CSCI 460 Operating Systems

Processes (Part II)

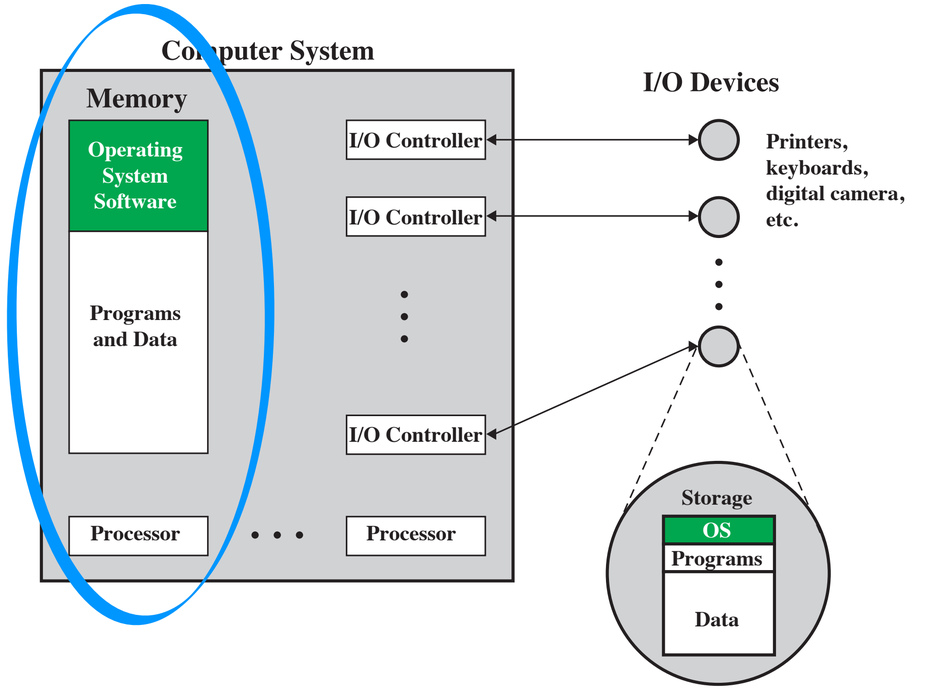
Professor Travis Peters
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

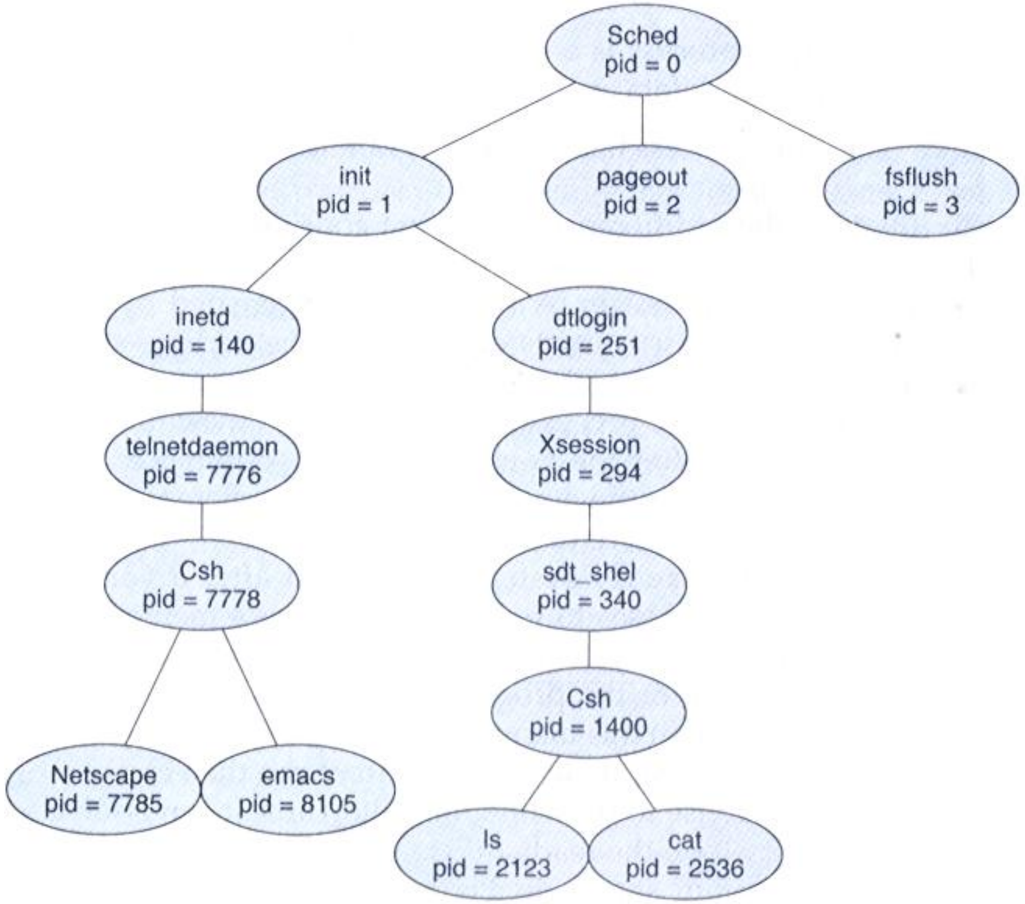
Goals for Today

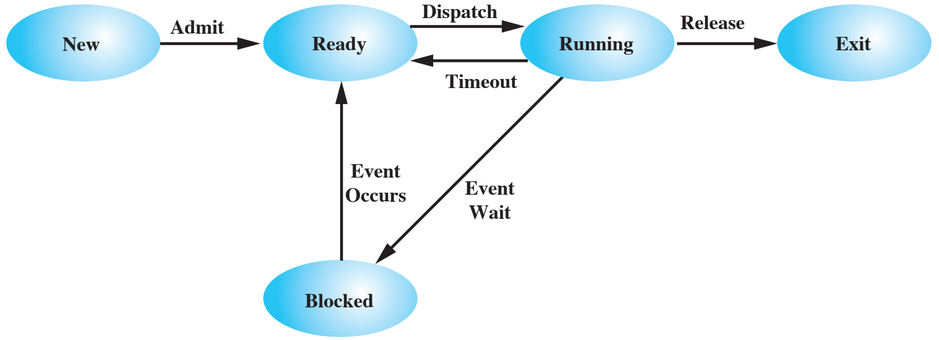
- **Learning Objectives**
  - Understand basic concept of process (control info, creation, termination, states, etc.)
  - Review some important UNIX syscalls and concepts for sys. programming
- **Announcements**
  - Grades should be fixed…
  - Rough schedule posted today
  - *Coming Soon…*
    - zyBook for OS (optional resource); link posted soon
    - 1st programming assignment
    - Details about project

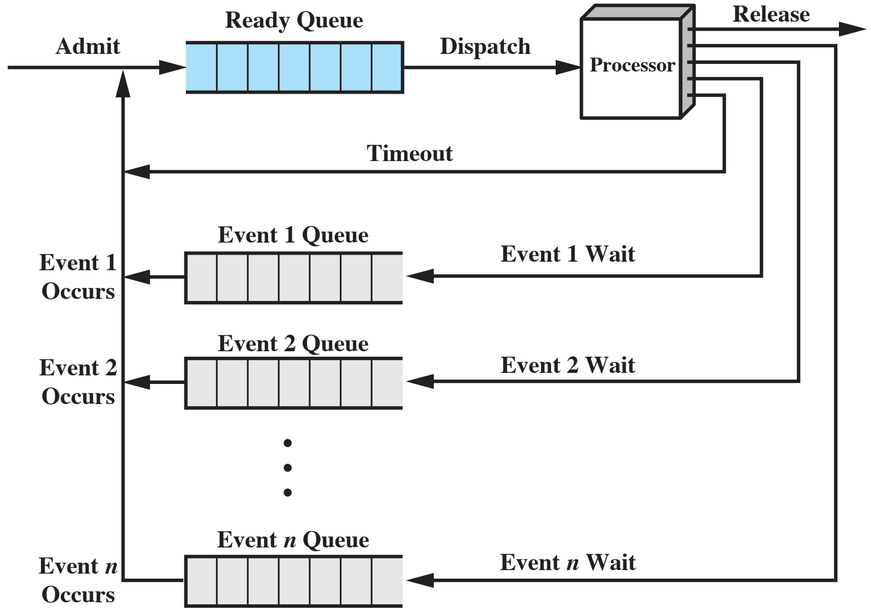


https://www.traviswpeters.com/cs460/

Last Time… Creating & Managing Processes







Creating a Process

```
vagrant@osbox:~/os$ ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S        0   0:00  \_ [ksoftirqd/0]
    2     4     0     0 ?           -1 S        0   0:00  \_ [kworker/0:0]
    2     5     0     0 ?           -1 S<       0   0:00  \_ [kworker/0:0H]
    2     7     0     0 ?           -1 S        0   0:01  \_ [rcu_sched]
```

DEMO

take a look at a process tree (already-created processes)

`ps axjf`

```
    0     1     1     1 ?           -1 Ss       0   0:00 /sbin/init
    1   524   524   524 ?           -1 Ss       0   0:00 dhclient -1 -v -pf /run/dhclient.eth0.pid -lf /va
    1   621   621   621 ?           -1 Ss       0   0:00 rpcbind
    1   663   663   663 ?           -1 Ss     107   0:00 rpc.statd -L
    1   741   741   741 ?           -1 Ss     102   0:02 dbus-daemon --system --fork
    1   806   806   806 ?           -1 Ss       0   0:00 rpc.idmapd
    1   826   826   826 ?           -1 Ss       0   0:00 /lib/systemd/systemd-logind
    1   832   832   832 ?           -1 Ssl    101   0:00 rsyslogd
    1   937   937   937 tty4       937 Ss+      0   0:00 /sbin/getty -8 38400 tty4
    1   940   940   940 tty5       940 Ss+      0   0:00 /sbin/getty -8 38400 tty5
    1   944   944   944 tty2       944 Ss+      0   0:00 /sbin/getty -8 38400 tty2
    1   945   945   945 tty3       945 Ss+      0   0:00 /sbin/getty -8 38400 tty3
    1   947   947   947 tty6       947 Ss+      0   0:00 /sbin/getty -8 38400 tty6
    1   987   987   987 ?           -1 Ss       0   0:00 /usr/sbin/sshd -D
  987  8560  8560  8560 ?           -1 Ss       0   0:00  \_ sshd: vagrant [priv]
 8560  8637  8560  8560 ?           -1 S     1000   0:00      \_ sshd: vagrant@pts/0
 8637  8638  8638  8638 pts/0    14242 Ss    1000   0:00          \_ -bash
 8638 14242 14242  8638 pts/0    14242 R+    1000   0:00              \_ ps axjf
    1   989   989   989 ?           -1 Ss       0   0:00 acpid -c /etc/acpi/events -s /var/run/acpid.socke
    1   990   990   990 ?           -1 Ss       0   0:00 cron
    1   991   991   991 ?           -1 Ss       1   0:00 atd
    1  1183  1183  1183 ?           -1 Ssl      0   0:01 /usr/bin/ruby /usr/bin/puppet agent
    1  1199  1196  1196 ?           -1 Sl       0   0:12 /usr/sbin/VBoxService --pidfile /var/run/vboxadd-
    1  1322  1319  1319 ?           -1 Sl       0   0:00 ruby /usr/bin/chef-client -d -P /var/run/chef/cli
    1  1349  1349  1349 tty1      1349 Ss+      0   0:00 /sbin/getty -8 38400 tty1
    1  2176  2175  2175 ?           -1 S        0   0:00 upstart-file-bridge --daemon
    1  2179  2178  2178 ?           -1 S        0   0:00 upstart-socket-bridge --daemon
    1  3800  3799  3799 ?           -1 S        0   0:00 upstart-udev-bridge --daemon
    1  3803  3803  3803 ?           -1 Ss       0   0:00 /lib/systemd/systemd-udevd --daemon
```

Some things to note:
names *(e.g., init),*
relationships *(parents, children, grandchildren),*
IDs *(PID, PPID)*

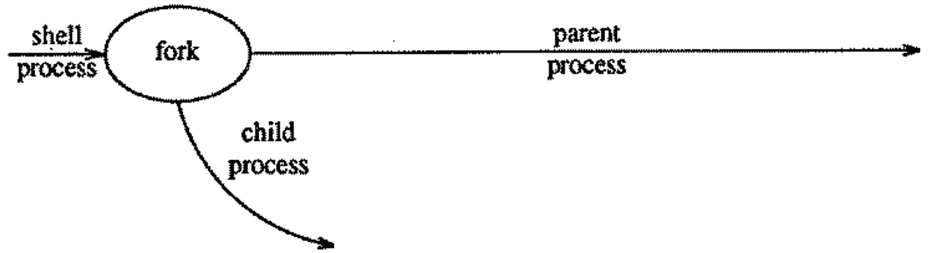
Creating a Process

- **Q:** But *how* to create a process? What UNIX call creates a process?

Creating a Process - `fork()`

- **Q:** But *how* to create a process? What UNIX call creates a process?
- `fork()` duplicates a process so that instead of one process, you get two!
  - *P1 and P2 continue in parallel from the statement that follows the* `fork()`
- **Q:** How can you tell P1 and P2 apart?

  …the return value of `fork()`!
  - `rval == 0 //to child`
  - `rval == child_pid //to parent`
  - `rval == -1 //fork() failed`



- **Q:** Will ***child*** see changes to global variable made by the ***parent***?

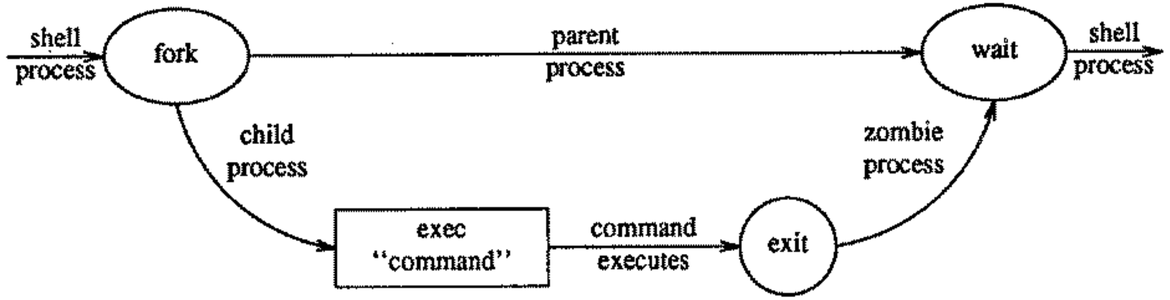
  No! On `fork()`, child gets new PC, stack, heap, globals, PID!

Creating a (Different) Process

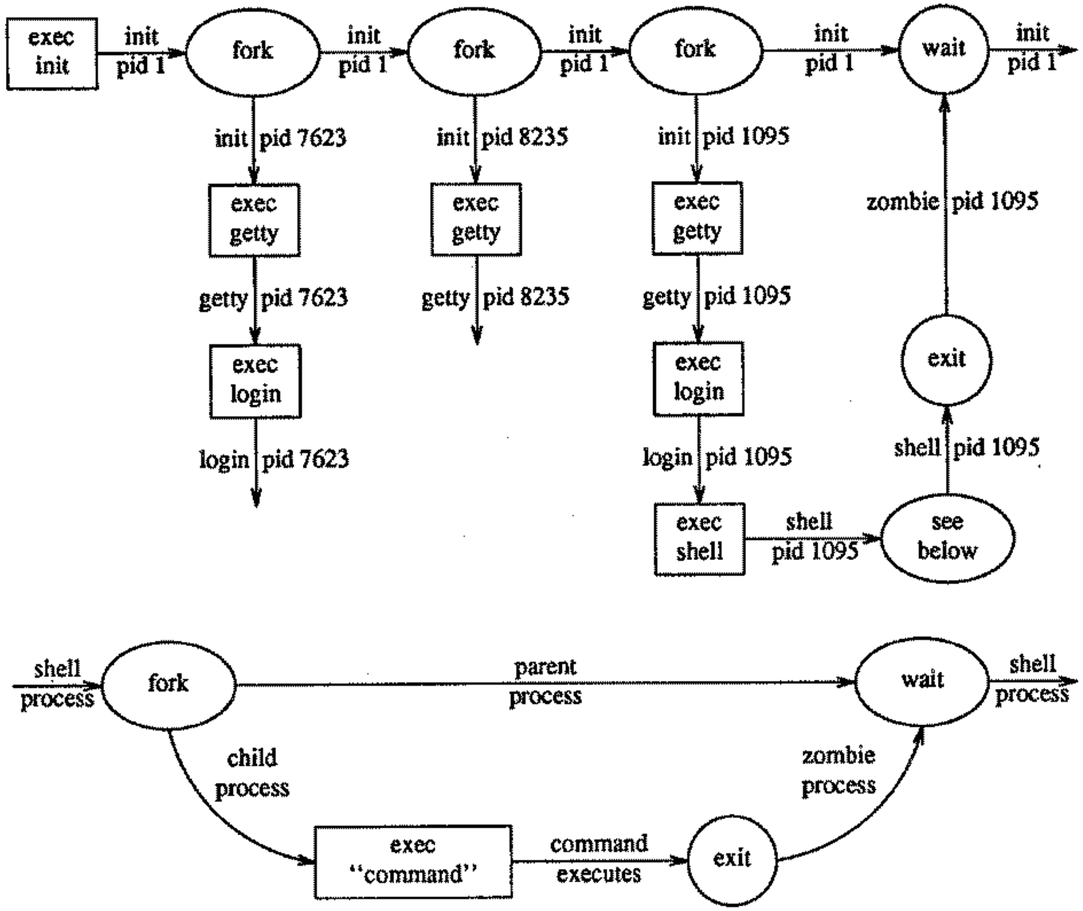
- **Q:** What if we want the child process to execute different code than the parent process?

Creating a (Different) Process - `exec()`

- **Q:** What if we want the child process to execute different code than the parent process?
- `exec()` …
  - …allows child to execute code that is different from the parent's code
  - …has a family of functions (`execl()`, `execv()`, `execle()`, `execlp()`, `execlvp()`) that provide a facility for overlaying the process image of the calling process with a new image
  - …returns -1 and sets `errno` if unsuccessful



Example: `fork()` and `exec()`



—John S. Quarterman, Abraham Silberschatz, and James L. Peterson, ``4.2BSD and 4.3BSD as Examples of the UNIX System'', Computing Surveys, Volume 17, Number 4, (December 1985), pages 379-418; translated to Japanese, Computer Science (BIT), Volume 18, Number 3, (1986), pages 175-213.

```c
/* Start a new process to do the job. */
cpid = fork();
// printf("process id is %d\n", cpid);

if(cpid < 0) {
    perror("fork");
    free(main_ptr); // clean-up
    return;
}

/* Check for who we are! */
if(cpid == 0) {
    /* We are the child! */
    execvp(main_ptr[0], main_ptr);
    perror("exec");
    free(main_ptr); // clean-up
    exit(127);
}

/* Have the parent wait for child to complete */
if(wait(&status) < 0)
    perror("wait");
// printf("wait result for process id %d is %d\n", cpid, status);
```

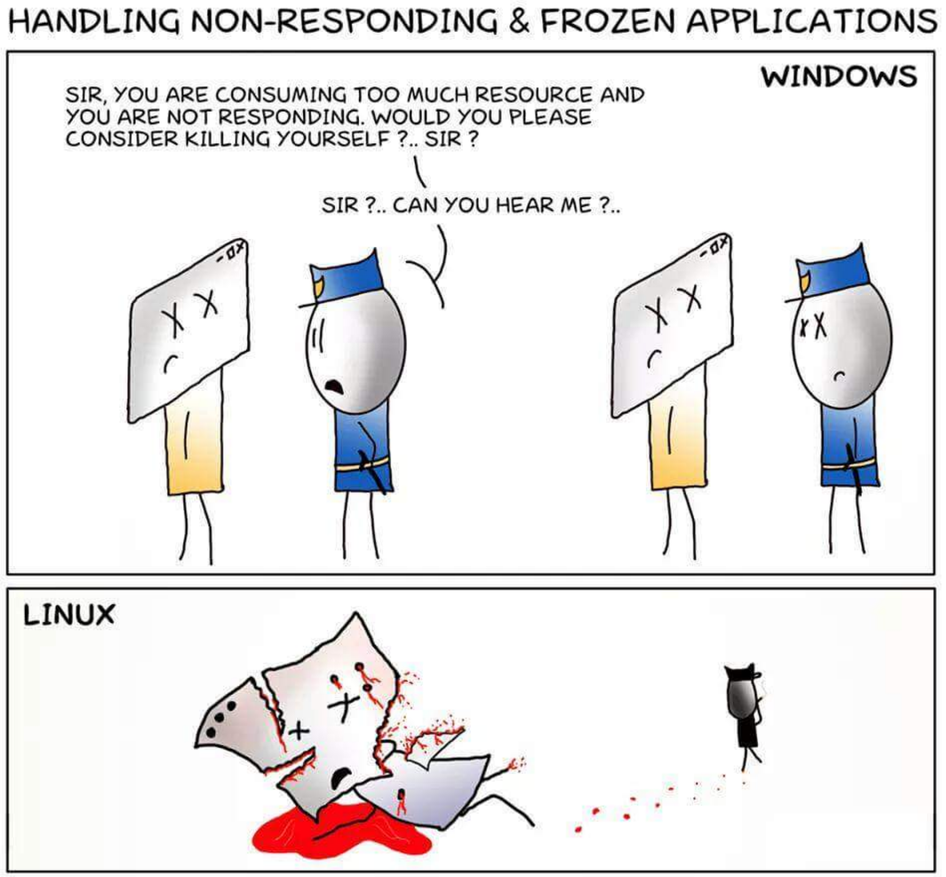
—fork() and exec() code snippet that Travis wrote back when he was an undergrad…

Terminating a Process

- There must *(at least should…)* be a means for a process to indicate its completion
- A batch job should include a HALT instruction (or something similar)
- An interactive process will terminate when a user, e.g., logs off, quits an app
- Again, *there is some discussion in the text on when/why processes terminate…*
  - *…normal completion*
  - *…timeout*
  - *…killed due to error/violation*
  - *…parent terminates*
  - *…explicitly requested (e.g., parent's request)*
  - *etc.*

Terminating a Process

- **Q:** What is the mechanism in UNIX to explicitly terminate a process?



Terminating a Process - `kill()`

- **Q:** What is the mechanism in UNIX to explicitly terminate a process?
- `kill()` …
  - …enables terminating a process by specifying the ***PID*** and a ***signal***
  - …default at commandline is to send signal 15 (SIGTERM) — *"please terminate…"*
- Read up on ***signals*** and different types of signals…
  - …signals are basically just *software interrupts* that can be sent to a program to indicate that an important event has occurred.
  - **Exercise:** run the following command in a terminal: `kill -l`

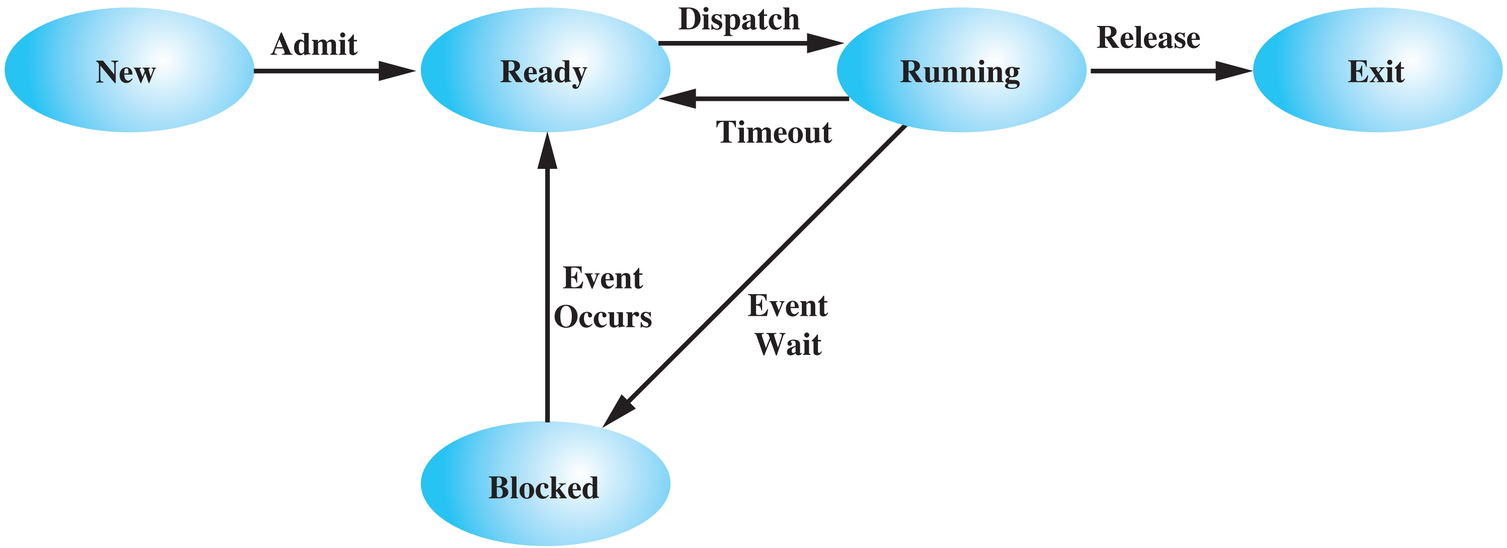
Suspended Processes & Swapping

- Swapping
  - Move parts of the process from main memory to disk to free up resources for other processes
  - OS will choose to swap blocked process(es) out to disk into a ***suspended state***
  - Swapping is an I/O operation; has the potential to make the problem worse…
    - on-system I/O is pretty fast though (relative to, e.g., network I/O)
    - Swapping *usually* improves performance
- **Q:** Why swap?

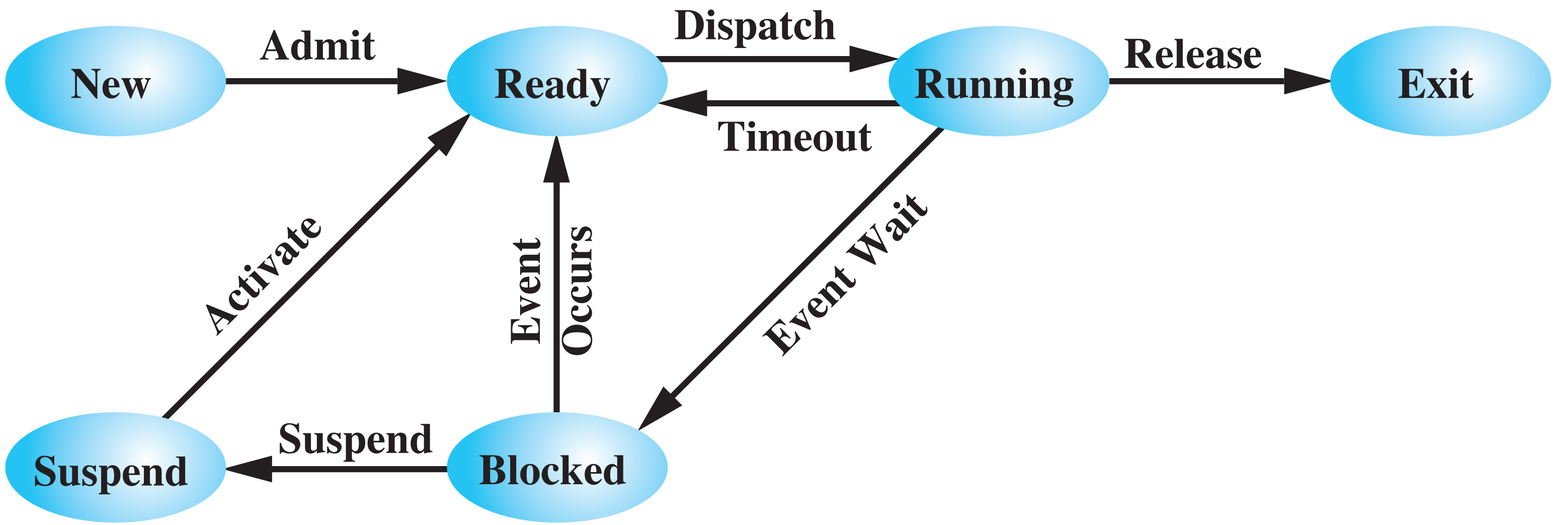
  OS needs to release resources, debugging, periodic task, etc.
- **Q:** Why not make memory bigger?!

  Cost; not more processes, usually bigger processes

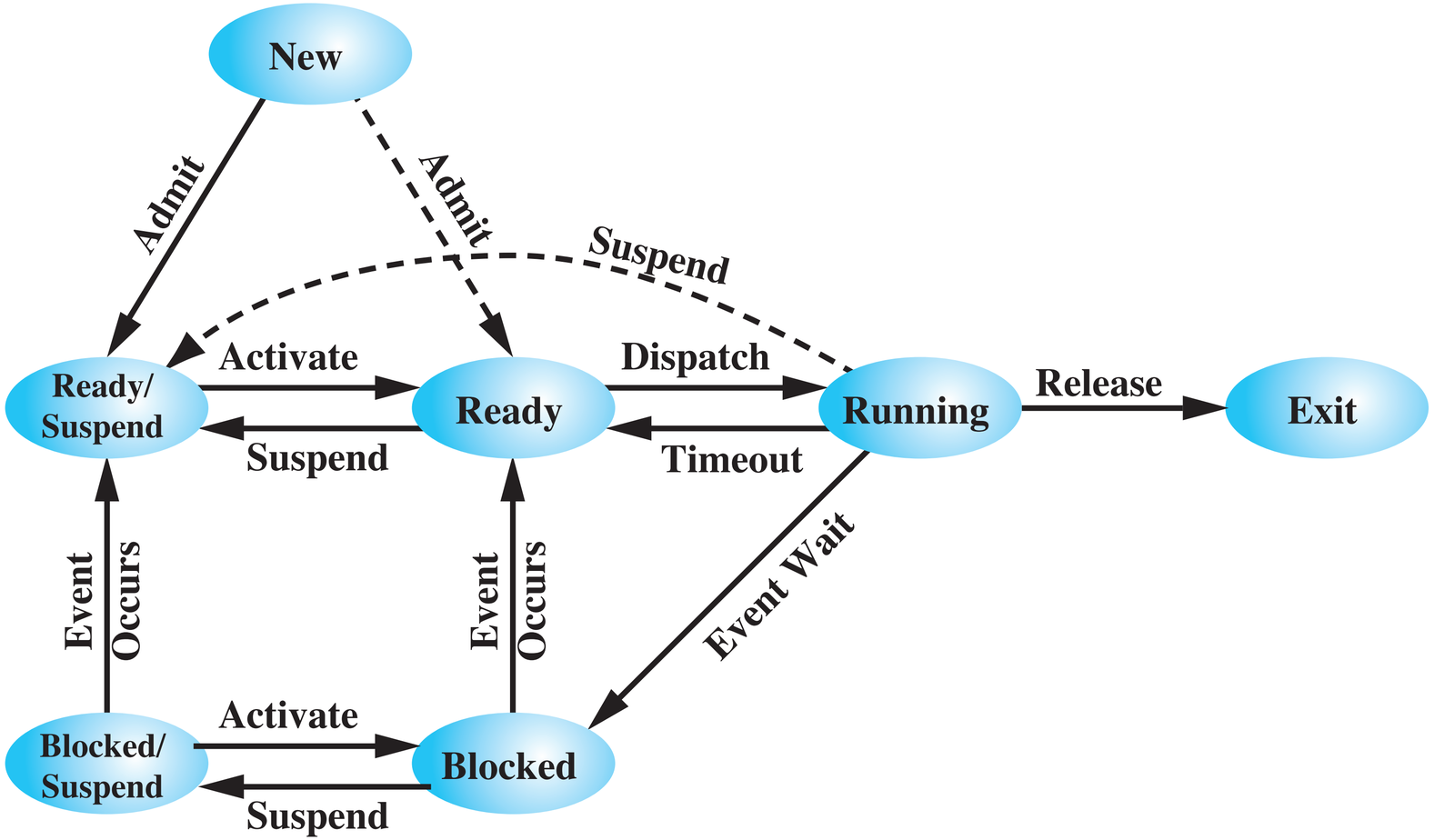
Suspended Processes & Swapping **(Before)**



Suspended Processes & Swapping **(After)**



Suspended Processes & Swapping **(After +Ready/Blocked)**



Summary: Process States

- It is **important for the OS to maintain information** about each process and its state **to manage system resources effectively**.
- Many, *many* **trade-offs** to consider…
  - More states (and more data structures for managing processes) are needed as we want to **reduce unncessary operations**, such as searching through queues to identify ready processes, higher-priority processes, etc.
  - But more data structures means **more memory is used** to maintain this state information, and **more overhead operations** to move processes between different data structures when they are blocked, suspended, ready, etc.

Suggestion:

know the differences between various process states,
and understand the reasons for (and pros/cons of) each.

Fun: The `fork()` Bomb

- **WARNING:** Run at your own risk. I'm running on a VM…

```c
#include <sys/types.h>
#include <unistd.h>

int main()
{
    while(1) {
      fork();
    }
    return 0;
}
```

—https://en.wikipedia.org/wiki/Fork_bomb

```
forkbomb() { forkbomb | forkbomb & } ; forkbomb
```