Processes Continued

Using the CPU, off the ready queue

RUNNABLE (x10) scheduled to run

Scheduled to run

Setup

ZOMBIE

Waiting for an event

Kernel schedules another process to run

as soon as a process blocks,

event occurred

kernel must track what event a process is waiting on

Ready

dispatch

Running

time slice expired

Blocking

timeout

unblocking

Blocked

Creation

Ready queue

Kernel schedules waiting another process to run

4/3/24
// Per-process state
struct proc {
    struct vspace vspace;                     // Virtual address space descriptor
    char* kstack;                           // Kernel stack
    enum procsate state;                   // Process state
    int pid;                               // Process ID
    struct proc *parent;                   // Parent process
    struct trap_frame *tf;                 // Trap frame for current syscall
    struct context *context;               // swtch() here to run process
    void *chan;                            // If non-zero, sleeping on chan
    int killed;                            // If non-zero, have been killed
    char name[16];                         // Process name (debugging)
};

// fd table

\[
\begin{align*}
\text{sleep}(\text{chan}) &= \text{waiting for event}(\text{chan}) \\
\text{wakeup}(\text{chan}) &= \text{event has taken place}, \text{unblock all processes waiting on chan}
\end{align*}
\]
Process APIs: *Fork*

→ Creates a new process that's an exact copy of the calling process at the time of `fork`.

- `fork()`
- `x = 2`
- `y = 3`

→ Where should the child start execution?
- Same as its parent, return from `fork`.

**Parents**
- Changes after `fork` not visible to child
- VAS
- Separate processes
- Separate translation tables
- Their own kernel stacks
- OS resources inherited (open files)

**Children**
- VAS
- Physical memory
- `x = 2`

Parent = actual caller, receives:
- Actual caller's PID

Child = didn't actually call `fork`, receives 0 for return value.

I should have the same trapframe & values except for `%rax`. 
man 2 fork (manpage)  How many processes in total?

fork();
  1 parent
  1 child
  1 parent
  1 child
  1 child (2nd fork)
  1 grandchild (2nd fork)
  parent
  child
  grandchild
  fork();
  if (pid == 0) {
    fork();
  }
Process APIs: `exec` → loads a new program into the current process
(replaces the current program!)

same process, different address space, different execution states

(pid10) process VAS according to program A

exec(`B`)  ➞  (pid10) process VAS according to program B

(rsp = program B's entry point)

(rip = program B's args)
Fork-exec combo

\[ \rightarrow \text{simple semantics} \]
\[ \rightarrow \text{easy to support redirect} \]

example: ls > output

```c
pid = fork();
if (pid == 0) {
  fd = open("output");
  close(stdout);
  dup(fd); // stdout \rightarrow output
  exec("ls"); // ls prints to stdout which is now output
}
```
Fork: copies parent’s memory, sets up appropriate translation table
Excl: gets rid of current VAS, set up a new VAS & V for the process

[ highly inefficient! ]

→ copy-on-write (cow)

- share the same phys. memory for as long as possible (until a write)
- upon write, makes a copy so the write can be carried out independently

Kernel needs to mark all shared memory as read only
Write will then cause a page fault exception (needs to differentiate cow from actual permission violation)