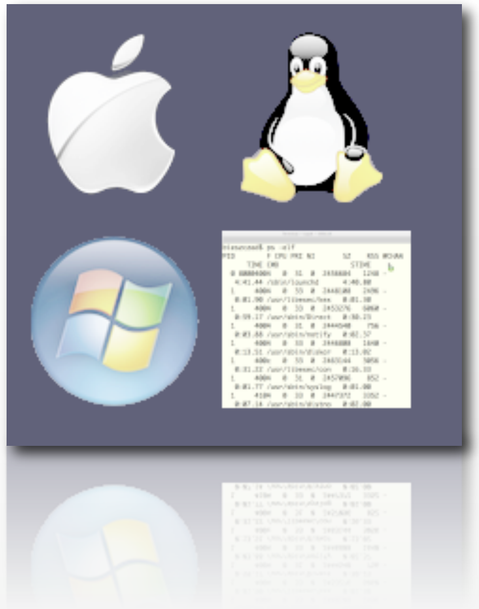


# Chapter 3: Processes

COMP362 Operating Systems
Prof. AJ Bieszczad

# Outline: Processes

- Process Concept
- Process Lifecycle
- Process Control Block (PCB)
- Process Scheduling
- Context Switch
- Process Creation
  - cloning with fork()
  - morphing with exec()
- Processes Termination

# Process Concept

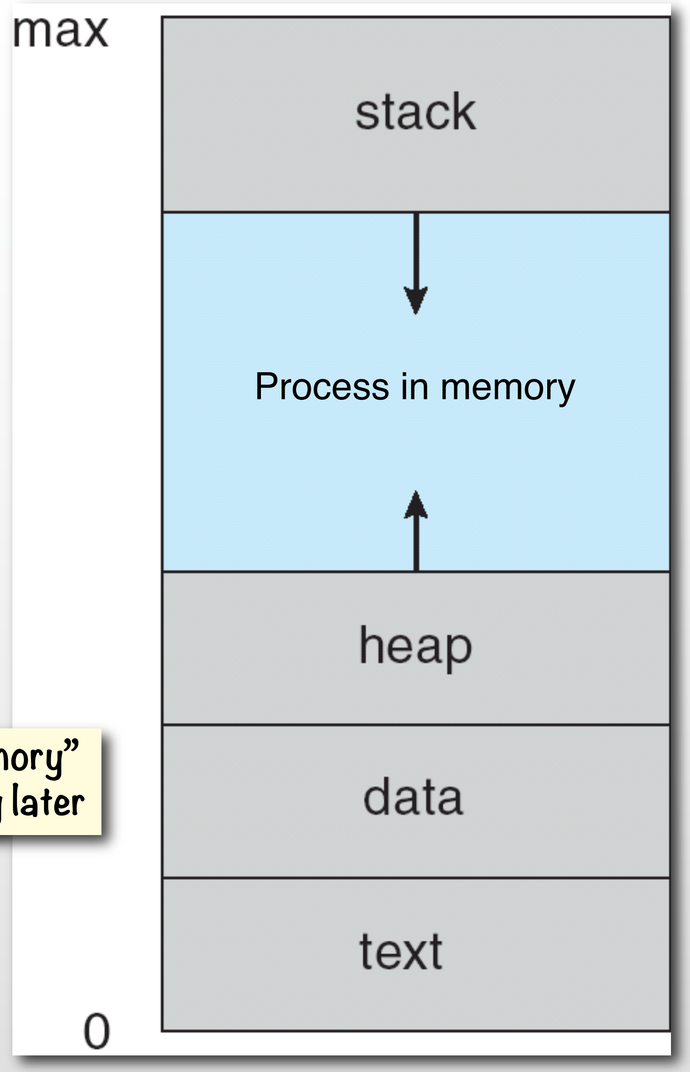
- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms **job** and **process** almost interchangeably
- **Process** – a program in execution
  - process execution must progress in sequential fashion
- A process includes memory for:
  - text
    - code
  - data
    - static data of the program
  - stack
    - for function calls; fixed size; grows “down”
  - heap
    - for dynamic allocation; OS can resize it as needed
- Some addresses randomized to increase security
- Resources can be controlled by system calls

Linux provides procfs
macOS does not

```
$ cat /proc/6514/maps
$ pmap 6514
```

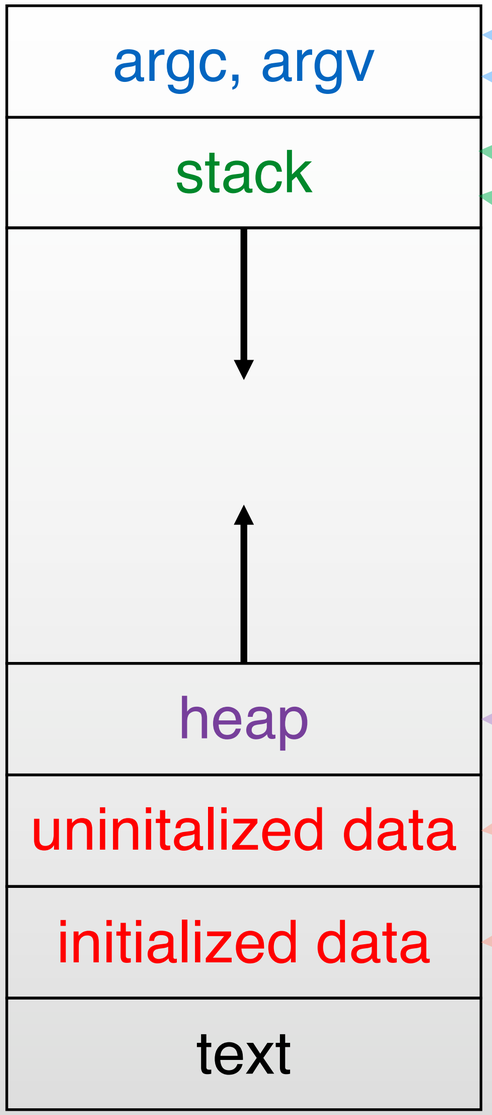
process ID

```
$ man ulimit
$ man setrlimit
```




“virtual memory”
we will study later

# Data Placement in a Process

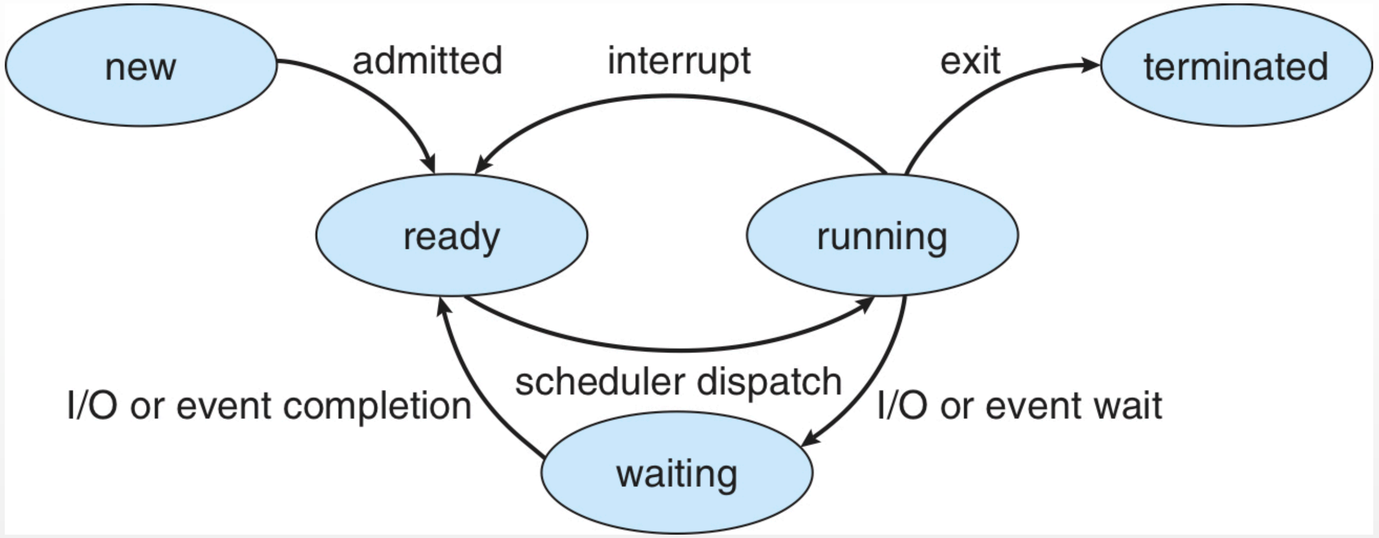



```
$ gcc memory_layout.c -o memory_layout
$ size memory_layout
```

```c
#include <stdio.h>
#include <stdlib.h>

int error_type;
char *error_msg = "ERROR";

int main(int argc, char **argv)
{
    int *val;

    val = (int *) malloc(5 * sizeof(int));

    for (int i = 0; i < 5; i++)
        val[i] = i;

    return EXIT_SUCCESS;
}
```

# Process Lifecycle

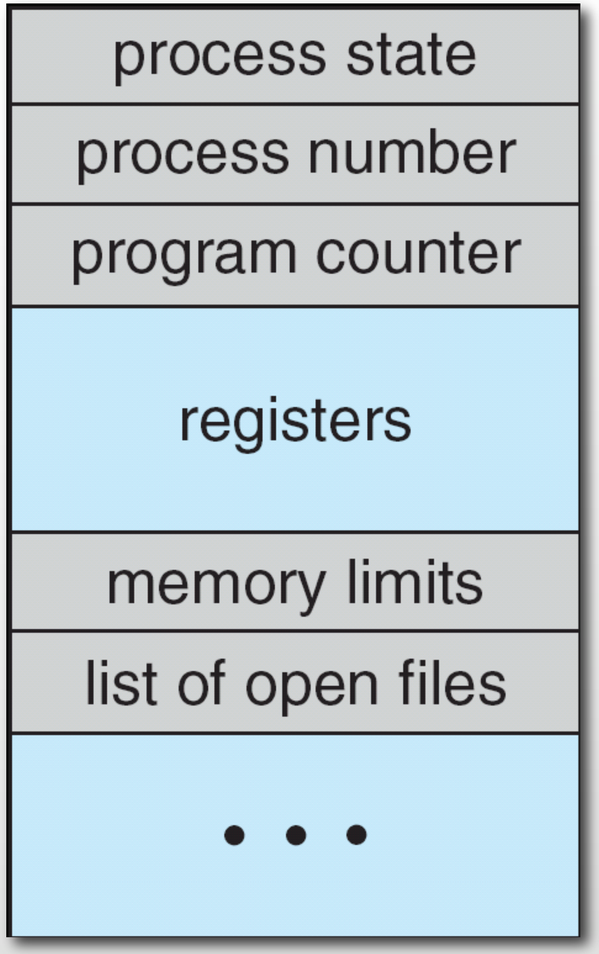
- Throughout its life in the system, a process changes states as follows




- **new**: The process is being created
- **ready**: The process is waiting to be assigned to a processor (CPU)
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **terminated**: The process has finished execution

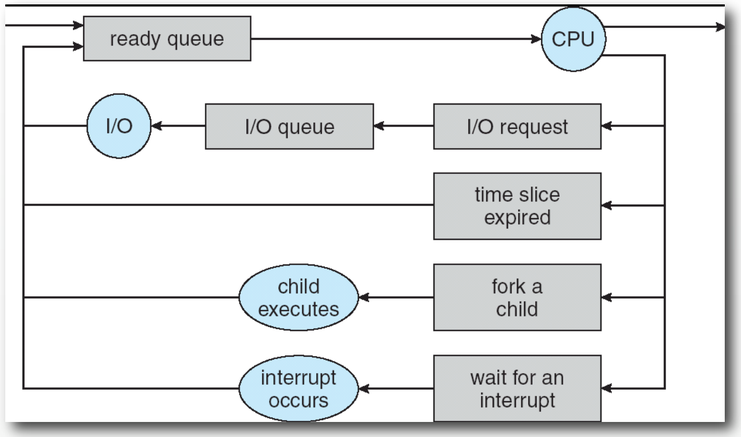
# Process Control Block (PCB)

- Information associated with each process
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information
- OS maintains a list of PCBs for all processes; e.g.,
  - array
  - linked list
  - hybrid

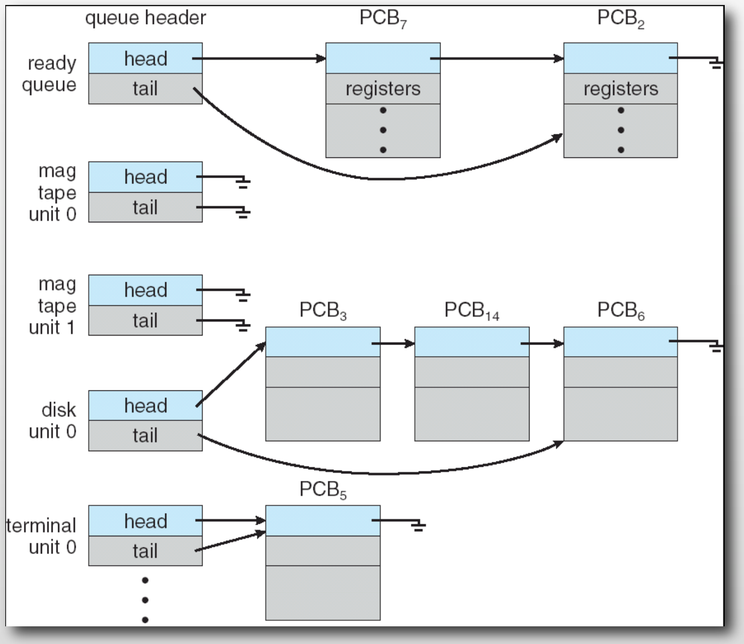


# Process Scheduling Queues

- Processes migrate among various OS queues
  - actually: references to PCBs used
- **Job queue**
  - set of all processes in the system
- **Ready queue**
  - set of all processes residing in main memory, ready and waiting to execute
- **Device queues**
  - set of processes waiting for an I/O device




some reasons for suspending a process



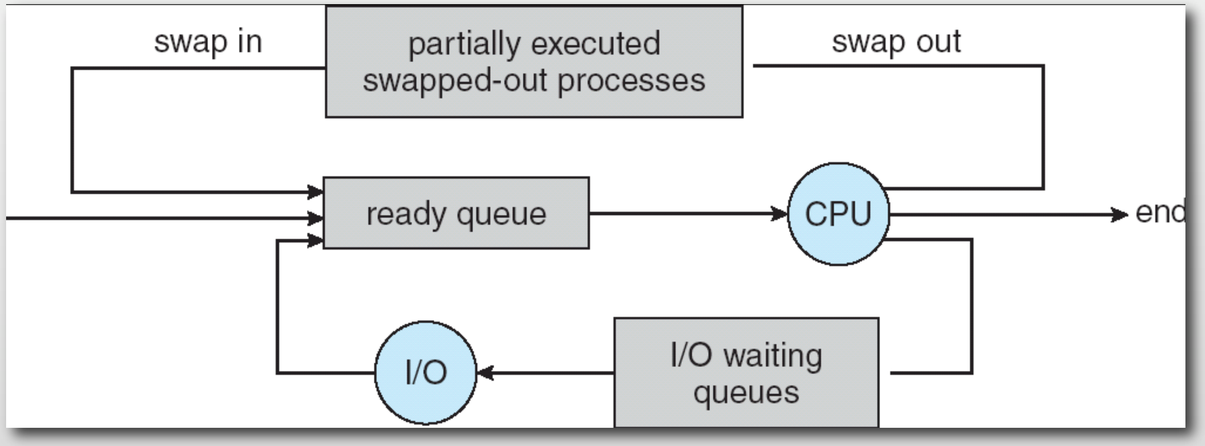

some device queues

# Schedulers

- **Long-term scheduler** (or **job scheduler**)
  - selects which processes should be brought into the ready queue
- **Short-term scheduler** (or **CPU scheduler**)
  - selects which process should be executed next and allocates CPU
- **Medium-term scheduler**
  - for swapping processes out and in
    - when inactive, consuming too much memory, etc.

Analogy: taking several courses

- ☑ how many can you take in a semester?
- ☑ which one should you attend now?
- ☑ which one can you put away for a week if you are swamped with work?



# Schedulers (Cont.)

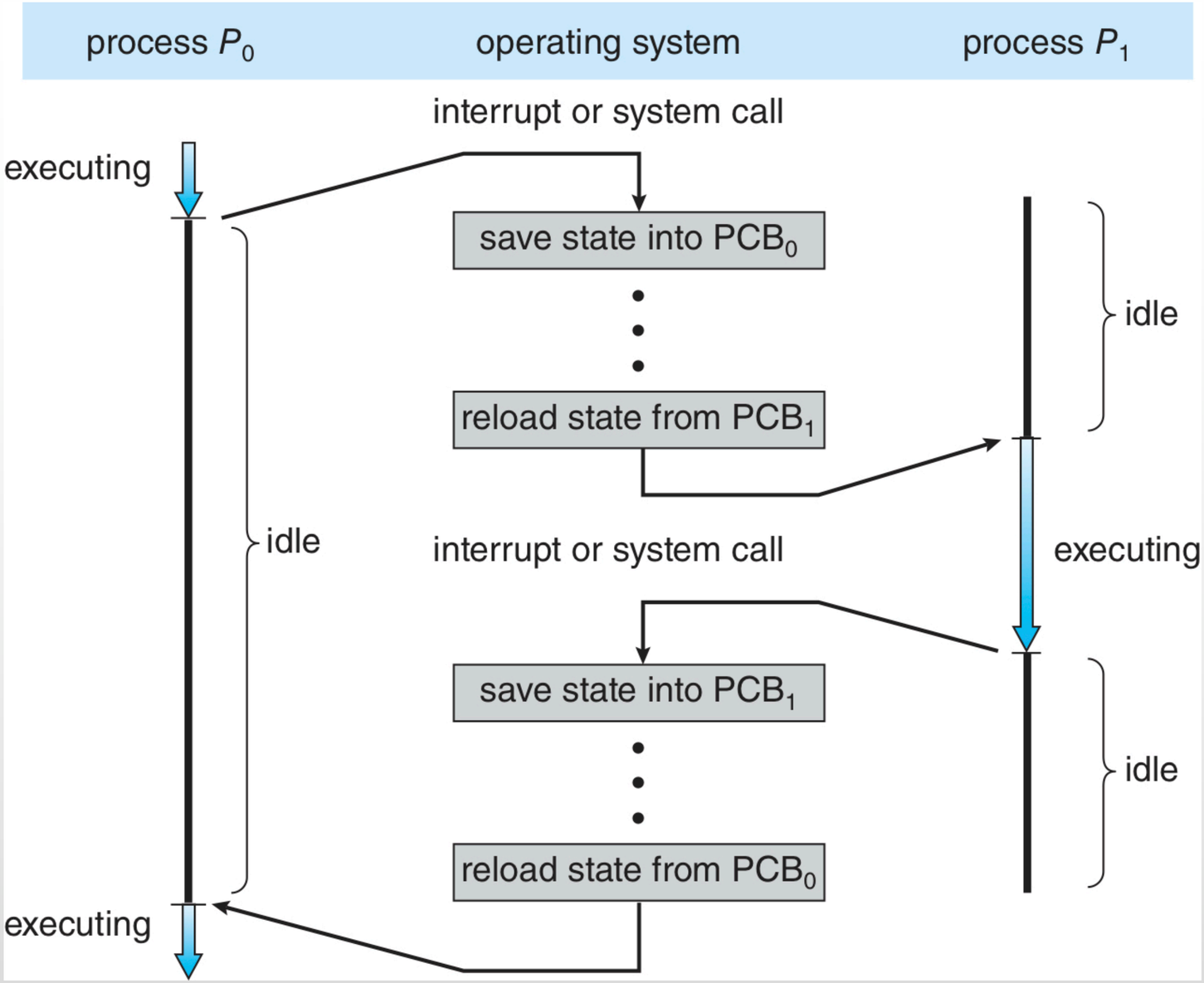
- Short-term scheduler is invoked very frequently (milliseconds)
  - ☛ must be fast

- Long-term scheduler is invoked very infrequently (seconds, minutes)
  - ☛ does not need to be very fast

- The long-term scheduler controls the degree of **multiprogramming**
  - long-term scheduler must take many things into account when admitting programs; that includes type of the process:
    - **I/O-bound process**
      - spends more time doing I/O than computations, many short CPU bursts
    - **CPU-bound process**
      - spends more time doing computations; few very long CPU bursts

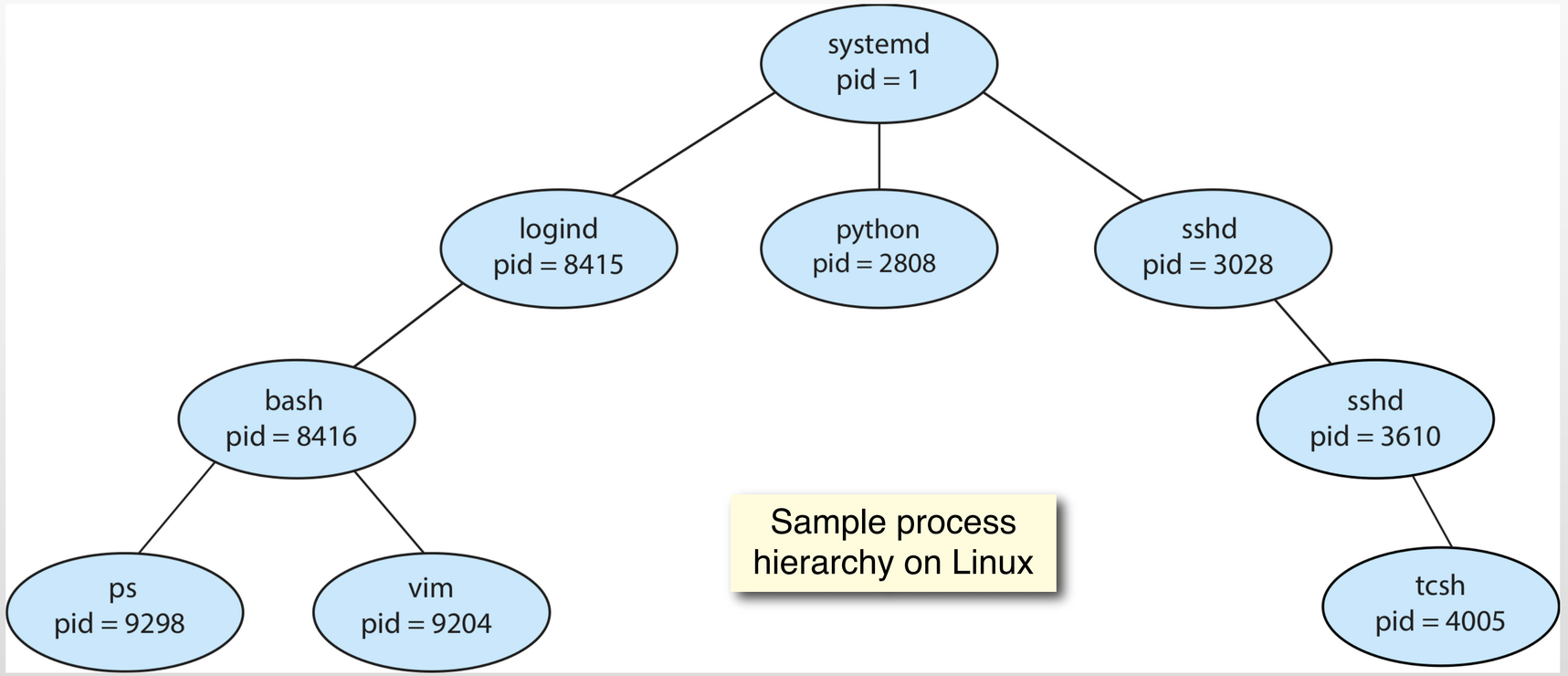
# Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- **Context-switch** time is an overhead
  - the system is not executing programs
- Time depends on hardware support



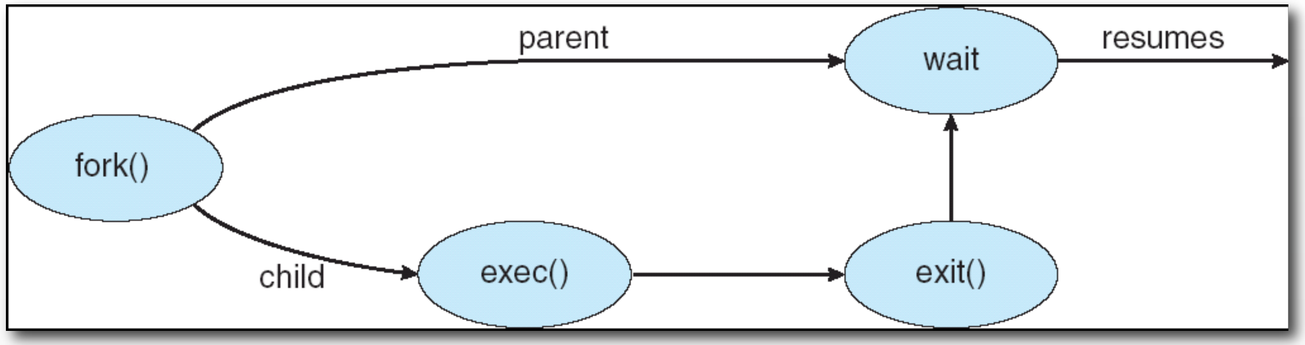
# Process Creation

- **Parent process** creates **children processes**, which, in turn create other processes, forming a hierarchy of processes
- Resource sharing options
  - Parent and children may share all resources
  - Children may share subset of parent's resources
  - Parent and children may share no resources
- Execution options
  - Parent and children may execute concurrently
  - Parent may wait until children terminate



# Process Creation

- Address space choices
  - Child may be a duplicate of the parent
  - Child has another program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call is used after a fork to replace the process' memory space with a new program




```
/* fork a child process */
pid = fork();
if (pid < 0)
{
   /* error occurred */
}
else if (pid == 0)
{
   /* child process */
}
else /* pid > 0 */
{
   /* parent process */
}
```

create_process.c

# Process Creation in POSIX

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
   pid_t pid;
   /* fork a child process */
   pid = fork();
   if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed\n");
       exit(EXIT_FAILURE);
   }
   else if (pid == 0) { /* child process */
       printf("I am the child %d\n", getpid());
       execlp("/bin/ls", "ls", NULL); // run /bin/ls in the child
       exit(EXIT_FAILURE); // executed only if execlp() fails
   }
   else { /* parent process; pid is the pid of the child */
       /* parent will wait for the child to complete */
       printf("I am the parent %d\n", getpid());
       waitpid(pid, NULL, 1); // wait for the child with the given pid
       printf("Child Complete\n");
       exit(EXIT_SUCCESS);
   }
}
```

POSIX standard API

# Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - output data from child to parent (via **wait**)
  - process' resources are **deallocated** by operating system
- Parent may terminate execution of children processes (**abort**)
  - for example, when:
    - child has exceeded allocated resources
    - task assigned to child is no longer required
    - if parent is exiting
      - some operating system do not allow a child to continue if its parent terminates
        - **cascading termination -** if all children along with their children, and so on, are being terminated
- Terminated process may enter so-called a "zombie" state
  - while the operating system is performing termination-related cleanup

# Process Management

```
$ man ps
$
$ man kill
$
$ man top
$ top
<exit by pressing Q>
$
$ sudo apt install htop
$ man htop
$ htop
<exit by pressing Q>
$
$ man procfs
$ cat /proc/12345/maps
$ cat /proc/12345/sched
```

htop is a better version of top

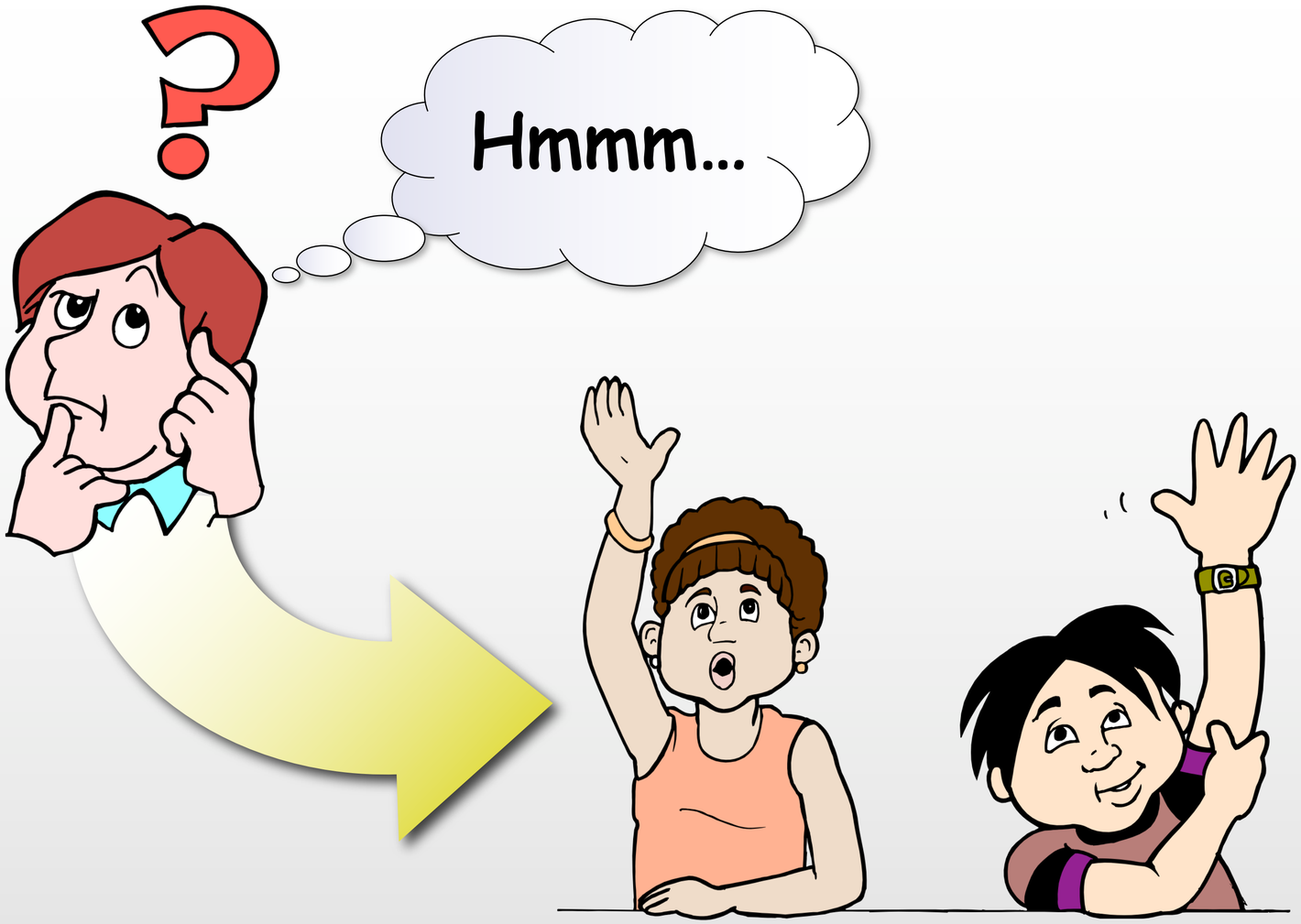
accessing process information using process pseudo file system (content of `/proc`) – only on Linux!

# Process Management *(cont.)*

```
$ ps -elf | grep proc_
0 S aj       29836  2122  0  80   0 -   620 do_wai 16:14 pts/0    00:00:00 ./proc_hier
0 S aj       30333 29860  0  80   0 -  4415 pipe_w 16:17 pts/2    00:00:00 grep --color=auto proc_
$ ps -elf | grep iam
0 R aj       29837 29836 30  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.1
0 R aj       29838 29836 30  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.2
0 R aj       29839 29837 30  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.1.1
0 R aj       29840 29837 31  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.1.2
0 R aj       29841 29838 31  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.2.1
0 R aj       29842 29838 31  80   0 -   620 -      16:14 pts/0    00:00:04 iam 1.2.2
0 S aj       29888 29860  0  80   0 -  4382 pipe_w 16:14 pts/2    00:00:00 grep --color=auto iam
$ kill -9 29839 29840
$ ps -elf | grep iam
0 R aj       29837 29836 31  80   0 -   620 -      16:14 pts/0    00:00:16 iam 1.1
0 R aj       29838 29836 31  80   0 -   620 -      16:14 pts/0    00:00:16 iam 1.2
0 Z aj       29839 29837 29  80   0 -     0 -      16:14 pts/0    00:00:15 [iam] <defunct>
0 Z aj       29840 29837 29  80   0 -     0 -      16:14 pts/0    00:00:15 [iam] <defunct>
0 R aj       29841 29838 31  80   0 -   620 -      16:14 pts/0    00:00:16 iam 1.2.1
0 R aj       29842 29838 31  80   0 -   620 -      16:14 pts/0    00:00:16 iam 1.2.2
0 S aj       29994 29860  0  80   0 -  4382 pipe_w 16:15 pts/2    00:00:00 grep --color=auto iam
$ kill -9 29837
$ ps -elf | grep iam
0 R aj       29838   913 45  80   0 -   620 -      16:14 pts/0    00:03:51 iam 1.2
0 R aj       29841 29838 45  80   0 -   620 -      16:14 pts/0    00:03:52 iam 1.2.1
0 R aj       29842 29838 46  80   0 -   620 -      16:14 pts/0    00:03:52 iam 1.2.2
0 S aj       31168 29860  0  80   0 -  4382 pipe_w 16:22 pts/2    00:00:00 grep --color=auto iam
```

zombies!

zombies gone after the parent terminated




# COMP362 Operating Systems

Prof. AJ Bieszczad