What is Process

A process is a program in execution
– the unit of execution
– the unit of scheduling
– the dynamic (active) execution context

Process is often called a job or task

Program is static, just a bunch of bytes

Process vs Program

No one-to-one mapping between processes and programs
• can have multiple processes of the same program
• One process can invoke multiple programs
**What is in a Process**

A process consists of (at least):
- an address space
- the code for the running program
- the data for the running program
- an execution stack and stack pointer (SP)
  - traces state of procedure calls made
- the program counter (PC), indicating the next instruction
- general-purpose processor registers and their values
- a set of OS resources
  - open files, network connections, sound channels, ...

**A Process’s Address Space**

```
0xFFFFFFFF
0x00000000
```

<table>
<thead>
<tr>
<th>Stack (dynamic allocated mem)</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap (dynamic allocated mem)</td>
<td></td>
</tr>
<tr>
<td>Static data (data segment)</td>
<td></td>
</tr>
<tr>
<td>Code (text segment)</td>
<td></td>
</tr>
</tbody>
</table>

**Process States**

Each process has an **execution state**, which indicates what it is currently doing:
- new: being created
- ready: waiting to be assigned to CPU
  - could run, but another process has the CPU
- running: executing on the CPU
  - is the process that currently controls the CPU
  - how many processes can be running simultaneously?
- waiting: waiting for an event, e.g., I/O
  - cannot make progress until event happens
- terminated: finished execution

As a process executes, it moves from state to state.

**Diagram of Process States**

[Diagram showing process states and transitions]

new → admitted → running → waiting →... → terminated
Process Control Block (PCB)

PCB is a data structure in the kernel containing the information needed to manage a process

• The PCB is identified by an integer process ID (PID)
• also called task control block

PCB is the “the manifestation of a process in the kernel”

OS keeps all of a process’s hardware execution state in the PCB when the process isn’t running

Context Switch

• The act of switching the CPU from one process to another is called a context switch

• Context switch time is overhead, no useful work is done while switching
  – timesharing systems may do 100 or 1000 switches/sec.
  – takes about 5 microseconds on today’s hardware
Context-switch Implementation

Basic idea: save current context including registers to PCB in memory
- Tricky: must execute code without using registers
- Machine-dependent code

Requires special hardware support

State Queues

The OS maintains a collection of queues that represent the state of all processes in the system
- typically one queue for each state
  - e.g., ready, waiting, ...
- each PCB is queued onto a state queue according to the current state of the process it represents
- as a process changes state, its PCB is unlinked from one queue, and linked onto another

The PCBs are moved between queues, which are represented as linked lists.

Ready and Waiting (I/O) Queues

PCBs and State Queues

PCBs are data structures
- dynamically allocated inside OS memory

When a process is created:
- OS allocates a PCB for it
- OS initializes PCB
- OS puts PCB on the correct queue

As a process computes:
- OS moves its PCB from queue to queue

When a process is terminated:
- PCB may hang around for a while (exit code...)
- eventually, OS deallocates its PCB
Process Creation

- Four major events that cause processes to be created
  - System initialization
  - Execution of a process creation system call by a running process
  - A user request to create a new process
  - Initiation of a batch job

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
  - Two ways to create a process
    - Build a new empty process from scratch
    - Copy an existing process and change it appropriately

Process Creation

Option 1: New process from scratch
- steps
  - Load specific code and data into memory; create empty call stack
  - Create and initialize PCB
  - Put process on ready queue
- pros: no wasted work
- cons: difficult to setup process correctly
  - process permissions, environment variables

Process Creation

Option 2: clone existing process and change
- Example: Unix fork() and exec()
  - fork(): clones calling (parent) process
  - exec(): replace the process’s address space with a new program
- Pros: flexible, clean, simple
- Cons: overhead in copy and then overwrite of memory
Unix process creation

- **fork()** system call
  - creates and initializes a new PCB
  - creates a new address space
  - initializes new address space with a copy of the entire contents of the address space of the parent
  - initializes kernel resources of new process with resources of parent (e.g., open files)
  - places new PCB on the ready queue

- the **fork()** system call "returns twice"
  - once into the parent, and once into the child
  - returns the child’s PID to the parent
  - returns 0 to the child

- **fork()** = "clone me"

exec() vs. fork()

- So how do we start a new program, instead of just forking the old program?
  - the **exec()** system call!
  - int exec(char * prog, char * argv[])

- **exec()**
  - stops the current process
  - loads program ‘prog’ into the address space
  - initializes hardware context, args for new program
  - places PCB onto ready queue
  - note: does not create a new process!

Tree of Processes on Solaris

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process’ resources are deallocated by operating system

- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates