CSCI 5103  
Operating Systems  
Instructor: Abhishek Chandra

Outline
- Processes:
  - Process Concept
  - Process States and Representation
  - Process Scheduling - Basics
  - Process Creation and Termination
  - IPC

What is a Process?
- Process is a program in execution
- Basic unit of work (activity)
- Enables multiprogramming
- Typically also provides a protection boundary
- Also referred to as “job” or “task”

Program vs. Process
- Program:
  - Passive entity
  - Set of instructions
  - A binary file
- Process:
  - Active entity
  - Executing path of instructions
  - Live set of resources (CPU cycles, memory, files)
- Multiple processes can correspond to the same program
### Process Memory Layout

<table>
<thead>
<tr>
<th>High Address</th>
<th>Stack</th>
<th>Activation Records (function params, local vars, saved registers)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low Address</td>
<td>Heap</td>
<td>Dynamic Memory</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>Static Data</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Binary Code</td>
</tr>
</tbody>
</table>

### Process Life Cycle

- **new**
- **ready**
- **running**
- **waiting**
- **scheduled**
- **interrupted**
- **admitted**
- **done**
- **terminated**
- **I/O or event completion**
- **wait for I/O or event**

### Process Control Block (PCB)

- Kernel structure maintaining info about a process
- Each process has a unique PCB
- Info used by OS to schedule process, find its contents, execute it

### What info does a PCB contain?

- Process state
- Registers. E.g., Program counter, Stack pointer
- CPU scheduling information. E.g.: priority, pointers to scheduling queues
- Memory-management information. E.g.: page table info
- I/O status information:
  - Files, sockets, I/O devices, locks
- Accounting information: E.g., runtime
Example PCB: Linux

```c
struct task_struct {
    pid_t pid;
    long state; /* runnable, stopped, etc. */
    struct sched_entity se; /* scheduling info */
    struct task_struct *parent;
    struct list_head children;
    struct mm_struct *mm; /* Mem mgmt info */
    struct files_struct *files; /* open files */
    struct timespec start_time;
    ...
}
```

Process Queues

- Run queue: Processes ready to run
- Wait queues: Processes waiting on an event
  - I/O completion on a device
  - Lock, alarm
  - Each device/event object would have a wait queue
- Processes move between queues through their lifetime
  - State of process is updated appropriately

Process Scheduling

- OS runs one process on a CPU at a time
- Selects a process to run when:
  - Quantum expires
  - Running process gets blocked or exits
  - A higher priority process becomes runnable

Context Switch

- Switching between two processes
  - Save state of one process and restore state of another process
- What does a process context contain?
- Context switch overhead depends on:
  - Architecture support
  - Amount of information needed by OS
- How to decide on the quantum length?
Process Creation

- A process can create a child process
- Main issues:
  - What should the new process inherit from the parent process?
  - How to execute a program in the child process?
  - Can the parent and child run concurrently?

Process Creation: UNIX

- Uses two syscalls: fork and exec
- fork(): creates new process
- Child inherits a copy of parent’s execution state
  - Code, data, open files, program counter, stack
- Two clones exist immediately after fork

Process Creation: fork

<table>
<thead>
<tr>
<th>Stack</th>
<th>Data</th>
<th>Program Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heap</td>
<td></td>
</tr>
</tbody>
</table>

parent

fork

child

<table>
<thead>
<tr>
<th>Stack</th>
<th>Data</th>
<th>Program Text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heap</td>
<td></td>
</tr>
</tbody>
</table>

Process Creation: exec

- exec starts execution of new program
- Overwrites the memory image inherited from the parent with a new state:
  - New code, program counter, data, stack
- Retains some things from before exec:
  - Open files
**Process Creation: Windows**

- CreateProcess(): creates a new process
- Child executes a new program
- Child has its own state

**Process Termination**

- Normal termination: Process exits when it’s done
- Abnormal: killed by another process, fault/error
- For resource reclamation: E.g., on Android
- What happens to parent/children?
  - Parent can wait for child’s status. E.g.: UNIX wait()
  - Children can be terminated. E.g.: VMS cascading termination
  - Children can be adopted. E.g.: UNIX adoption by init process

**Interprocess Communication**

- Multiple processes may want to communicate
  - Examples?
- IPC mechanisms provided by OS to allow communication/data sharing

**Message-Passing**

- Sender and receiver processes
  - Explicitly send and receive messages
  - OS provides mechanism to deliver messages
- Issues:
  - Naming: Direct addressing (Process ID) vs. indirect addressing (mailbox)
  - Synchronization: Blocking vs. non-blocking
  - Buffering: Zero capacity, bounded or unbounded
- Example: MACH ports and messages
**Shared Memory**

- Two processes explicitly share part of their memory
  - One process creates a shared region
  - Other process attaches it in its memory space
- Communication: Read/write to shared memory
  - OS only facilitates creation/attaching of shared spaces
- Example: POSIX Shared Memory

**Message Passing vs. Shared Memory**

**Other IPC Mechanisms**

- Pipes and FIFOs:
  - Provides unidirectional flow of data
  - Uses kernel buffer
- Sockets:
  - Used for internetworking
- Remote Procedure Calls (RPCs):
  - Higher level client-server programming abstraction
  - Uses interfaces and code stubs