The Life Cycle for a Process
Process states

- Runnable
- Running
- Waiting
- Terminated

- Load executable binary from disk
- Copy existing process via fork()
Process states

- Runnable
- Running
- Waiting
- Terminated

Scheduler

Ready queue

Scheduler Algorithms
- FIFO?
- Priorities?
Process states

- **Runnable**
  - Transitions to **Running** via Scheduler
  - Transitions to **Waiting** via Operation completes

- **Running**
  - Transitions back to **Runnable** via Scheduler
  - Transitions to **Terminated** via Blocking IO Synchronization

- **Waiting**
  - Transitions to **Runnable** via Scheduler

- **Terminated**
  - No direct transitions

**Scheduler**

**Ready queue**

**Blocked queue**
OS keeps the PCB for the process, so that parent process can wait().
Process termination

Parent

```
child_pid = fork();
//Parent does some
//stuff, and then
//does this . . .
int child_status;
waitpid(child_pid,
        &child_status);
```

//Child starts
//executing.

Child

```
//Later...
exit(42);
```

//waitpid() returns!
printf("%d", child_status);
//Displays "42".

Time flows down
In the state diagrams in this video, we assumed that a process can only reach the "Terminated" state from the "Running" state. Can you think a way that a process in the "Runnable" or "Waiting" state could transition to the "Terminated" state?