Operating Systems!

OS Interaction (Part 2)

How to Get Stuff Done Using the OS, Processes, Threads, Etc.

Prof. Travis Peters

Montana State University

CS 460 - Operating Systems

Fall 2020

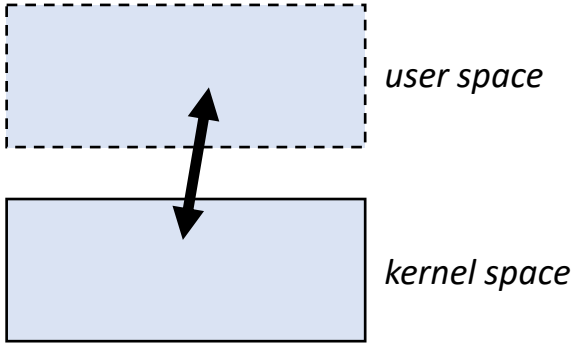
https://www.cs.montana.edu/cs460

Today

- Announcements
  - *Heads up.... PA1 has been released!*
    Sunday [09/20/2020] @ 11:59 PM (MST)

    NO D2L! *Code pushed to GitHub == submitted!*

- Learning Objectives
  - Understand the big ideas behind the "OS API"
    - ~~**user vs. kernel, modes,**~~ **syscalls, libraries**
  - Understand the big ideas behind process control
    - *[theory]* **control info, creation, termination, states, etc.**
    - *[reality]* **fork, exec, getpid, waitpid, etc.**



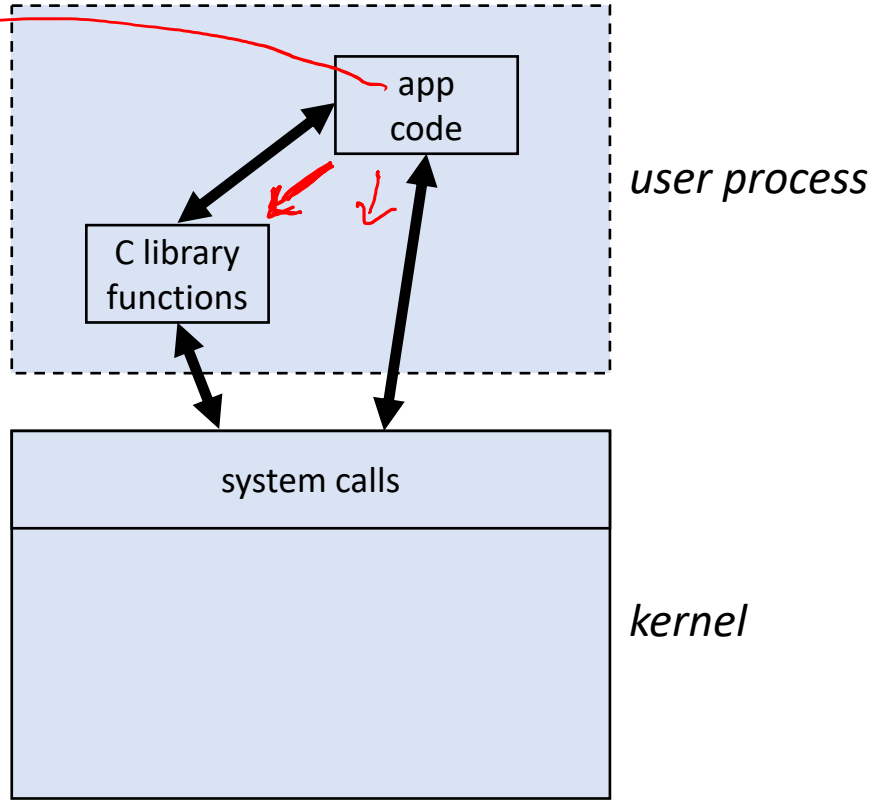
System Calls ("Syscalls")

An example!

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
    return 0;
}
```

```
# Run myexec. What happens?
$ gcc exec1.c -o exec1
$ ./exec1
```



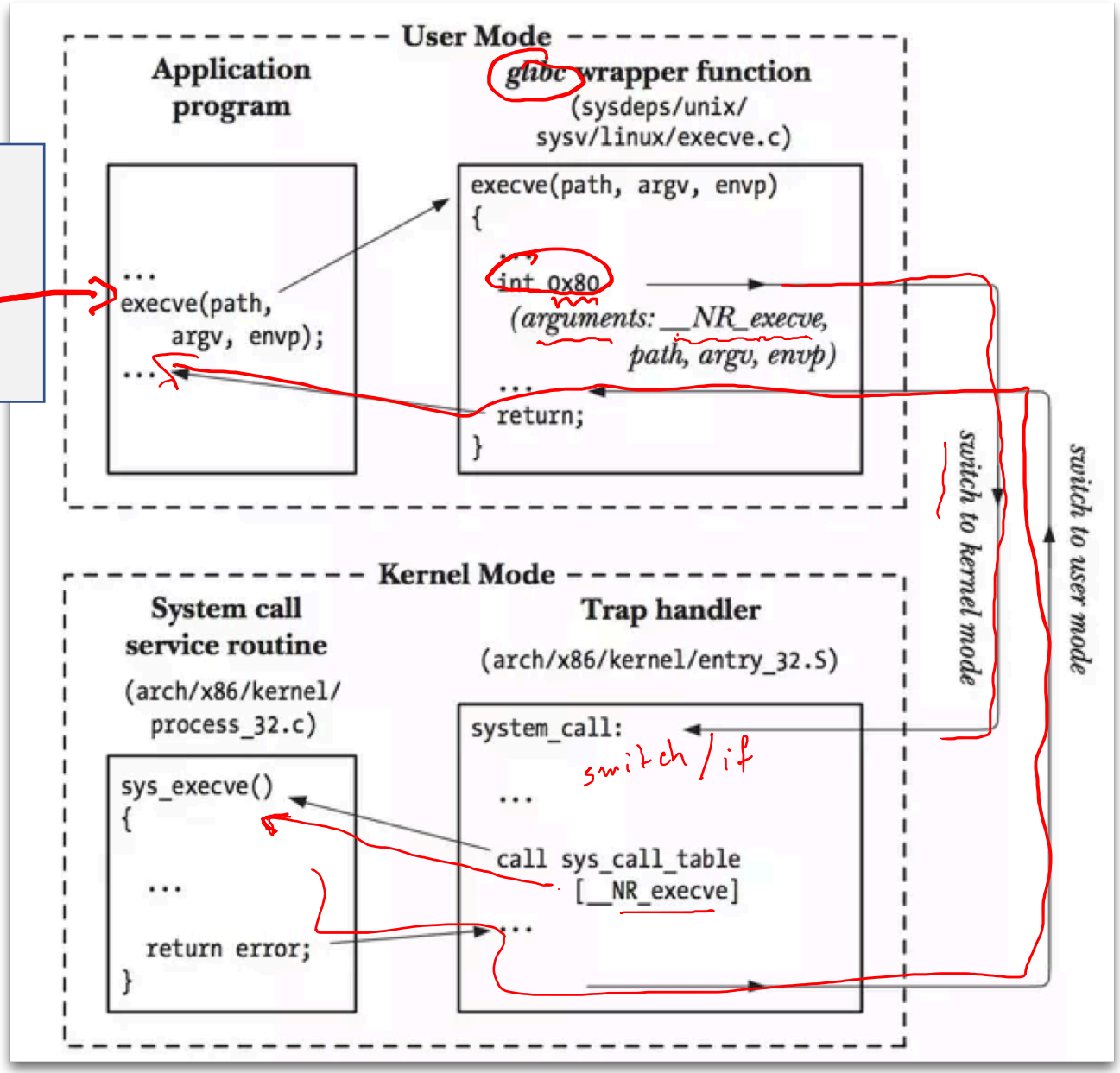
Syscalls *(cont.)*

An example!

```c
int main()
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
    return 0;
}
```

Registers & Invoking Syscalls

EAX	**System Call Number**
EBX	**1st Argument**
ECX	**2nd Argument**
EDX	**3rd Argument**



Syscalls *(cont.)*

An example!

```
int main()
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
    return 0;
}
```

Registers & Invoking Syscalls

EAX

System Call Number

`0x0000000b` (11) — **the value for the execve syscall*

EBX

1st Argument

address of "/bin/sh"

ECX

2nd Argument

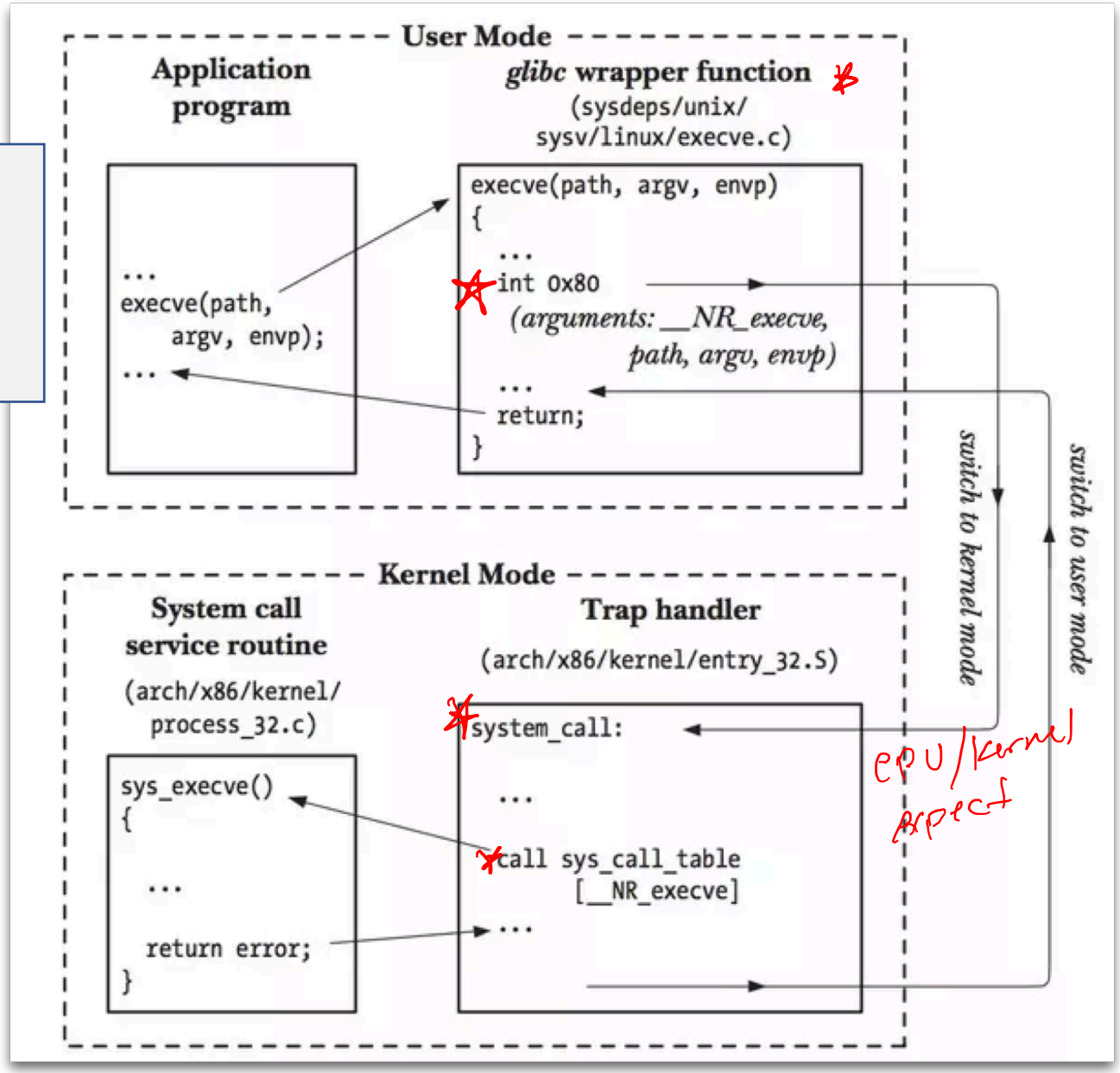
`0` — *a.k.a. NULL - i.e., don't pass on environment variables*

EDX

3rd Argument

`INT 0x80` — *trap to kernel and invoke the syscall identified in the EAX register*

**Example based on program compiled for 32-bit systems*

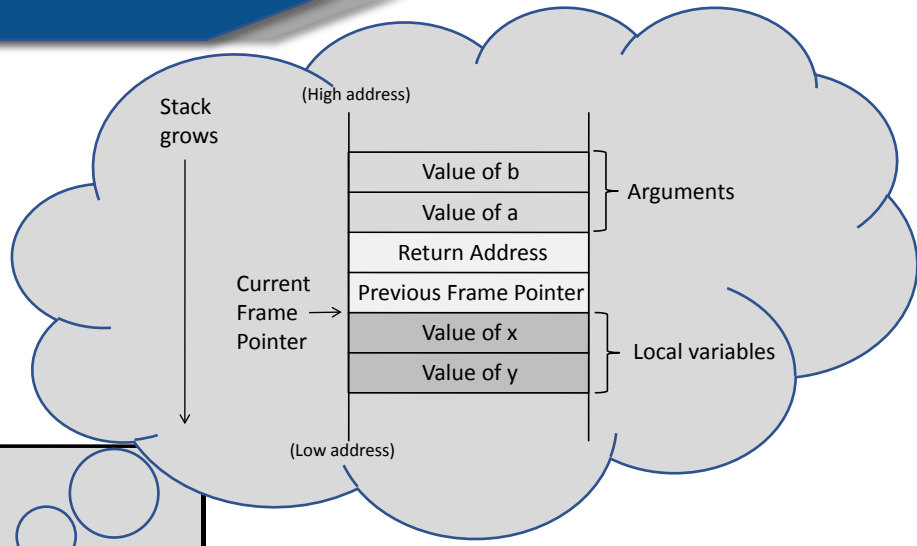


Processes & Process Control

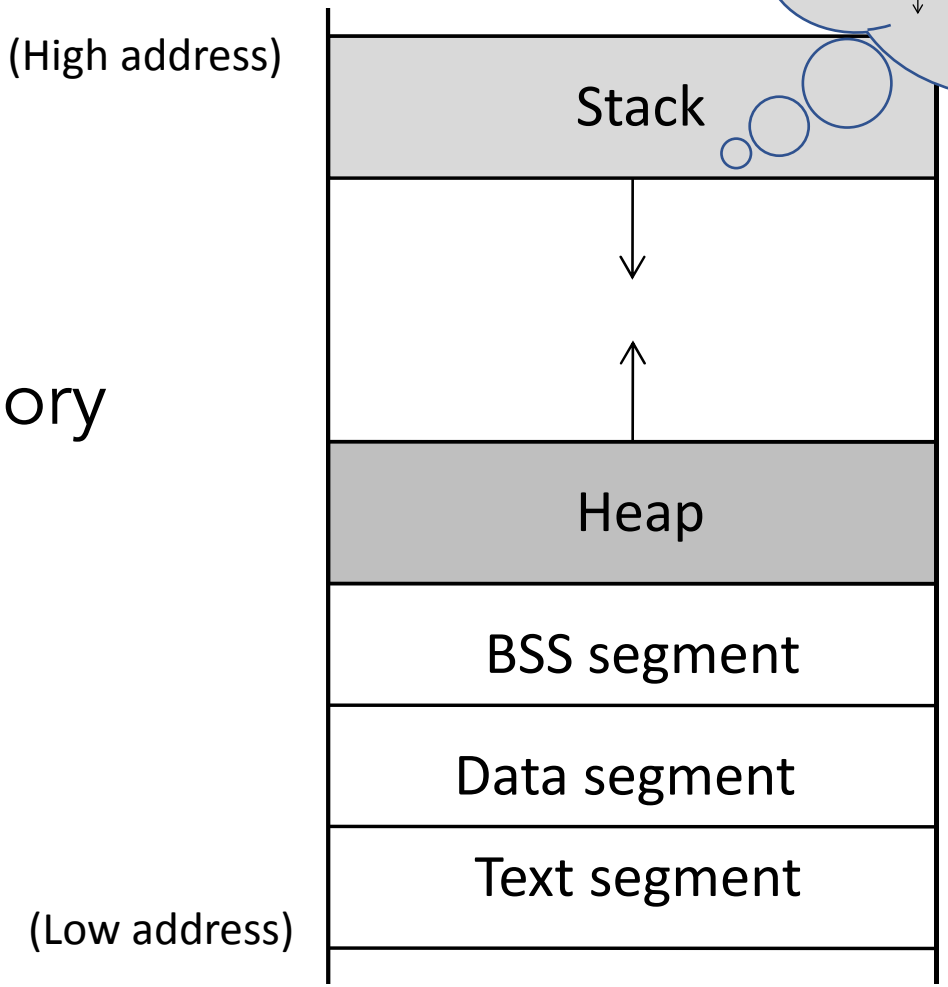
The *WHAT* and *WHY (and a little of the HOW)*

A (User) Process In Memory

(Revisited)



The standard way to set up a process in memory

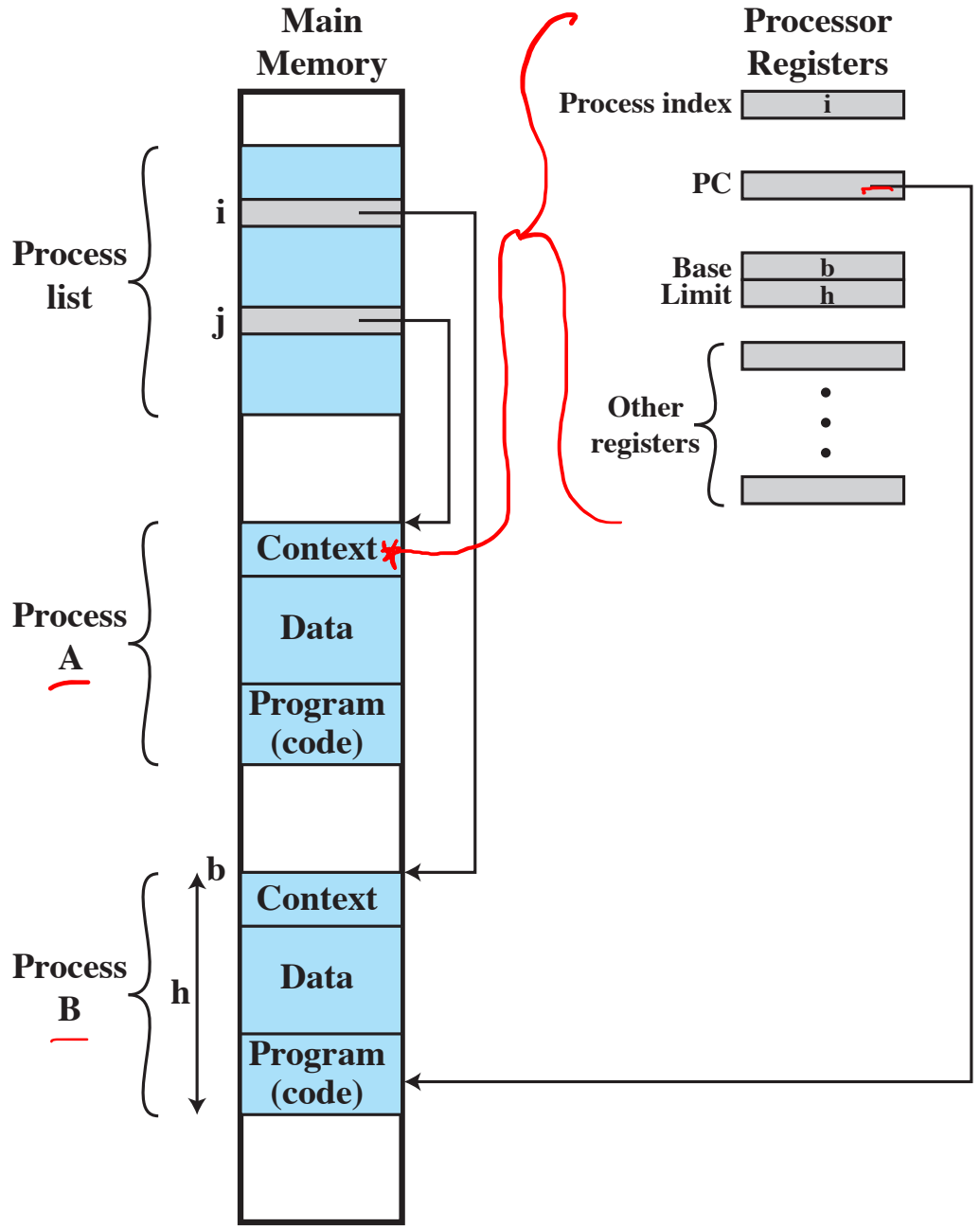


The Idea of a "Process"

(Revisited)

- *An instance of a program running on a computer*

- Consists of
  - (1) an executable program (**code**),
  - (2) associated **data**,
  - (3) **execution context**
    (i.e., info the OS needs to manage the process)



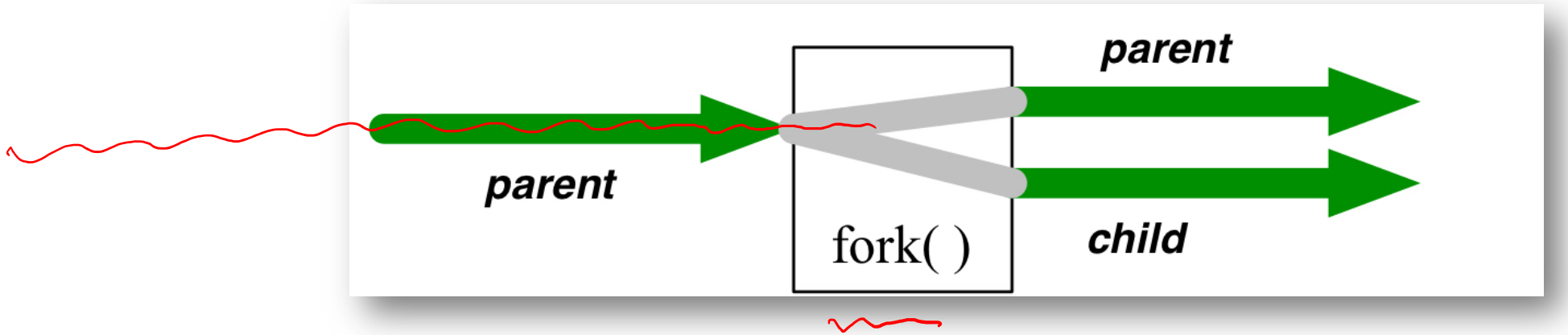
Processes & Process Control

The *HOW: create, control, identify, synchronize, terminate, etc.*

Creating a Process *(Who's Your Daddy? Or Mommy? Parent? Guardian?)*

- In the Unix family (and others), we use **`fork()`** to create a new process

- **`fork()`** returns twice…
  - It creates a ***child process*** with an exact copy of the ***parent process***.
  - same PC, same address space, …
  - since PC is the same, both pick up executing at the same place (right after **`fork()`**)



Creating a Process *(Who's Your Daddy? Or Mommy? Parent? Guardian?)*

- In the Unix family (and others), we use **`fork()`** to create a new process

- **`fork()`** returns twice…
  - It creates a ***child process*** with an exact copy of the ***parent process***.
  - same PC, same address space, …
  - since PC is the same, both pick up executing at the same place (right after **`fork()`**)

Q: How do you tell the difference?

A. You just know…
B. You control this by setting the input arguments to fork()
C. Check the return value of fork()
D. Use another syscall to query the OS

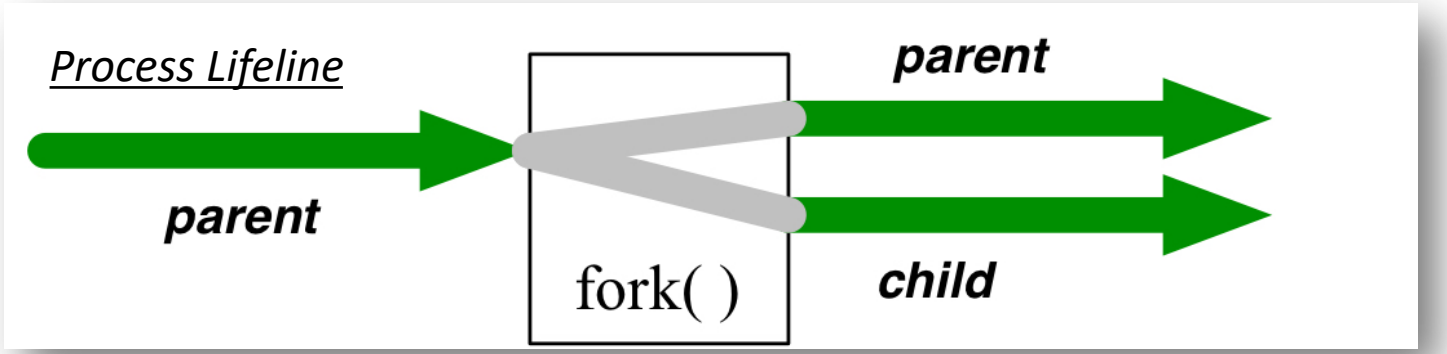
Creating a Process *(cont.)*

~~A. You just know…~~

~~B. You control this by setting the input arguments to fork()~~

C. Check the return value of fork()

~~D. Use another syscall to query the OS~~



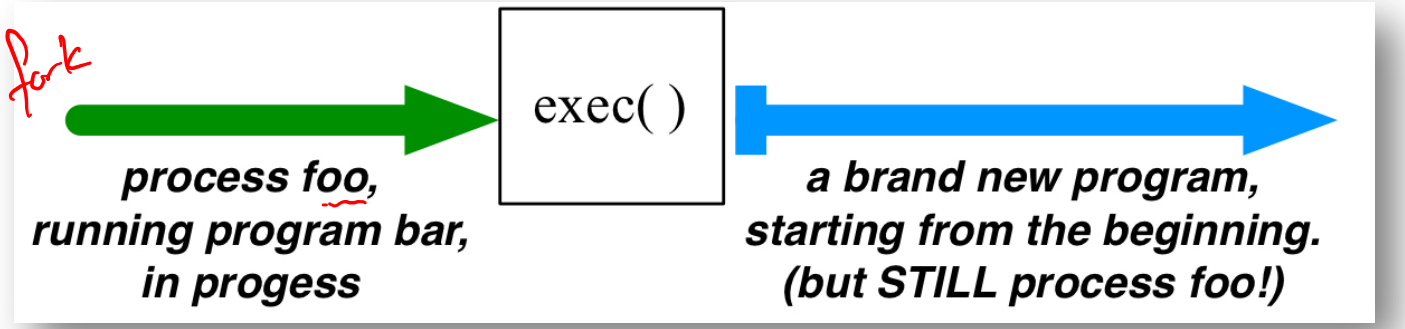

```c
pid = fork();
if (0 == pid) {
    // I'm the child
    printf("Hi, I'm the child. Me no know how talk talk.\n");
    exit(0);
}
printf("I'm the parent. My child has pid %d\n", pid);
```

pid 20

Changing Your Program *(But I don't want to be like my parents...)*

- *But what if I don't want my child to be EXACTLY like me...?*
  "But I want the child to run a different program!! What do I do?"

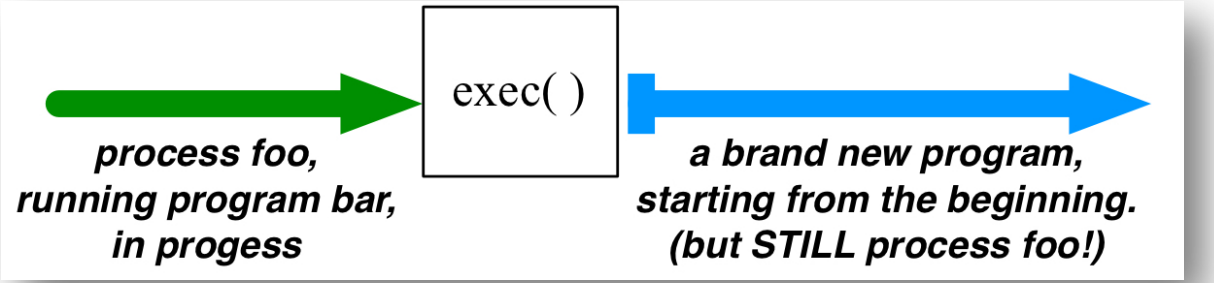
- The **exec** family of function calls
  ...tell the OS to wipe out the address space in the calling process
  ...+ repopulate with a new program and start running again from scratch → *think `main()`*




```
int main()
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
    return 0;
}
```

Changing Your Program *(But I don't want to be like my parents...)*

- *But what if I don't want my child to be EXACTLY like me...?*
  "But I want the child to run a different program!! What do I do?"



- There are a bunch of versions, varying on:
  - Do you specify the new program as a filename, or a full path?
  - Do you give the arguments as a list of NULL-terminated parameters, or as an `argv[]` array?
  - Do you want to bother specifying a new set of environment variables?
  - *...lots of conventions & special semantics*

execlp

Try `man exec`
...oops, that's the wrong answer.
...try `man -k exec` → *there's a bunch!*
... try `man -S3 exec` (or `man 3 exec` or `man exec.3`)
... try `man execve`

Changing Your Program

How we typically create a new process that runs a different program!

Process Identity *(And who exactly do you think you are?!)*

A process can also explore its "family tree".

- `getpid` gets *your* PID / *this process's* PID.
- `getppid` gets *your parent's* PID.

- The **Init** process is typically distinguished by PID = 1

```
[vagrant@vbox][~/code/week04]$ ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     4     0     0 ?           -1 I<       0   0:00  \_ [kworker/0:0H]
    0     1     1     1 ?           -1 Ss       0   0:01 /sbin/init
    1   418   418   418 ?           -1 S<s      0   0:00 /lib/systemd/systemd-journald
    1   425   425   425 ?           -1 Ss       0   0:00 /sbin/lvmetad -f
    1   437   437   437 ?           -1 Ss       0   0:00 /lib/systemd/systemd-udevd
    1   468   468   468 ?           -1 Ss       0   0:00 /sbin/rpcbind -f -w
    1   478   478   478 ?           -1 Ss     100   0:00 /lib/systemd/systemd-networkd
    1   514   514   514 ?           -1 Ss     101   0:00 /lib/systemd/systemd-resolved
    1   621   621   621 ?           -1 Ssl      0   0:00 /usr/bin/lxcfs /var/lib/lxcfs/
    1   624   624   624 ?           -1 Ss     103   0:11 /usr/bin/dbus-daemon --system --address=
    1   676   676   676 ?           -1 Ssl      0   0:00 /usr/bin/python3 /usr/bin/networkd-dispa
    1   677   677   677 ?           -1 Ss       1   0:00 /usr/sbin/atd -f
    1   678   678   678 ?           -1 Ssl    102   0:00 /usr/sbin/rsyslogd -n
    1   680   680   680 ?           -1 Ssl      0   0:03 /usr/lib/accountsservice/accounts-daemon
    1   684   684   684 ?           -1 Ss       0   0:00 /usr/sbin/cron -f
    1   691   691   691 ?           -1 Ss       0   0:00 /lib/systemd/systemd-logind
    1   763   763   763 ?           -1 Ssl      0   0:00 /usr/lib/policykit-1/polkitd --no-debug
    1   881   881   881 ?           -1 Ss       0   0:00 /usr/sbin/sshd -D
  881  1730  1730  1730 ?           -1 Ss       0   0:00  \_ sshd: vagrant [priv]
 1730  1821  1730  1730 ?           -1 S     1000   0:04      \_ sshd: vagrant@pts/0
 1821  1822  1822  1822 pts/0    12346 Ss    1000   0:01          \_ -bash
 1822 12346 12346  1822 pts/0    12346 R+    1000   0:00              \_ ps axjf
```

Termination *(I'll be back.... No you won't.)*

- Most often* a process terminates by...
  - `exit()`ing
  - "falling off" the end of main
  - receiving/catching a "terminate" **signal** *(we'll talk more about signals soon...)*
    - signal == software interrupt
    - e.g., SIGTERM (15) - *"please terminate...pretty please..."*
  - being killed by the OS

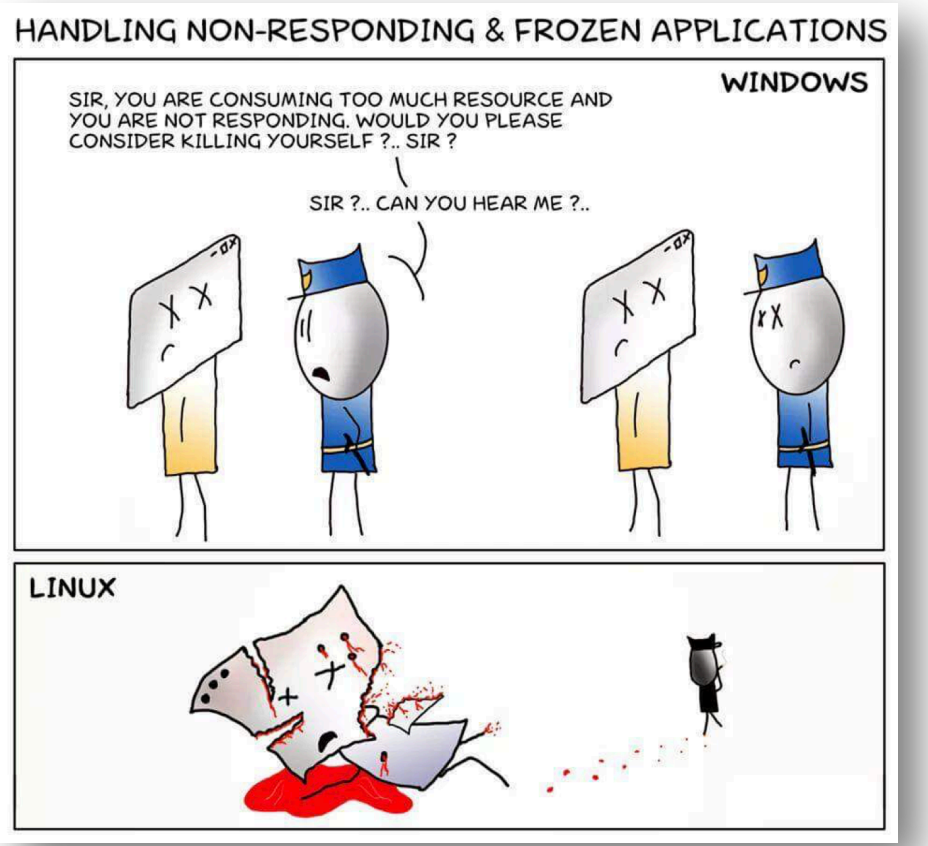
**Occasionally, some systems will enforce cascading termination: if a parent dies, its child must die too.*

Name	Description	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Default action
SIGABRT	abnormal termination (abort)	•	•	•	•	•	•	terminate+core
SIGALRM	timer expired (alarm)		•	•	•	•	•	terminate
SIGBUS	hardware fault		•	•	•	•	•	terminate+core
SIGCANCEL	threads library internal use						•	ignore
SIGCHLD	change in status of child		•	•	•	•	•	ignore
SIGCONT	continue stopped process		•	•	•	•	•	continue/ignore
SIGEMT	hardware fault			•	•	•	•	terminate+core
SIGFPE	arithmetic exception	•	•	•	•	•	•	terminate+core
SIGFREEZE	checkpoint freeze						•	ignore
SIGHUP	hangup		•	•	•	•	•	terminate
SIGILL	illegal instruction	•	•	•	•	•	•	terminate+core
SIGINFO	status request from keyboard			•		•		ignore
SIGINT	terminal interrupt character	•	•	•	•	•	•	terminate
SIGIO	asynchronous I/O			•	•	•	•	terminate/ignore
SIGIOT	hardware fault			•	•	•	•	terminate+core
SIGJVM1	Java virtual machine internal use						•	ignore
SIGJVM2	Java virtual machine internal use						•	ignore
SIGKILL	termination		•	•	•	•	•	terminate
SIGLOST	resource lost						•	terminate
SIGLWP	threads library internal use			•			•	terminate/ignore
SIGPIPE	write to pipe with no readers		•	•	•	•	•	terminate
SIGPOLL	pollable event (poll)				•		•	terminate
SIGPROF	profiling time alarm (setitimer)			•	•	•	•	terminate
SIGPWR	power fail/restart				•		•	terminate/ignore
SIGQUIT	terminal quit character		•	•	•	•	•	terminate+core
SIGSEGV	invalid memory reference	•	•	•	•	•	•	terminate+core
SIGSTKFLT	coprocessor stack fault				•			terminate
SIGSTOP	stop		•	•	•	•	•	stop process
SIGSYS	invalid system call		XSI	•	•	•	•	terminate+core
SIGTERM	termination	•	•	•	•	•	•	terminate
SIGTHAW	checkpoint thaw						•	ignore
SIGTHR	threads library internal use			•				terminate
SIGTRAP	hardware fault		XSI	•	•	•	•	terminate+core
SIGTSTP	terminal stop character		•	•	•	•	•	stop process
SIGTTIN	background read from control tty		•	•	•	•	•	stop process
SIGTTOU	background write to control tty		•	•	•	•	•	stop process
SIGURG	urgent condition (sockets)		•	•	•	•	•	ignore
SIGUSR1	user-defined signal		•	•	•	•	•	terminate
SIGUSR2	user-defined signal		•	•	•	•	•	terminate
SIGVTALRM	virtual time alarm (setitimer)		XSI	•	•	•	•	terminate
SIGWAITING	threads library internal use						•	ignore
SIGWINCH	terminal window size change			•	•	•	•	ignore
SIGXCPU	CPU limit exceeded (setrlimit)		XSI	•	•	•	•	terminate or terminate+core
SIGXFSZ	file size limit exceeded (setrlimit)		XSI	•	•	•	•	terminate or terminate+core
SIGXRES	resource control exceeded						•	ignore

Figure 10.1 UNIX System signals

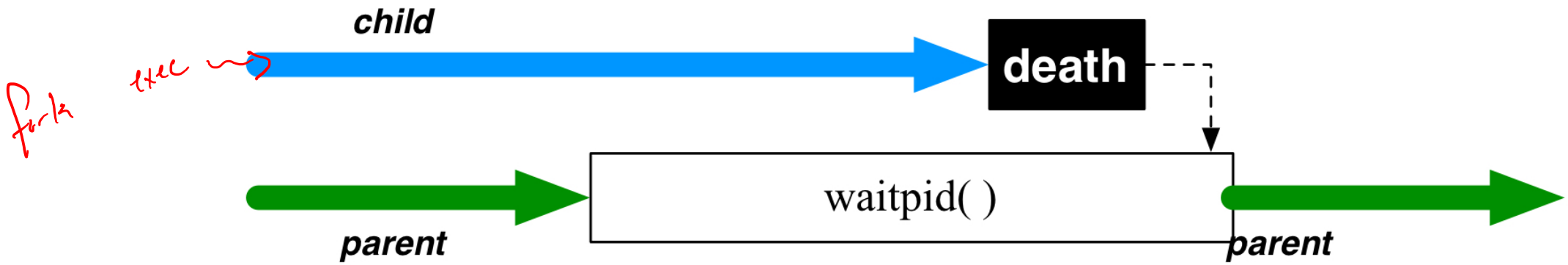
Try:

```
kill -l
```



Waiting *(Hey! Wait for meeeee)*

- A process might want to check if another process has finished.
- The "classic" way of doing this is with **`wait()`**:
  "please block me until the child process I named terminates, and then tell me its exit code."



- In Linux, **`waitpid()`** is maybe the simplest way to do this
  - *the parent can specify exactly which child (PID) it is waiting for.*

Demos

- Go and try out **`week04/`** demo code
  - fork
  - exec
  - waitpid
  - kill
  - getpid
  - *+ more!*