Shells and Processes

Bryce Boe

2012/08/08

CS32, Summer 2012 B

Outline

- Operating Systems and Linux Review
- Shells
- Project 1 Part 1 Overview
- Processes
- Overview for Monday (Sorting Presentations)

OS Review

- Operating systems
  - Manages system resources: cpu, memory, I/O
  - Types: single/multi-user and single/multi-process
  - Provides fairness, security

Self Check Questions

- What is the primary benefit of a multi-process OS over a single process OS? How is this accomplished?
- Explain the difference between multiprogramming and multitasking

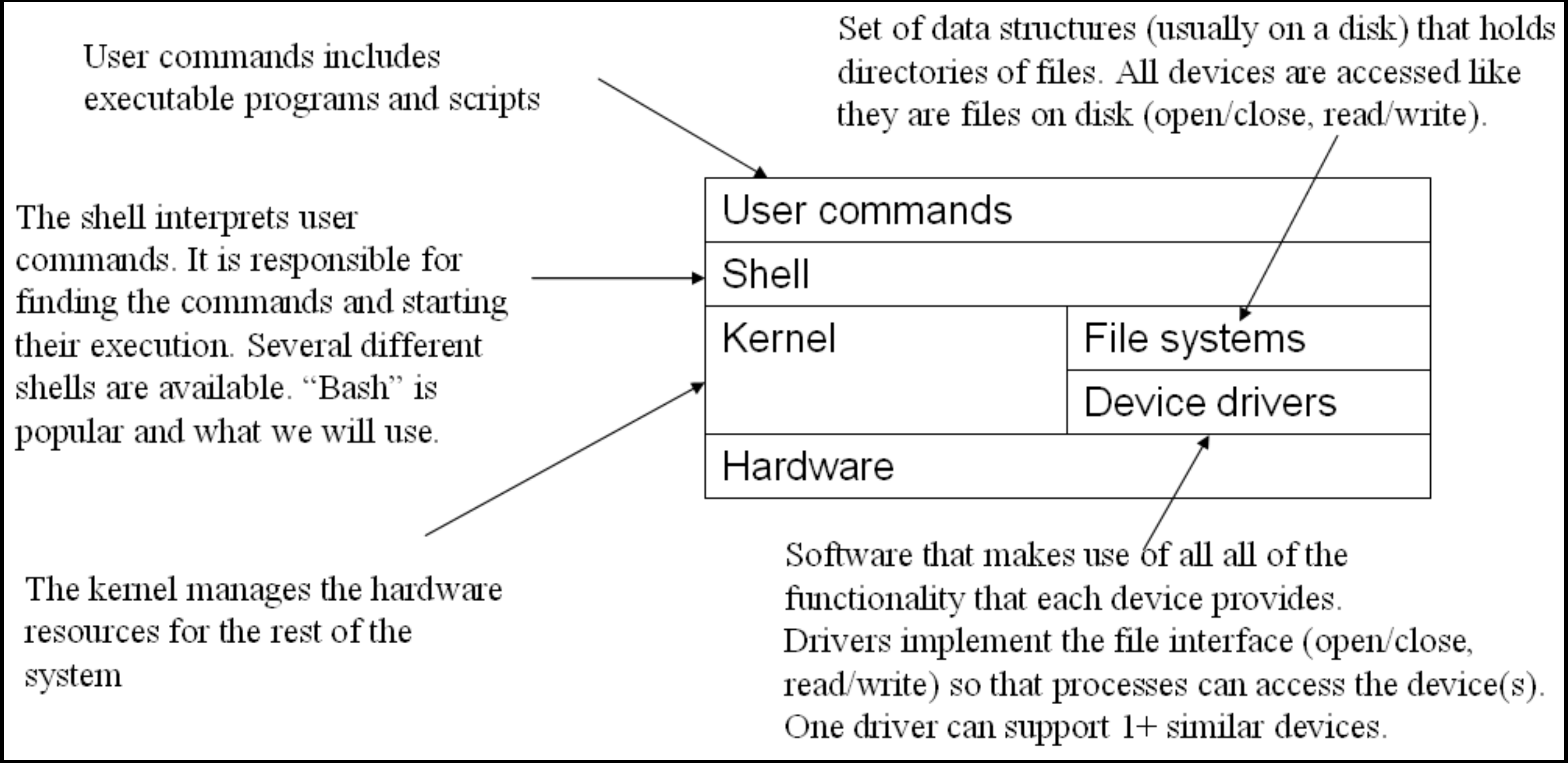
Self Check Answers

- What is the primary benefit of a multi-process OS over a single process OS? How is this accomplished?
  - Increased resource utilization (primarily of the CPU) accomplished by scheduling other processes when the currently running process requires I/O

Self Check Answers cont.

- Explain the difference between multiprogramming and multitasking
  - Multiprogramming systems switch the running process when that process requires I/O.
  - Multitasking systems periodically switch the running process after some (typically minute) period of time

Linux Architecture



User commands includes executable programs and scripts

The shell interprets user commands. It is responsible for finding the commands and starting their execution. Several different shells are available. "Bash" is popular and what we will use.

The kernel manages the hardware resources for the rest of the system

Set of data structures (usually on a disk) that holds directories of files. All devices are accessed like they are files on disk (open/close, read/write).

Software that makes use of all all of the functionality that each device provides. Drivers implement the file interface (open/close, read/write) so that processes can access the device(s). One driver can support 1+ similar devices.

Shells

What is a shell?

- A shell is a program that provides the interface between the user and the operating system
- Can be used to tell the OS to:
  - Execute programs (as processes)
  - Stop, or pause processes
  - Create, copy, move, remove files
  - Load or unload device drivers

Types of Shells

- Command line shells:
  - Provide a textual input as the user-interface
  - Bourne shell (sh), C shell (csh), Bourne-Again shell (bash), cmd.exe
- Graphical shells
  - Provide a point-and-click type interface
  - Windows shell, Gnome, KDE, Xfce, Xmonad

Login Shell

- The shell presented to the user upon login
- Typically changeable on Linux via `chsh`

Configuration Files

- Unix shells typically load configuration settings on launch
  - Bourne shell: ~/.profile
  - C shell: ~/.login, ~/.cshrc
  - Bash: ~/.bashrc, ~/.bash_profile
- Useful to adjust environment variables such as the PATH
  - Examples are provided in the reader on page 29 and 30

Unix Shells

- Contain built-in commands
  - cd, eval, exec, exit, pwd, test, umask, unset
- Launch external programs
  - cat, cp, mv, touch, wc
- Continue executing until their input stream is closed via <ctrl+d>
- External commands are searched for according to the PATH environment variable

Launching shells

- Shells can be launched within shells (they're just applications after-all)
  - Demo pstree with nested shells
- Shells process commands from stdin
  - Run: echo "echo foo" | sh
  - Combined with stdin redirection we have the ability to write shell scripts
  - More on shell scripts in lab1 and project 1

Working with the PATH

- The PATH environment variable specifies directories containing executable file
- Commands to demo:
  - echo $PATH
  - which -a <PROG_NAME>
- Bad things can happen with '.' is on the PATH
  - Shell script wrapper program

Shell meta characters

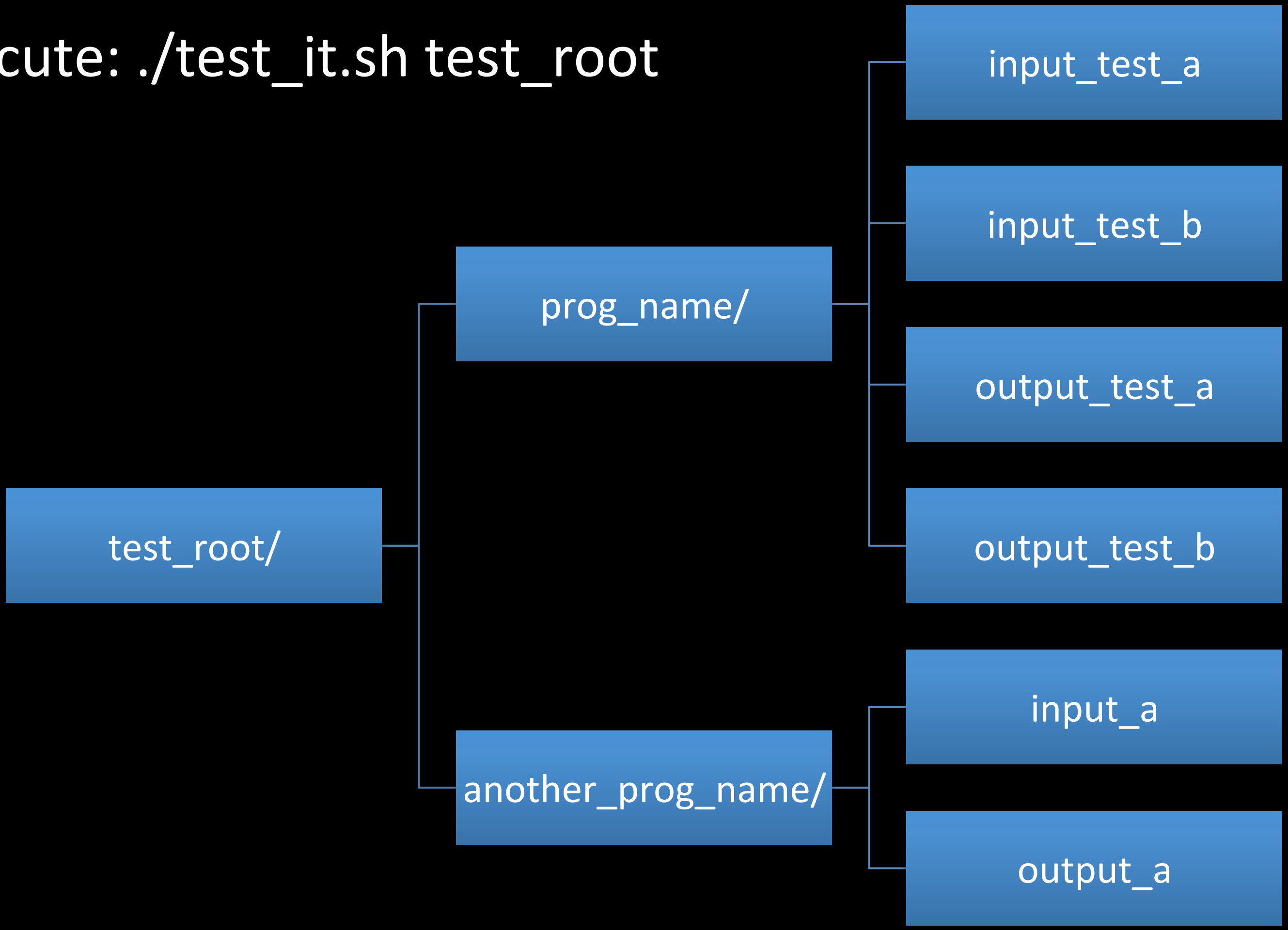
- Support for globbing
  - Filename expansion using:
    - * - wildcard to match 0 or more characters
    - ? – wildcard to match exactly one character
    - [ ] – matches one character if it’s contained in the character list
      - [0-9A-Za-z] will match a single character if it’s alphanumeric
- Home directory substitution via ~

Project 1 Part 1

- Automated testing bourne script
  - Usage: test_it.sh DIRECTORY
- Given a directory as input run tests against programs specified by DIRECTORY's subdirectory names
  - Individual test inputs are files prefixed with "input_" and should be compared with the corresponding "output_" file

DIRECTORY Hierarchy

Execute: ./test_it.sh test_root

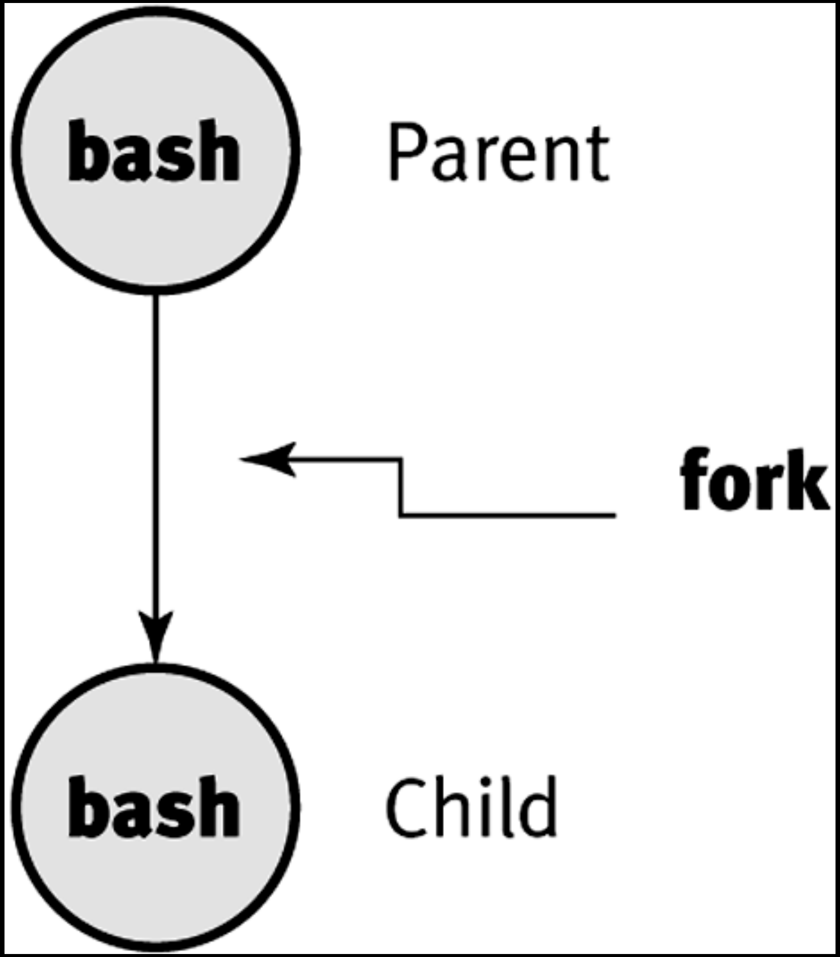


Project 1 Part 1 Demo

Processes (in Linux)

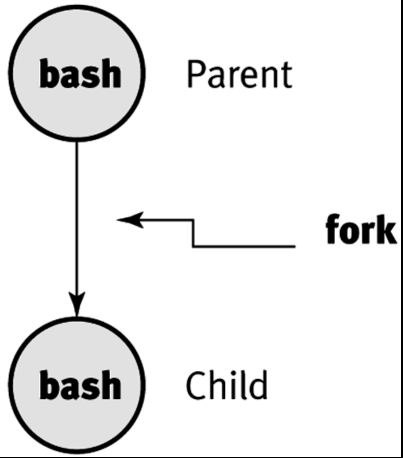
- A process is a program in execution
  - Copied to memory and assigned a process ID (PID)
- Multiple processes run *simultaneously* via multitasking
- Processes are created when an existing process makes a fork or clone system call
- Processes can have different scheduling priority (nice values in Linux)

Simple Fork Example

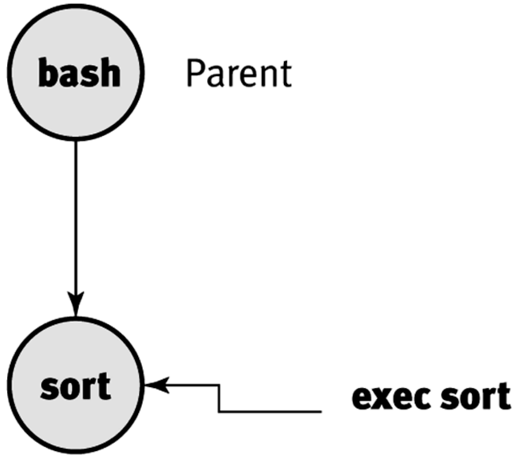


Running sort from bash

Step 1: Shell uses **fork** to create a child



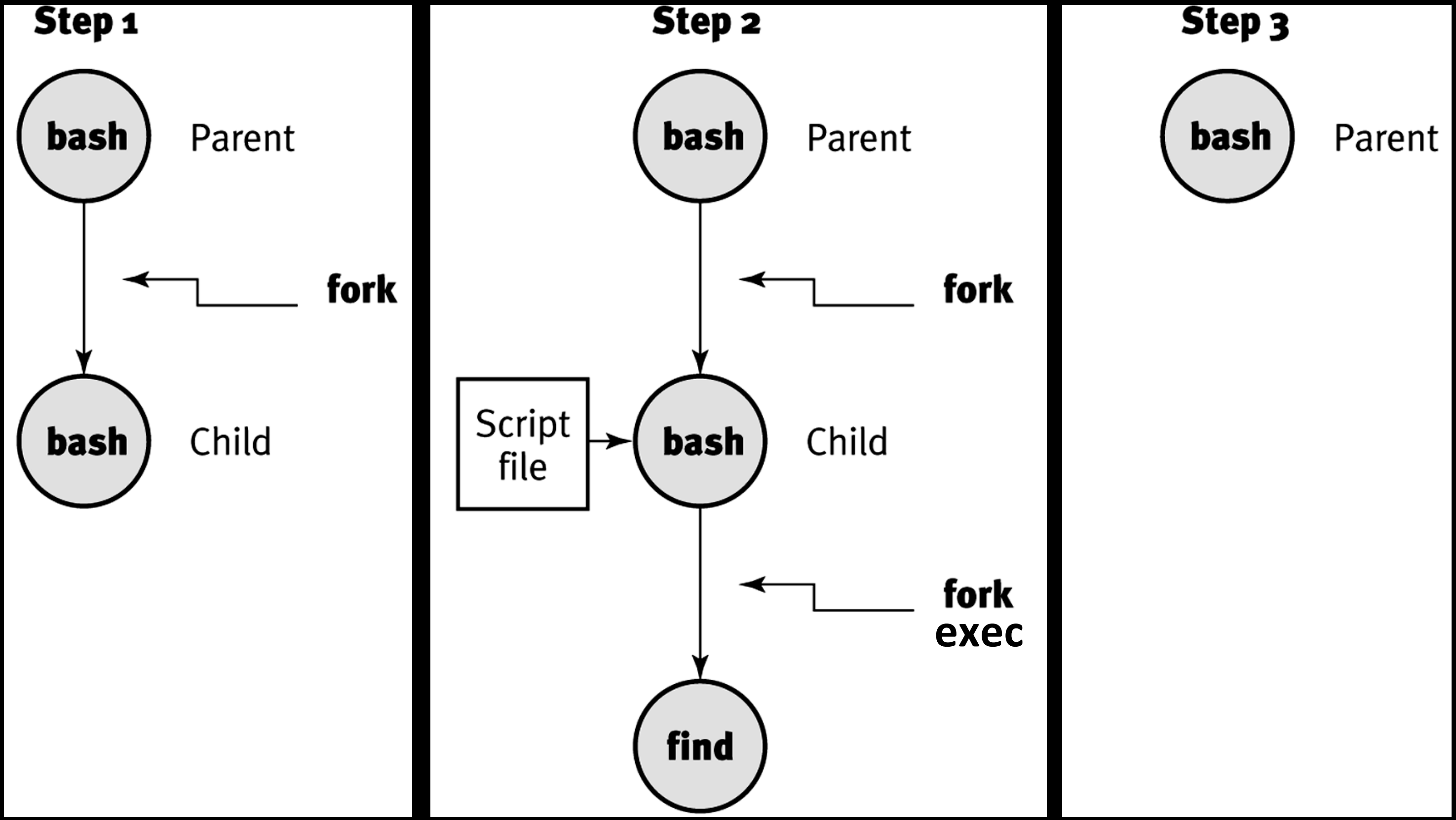
Step 2: Child uses **exec** to overwrite itself with the executable file corresponding to the **sort** command.



Step 3: **sort** starts execution while 'bash' waits for the command to finish. When **sort** finishes, the child process terminates and 'bash' starts execution again, waiting for the user to give it another command to execute.

bash
Parent

Running a shell script that runs find



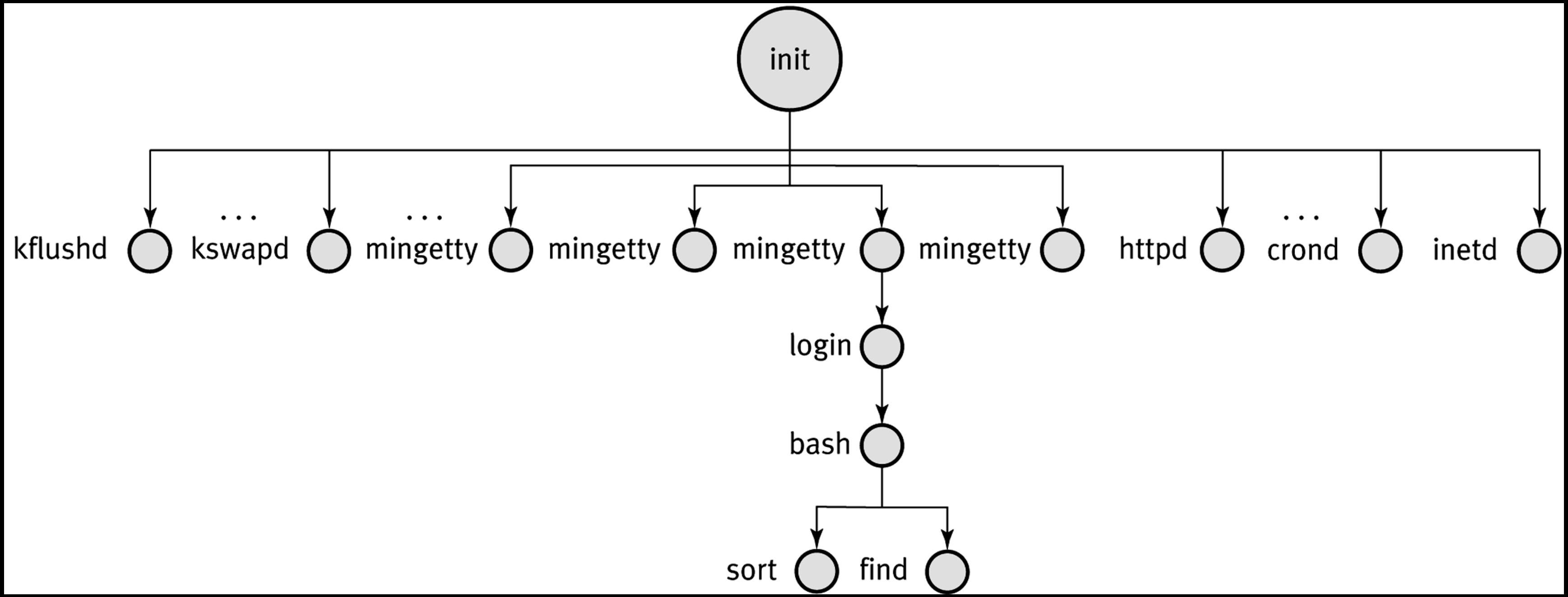
Select Process Attributes

- The column names as listed in `ps -l` output
- S – the state of the process
- PID – the process id
- PPID – the parent process id
- UID – process owner's user id
- WCHAN – the event a non-running process is waiting for

Process Hierarchy

- init (now systemd) is the root of all processes (PID 1)
- The process hierarchy's depth is limited only by available virtual memory
- A process may control the execution of any of its descendants
  - Can suspend or resume it
  - Can even terminate it completely
- By default, terminating a process will terminate all of its descendants too
  - So terminating the root process will terminate the session

Example Process Hierarchy

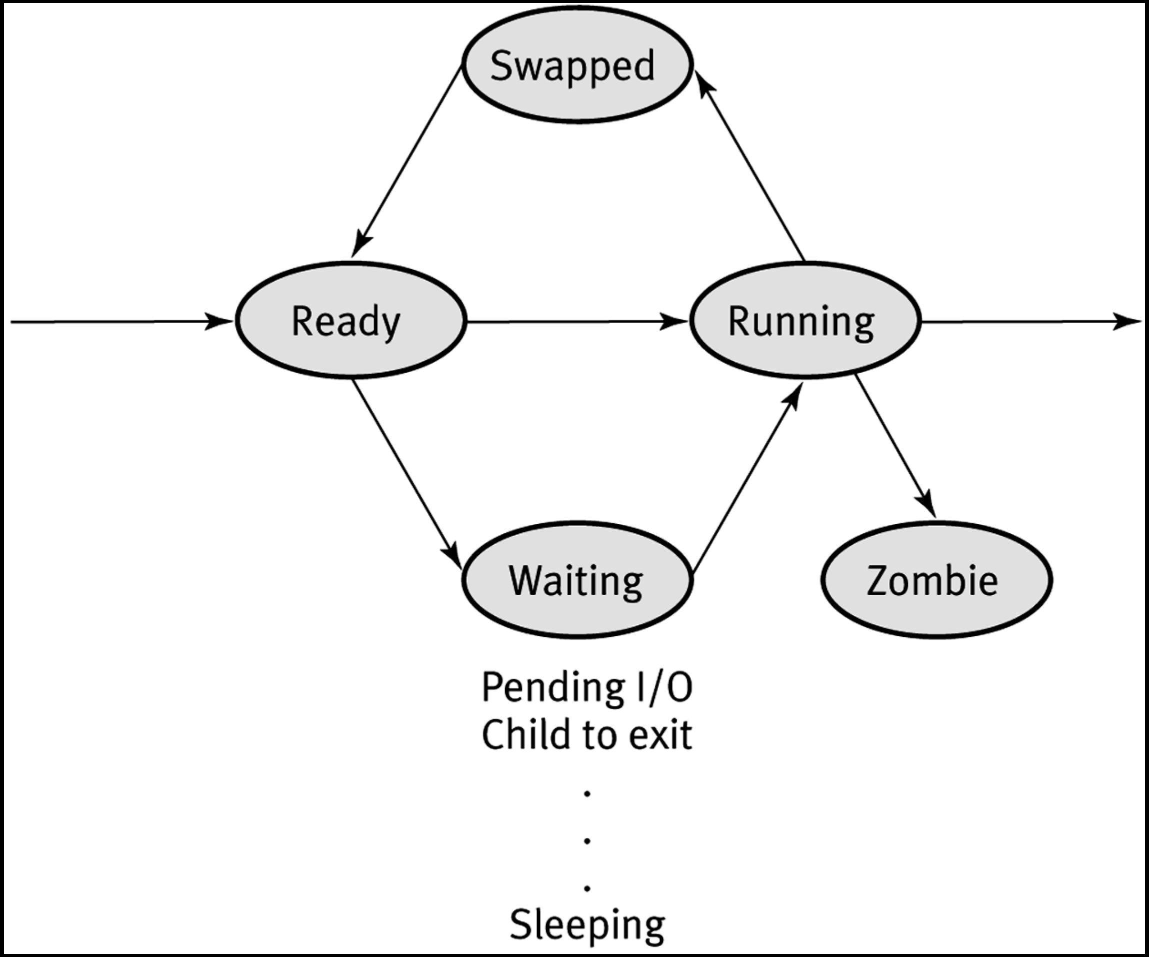


Process States

- A process exist in a number of different states

- Ready
  - The process is ready to be scheduled
- Running
  - The process is currently runny
- Swapped
  - Part or all of the process's memory is on disk
- Zombie
  - The parent of the process no longer exists

Process States Diagram



Observing Process States and Hierarchy

- ps
  - Output a snapshot of the running process (many options)
- pstree
  - Output a text-based view of the process hierarchy tree
- top
  - A terminal-based process monitoring program

Process Exit Status

- Each process exits with some status 0-255
  - 0 is typically used to indicate success
  - All other numbers are used to indicate some “error” condition that is application specific
  - In C/C++ the int return value from the main is the exit status

Processes and the shell

- The shell can run processes in the foreground (fg) and the background (bg)
- Multiple processes can be run in succession or in parallel via a single command

Foreground and background

- The shell normally runs processes in the foreground
- Launch a process in the background via &
  - sleep 500 &
- See a list of background processes (jobs) associated with your current shell via
  - jobs

Background -> foreground

- Type: fg (note there must be a background processes running)
- You can also explicitly foreground a specific job by number:
  - fg %3

Foreground -> background

- When a process is running, suspend it:
  - <ctrl> + z
  - This will bring you back to the terminal
- Then run bg to resume the process running in the background
- As with the fg command, you can provide an explicit job number:
  - bg %2

Sequentially executing programs

- Separate via ; on the command line
  - sleep 5; ls; sleep 5; ls
  - Processes run regardless of previous process's exit status
- Conditionally execute sequentially based on exit status: separate via &&
  - sleep 5 && ls -l foo && sleep 5 && ls –l
  - Command stops when a non-zero exit status is returned

Executing programs in parallel

- Separate via &, the background process indicator
  - echo foo & echo bar & echo somethingelse &
  - If process is running in the background, the command's exit status will be zero

Mix and match

- sleep 5; echo foo& echo bar & ; ls
  - sleeps 5 seconds
  - Concurrently runs echo foo, echo bar and ls
    - Both echo commands run in the background
    - ls runs in the foreground

For Monday

- Prepare 10-15 minute presentation on an assigned sorting algorithm
  - Provide a number of examples and detail and possible *corner* cases
- 1 volunteer will be asked to present each sorting algorithm
  - If no volunteers, then someone will be picked randomly
- Complete instructions will be posted on Piazza sometime before Thursday's lab