Last day of multiprocessing
./guitar example

recall blockMonitoredSet(monitoredSignals); — tell OS "I will handle these particular signals using SIGWAIT

```
static void reapChildProcesses(set<pid_t>& processes) {
  while (true) {
     pid_t pid = waitpid(-1, NULL, WNOHANG);
     if (pid <= 0) break;
     processes.erase(pid);
  }
}
```

```
static void stopPlaying(const set<pid_t>& processes) {
   for (int pid: processes) kill(pid, SIGKILL);
   setAlarm(0); // disable all future-scheduled alarms
}
```

signal blocks preserved over both fork & execvp boundaries!! Remember to unblockMonitoredSet(monitoredSignals);

Virtual Memory

Main memory (RAM) is organized array of contiguous bytes
- physical hardware
- physical addressing: addrs generated & dereferenced by the OS mapped to data at same address (the actual one)

$\downarrow$

- virtual addressing: to allow true multiprocessing
  - all processes operate as if they own all of main memory
  - CPU+OS treat processes' addresses as virtual, and translate to a physical addr before accessing RAM
    - address mapping has to happen w/in the same clock cycle — super fast!
      - translation has to happen using bitwise and/or/left shift/right shift
  - support for virtual memory has similar scheme to file systems (slides are very informative here)
    - uses "page" instead of blocks — multiples of 4096 (meaning everything a multiple of 0x000 (ends w/ this))
    - mapping scheme is simpler — OS stores page table of virtual → physical page mappings

view the hard drive as physical memory
- ↳ & main memory as a kind of cache
- executables stored on disk & loaded to main memory when needed
- ↳ OS will evict pages to the hard drive if they become inactive (in swap files)