Processes

• In most contemporary Operating Systems such as Windows and Linux/UNIX, the unit of management is called a **process**

• A process is a resource container
  – Depending on the specific operating system, a process will have a set of defining attributes
  – At any given moment, the collection of processes in a system completely defines the system

• All computations must be done in the context of a process
Processes (cont’d)

• While processes on various systems share much more in common than in difference, we will focus on the process model used in Linux.

• A Linux process is characterized by many attributes, but foremost among these are:
  – An executable program
  – One or more threads that can run the program
  – An address space to contain all process memory objects (i.e. text, data, stack, etc.)
A Linux Process Tree

PID 0
kernel proc

PID 1
systemd

PID 2
kthreadd

PID 5
kworker

PID 11
watchdog

PID 13
ksoftirqd

PID 517
udevd

PID 918
bluetoothd

PID 972
sshd

PID 8341
sshd

PID 8345
bash

PID 3882
ps

TCP port 22

$ ps
----------

PID 8345 control terminal
Inherited through hierarchy

Inherited through hierarchy

ch 0, 1, 2

stderr 2
stdin 0
stdout 1

PID 8345 control terminal
Inherited through hierarchy

PID 8345 control terminal
Inherited through hierarchy
Process Address Space

- Each memory object is a contiguous range of bytes within the address space
- The size of the address space is limited by the CPU architecture and the operating system version
- In a 32 bit Linux system on an x86 processor, the user default space is 3 GB (it’s 128 TB in a 64 bit x86 system)
Threads

• The executable (schedulable) elements in a Linux system
• Each thread in the system is uniquely contained by some process
  – Each user thread is contained by some user PID
  – Each kernel thread is contained in PID 0
• When a new process is created, it is populated by exactly one executable thread, known as the Initial Thread (IT) of the new process
• The IT of a process can create new threads only within its own process
• While the IT must create the second thread in a process, any subsequent threads can then create new threads, but only within their own process
switch (int pid = fork()){
  case -1: perror("fork failed ");
      exit(1);
  case 0: printf("child alive\n");
      execl("./myprog", "myprog", NULL);
  default: printf("created PID %d \n", pid);
} // end switch
The new child program *myprog* executes from the first statement in its *main()* function.

If the new program executes the following statement:

```c
pthread_create(&tid_id, NULL, th_fun1, NULL);
```

a **new stack** will be mapped into the address space
Thread States and Transitions

- Run K/U
- Block
- Ready
- fork()
  pthread_create()
Thread Access Example

- A **system call** made by a thread in PID x is:
  
  ```c
  int channel = open("/usr/faculty/bill/mmap_rw_var.c", O_RDWR, 0);
  ```

- The system call **succeeds** and returns a valid channel to read and write

- A second call made by a thread in PID x is:
  
  ```c
  int channel = open("/usr/faculty/bill/malloc_th.c", O_RDWR, 0);
  ```

- This call **fails**, since the calling process is the owner, and owner permissions don't allow WRITE, even though group and other do