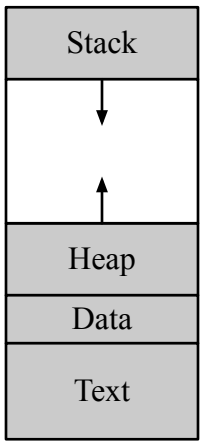
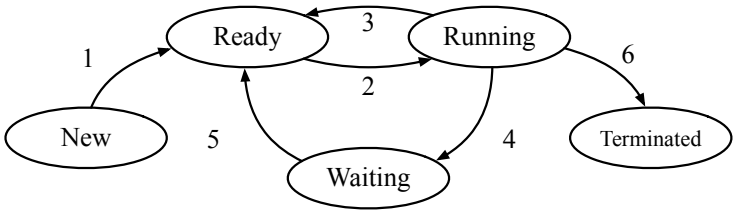
Lecture Notes 3

Processes

- Process Concept – Program in execution
  - Process, User Program, Task, or Job
    - Text Section – The instructions of process
    - Program Counter – The current CPU instruction
    - Stack – Automatically allocated variables "Local Variables"
    - Data Section – "Global Variables" and constants
    - Heap – Dynamically allocated variables



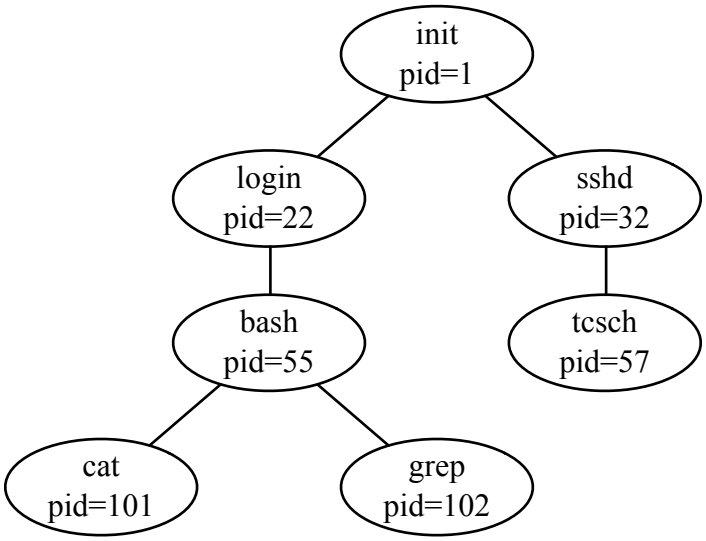
  - Executable File – Stored Program on Disk
  - Process State
    - New – Process being created
    - Running – Process is executing instructions
    - Waiting – Process is blocked waiting for I/O, signal, or a resource
    - Ready – Process is able to run when assigned a CPU
    - Terminated – Process has finished executing



1. Process created
2. Scheduler selects process
3. Scheduler selects another process
4. Process blocks for I/O
5. I/O completes
6. Process terminates

- Process Control Block (PCB) or Task Control Block – Maintains information about the process
  - Process State – Running, Ready, etc.
  - CPU Registers – Program counter, stack pointer, general purpose, etc.
  - Scheduling Information – Process priority, pointers to queues, etc.
  - Memory Management Information – Page/Segment table information, and/or base/limit values
  - Accounting Information – Read time used, process identification, etc.
  - I/O Status Information – Open file list, I/O status, etc.
- Threads – Sequence of programmed instructions managed by scheduler
  - Heavy Weight Threads – Kernel space, visible to the OS, can concurrently executing on independent cores
  - Light Weight Threads – User space only, scheduled by application, all share a core
  - Fibers – Light Weight Threads that are cooperatively scheduled (non-preemptive threads)

- Process Scheduling
  - Process Scheduler – Selects for process to execute on CPU
  - Scheduling Queues – Queues that maintain processes in the system
    - Job Queue – All processes that have entered the system
    - Ready Queue – All processes ready to execute, but waiting for CPU time
    - Device Queue – All processes waiting for a particular I/O device
  - Schedulers – Decides what to execute and when
    - Long-term Scheduler (or Job Scheduler) – Selects from pool to load processes into memory
      - Degree of Multiprogramming – The number of processes in memory
      - Controls mix of I/O-bound and CPU-bound processes
    - Short-term Scheduler (or CPU Scheduler) – Selects from ready processes to execute on the CPU
    - Medium-term Scheduler – Controls swapping to control Degree of Multiprogramming
  - Context Switch – Switching between contexts
    - Context – Current state of the CPU and process
    - State Save – Storing the context so that it can be restored later
    - State Restore – Restoring the saved context so that it can begin executing again
    - Stack Switch – Common implementation of context switch
- Operations on Processes
  - Process Tree – A tree structure of all processes with connections between Parent/Children



- Process Creation
  - Process Identifier (PID) – The unique identifier for a process
  - fork(), exec() – Typical combination of system calls to create a process (*nix OSes)
  - CreateProcess() – Windows system call to create a process
- Process Termination
  - exit(), kill() – Typical system calls to terminate a process either normally with exit or forced/signaled through kill (*nix OSes)
  - TerminateProcess() – Windws system call to terminate a process
  - Cascading Termination – Initiated at root of process tree to terminate all processes
  - Zombie Process – Child process that has terminated, but parent has not called wait
  - Orphan Process – Child process whose parent has terminated
- Interprocess Communication (IPC)
  - Independent Process – Is not affected nor affects other processes
  - Cooperating Process – Affects or is affected by other processes
  - Need for IPC
    - Information Sharing – Ability to share or concurrently access same information
    - Computation Speedup – Ability to execute a task in parallel subtasks
    - Modularity – Break up task into modular chunks
    - Convenience – Easier to move information between tasks
  - IPC Problems
    - Producer Consumer – Producer produces items that the consumer consumes
    - Bounded vs. Unbounded Buffer – Typically the producer consumer has a bounded buffer of a finite size
  - Shared Memory Systems

        - shm_open(), shm_unlink(), mmap(), munmap(), ftruncate() – POSIX compatible shared memory system calls
    - Message Passing Systems
        - Send/Receive – Message passing interface typically uses send and receive as the functions for sending and receiving a message
        - Direct Communication – Must explicitly state to process on send and from process on receive
        - Indirect Communication – Messages are sent to mailboxes or ports
        - Blocking or Synchronous – Blocks until the action has completed
            - Blocking Send – Sending process blocks until message is received
            - Blocking Receive – Receiver blocks until message arrives
        - Non-blocking or Asynchronous – Process continues if the communication cannot complete
            - Non-blocking Send – Sending process sends message and resumes operation
            - Non-blocking Receive – Receiver either retrieves message or detects no message available
        - Rendezvous – Both blocking send and receive are being used
        - msgget(), msgctl(), msgsnd(), msgrcv() – System V message queue system calls
        - mq_open(), mq_unlink(), mq_send(), mq_receive(), mq_notify(), mq_close() – POSIX message queue system calls
- Client-Server Systems
    - Sockets – Used in network programming usually built upon Internet Protocol (IP)
        - Transmission Control Protocol (TCP) – Connection oriented sockets protocol
        - User Datagram Protocol (UDP) – Connectionless sockets protocol
        - Loopback – The local connection so that sockets can be used for local IPC
    - Remote Procedure Calls – Ability of sever to execute a function as if it were executed on the local machine
    - Pipes – FIFO allowing processes to communicate
        - Ordinary Pipes (or Anonymous Pipes) – Created and are shared typically between parent/child or between children of a process
        - pipe() – *nix system call to create an ordinary pipe
        - CreatePipe() – Windows system call to create an ordinary pipe
        - Named Pipes – FIFOs that are created in the file system
        - mkfifo() – *nix system call to create a named pipe
        - CreateNamedPipe() – Windows system call to create a named pipe