15-410

"Now I know why they call it a 'cursor'"

The Process
Sep. 13, 2004

Dave Eckhardt

Bruce Maggs

Synchronization

Project 1

- Hope you've run simics by now!
- “End of today”
  - paint character on screen
  - position cursor

Anybody reading comp.risks?

This lecture

- Chapter 4, but not exactly!

Outline

Process as pseudo-machine

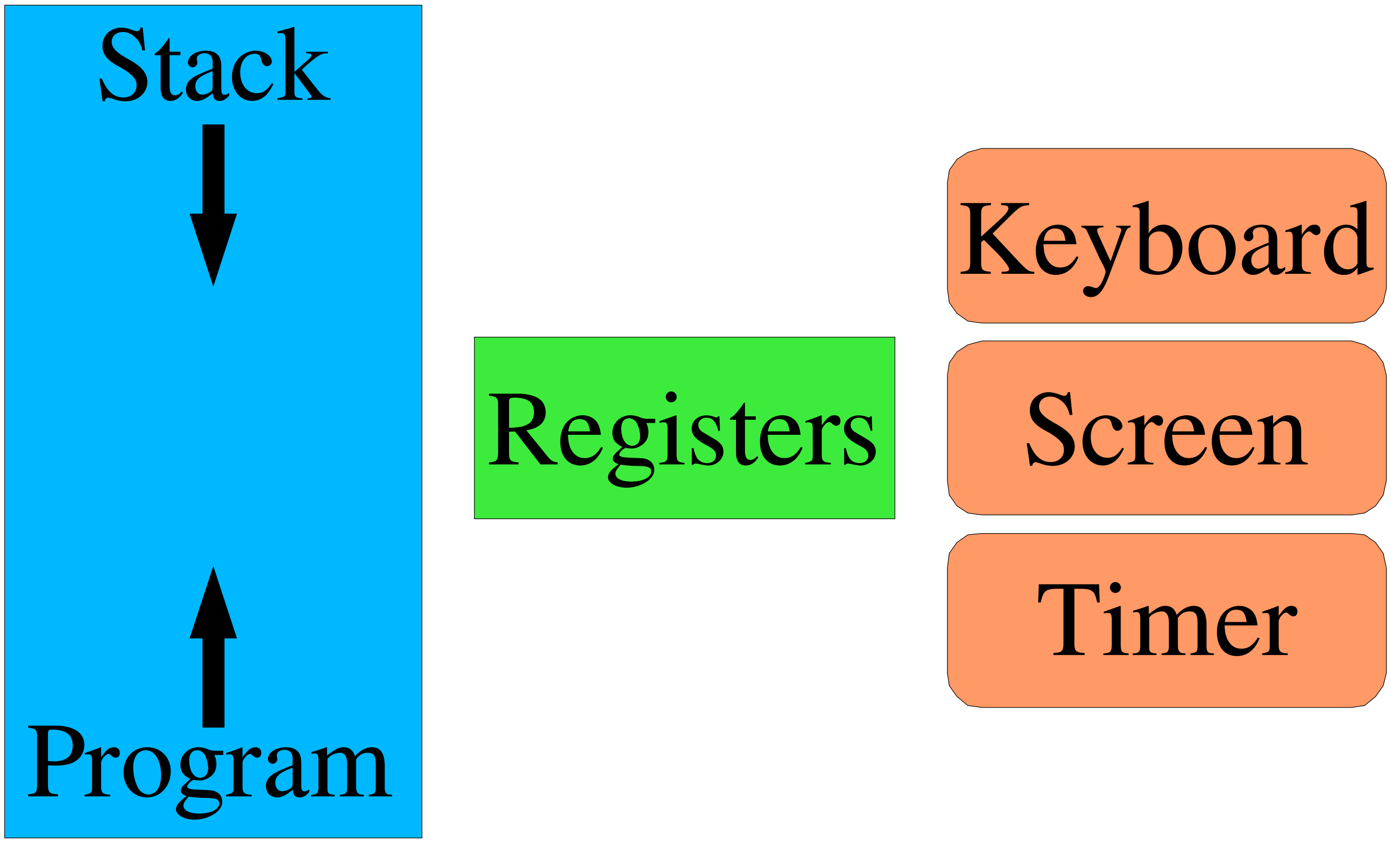
- (that's *all* there is)

Process life cycle

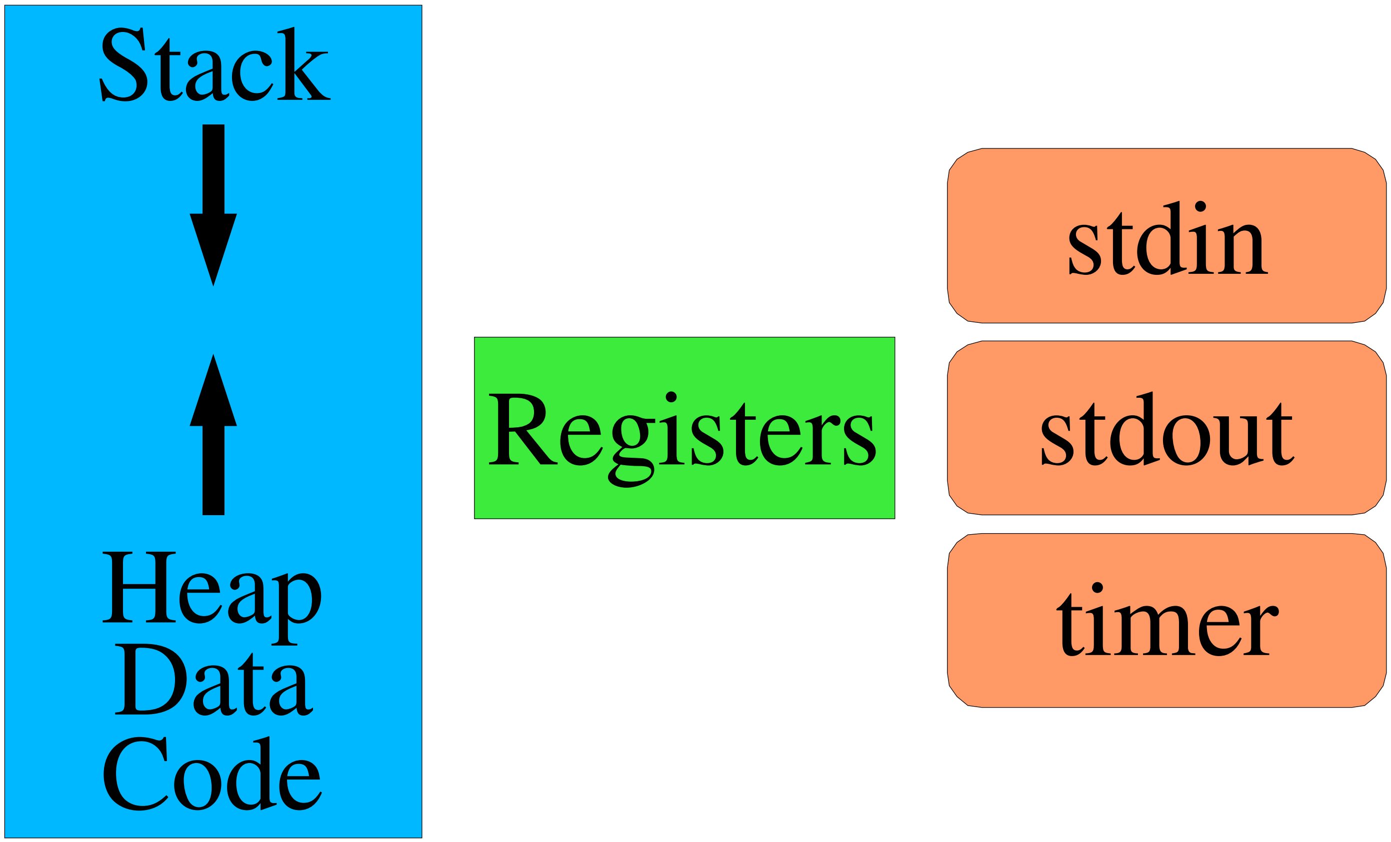
Process kernel states

Process kernel state

The Computer



The Process



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
  - Text, data, stack
- CPU register contents (N of them)
- "I/O ports"
  - File descriptors, e.g., stdin/stdout/stderr
- 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...

- Maybe we could use its settings to make a new one...
- Birth via "cloning"

Birth – fork() - 1

"fork" - Original Unix process creation system call

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() - 2

File descriptors

- Copy all of them
- Can't copy the *files!*
- Copy *references* to open-file state

Hidden stuff

- Do whatever is "obvious"

Result

- Original, “parent”, process
- Fully-specified “child” process, with 0 fork() parameters

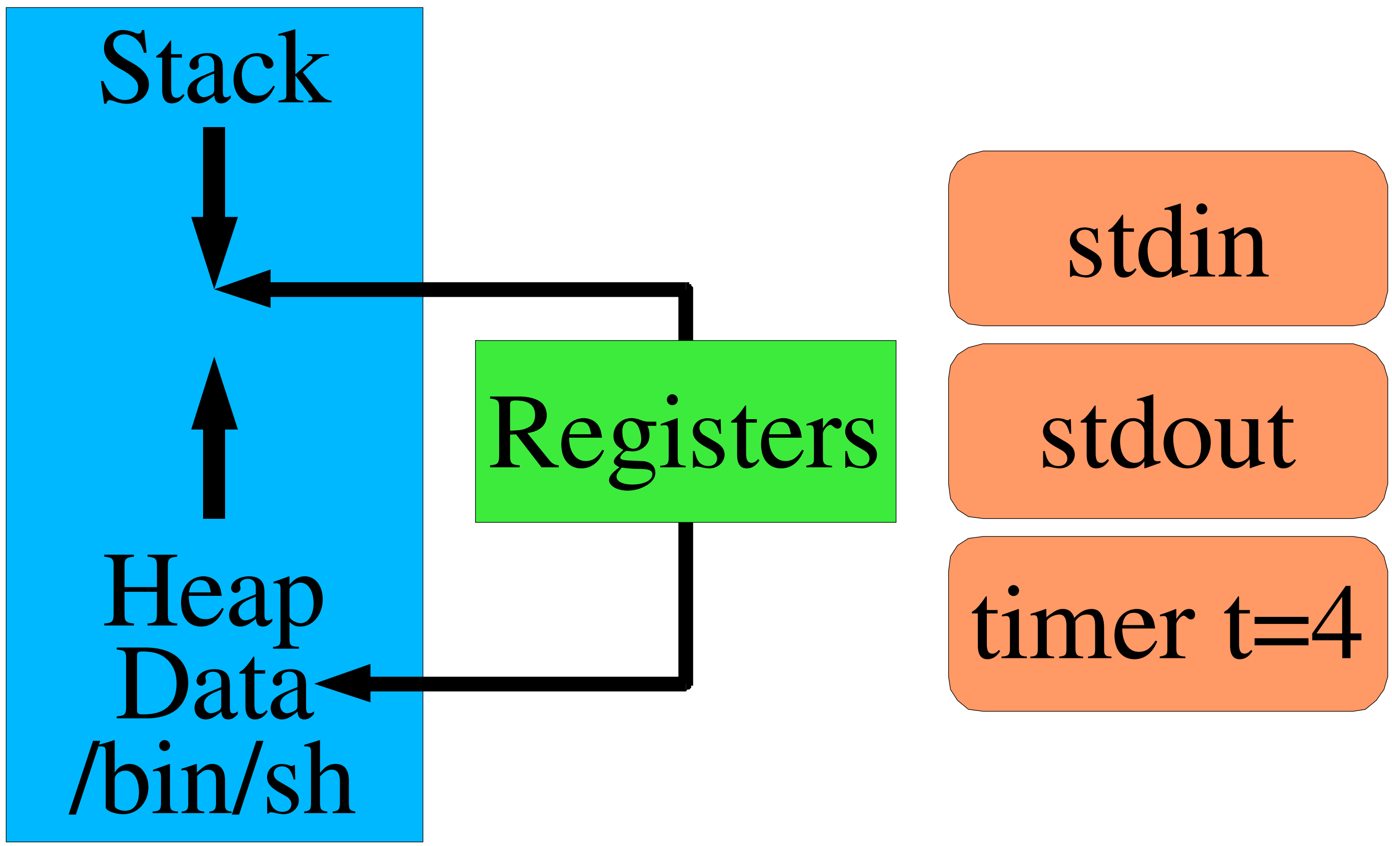
Now what?

Two copies of the same process is *boring*

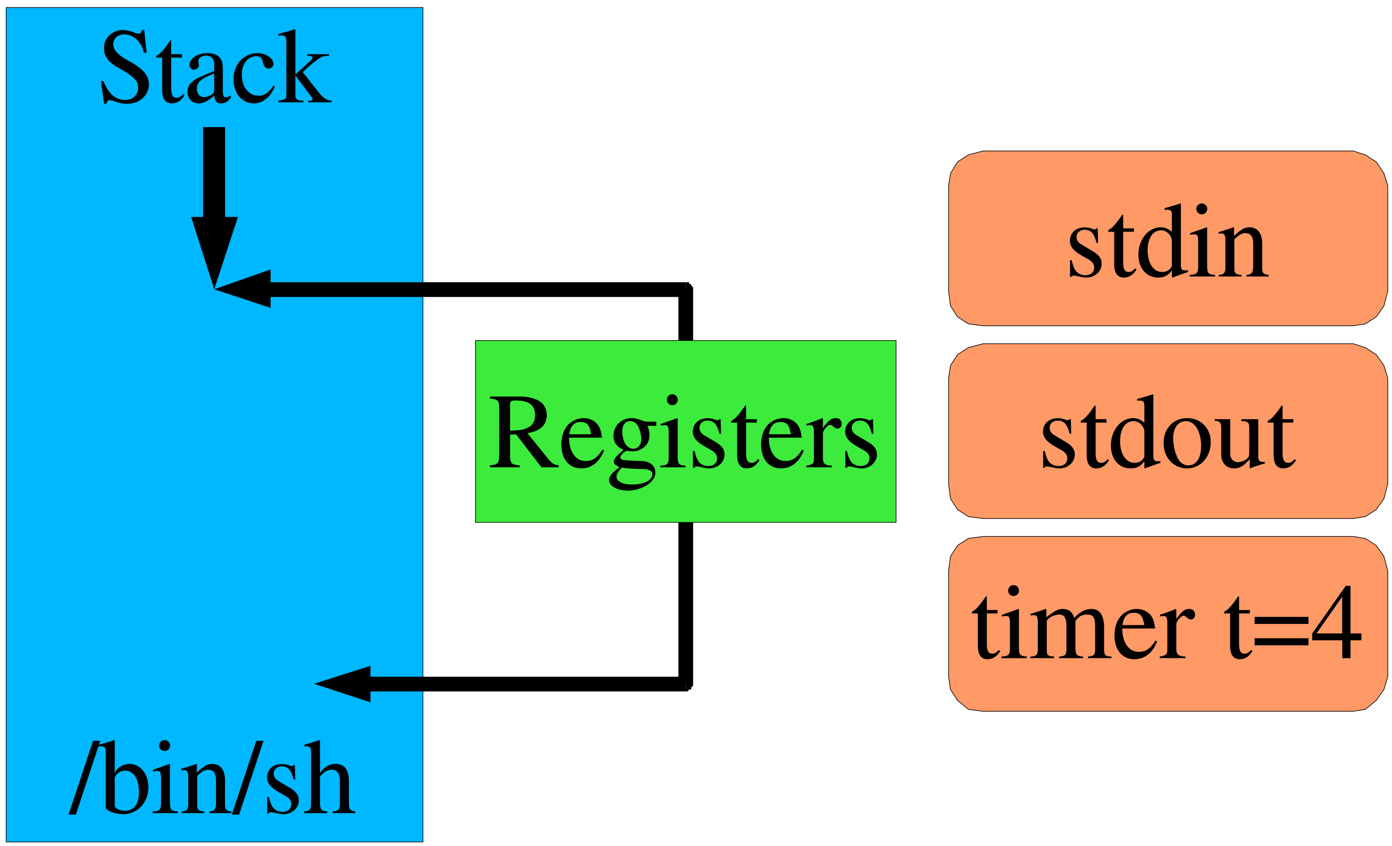
Transplant surgery!

- 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 what's “obvious”

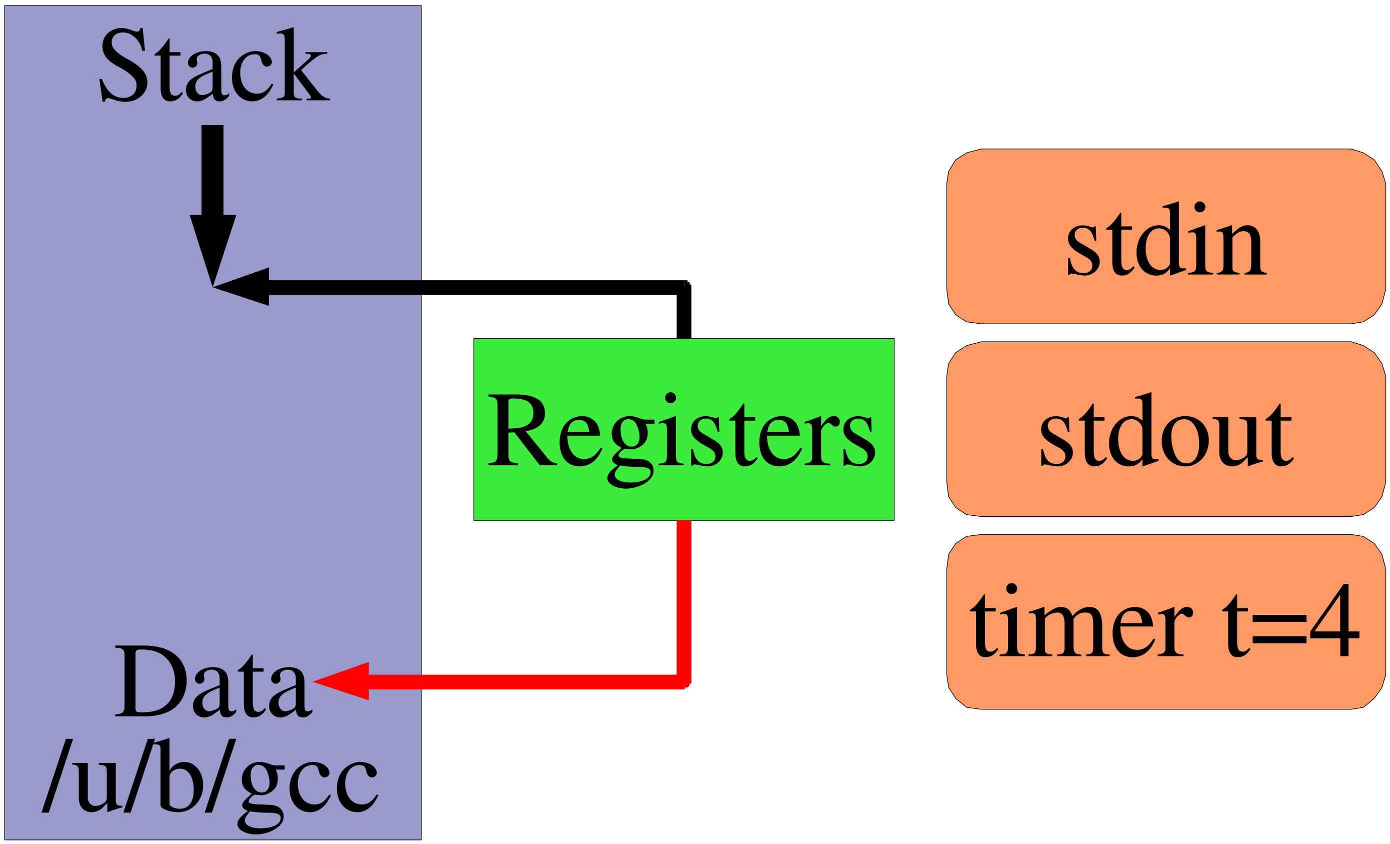
Original Process



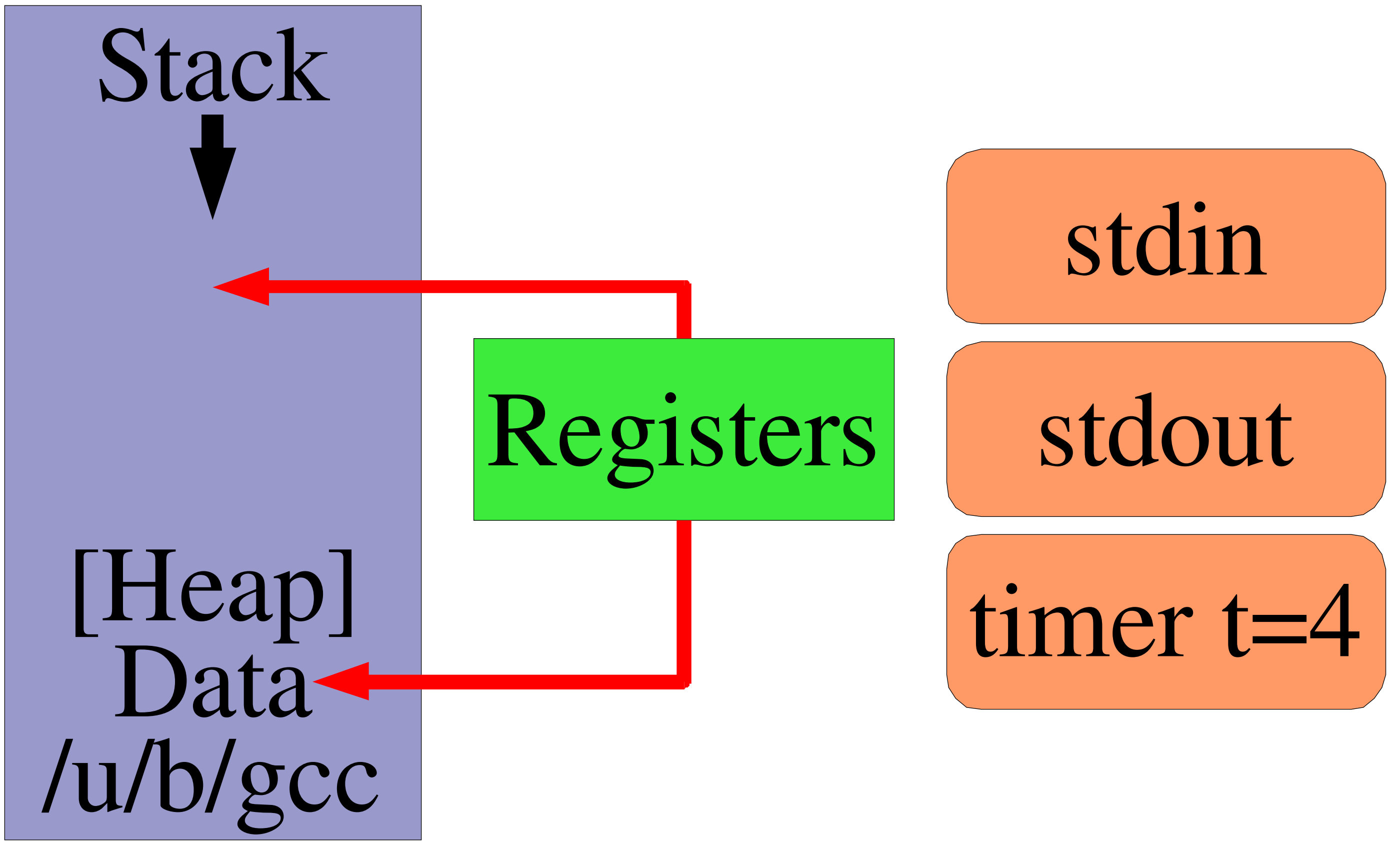
Toss Heap, Data



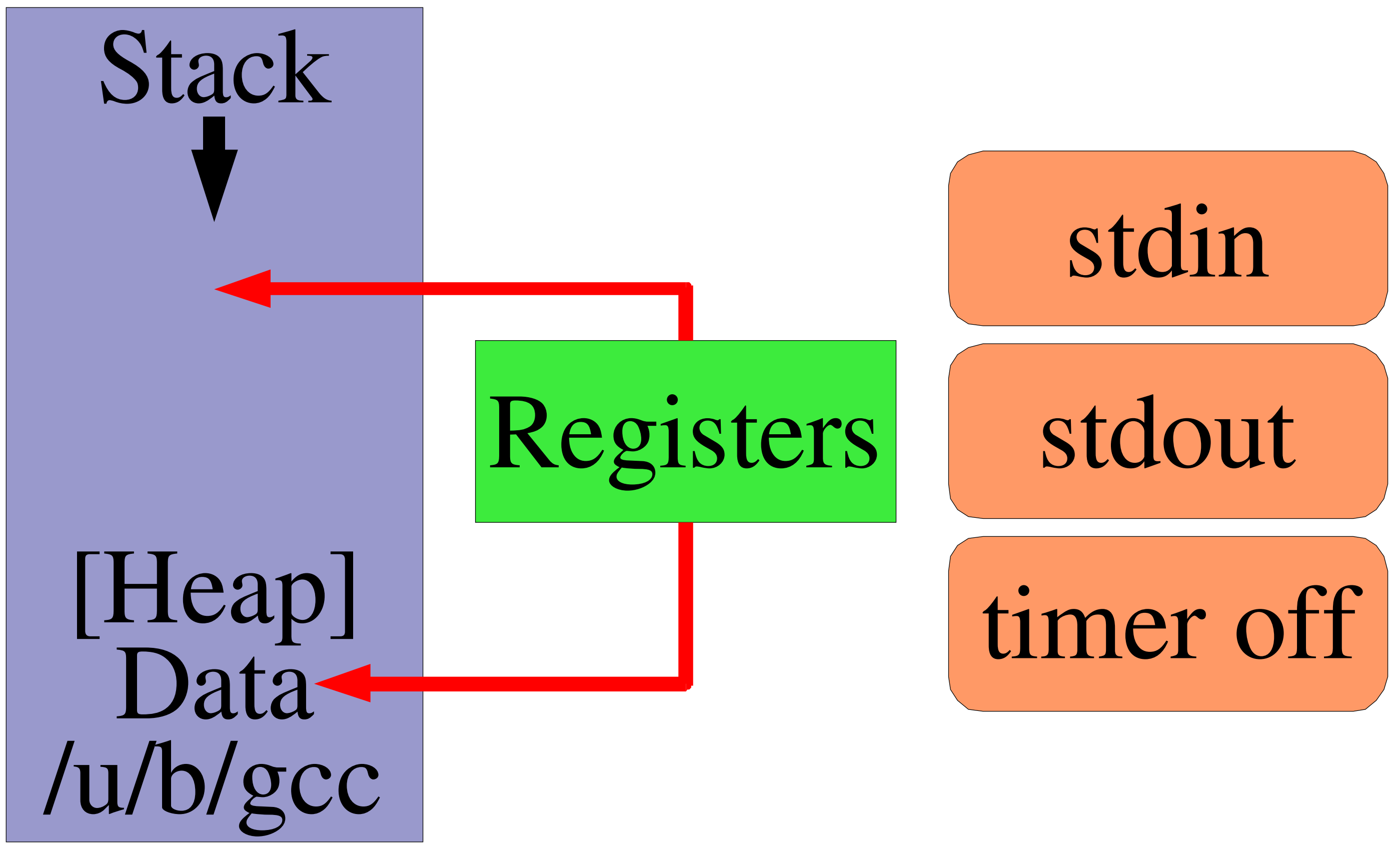
Load New Code, Data From File



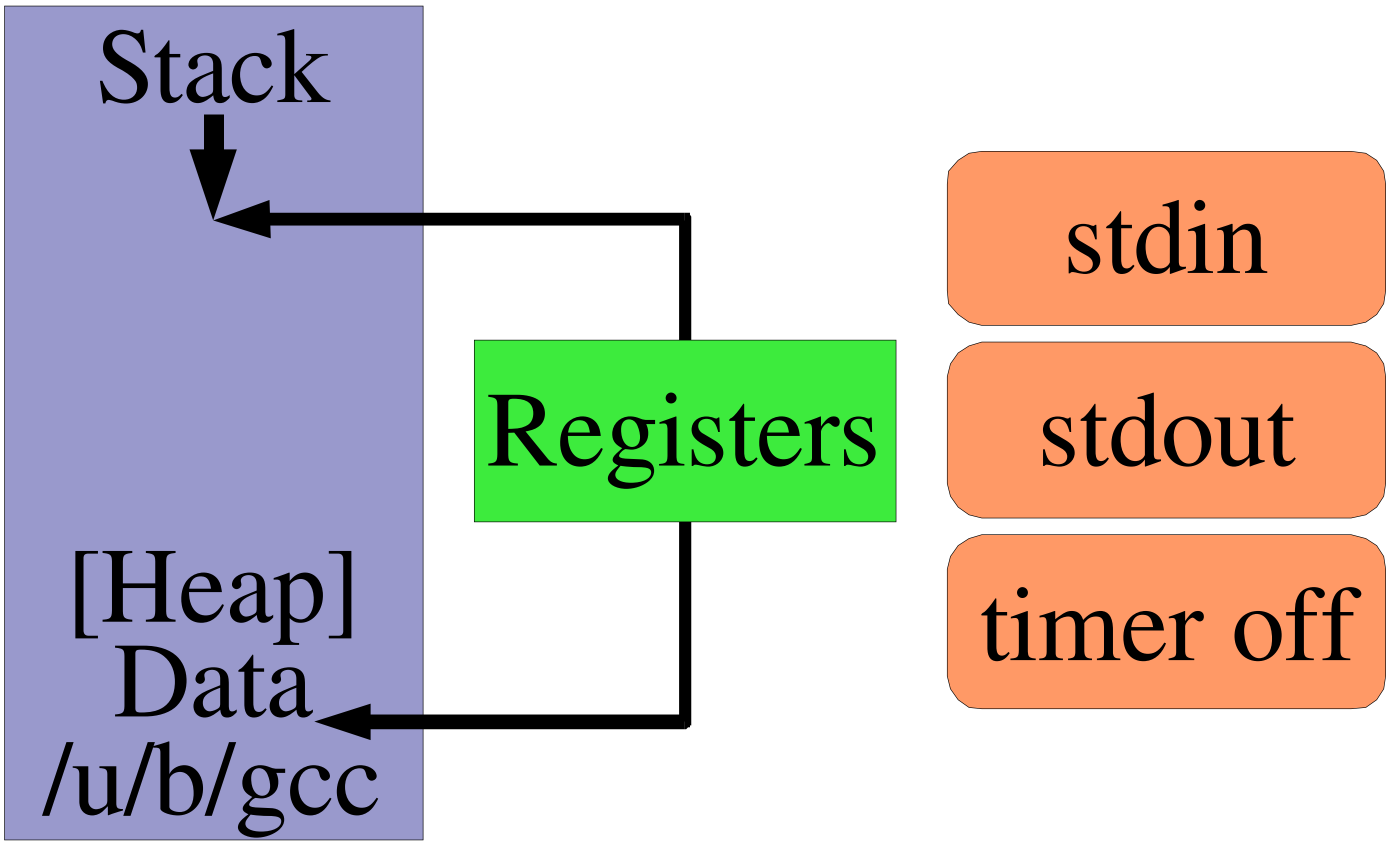
Reset Stack, Heap



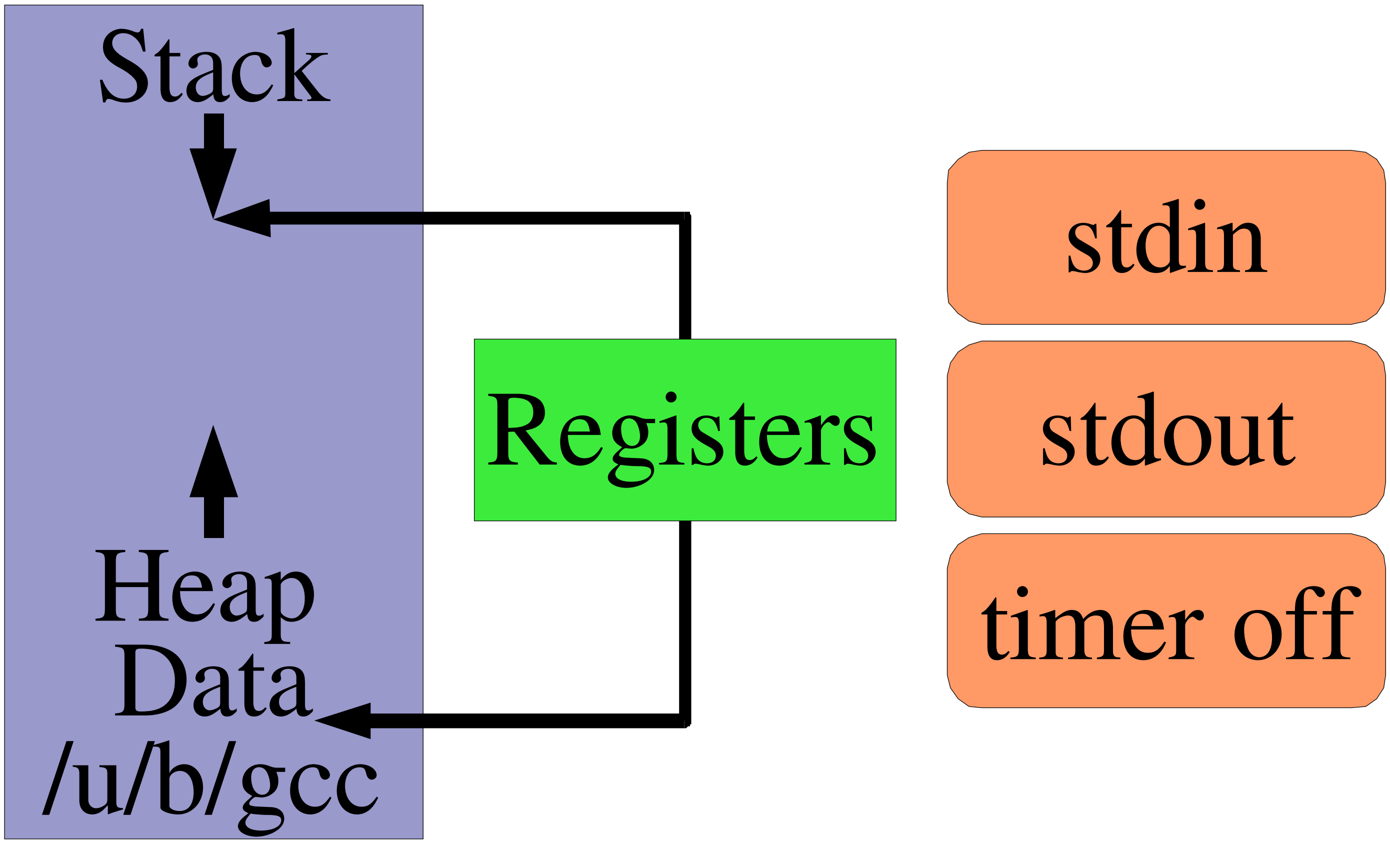
Fix "Stuff"



Initialize Registers



Begin Execution



What's The Implant Procedure Called?

```
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
  - Complicated
- Win: 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
  - "Copy memory, share files, copy environment, share ..."

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: we saw...

- **Toss old data memory**
- **Toss old stack memory**
- **Load executable file**

Also...

The Stack!

Kernel builds stack for new process

- Transfers argv[] and envp[] to top of new process stack
- Hand-crafts stack frame for __main()
- Sets registers
  - Stack pointer (to top frame)
  - Program counter (to start of __main())

Work

Process states

- Running
  - User mode
  - Kernel mode
- Runnable
  - User mode
  - Kernel mode
- Sleeping
  - "Blocked" awaiting some event
  - Not run by scheduler
  - Q: Is this user mode or kernel mode?

Work

Other process states

- Forking
  - Probably obsolete, once used for special treatment
- Zombie
  - Process has called exit(), parent hasn't noticed yet

"Exercise for the reader"

- Draw the state transition diagram

Death

Voluntary

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

Hardware exception

- SIGSEGV - no memory there for you!

Software exception

- SIGXCPU – used "too much" CPU time

Death

kill(pid, sig);

- `keyboard` **^C** $\Longrightarrow$

  `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
- Waits around until parent calls wait()
    - Copies exit code to parent memory
    - Deletes PCB

Kernel process state

The dreaded "PCB"

- (polychlorinated biphenol?)

Process Control Block

- "Everything without a user-visible memory address"
  - Kernel management information
  - Scheduler state
  - The "stuff"

Sample PCB contents

Pointer to CPU register save area

Process number, parent process number

Countdown timer value

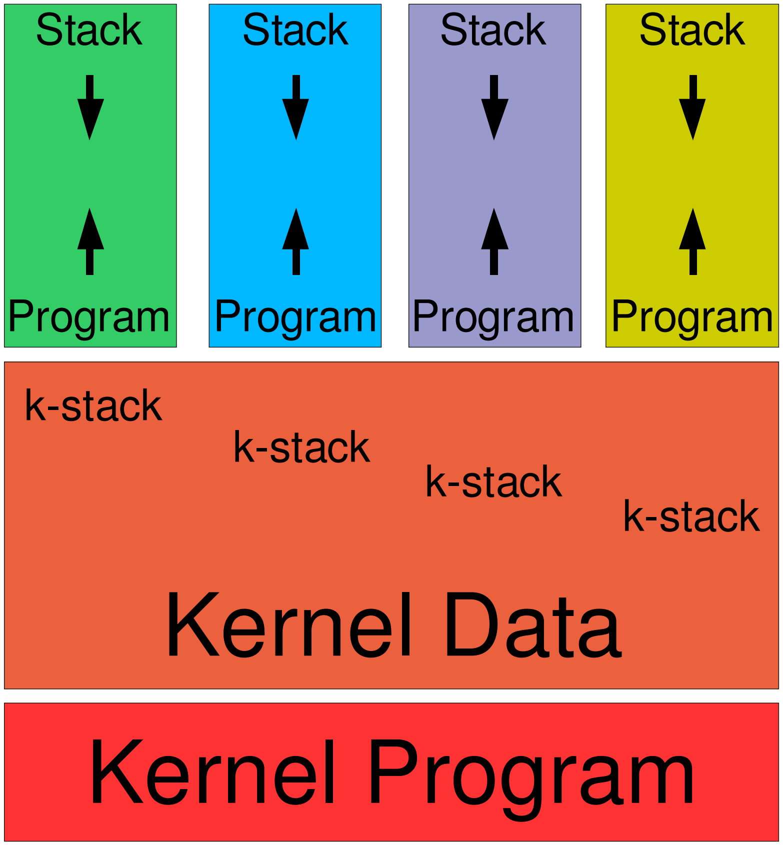
Memory segment info

- User memory segment list
- Kernel stack reference

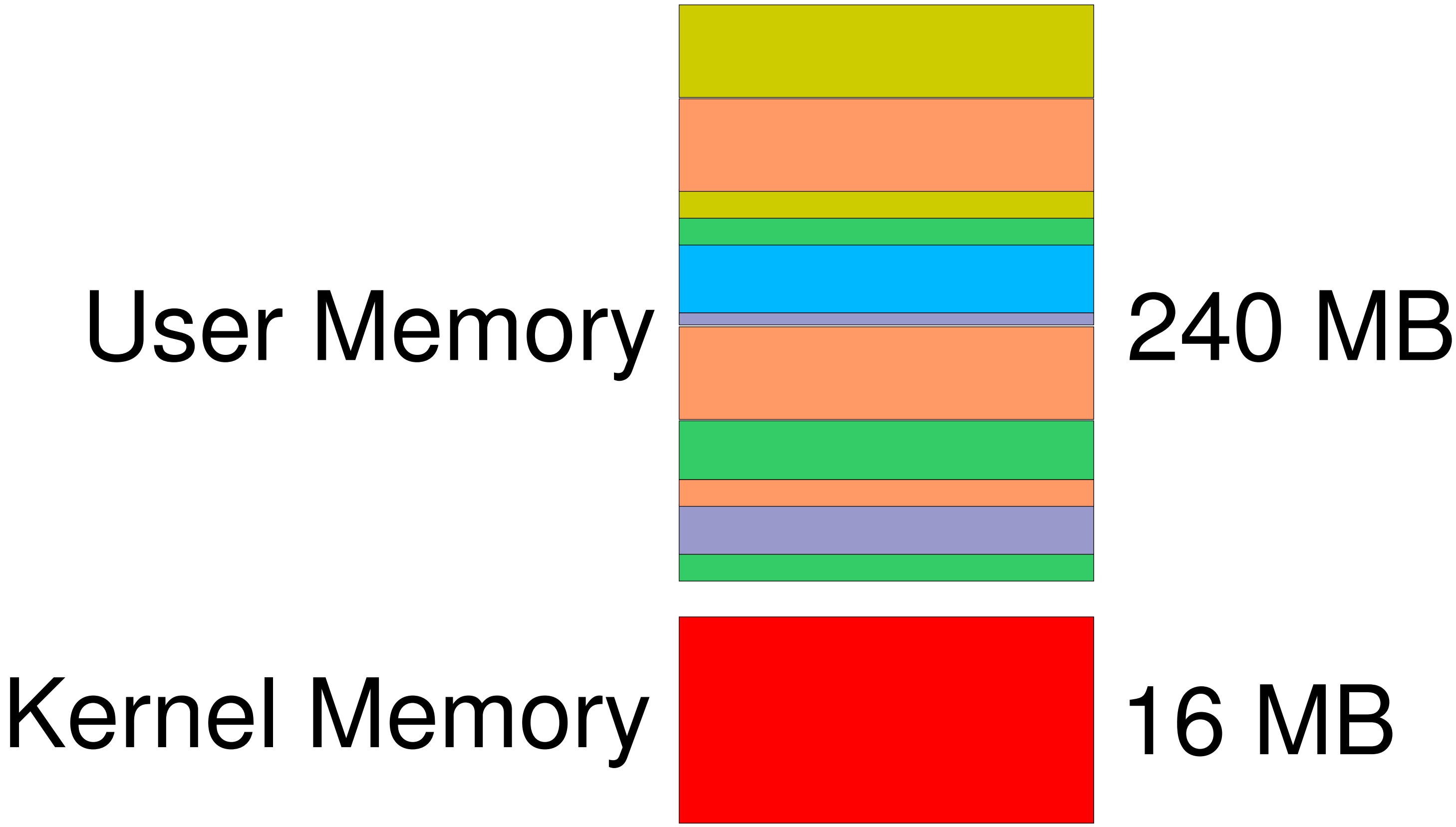
Scheduler info

- linked list slot, priority, “sleep channel”

15-410 Virtual Memory Layout



15-410 Physical Memory Layout



Ready to Implement All This?

Not so complicated...

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

What could possibly go wrong?

Summary

Parts of a Process

- **Physical – Memory pages, registers, I/O devices**
- **Virtual – Memory regions, registers, I/O "ports"**

Birth, School, Work, Death

"Big Picture" of system memory – both of them

- **(Numbers & arrangement are 15-410–specific)**