The Process

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Exam flavor?
  - Is 50 minutes enough?
  - Two mid-terms? Evening exam?
- Concurrency expertise?
  - Monitor? P()/V()? Mutex? Condition?
- Anybody reading comp.risks?
- Today
  - Chapter 4, more or less

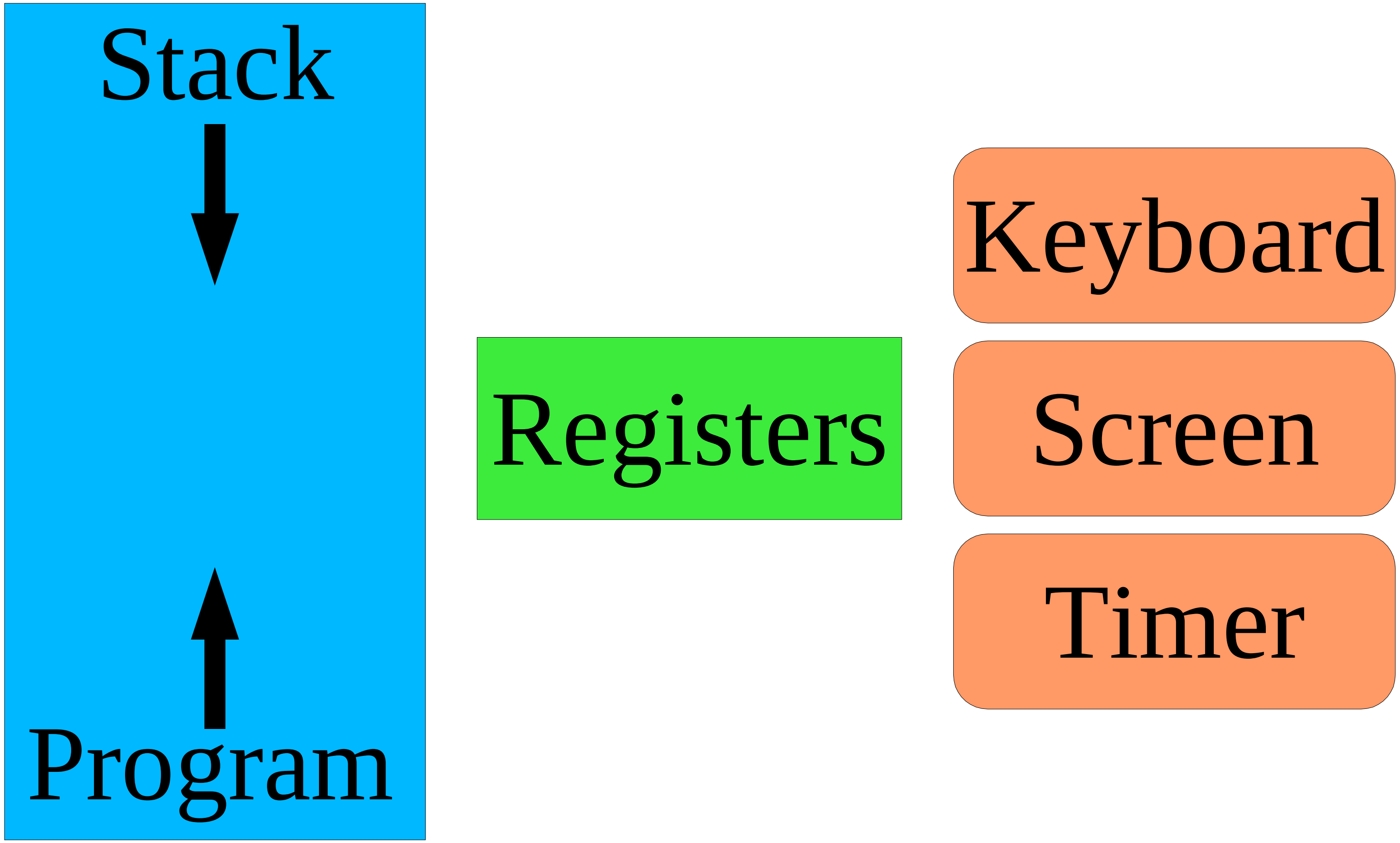
How's it going?

- You should have tried simics
  - (really)
- Just do something
  - Put some characters somewhere on the screen
  - Then loop forever
- Weekends are fine
  - but please don't skip this one!

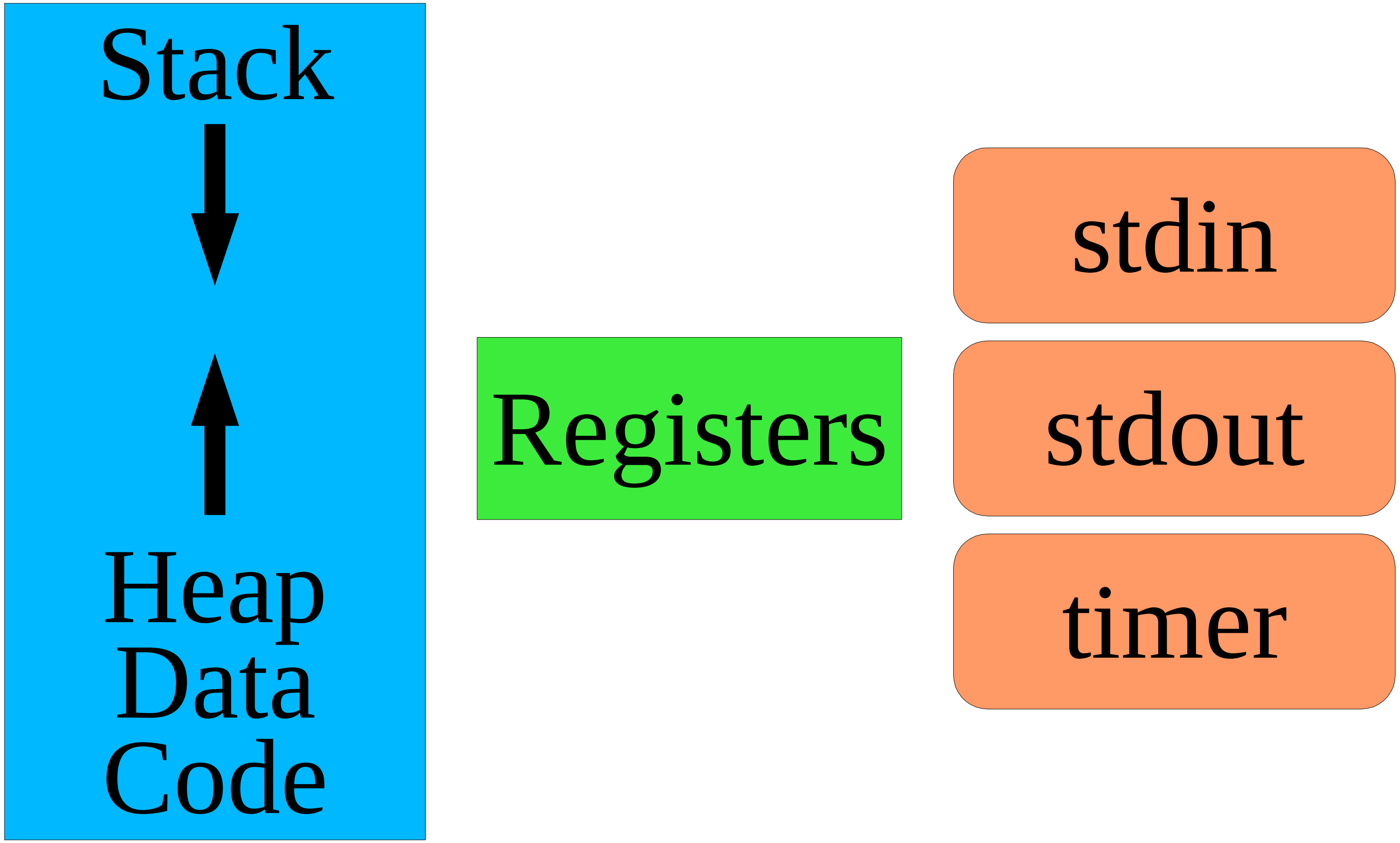
Outline

- Process as pseudo-machine
  - (that's ***all*** there is)
- Process life cycle
- Process kernel states
- Process kernel state

The Computer



The Process



Stack

Heap
Data
Code

Registers

stdin

stdout

timer

Process life cycle

- Birth
  - (or, well, fission)
- School
- Work
- Death
- (Nomenclature courtesy of The Godfathers)

Birth

- Where do new processes come from?
  - (Not: under a cabbage leaf, by stork, ...)
- What do we need?
  - Memory contents
  - CPU register contents (all N of them)
  - "I/O ports"
  - File descriptors
  - Hidden stuff (timer state, current directory, umask)

Birth

- Intimidating?
- How to specify all of that stuff?
  - What is your {name,quest,favorite_color}?
- Gee, we already have ***one*** process we like...

Birth - fork()

- Memory
  - Copy all of it
  - Maybe using VM tricks so it' s cheaper
- Registers
  - Copy all of them
    - All but one: parent learns child's process ID, child gets 0

Birth - fork()

- File descriptors
  - Copy all of them
  - Can't copy the ***files!***
  - Copy ***references*** to open-file state
- Hidden stuff
  - Do whatever is "obvious"

Now what?

- Two copies of the same process is ***boring***
- Transplant surgery!
  - Implant new memory!
    - New program text
  - Implant new registers!
    - Old ones don't point well into the new memory
  - Keep (most) file descriptors
    - Good for cooperation/delegation

Now what?

- Hidden state?
  - Do what's “obvious”
- What do we call this procedure?

```
int execve(
  char *path,
  char *argv[ ],
  char *envp[ ])
```

Birth - other ways

- There is another way
  - Well, two
- spawn()
  - Carefully specify all features of new process
  - Don't need to copy stuff you will immediately toss
- Plan 9 rfork() / Linux clone()
  - Build new process from old one
  - Specify which things get shared vs. copied

School

- Old process called

```
execve(
  char *path,
  char *argv[ ],
  char *envp[ ]);
```

- Result is

```
char **environ;
main(int argc, char *argv[ ]) {
  ...
}
```

School

- How does the magic work?
  - ***15-410 motto: No magic***
- Kernel process setup
  - Toss old data memory
  - Toss old stack memory
  - Load executable file
  - and...

The stack!

- Kernel builds stack for new process
  - Transfer argv[] and envp[] to top of new process stack
  - Hand-craft stack frame for ~main()
  - Set registers
    - stack pointer (to top frame)
    - program counter (to start of ~main())
      - (What's a ~main()?)

The mysterious ~main()

- What's in a name?
  - may be ~main(), _main(), @main()
  - Any illegal name will do
- ~main(argc, argv, envp)

```
environ = envp;
exit(main(argc, argv));
```

- Where does ~main() come from?
  - .../.../crt0.o

Work

- Process states
  - Running
    - user mode
    - kernel mode
  - Runnable
    - user mode
    - kernel mode
  - Sleeping
    - in condition_wait(), more or less

Work

- Other process states
  - Forking
  - Zombie
- “Exercise for the reader”
  - Draw the state transition diagram

Death

- Voluntary

```
void exit(int reason);
```

- Software exception
  - SIGXCPU - used "too much" CPU time
- Hardware exception
  - SIGSEGV - no memory there for you!

Death

- kill(pid, sig);
  - ^C ⇒ kill(getpid(), SIGINT);
  - Start logging

    ```
    kill(daemon_pid, SIGUSR1);
    % kill -USR1 33
    ```
  - Lost in Space

    ```
    kill(Will_Robinson, SIGDANGER);
    ```
    - I apologize to IBM for lampooning their serious signal
    - No, I apologize for that apology...

Process cleanup

- Resource release
  - Open files: close()
    - TCP: 2 minutes (or more)
    - Solaris disk offline - forever ("***None*** shall pass!")
  - Memory: release
- Accounting
  - Record resource usage in a magic file
- Gone?

“All You Zombies”

- Zombie process
  - Process state reduced to exit code
  - Wait around until parent calls wait()
    - Copy exit code to parent memory
    - Delete PCB

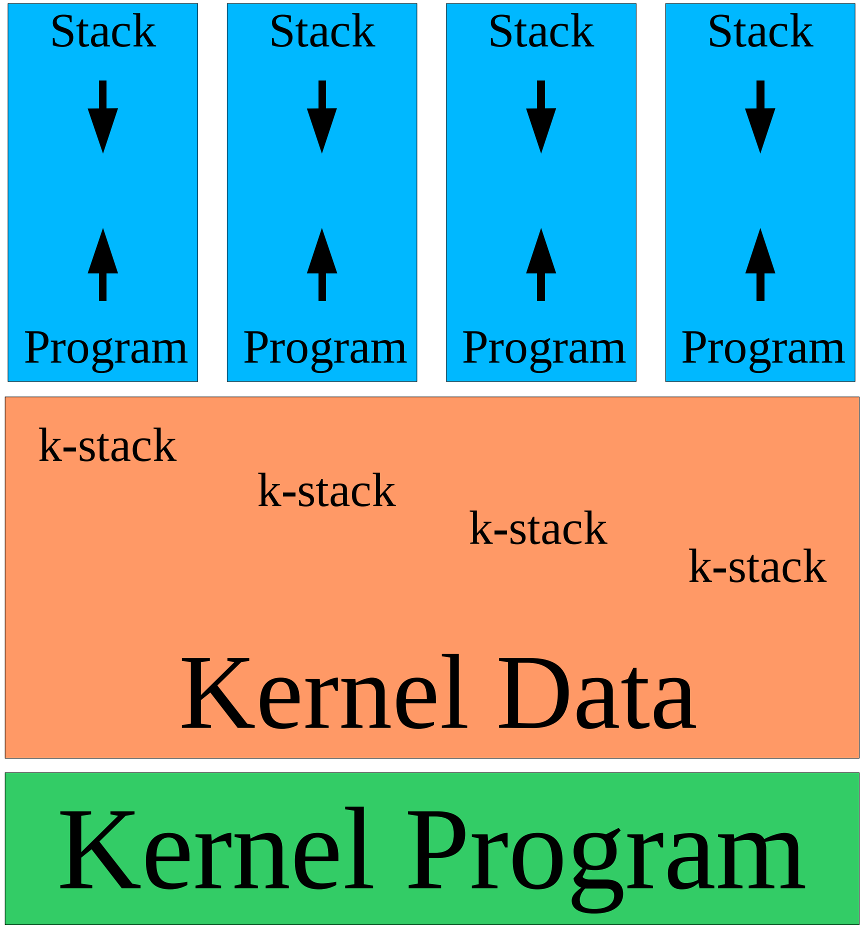
Kernel process state

- The dreaded "PCB"
  - (poly-chloro-biphenol?)
- Process Control Block
  - “Everything without a memory address”

Sample PCB contents

- CPU register save area
- Process number, parent process number
- Countdown timer value
- Memory segment info
  - User memory segment list
  - Kernel stack reference
- Scheduler info
  - linked list slot, priority, “sleep channel”

The Big Picture



Ready to start?

- Not so complicated...
  - getpid()
  - fork()
  - exec()
  - exit()
  - wait()
- What could possibly go wrong?