4/3/24

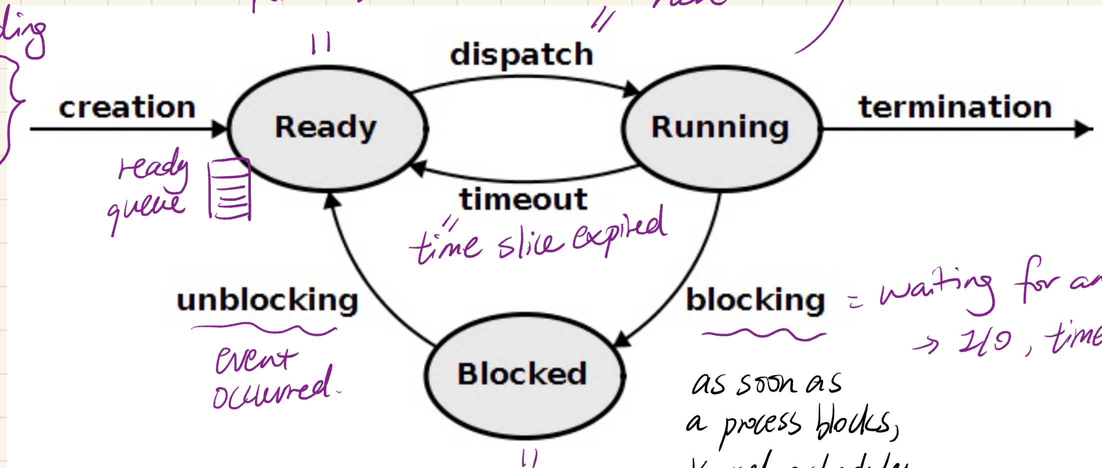
Processes Continued

using the CPU,
off the ready queue

RUNNABLE (xk)
=

scheduled to run
=

ELF loading
Set up VAS & PCB



ZOMBIE

ready queue

= time slice expired

event occurred

= waiting for an event
→ I/O, timer, etc.

as soon as
a process blocks,
kernel schedules
another process to run

= waiting

☆ kernel must track what event a process is waiting on

Process Control Block

```
// Per-process state
struct proc {
  struct vspace vspace;          // Virtual address space descriptor
  char* kstack;                  // Kernel stack
  enum procstate 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)
};
```

scheduling state

event waiting for

fd table

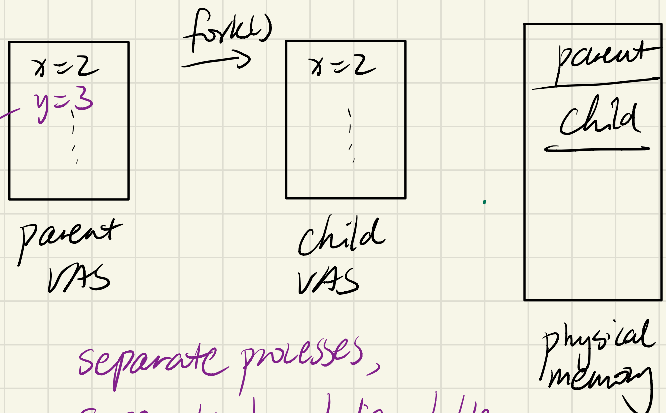
xk
- sleep(chan...) = waiting for event (chan)
- wakeup(chan) = event has taken place, unblock all processes waiting on chan

Process APIs: Fork

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

child (the new process) · parent (the calling process)

changes after fork not visible to child



separate processes,
separate translation tables,
their own kernel stacks,
OS resources inherited (open files)

→ where should the child start execution?

→ same as its parent, return from fork.

[should have the same trapframe * values * except for %rax]

the syscall return value.

parent = actual caller, receives child's pid

child = didn't actually call fork, receives 0 for return val.

man 2 fork (manpage)

How many processes in total?

```
fork();
```

| parent

| child

```
fork();
fork();
```

| parent

| child

| child (2nd fork)

| grandchild (2nd fork)

```
pid = fork();
if (pid == 0) {
    fork();
}
```

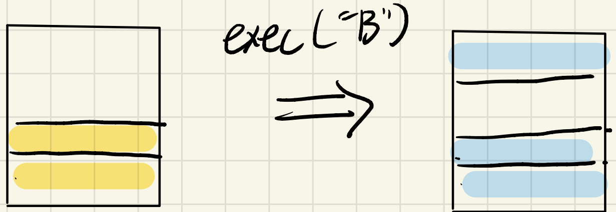
| parent

| child

| grand child

Process APIs : exec

→ loads a new program into the current process (replaces the current program!)



(pid 10) process VAS according to program A

(pid 10) process VAS according to program B

☆ same process, different address space, different execution states.

(rip = program B's entry point)

(rsp = program B's args).

Fork exec combo

- simple semantics
- easy to support redirect

example: ls > output

```
pid = fork();
if (pid==0){
    fd = open("output");
    close(stdout);
    dup(fd);      // stdout => output.txt
    exec("ls");   // ls prints to stdout which is now output.txt
}
```

Fork: copies parent's memory, sets up appropriate translation table

Exec: gets rid of current VAS, set up a new VAS & for the process

[highly inefficient!]

→ Copy-on-write (cow)

hw detects perm violations

→ 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)