Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 2 is due **next Monday** at 11:59 pm
  - Lab 1 is due on Tuesday 2/7 at 11:59 pm
  - Project 1 is due on Friday 2/17 at 11:59 pm
    - Discussed in this week's recitations
- AFS Quota
  - You can check it using the command **fs quota**
  - You can increase it from accounts.pitt.edu.
    - Check README of Lab 1
- VS Code issue
  - Turn off the usage of **flock** to lock files
  - Check my Piazza reply in **Thoth Password** thread

Previous Lecture …

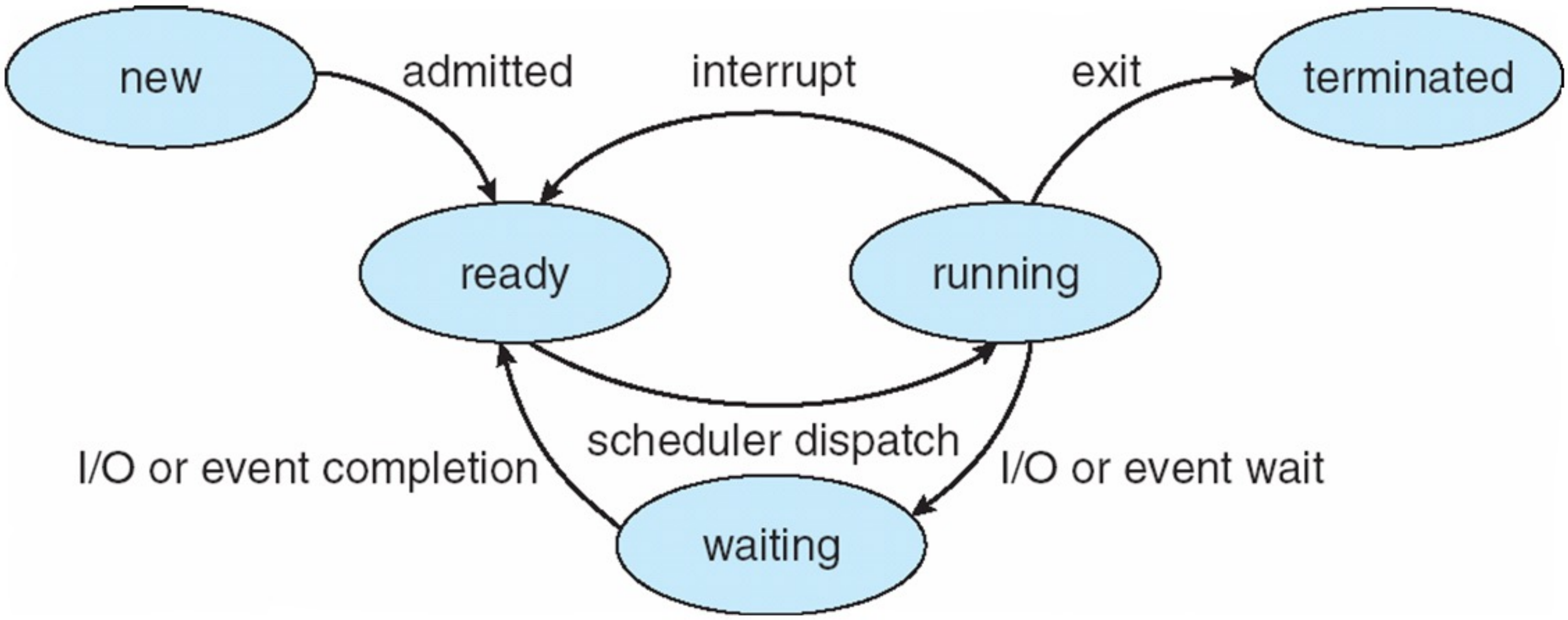
Three usage **problems** of Semaphores

- compromising mutual exclusion
  - **Solution: Mutex**
- deadlock
  - **Solution: Not yet discussed**
- priority inversion
  - **Solution: priority inheritance**

Question

How are processes created, maintained, and terminated?

Process Lifecycle (AKA Process States)



Process Creation

- Via fork() syscall
- Parent process: the process that calls fork()
- Child process: the process that gets created
- Child process has a new context
  - new PCB

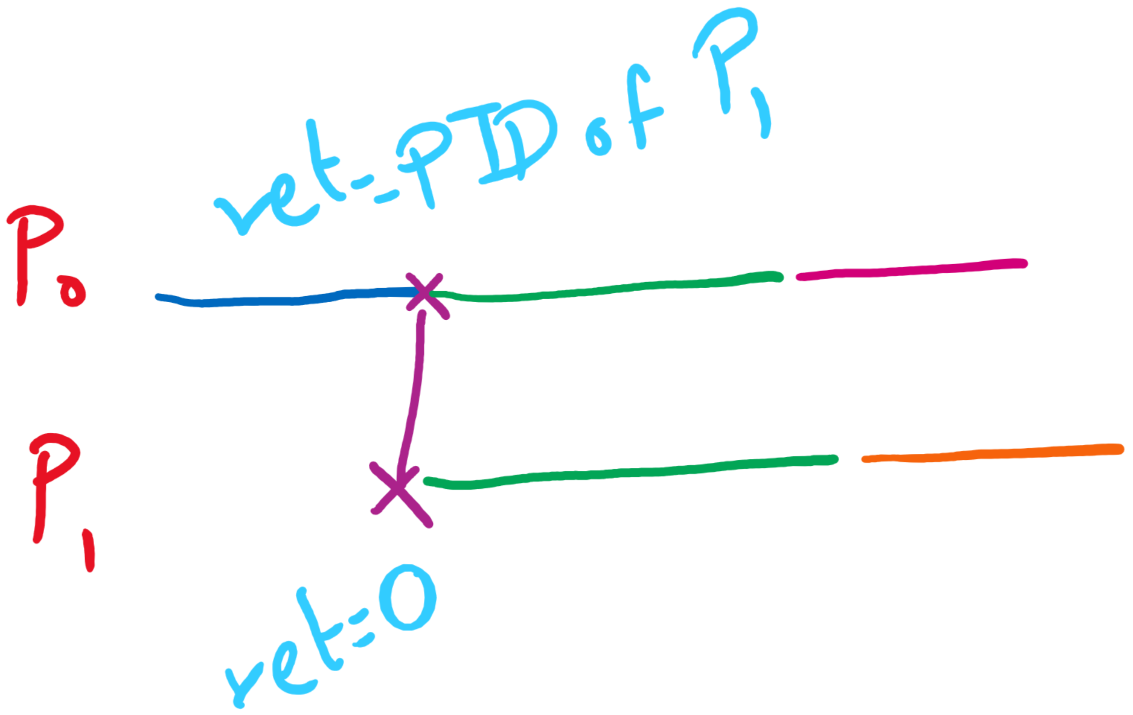
Process Creation

Memory of parent process copied to child process

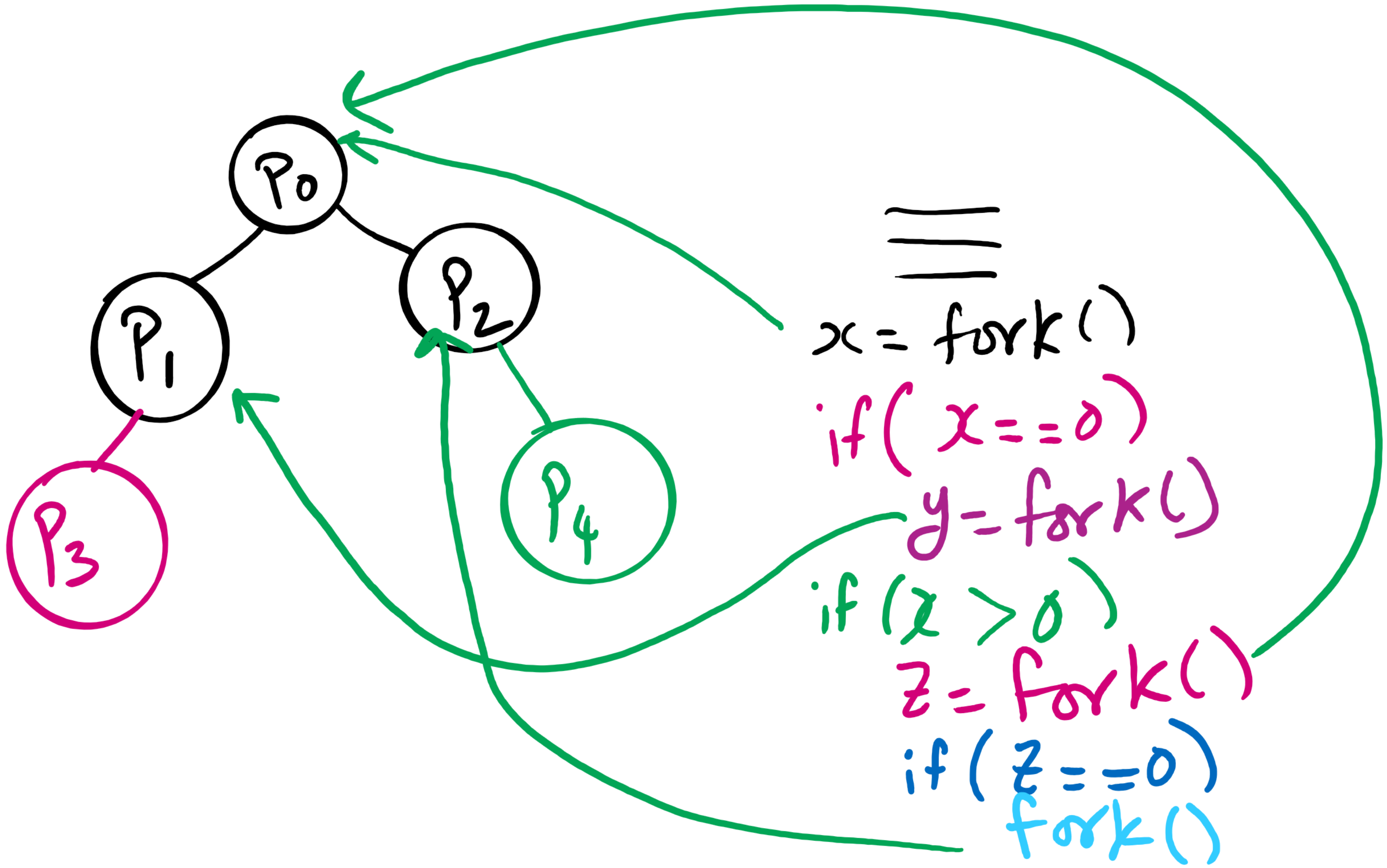
- Too much copying
- Even not necessary sometimes
  - e.g., fork() followed by exec() to run a different program
- Optimization trick:
  - **copy-on-write**
  - copy when any of the two processes writes into its memory
  - copy the affected memory “part” only
  - How would the OS know when a process writes to its memory?

fork() tracing

```
int ret = fork();
if(ret==0)
if(ret >0)
```

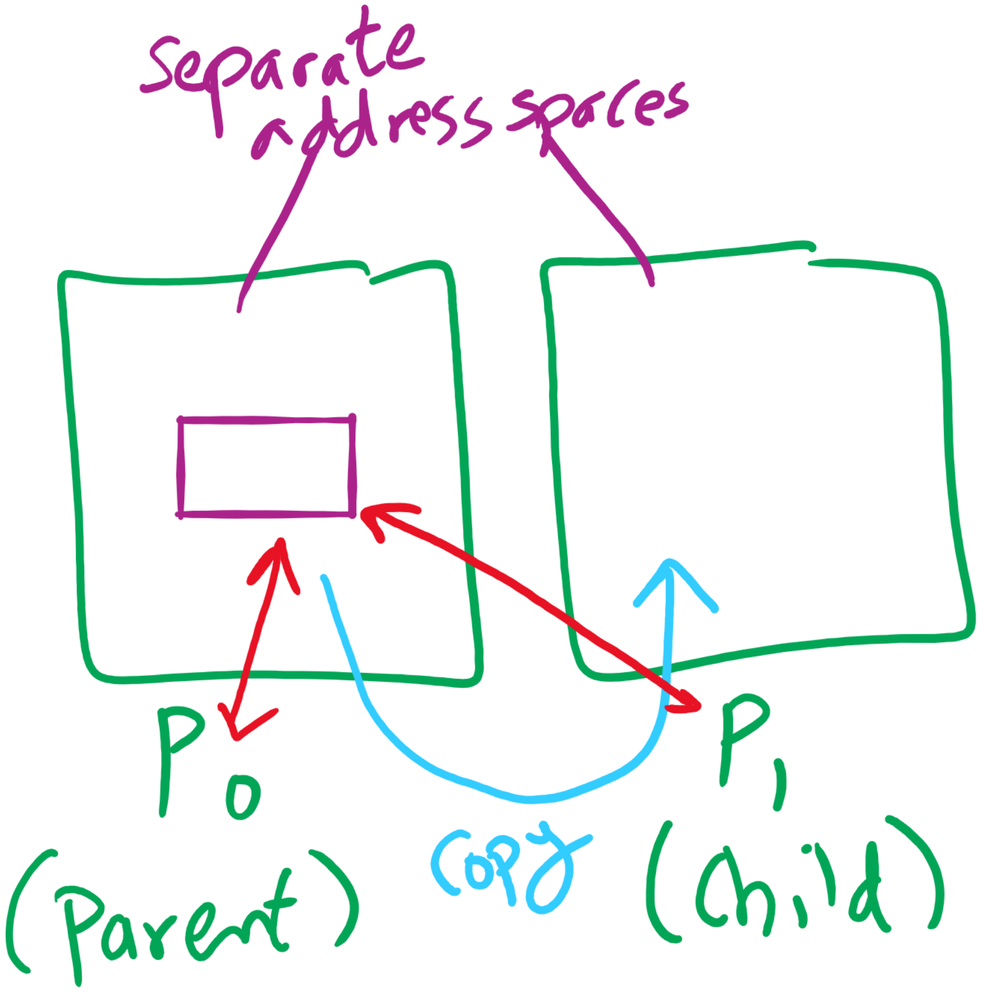


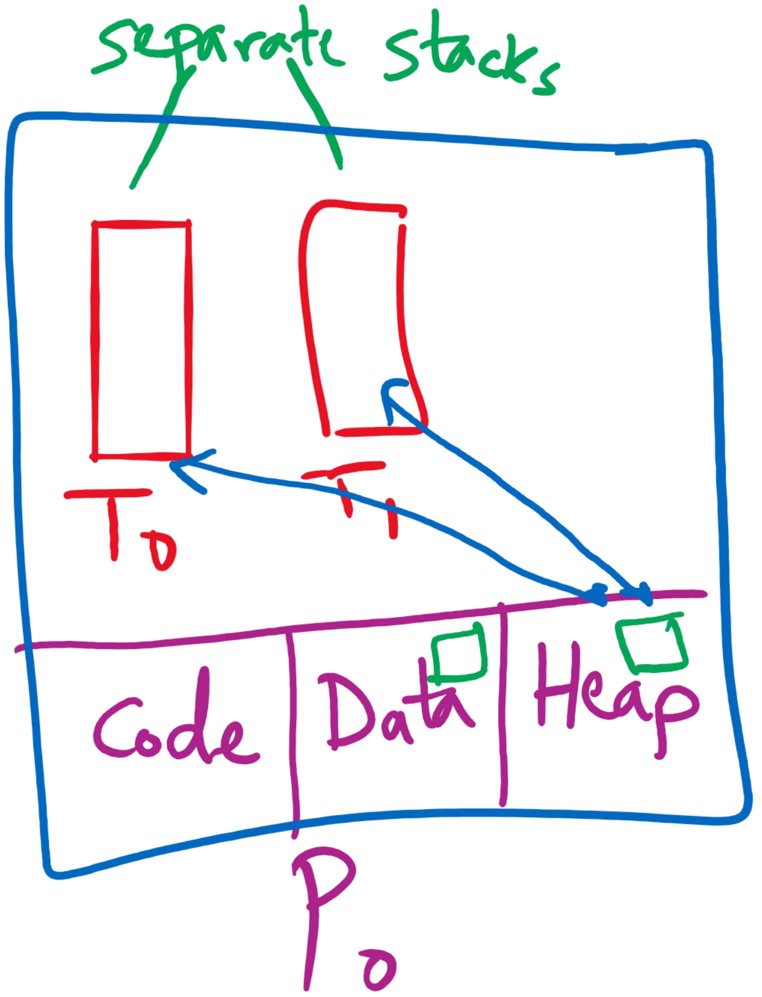
fork()'s of fork()'s



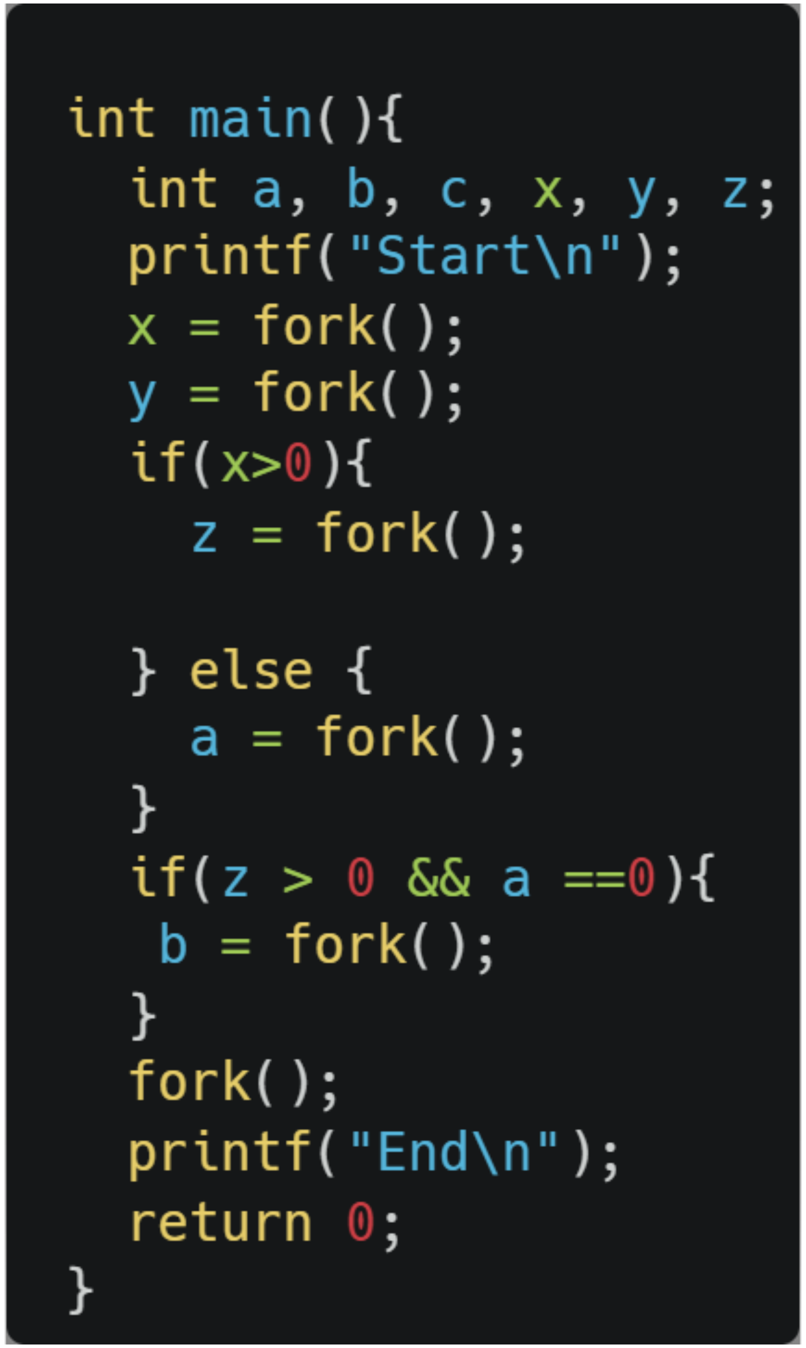
```
≡
x = fork()
if( x==0)
    y = fork()
if (z >0)
    z = fork()
    if ( z==0)
        fork()
```

Process vs. Thread





fork() example



```c
int main(){
  int a, b, c, x, y, z;
  printf("Start\n");
  x = fork();
  y = fork();
  if(x>0){
    z = fork();

  } else {
    a = fork();
  }
  if(z > 0 && a ==0){
   b = fork();
  }
  fork();
  printf("End\n");
  return 0;
}
```

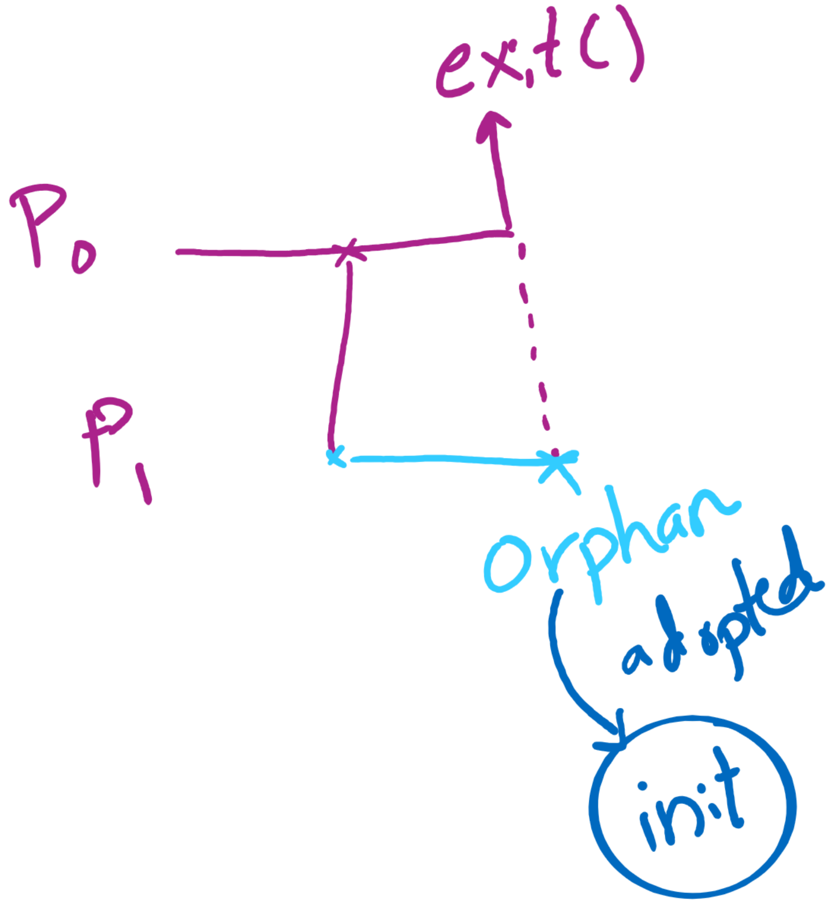
Process Termination

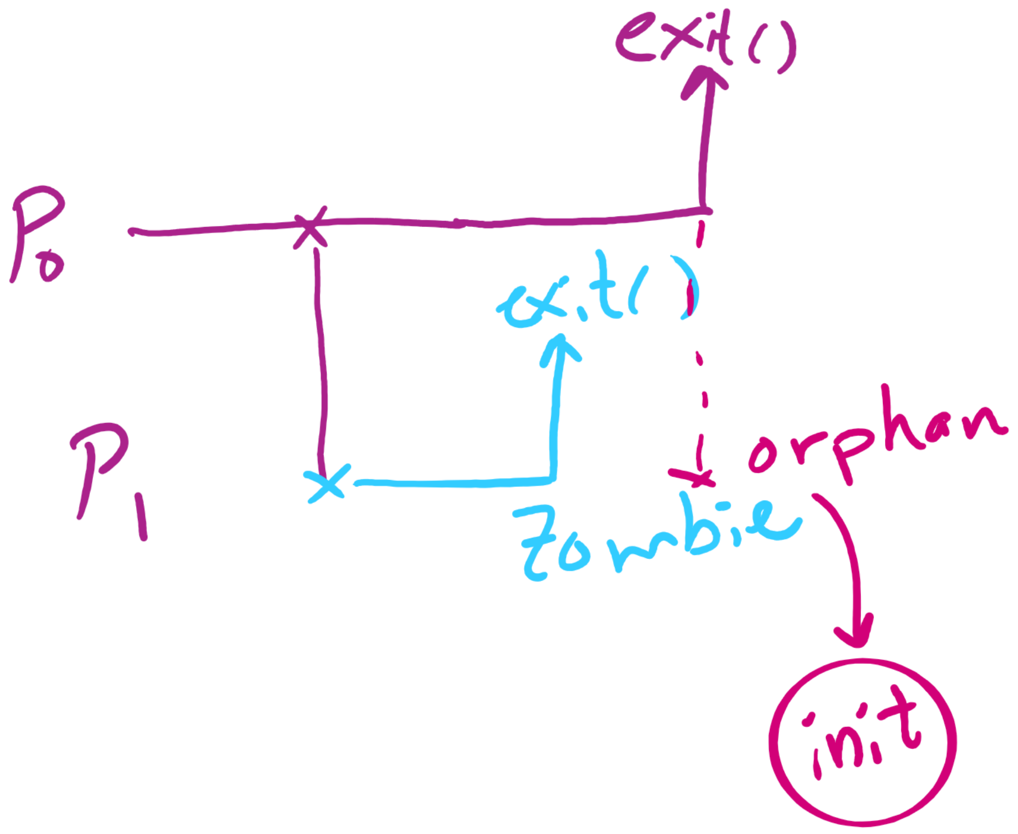
- Via **`exit`**(), **`abort`**(), or **`kill()`** syscalls
- The parent process may wait for termination of a child process by using the **`wait()`** system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- When a process terminates
  - If no parent waiting (did not invoke **`wait()`**) process is a **zombie**
  - If parent terminated without invoking **`wait`**, process is an **orphan**
    - adopted by the **`init`** process

Orphan vs. Zombie Processes





Benefits of Orphan Processes

- Allow a long-running job to continue running even after session (e.g., ssh connection) ends.
  - The nohup command does that
- Create daemon processes
  - Long-running background processes adopted by the init process.

Thread Synchronization

Synchronization issues apply to *threads* as well

- Threads can share data easily (same address space)
- Other two issues apply to threads