The Process

David A. Eckhardt
School of Computer Science
Carnegie Mellon University

de0u@andrew.cmu.edu

How’s it going?

You should have tried simics

- (really)

How about a blob?

- Put some characters somewhere on the screen
- Then loop forever

Weekends are fine

- but please don’t skip this one!

Polls

- Concurrency expertise: Monitor? P()/V()? Mutex? Condition?
- Anybody reading comp.risks?
  - Hmm... theoretically *possible* to do an exam question...

Outline

Lecture versus book

- Parts of Chapter 3
- Most of Chapter 4

Process as pseudo-machine

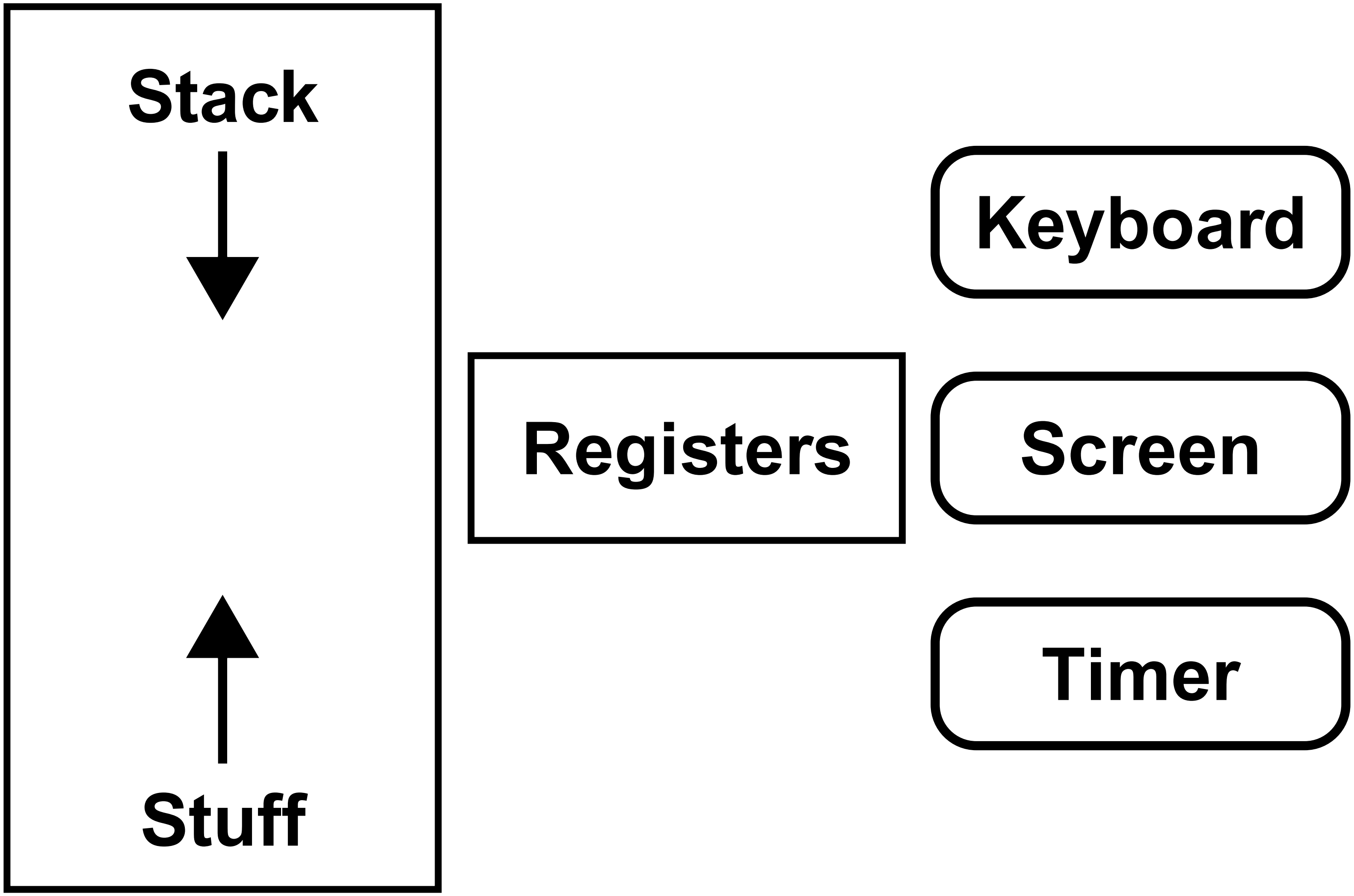
- (that's *all* there is)

Process life cycle

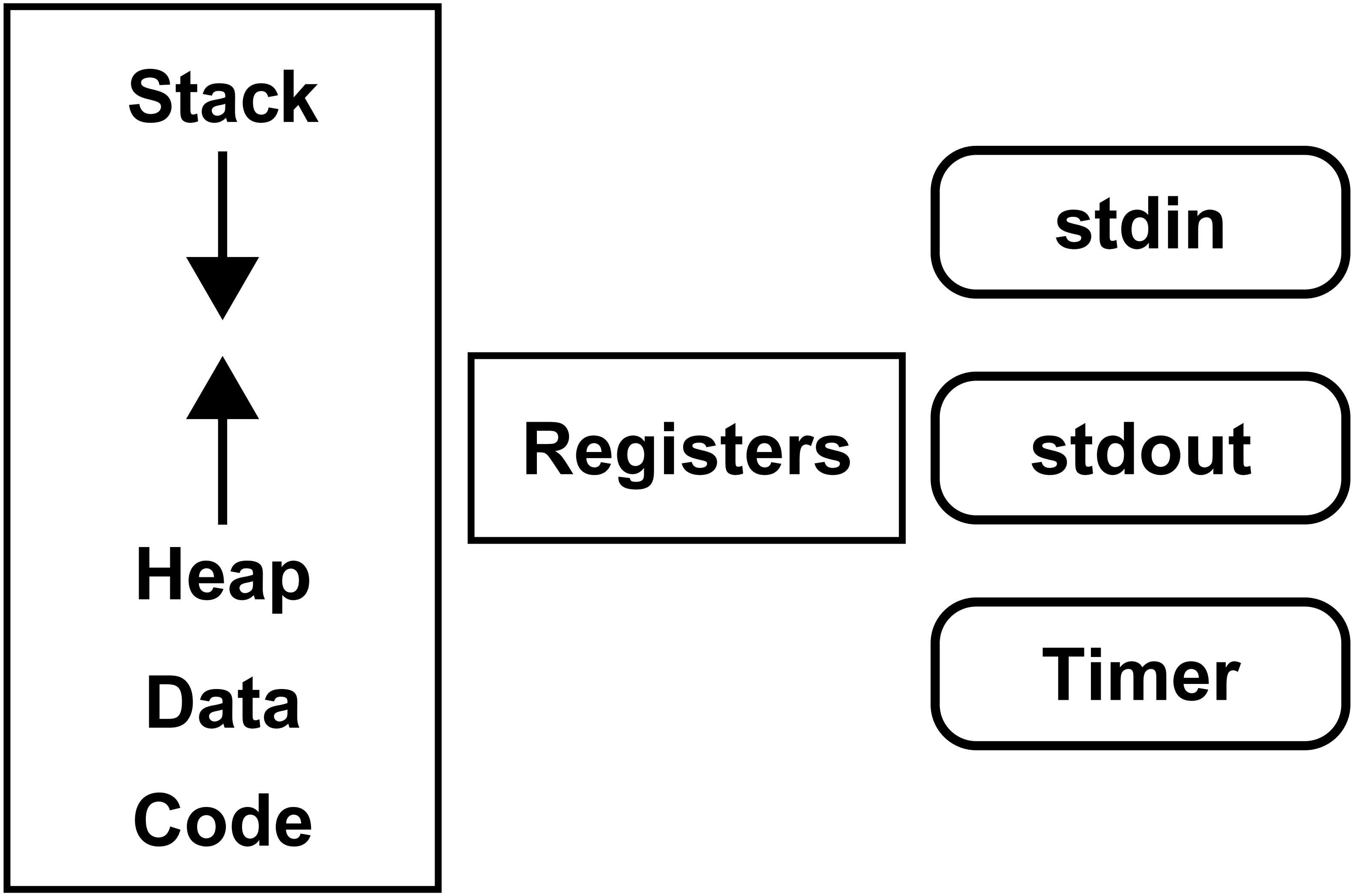
Process kernel states

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)

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

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
- Hidden state?
  - Do whatever is "obvious"

What do we call this procedure?

- 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

rfork() (Plan 9), clone() (Linux)

- build new process from old one
- specify which things get shared vs. copied

School

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

- becomes

char **environ;

main(int argc, char *argv[ ])

- How does the magic work?

Kernel process setup

- Toss old data memory (or page references)
- 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())

_:~:_main(argc, argv, envp)

- (not its real name: _main(), ~main())
- 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()
- [Forking]
- Zombie

Exercise for the reader

- Draw the transition diagram

Death

exit(reason);

Software exception

- SIGXCPU - used “too much” CPU time

Hardware exception

- SIGSEGV - no memory there for you!

kill(pid, sig);

- ^C - kill(getpid(), SIGINT);
- start logging - kill(daemon_pid, SIGUSR1);
- Lost in Space - kill(Will_Robinson, SIGDANGER);
    - I apologize to IBM for lampooning their serious signal
    - No, I apologize for the 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

Zombie

- process state reduced to exit code
- wait around until parent calls wait()

Kernel process state

The dreaded "PCB" (poly-chloro-biphenol?)

- Process Control Block

The laundry list

- CPU register save area
- process "identifier" (number), parent process identifier
- countdown timer value
  - (maybe: in-line linked list)
- memory segment info
  - user memory segment list
  - kernel stack reference
- scheduler info
  - linked list slot
  - priority
  - kernel sleep "channel" (condition variable)

Ready to start?

Not so complicated...

- fork()
- exec()
- getpid()
- exit()
- wait()

What could possibly go wrong?