# 15-410

***"System call abuse for fun & profit"***

## The Process
## Jan. 21, 2004

**Dave Eckhardt**

**Bruce Maggs**

# Synchronization

**Project 0 due at midnight**

- **Please go through the hand-in page *now***

**Anybody reading comp.risks?**

**Today**

- **Chapter 4, but not exactly!**

# Outline

**Process as pseudo-machine**

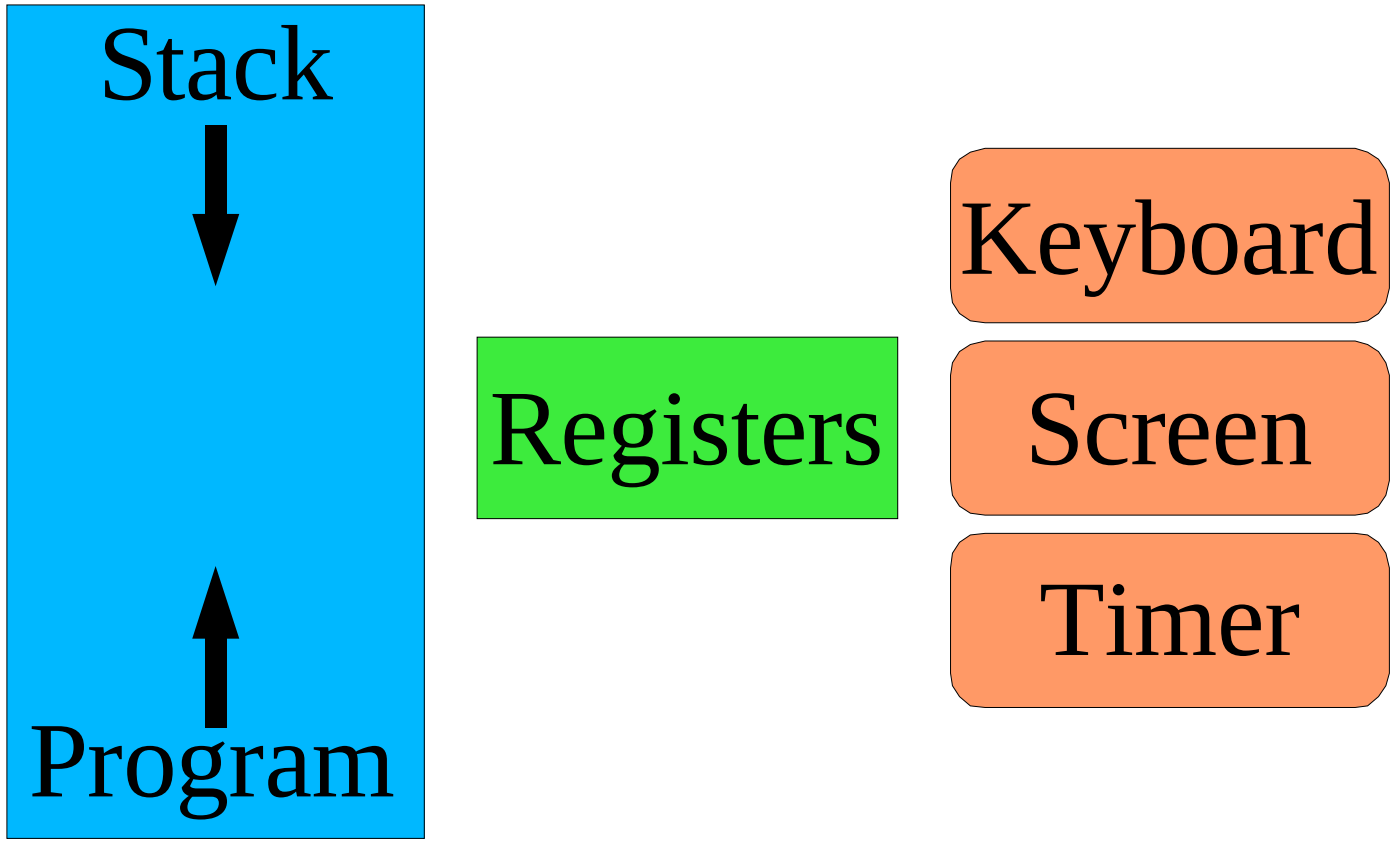
- **(that's *all* there is)**

**Process life cycle**

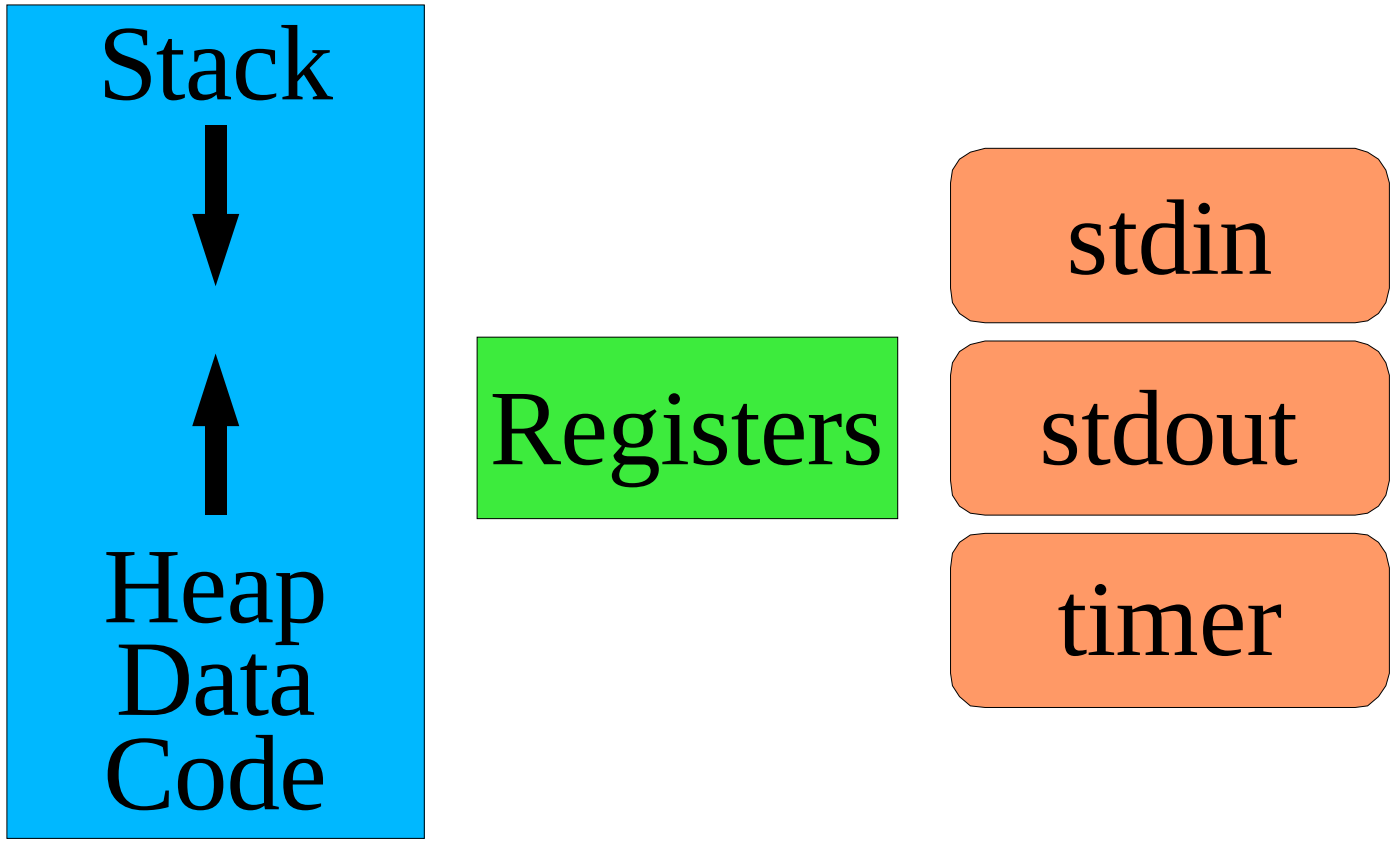
**Process kernel states**

**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...**

# Birth – fork() - 1

**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

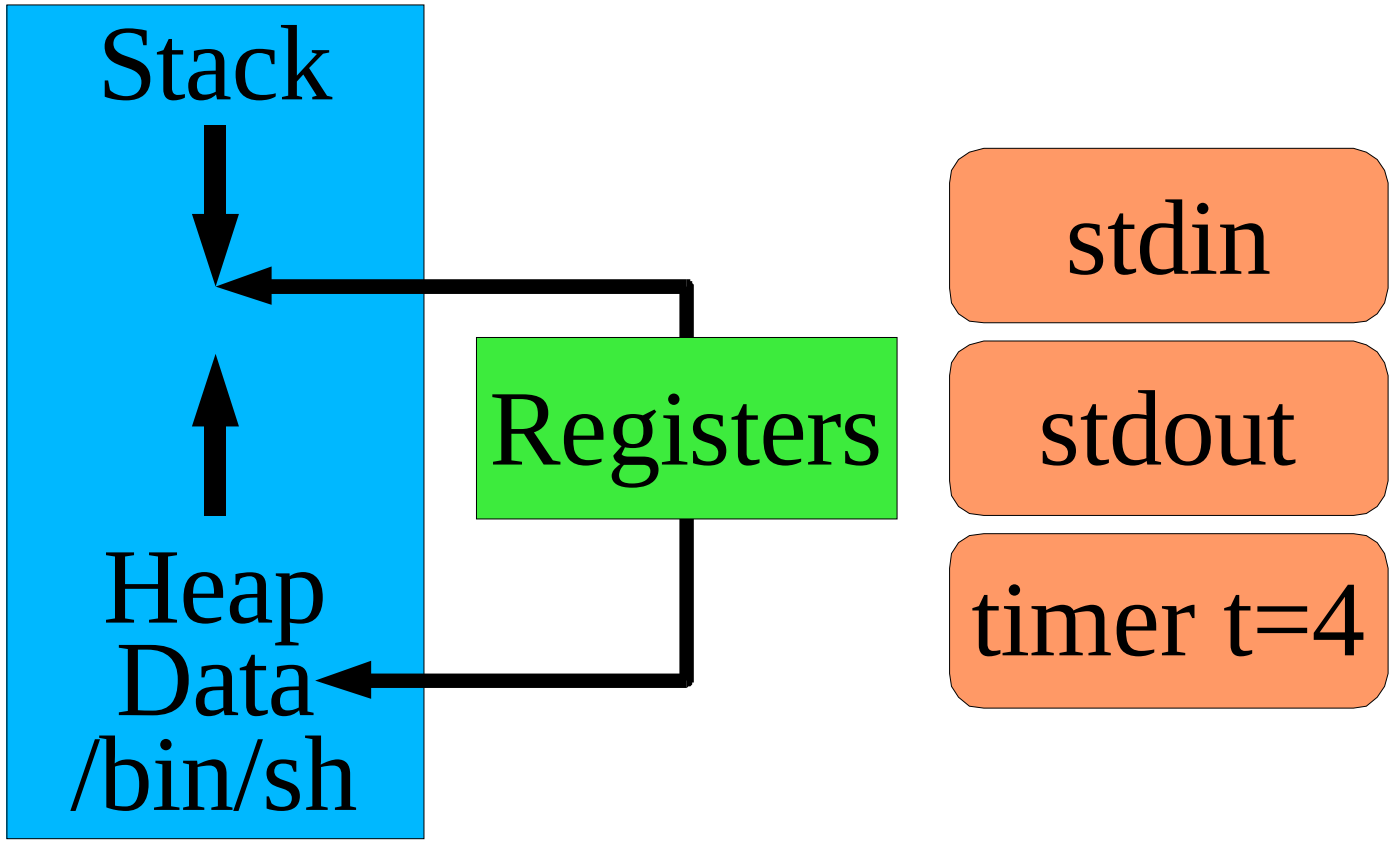
# Now what?

**Two copies of the same process is *boring***

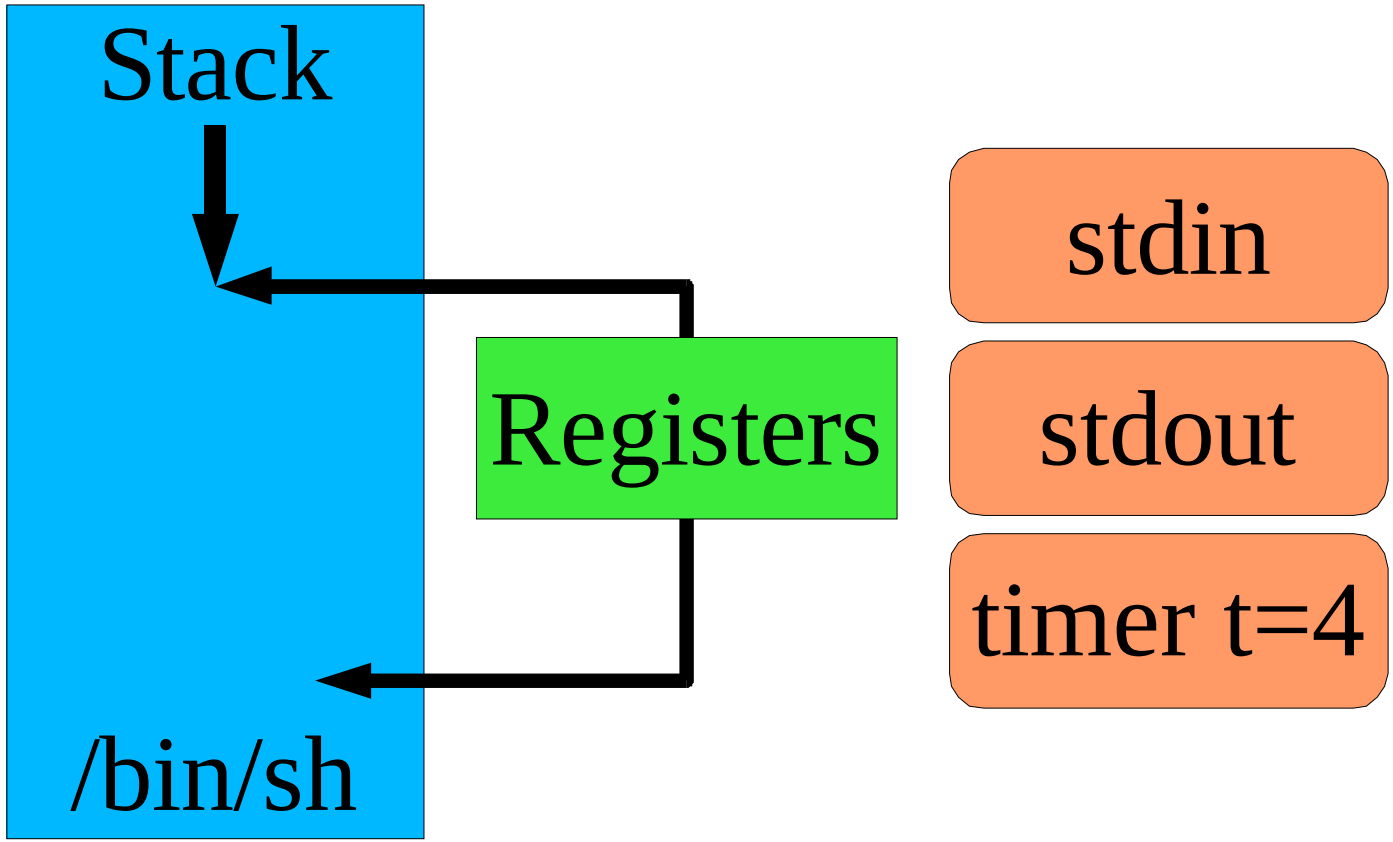
**Transplant surgery!**

- **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”**

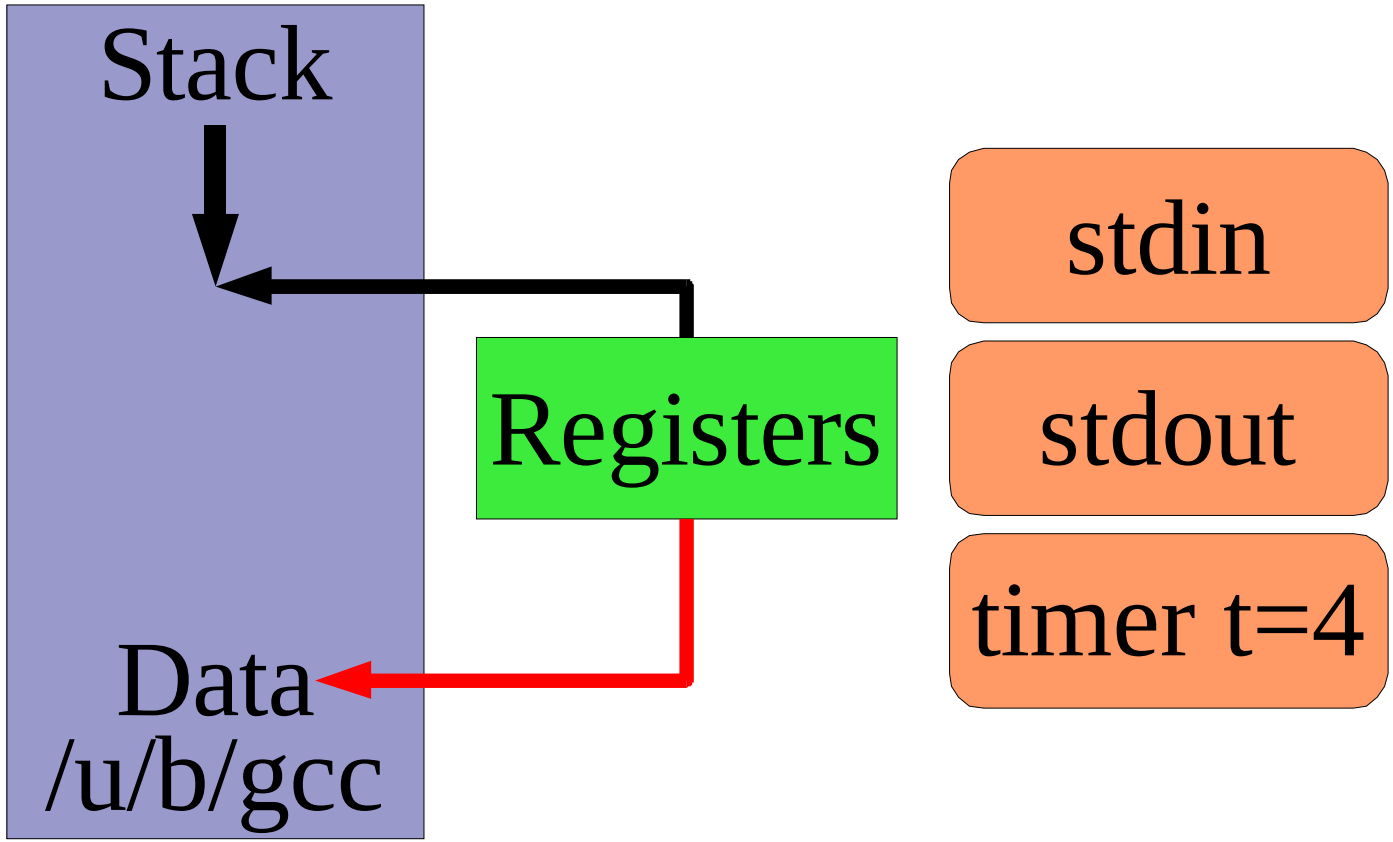
# Original Process



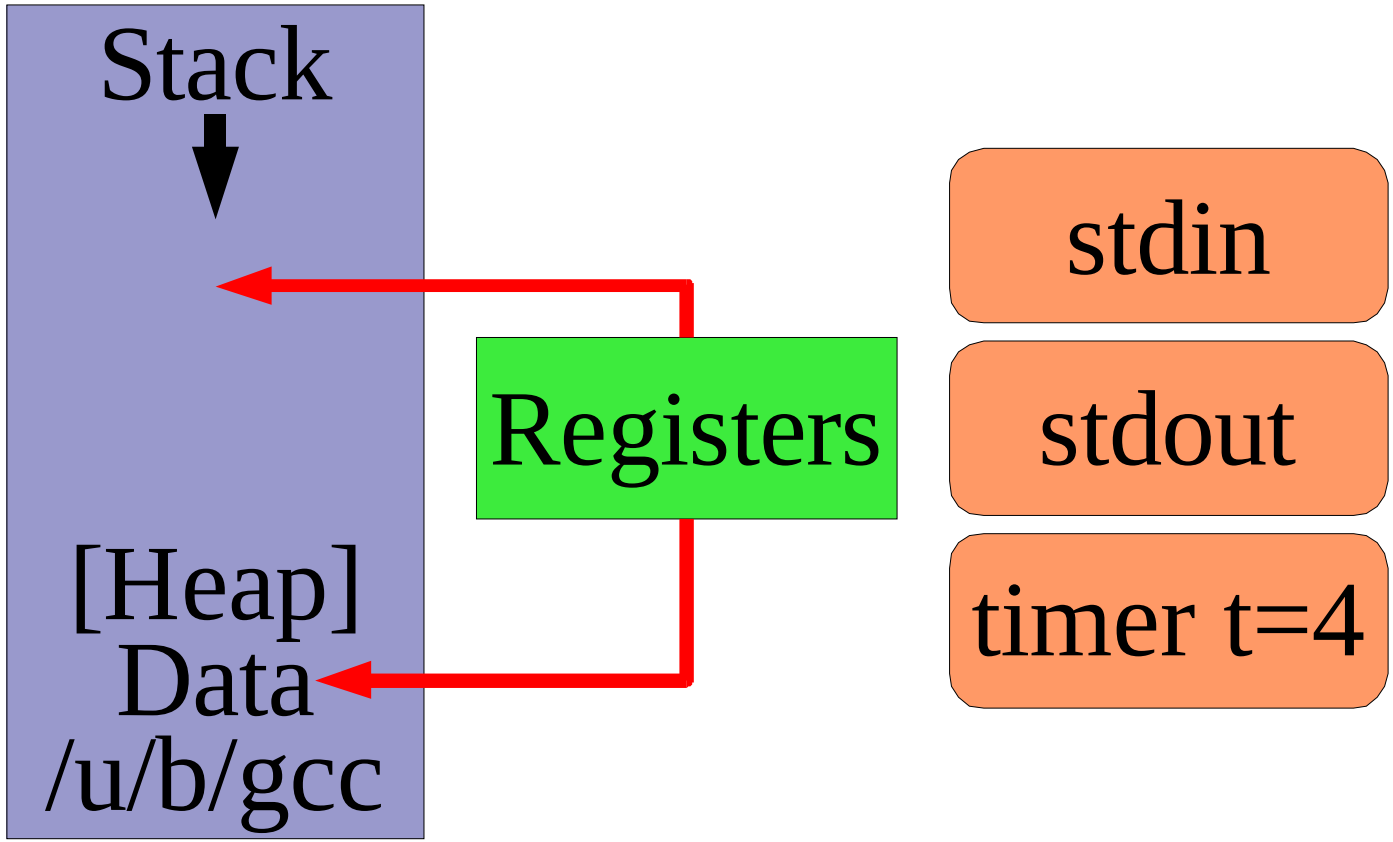
# Toss Heap, Data



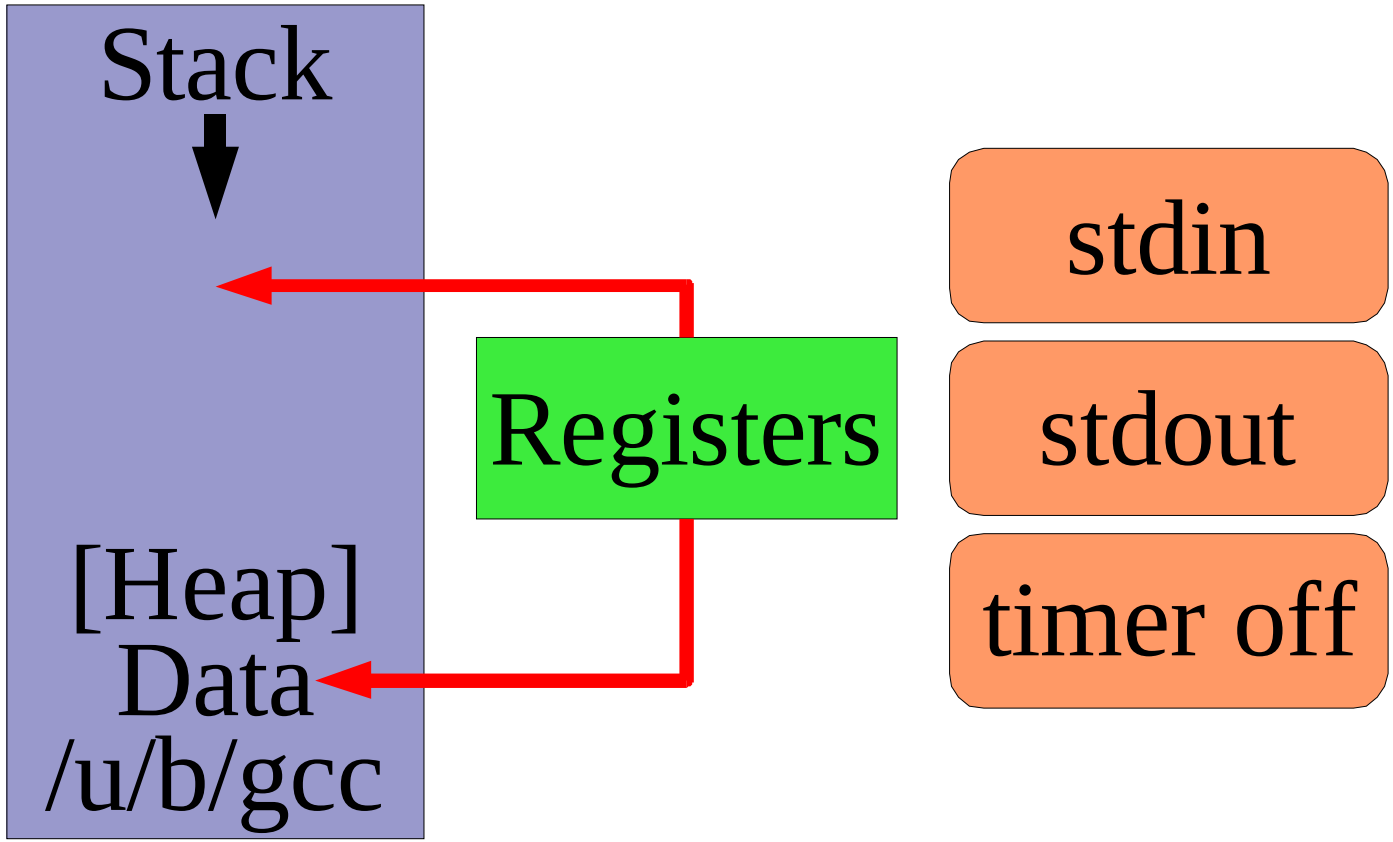
# Load New Code, Data From File



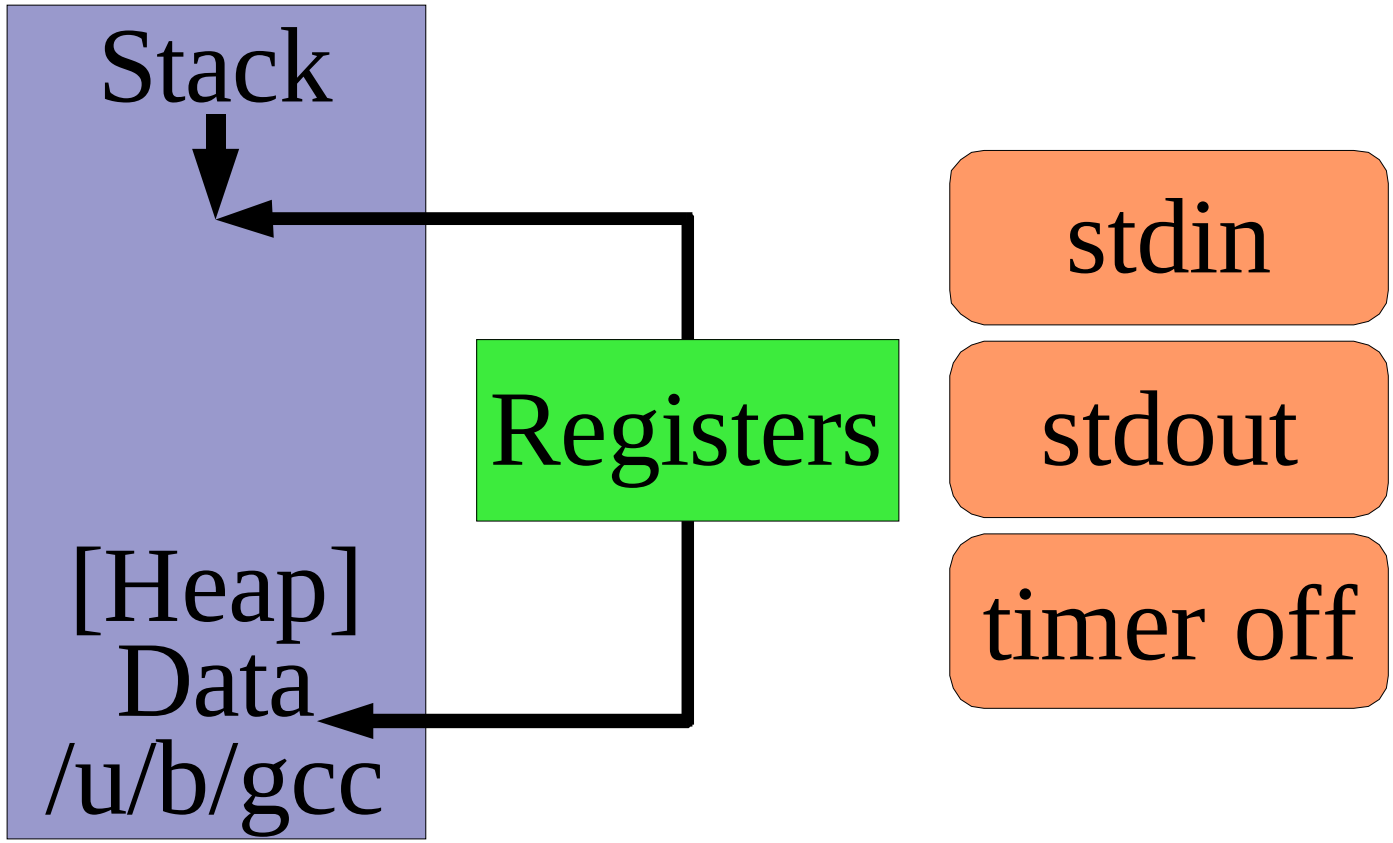
# Reset Stack, Heap



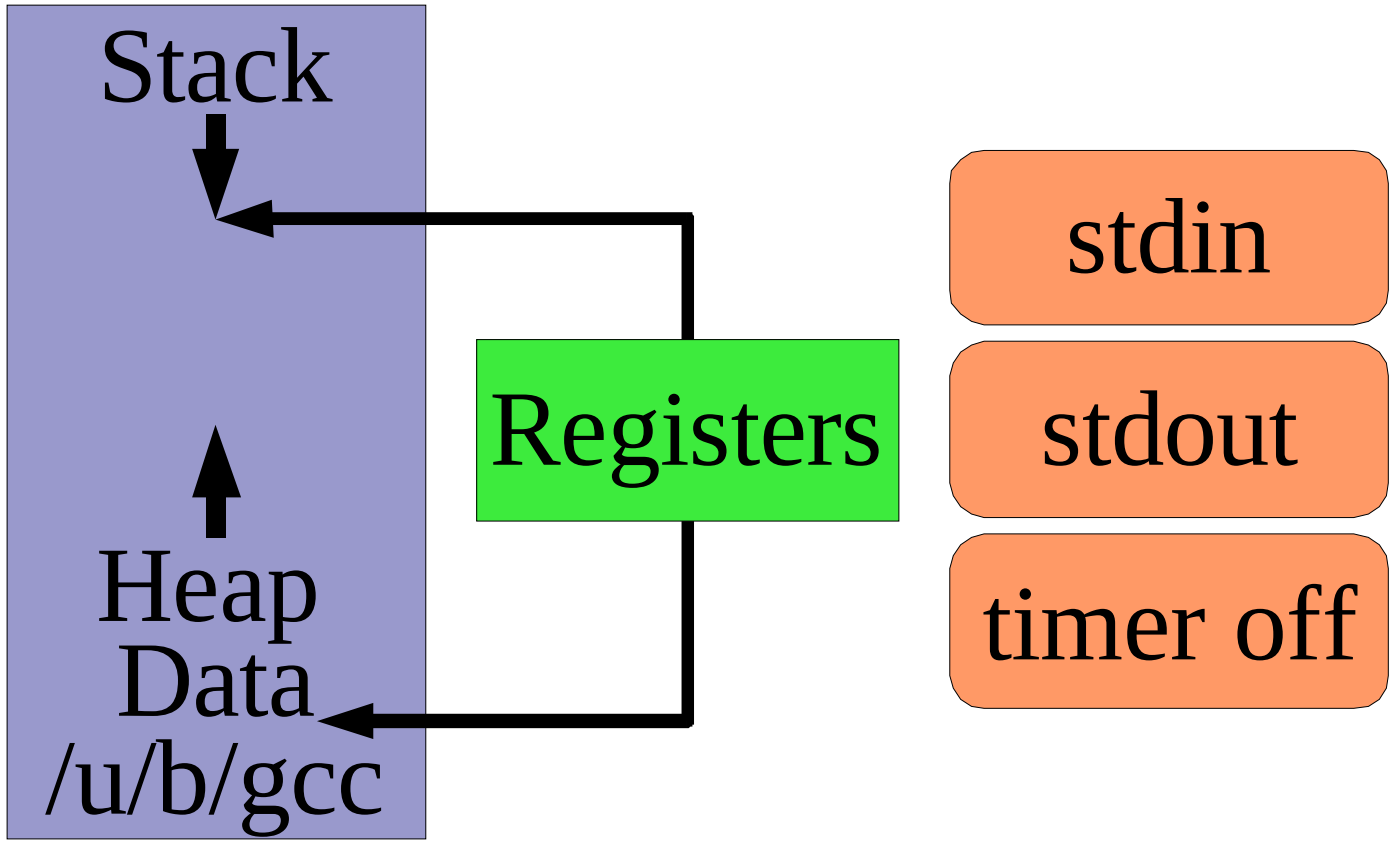
# Fix “Stuff”



# Initialize Registers



# Begin Execution



# What's This 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
- 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: 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
  - 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"**

- (polychlorinated biphenol?)

**Process Control Block**

- “Everything without a 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**

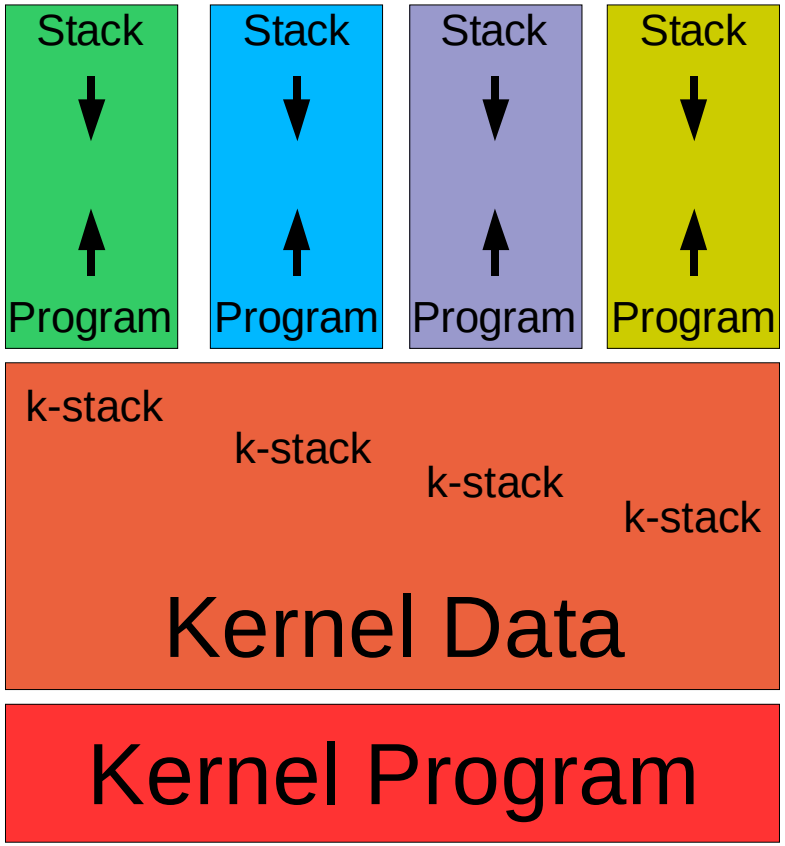
**Memory segment info**

- **User memory segment list**
- **Kernel stack reference**

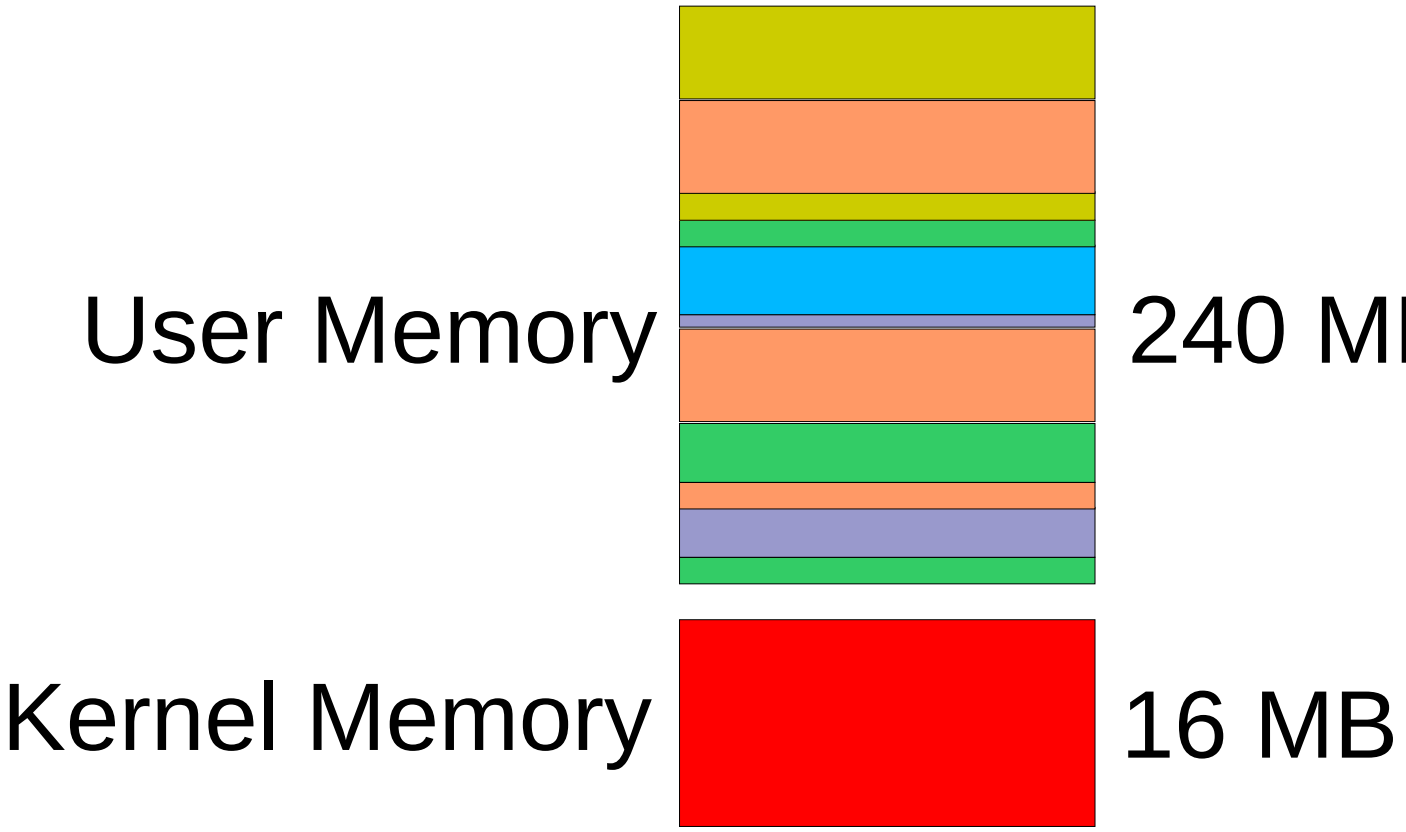
**Scheduler info**

- **linked list slot, priority, “sleep channel”**

# Conceptual Memory Layout



# 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**

- Virtual – Memory regions, registers, I/O “ports”
- Physical – Memory pages, registers, I/O devices

**Birth, School, Work, Death**

**“Big Picture” of memory – both of them**

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