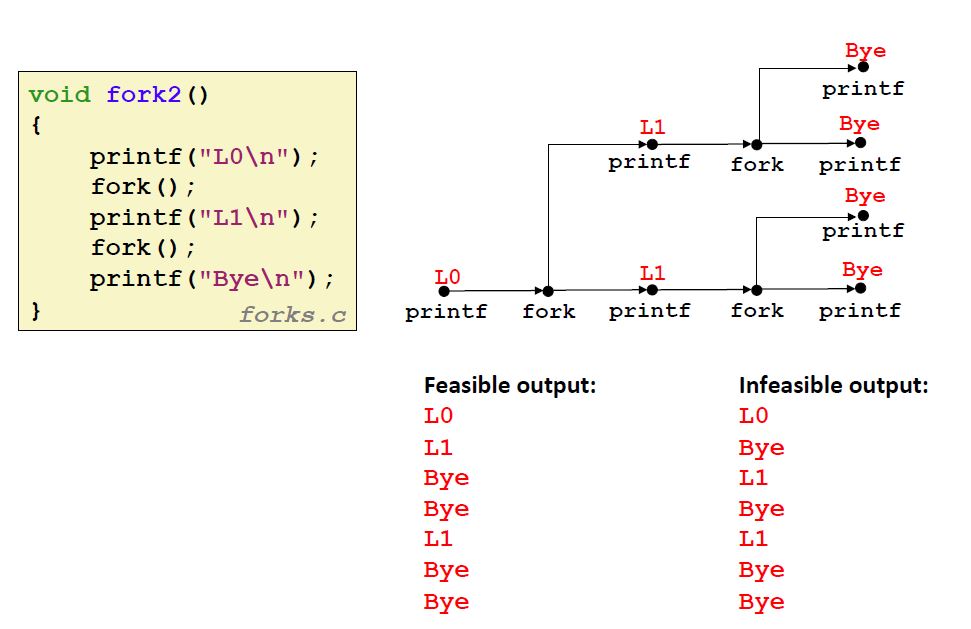
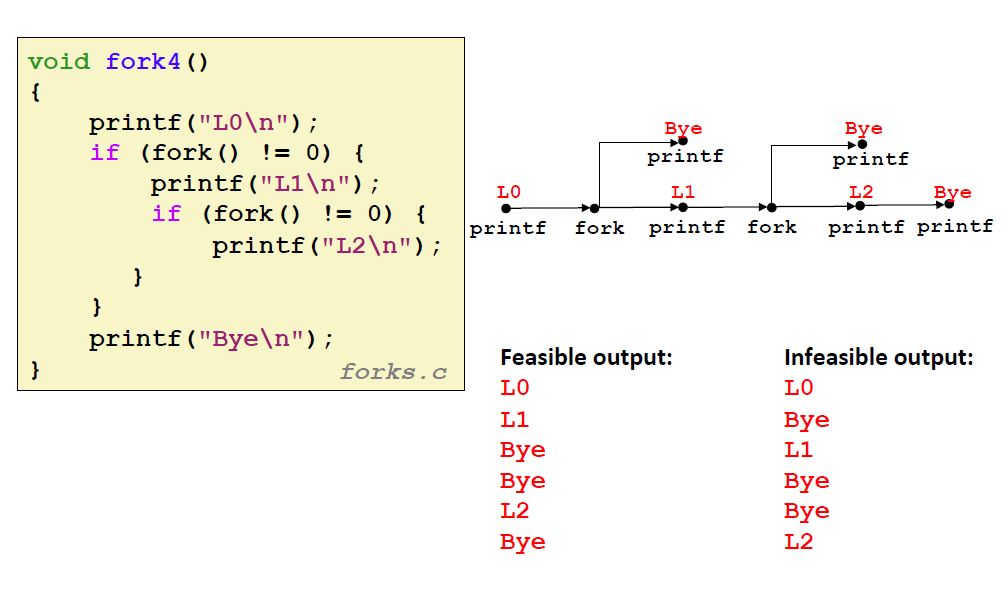
* To obtain a process ID
  + pid\_t getpid (void) - return PID of current process
  + pid\_t getppid(void) - return PID of parent process
* A process can be under three states
  + Running - a process is either executing or waiting to be executed
  + Stopped - a process executing is suspended and will be scheduled later
  + Terminated - a process is stopped permanently
* Terminating process
  + Receiving a signal whose default action is to terminate
  + Returning from the main routine
  + Calling the exit function
* Creating processes
  + Parent process creates a new running child process by calling fork
  + Int (fork) - returns 0 to the child process, returns the child’s PID to the parent process
* Process Graph
  + It is a useful tool for capturing the partial ordering of statements in a concurrent program.
  + Two consecutive forks



* + Nested Forks



* Reaping child process
  + When a process is terminated, it will not entirely be removed from the system. It is called a child zombie.
  + The parent will reap the child zombie by a statement wait or waitpid. Then the kernel deletes the zombie child process
  + If a parent terminates without reaping its child, then the orphaned child will be reaped by the **imit** process.
  + Wait
    - Parent reaps a child by calling the wait function
    - Int wait (int \*child\_status)
    - To get info on the exit status we can use macros like WIFEXCITED and WEXITSTATUS
    - Pid\_t waitpid(pid\_t pid, int &status, int options)
* Loading and running process
  + A fork() will create a child process which works in the same program
  + If to run a different program inside a process we use
  + Int execve(char \*filename, char \*argv[], char \*envp[])
  + Execve is called once and never returns - except if there is an error. If error then it will return -1