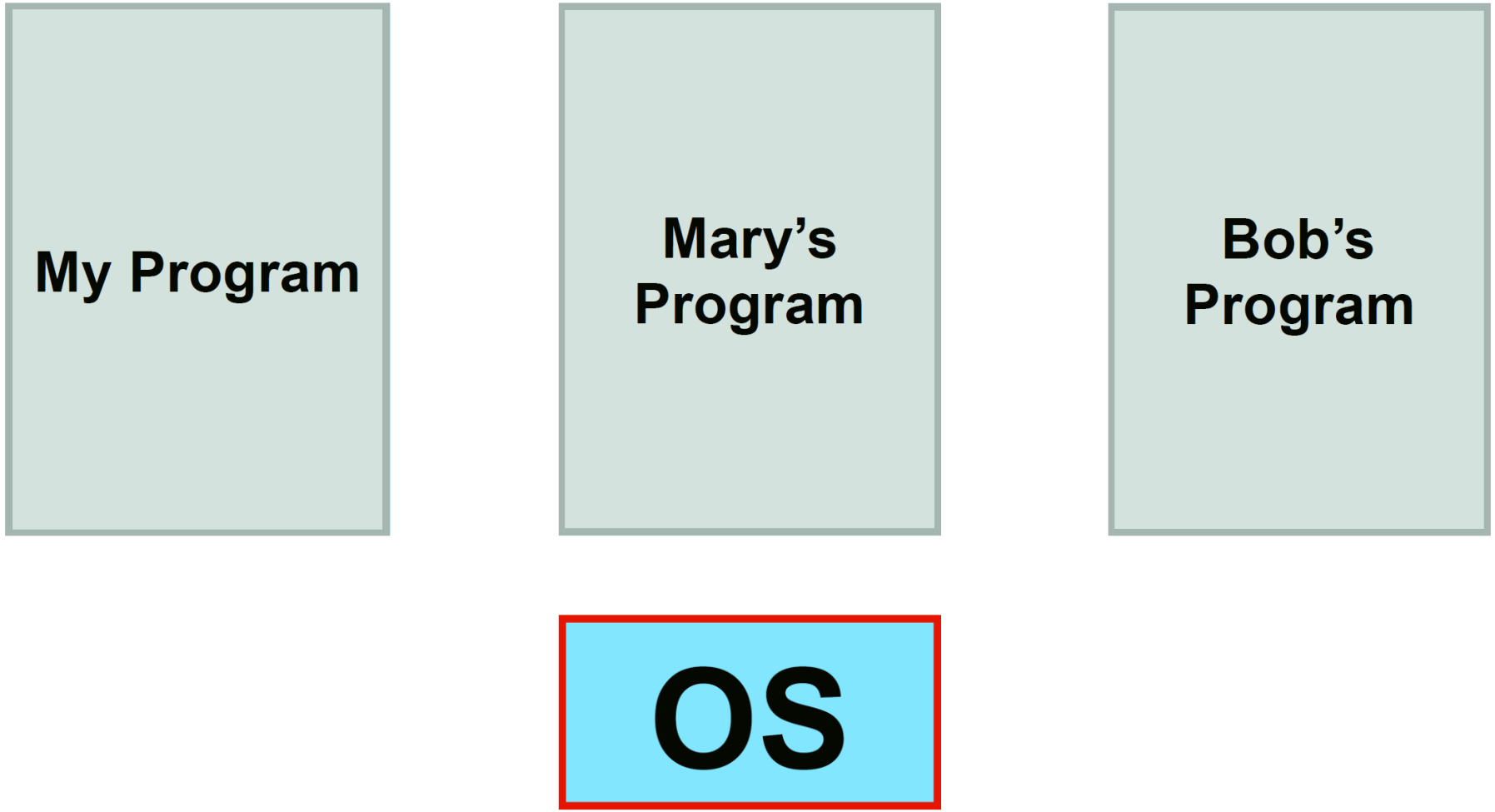
Architecture and OS, Process

* Some material adapted from CS:APP, Prof. Bryant, O'Hallaron, Doeppner, Galvin, etc.

The Operating System



Processes

- **A program in execution**
- **Containers for programs**
  - **virtual memory**
    - **address space**
  - **scheduling**
    - **one or more threads of control**
  - **file references**
    - **open files**
  - **and lots more!**

Fair Share

```
void runforever( ){
    while(1);
}
int main( ) {
    runforever();
}
```

Bob's Program

Architectural Support for the OS

- **Not all instructions are created equal ...**
  - **non-privileged instructions**
    - **can affect only current program**
  - **privileged instructions**
    - **may affect entire system**
- **Processor mode**
  - **user mode**
    - **can execute only non-privileged instructions**
  - **privileged mode**
    - **can execute all instructions**

Which Instructions Should Be Privileged?

- **I/O instructions**
- **Those that affect how memory is mapped**
- **Halt instruction**
- **Some others ...**

Who Is Privileged?

- You’re not
  - and neither is anyone else
- The operating-system kernel runs in privileged mode
  - nothing else does
  - not even super user on Unix or administrator on Windows

Entering Privileged Mode

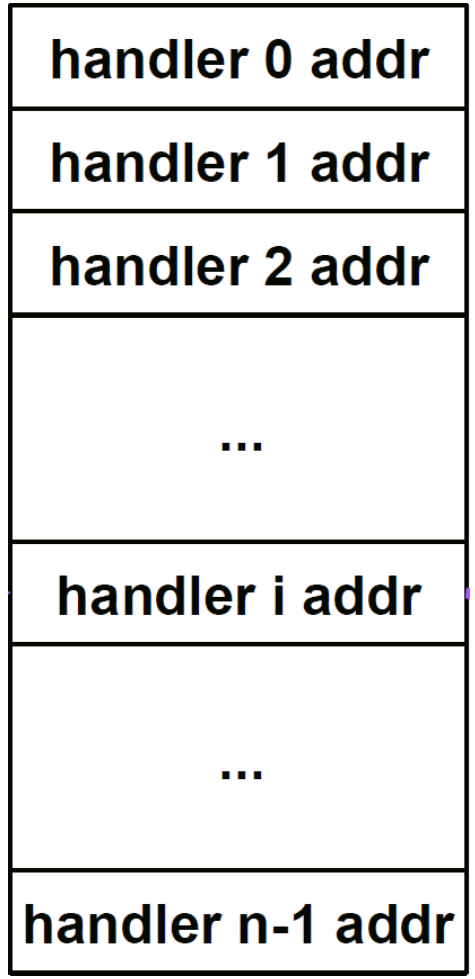
- **How is OS invoked?**
  - **very carefully ...**
  - **strictly in response to interrupts and exceptions**
  - **(booting is a special case)**

Interrupts and Exceptions

- **Things don’t always go smoothly ...**
  - **I/O devices demand attention**
  - **timers expire**
  - **programs demand OS services**
  - **programs demand storage be made accessible**
  - **programs have problems**
- **Interrupts**
  - **demand for attention by external sources**
- **Exceptions**
  - **executing program requires attention**

Interrupt and Exception Handling

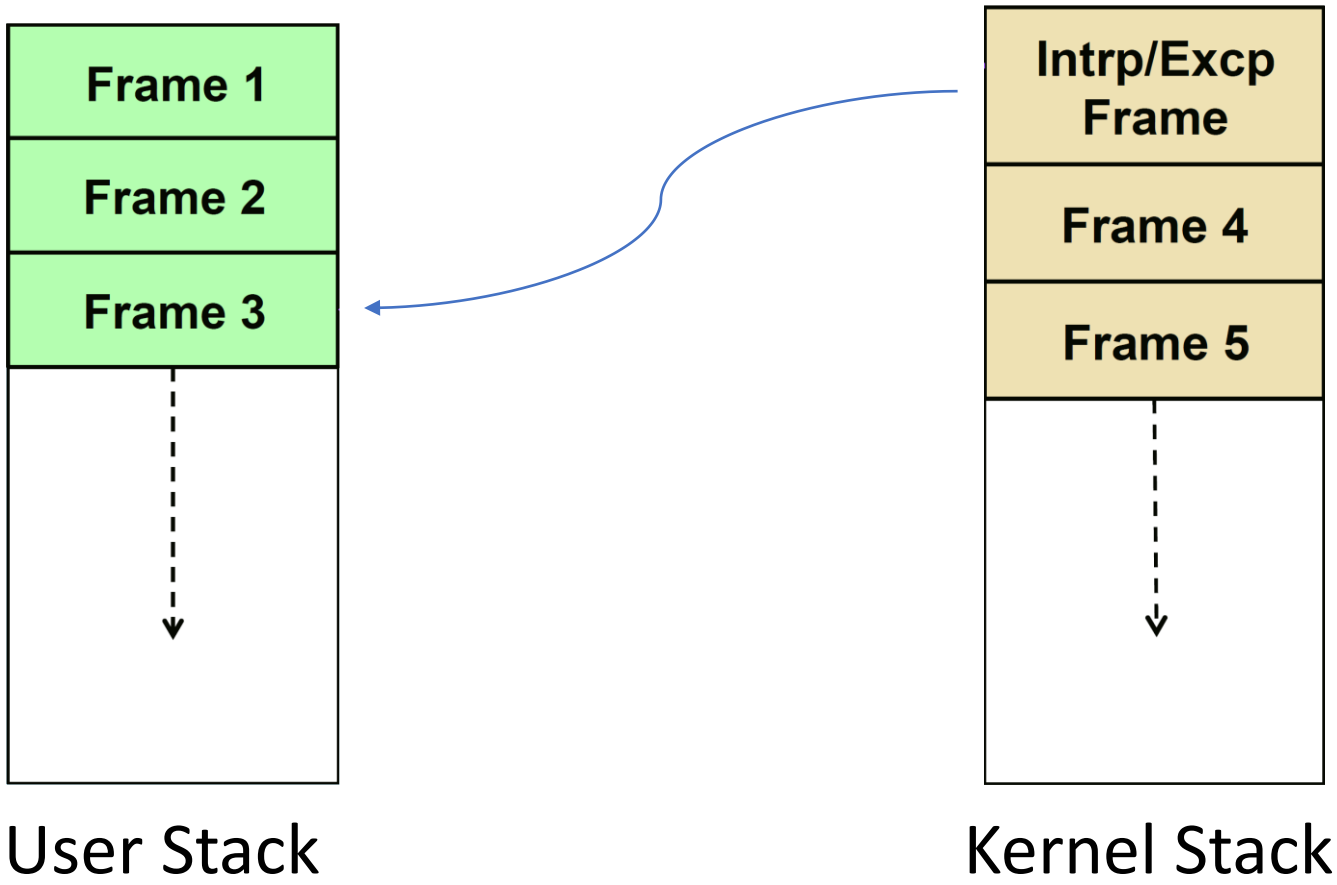
- Interrupt or exception invokes handler (in OS)
  - via interrupt and exception vector
  - one entry for each possible interrupt/exception
    - contains
      - address of handler
  - code executed in privileged mode



Entering and Exiting

- **Entering/exiting interrupt/exception handler more involved than entering/exiting a procedure**
  - **must deal with processor mode**
    - **switch to privileged mode on entry**
    - **switch back to previous mode on exit**
- **stack in kernel must be different from stack in user program**

One Stack Per Mode

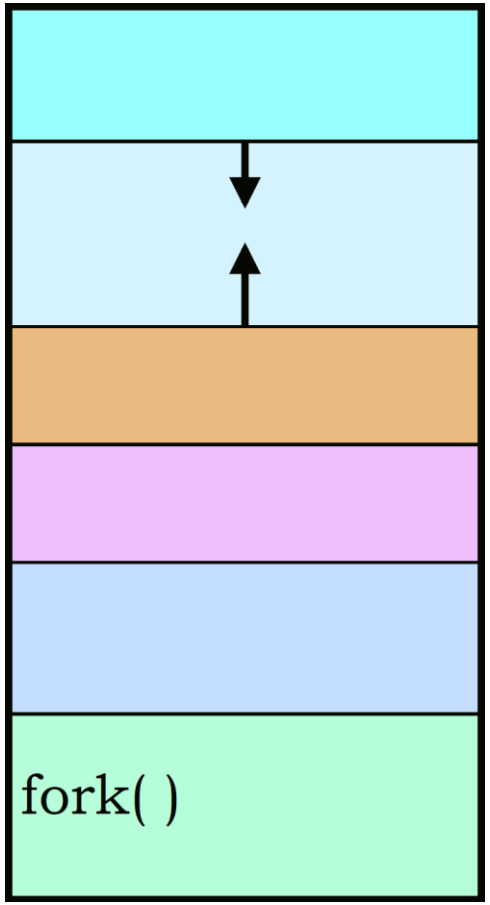


- If an interrupt occurs, which general-purpose registers must be pushed onto the kernel stack?
  - **a) none**
  - **b) callee-save registers**
  - **c) caller-save registers**
  - **d) all**

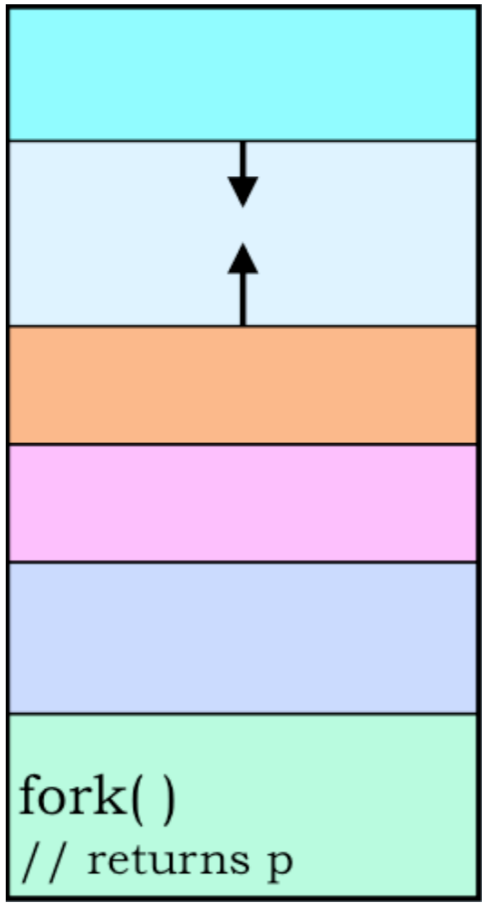
Creating Your Own Processes

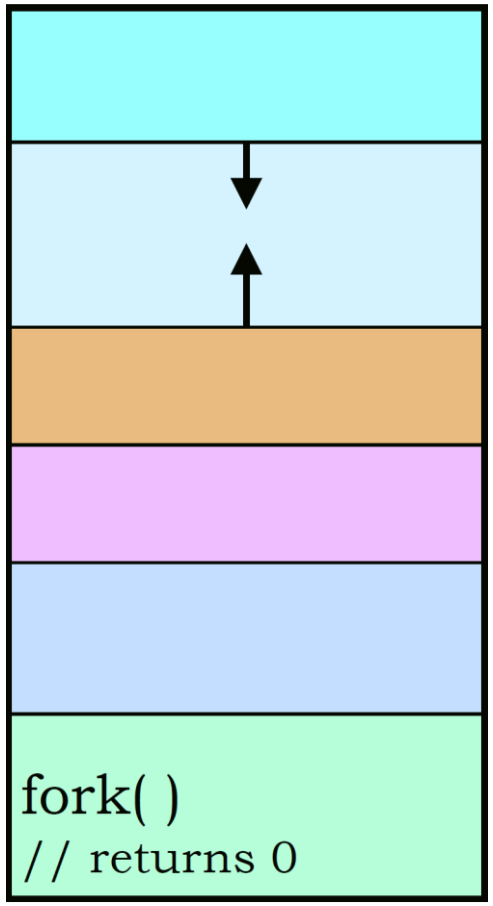
```
#include <unistd.h>
int main( ) {
    pid_t pid;
    if ((pid = fork()) == 0) {
        /* new process starts running here */
    }
    /* old process continues here */
}
```

Creating a Process: Before



Creating a Process: After





Creating Your Own Processes

```
#include <unistd.h>
int main( ) {
    pid_t pid;
    if ((pid = fork()) == 0) {
        /* new process starts running here */
    }
    /* old process continues here */
}
```

Process IDs

```
int main( ) {
    pid_t pid;
    pid_t ParentPid = getpid();
    if ((pid = fork()) == 0) {
        printf("Child: %d, %d, %d\n", pid, ParentPid, getpid());
        return 0;
    }
    printf("Parent: %d, %d, %d\n", pid, ParentPid, getpid());
    return 0;
}
```

parent prints:

27355, 27342, 27342

child prints:

0, 27342, 27355

parent process ID: 27342

child process ID: 27355

```
int flag;
int main() {
    while (flag == 0) {
        if (fork() == 0) {
            flag = 1;
            exit(0); // causes process to terminate
        }
    }
}
```

parent process: infinite loop

child process: terminate

```
int main()
{
int value = 5;
pid_t pid;
if ((pid = fork()) == 0) { /* child process */
                value += 15;
                printf("CHILD: value = %d\n",value); /* LINE A */
                return 0;
}
else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d\n",value); /* LINE A */
                return 0;
}
}
```