides communication channel, like shared buffer
- Processes write (send) / read (recv) messages to/from channel

Can Processes Interact? (3)



Shared memory IPC:
- OS establishes a shared channel and maps it into each process adress space
- Processes directly read/write from this memory

+: - OS is out of the way!

-: (re-) implement code