Process

- OS abstraction to virtualize the CPU ; process is a program in execution
- Multiple process can be associated to the same program
- Multiple Process run on a system
  - OS / System Process
  - User Process
- the process is defined by
  - code (aka text)
  - program (static) data
  - program counter
  - content of all registers which include the Program Counter (PC)
  - heap
  - runtime stack
  - page table

Process Address Space

- defined by the code + data + heap + stack
- bounded by the OS configuration
- Heap
  - where the objects/structures/arrays are created
  - can be handled by memory manage but it is the OS that provides the dynamic memory allocation
- Runtime Stack
  - helps manage method/procedure/function calls and how to return from them
  - Items on the stack are pushed/popped in groups, or activation records, or frame
  - activation record contains all the bookkeeping necessary for placing and returning from function calls

Process Life Cycle

- a process can be in finite number of different states
- allowed transitions between some pairs of states
- transitions happen when some event occurs

Multi-Tasking (Multi Programming)

- multiple processes can co-exist in memory
- all processes have their own separate address space

CPU Virtualization

- context switching
  mechanism by which a running program is kicked off the CPU and another one which is done fast and really frequent

Process Control Block

- Process State
- Process ID (PID)
- User ID
- Saved Register Values
- CPU scheduling information
- Memory Management Information
- Accounting Information
- I/O Status Info

Process Table

- a list of PCB kept by the OS
- because the Kernel size is bounded so is the Process Table

Process API

Process Creation

- Process from a genealogy tree
- If process A creates process B implies that A is the parent of B and B is the child of A
- Process can have at most one parent
- Process can have many children
- Each Process has a PID $\in \mathbb{Z}$
  - picked by the OS and is increasing
  - PPID: PID of the parent of a process
- after creating the child the parent continues executing but at any point it can wait for the child's completion
- child can be
  - complete copy of the parent (have an exact copy of the address space)
  - be a new program

fork() System Call

- a system call that creates a new process
- child is almost exact copy of the parent except
  - PID
  - PPID
  - resource utilization
- after the call, the parent continues executing and the child begins executing
- fork() returns an integer value
  - fork() returns 0 to a child
  - fork() returns the child PID to the parent

Zombies

- a terminated child process remains as a zombie
- OS keeps Zombies for the following
  - Zombies do not use hardware resource
  - uses a slot in the Process Table
  - may fill up due to Zombies and cause fork() to fail
- A zombie lingers on until its parent has acknowledged its termination or parent dies
- Zombies is reaped by the OS
- frowned upon to leave zombies around unnecessarily

Process Termination

- process terminate itself with exit() which takes as argument an integer called the process exit/return/error/value/code
- process can terminate another process done using signals and kill() system call

Signals

- software interrupts
- OS defines a number of signals each with a name and a number
- signals happen for various reasons
  - invalid access to valid memory sends a SIGSEGV signal to the running process
  - SIGINT signal to the running program in the Shell
  - tying to access an invalid address sends a SIGBUS signal to the running process
  - can send SIGKILL to another process to kill it
- signals can be used for process synchronization

wait() and waitpid()

- wait()
  blocks until any child completes and returns the pid of completed child and child's exit code
- waitpid()
  blocks until a specific child completes; can be made nonblocking
- SIGCHILD
  when a child exits signal is sent to the parent
  - convenient way to avoid zombies
  - parent associates a handler to SIGCHILD
  - handler calls wait()
  - way all children terminations are acknowledged

Inter-Process Communication

Basics

- Independent Process
  each process runs code independently; both the parent and child are aware of each other but they do not interact
- Cooperate
  process needs to share information, speed up computation, and because it's convenient
- IPC
  means of communication between cooperating process
- All OSes provide system calls for Shared Memory
- All OSes provide system calls fro Message Passing

Remote Procedure Calls

- provides a procedure invocation abstraction across processes
- performed through a client stub with automatically generated code to
  - Marshal the param
  - Send the data over to the server
  - Wait for the server‘s answer
  - Unmarshal the returned values

Pipes

- powerful IPC abstraction provided by OSes
- an abstraction of an actual physical pipe with data that flows between two processes
- two ends: write and read end
- process can write a stream of bytes to the write end
- process can read from the read end
- process closes one of the ends the other will get some EOF notification

File Descriptors

- File Descriptor
  an integer associated with the open file
  file descriptor is the index in the array of process opened files
- Three Kind of File Descriptor
  - File descriptor 0: standard input (stdin)
  - File descriptor 1: standard output (stdout)
  - File descriptor 2: standard error (stderr)

OS Mechanisms

Limited Execution: Restricted Operations

- 2 kinds of instructions
  unprotected and protected

User Mode vs Kernel Mode

- User Mode
  protected instructions cannot be executed; where the user code is executed
- Kernel Mode
  where all instruction is executed and executes the kernel code
- Mode is indicated by status bit (mode bit) protected control register in the CPU
- Decode Stage
  If the instruction is protected and the mode bit is not set to "Kernel mode", abort and raise a trap (that the OS will answer by terminating the program saying something like "not allowed")
  else execute the instruction
- Protected Instructions
  - Updating the mode bit
  - halt the CPU
  - Update CPU control registers
  - Change the system clock
  - Read/Write I/O device control/status registers
  - general interact with hardware components

  all of these operations can happen in Kernel Mode and only the kernel code can use them

Trap Table

- stored in RAM
- CPU has a register that points to it
- For each event type that the CPU could receive, this table indicates the address in the kernel of the code that should be run to react to the event

Context Switching

- mechanism to kick a process off the CPU and give the CPU to another process
  - Save the context of the running process to the PCB in RAM
  - Make its state Ready
  - Restore from the PCB in RAM the context of a Ready process
  - Make its state Running
  - Restart its fetch-decode-execute cycle
- should be as fast as possible because it is pure overhead
- context switch is mechanism and deciding when to context switch is a policy which is called scheduling