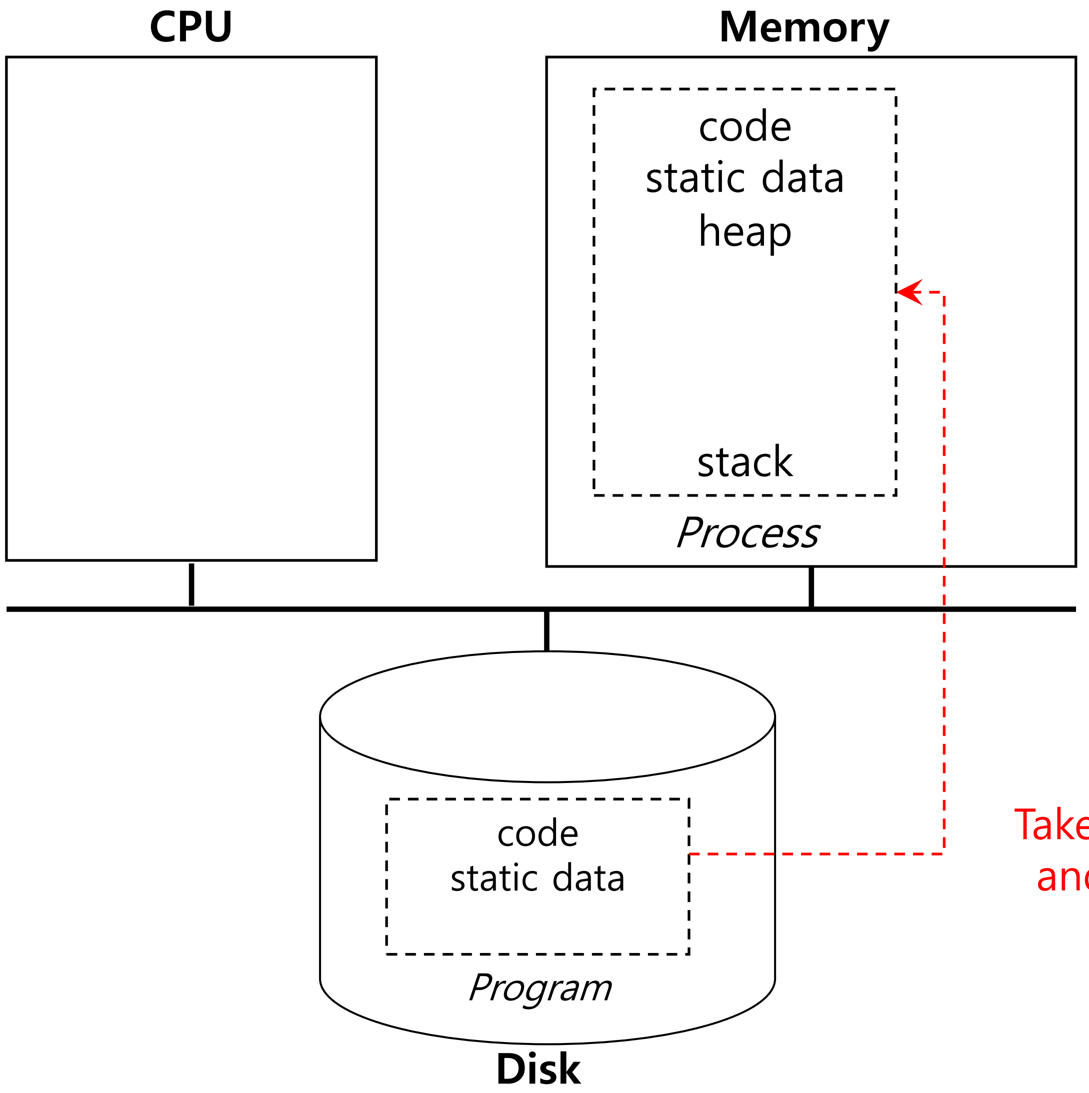
Process

Sridhar Alagar

Executing a Program



Loading:
Takes on-disk program and loads it into the memory

What is a Process?

- A program in execution

- What constitutes a process?
  - Memory space – code, data, heap, stack
  - Registers, IP
  - Open files
  - Many overheads

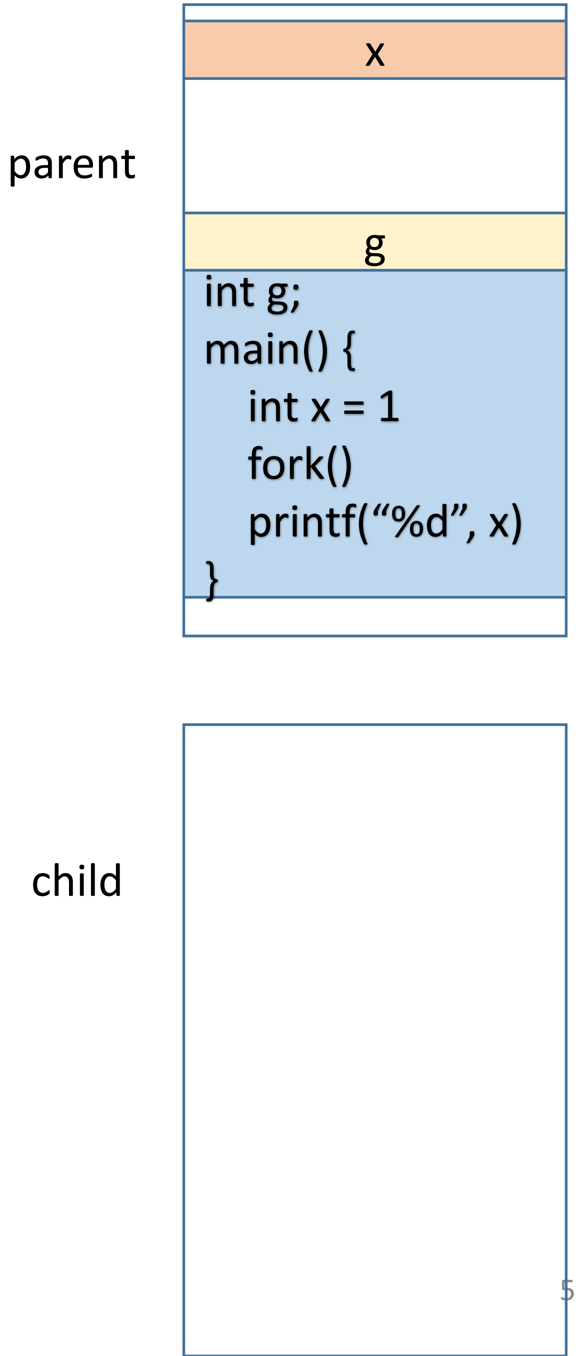
Process API

- Creation
- Terminate
- Wait
  - wait for a process to stop running
- Control
  - suspend and resume
- Status
  - get some status info about the process

Process creation

UNIX/Linux:

- **`fork()`** system call creates a new child process



```
int g;
main() {
   int x = 1
   fork()
   printf("%d", x)
}
```

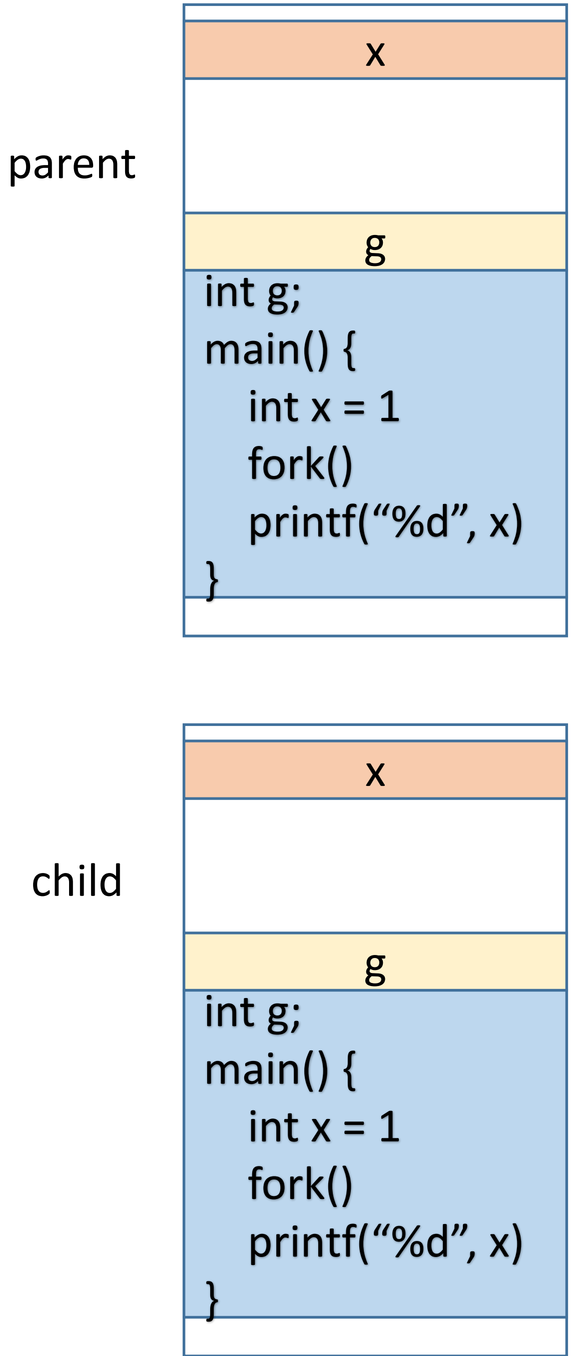
Process creation

UNIX/Linux:

`fork()` system call creates a new child process

Initially, child is a duplicate of the parent

Parent and child processes have separate memory spaces and execute independently



```
int g;
main() {
   int x = 1
   fork()
   printf("%d", x)
}
```

```
int g;
main() {
   int x = 1
   fork()
   printf("%d", x)
}
```

fork()

```
fork();

printf("hello, who am I? \n")
```

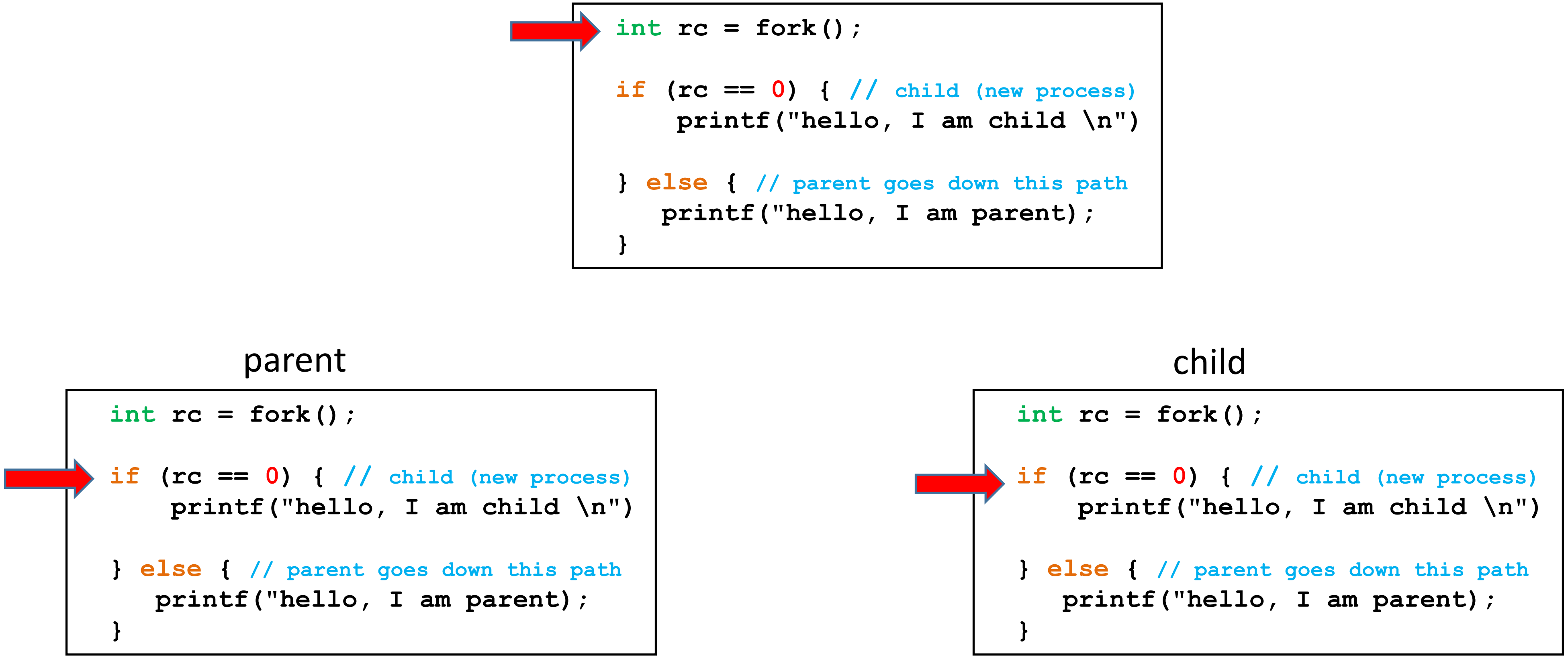
```
fork();

printf("hello, who am I? \n")
```

```
fork();

printf("hello, who am I? \n")
```

fork() usage



```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

parent

```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

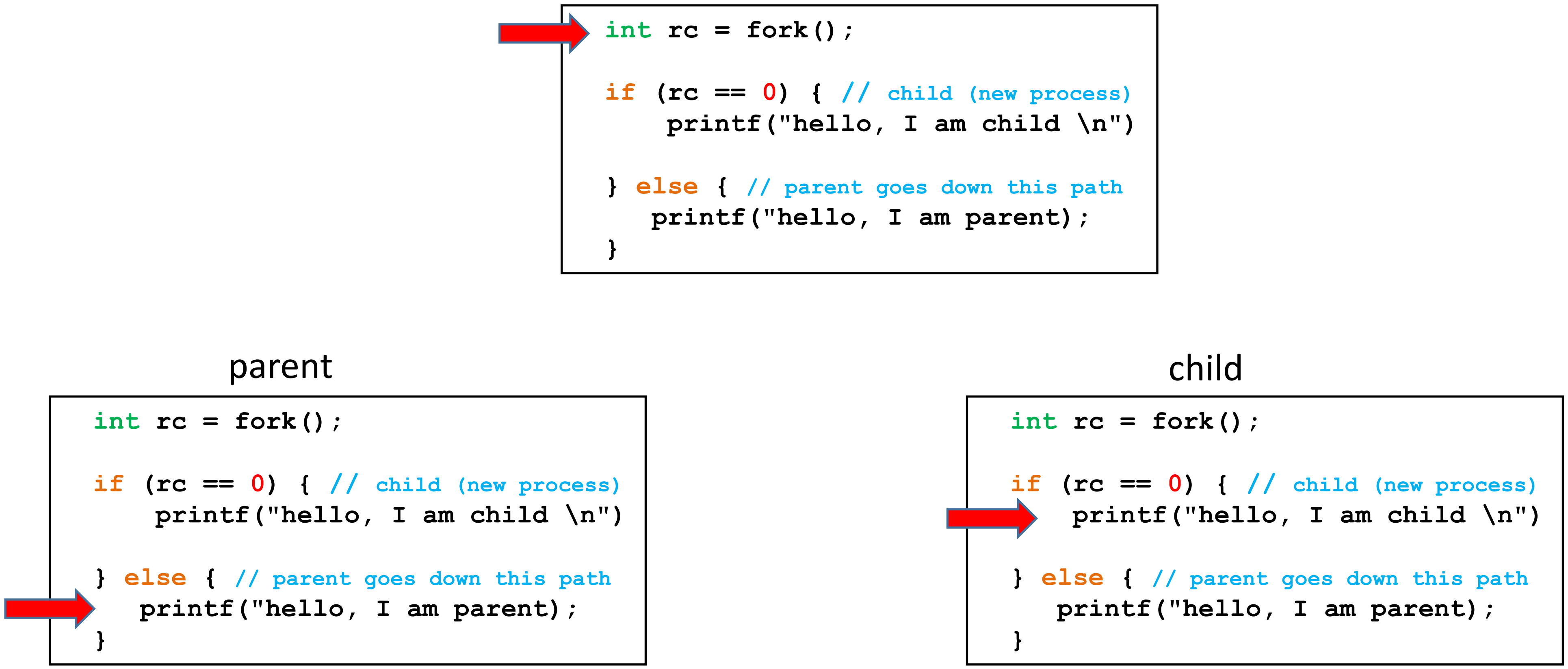
child

```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

fork() usage



```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

parent

```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

child

```
int rc = fork();

if (rc == 0) { // child (new process)
    printf("hello, I am child \n")

} else { // parent goes down this path
   printf("hello, I am parent);
}
```

fork() summary

- Clones another process -> child

- Child is a duplicate of parent (caller of fork())

- By returning different values to parent and child, OS indirectly tells them who they are

- The value returned to the parent is child's pid
  - This is the only way through which a parent will know the child pid

- Parent can wait for child to terminate

fork() quiz

```
int x = 10;
int rc = fork();
if (rc == 0) { // child (new process)
    printf(“Child: x = %d \n“, x)
    x = 100;
} else { // parent goes down this path
   wait();
   printf (“parent: x = %d \n“, x)
}
```

parent

```
int x = 10;
int rc = fork();
if (rc == 0) { // child (new process)
    printf(“Child: x = %d \n“, x)
    x = 100
} else { // parent goes down this path
   wait();
   printf (“parent: x = %d \n“, x);
}
```

child

```
int rc = fork();

if (rc == 0) { // child (new process)
    printf(“Child: x = %d \n“, x)
    x = 100
} else { // parent goes down this path
   wait();
   printf (“parent: x = %d \n“, x)
}
```

What are the printed values of parent and child process?

Where is the meta-data of a process is stored?

Process Control Block (PCB)

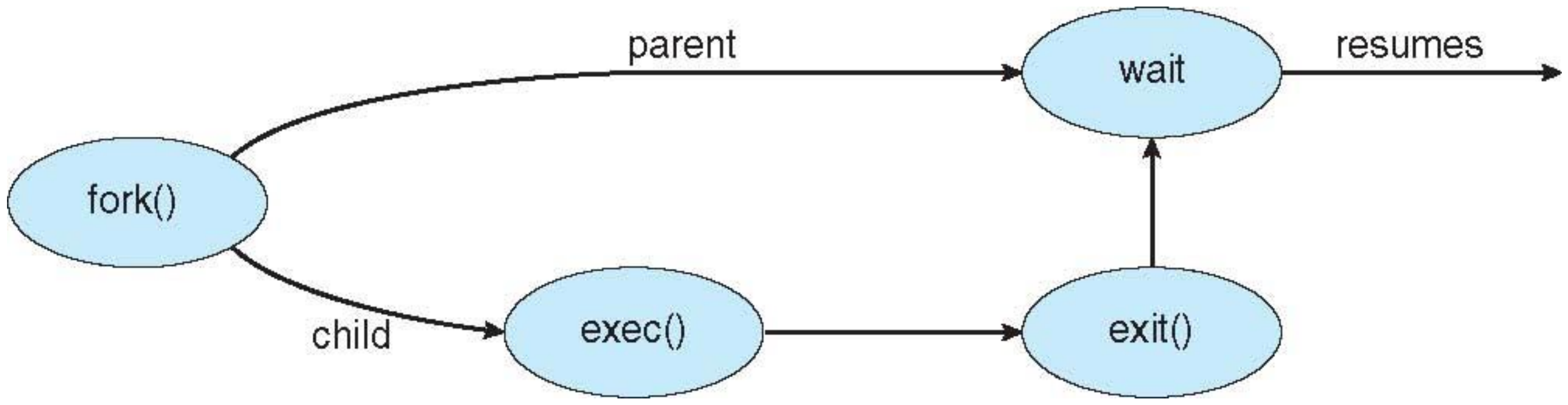
- PID
- Process state (i.e., running, ready, or blocked)
- Execution state (all registers, PC, stack ptr)
- Scheduling priority
- Accounting information (parent and child processes)
- Credentials (which resources can be accessed, owner)
- Pointers to other allocated resources (e.g., open files)

Executing a program

UNIX example:

`exec()` - system call to replace the process' memory space with a new program

Typically, used after a **`fork()`**



fork and exec

```
main(int argc, char *argv[]){
    int rc = fork();
    if (rc == 0) { // child:
        // now exec “ls"...
        char* myargs[2];
        myargs[0] = strdup(“ls"); // program: “ls"
        myargs[2] = NULL;          // marks end of array
        execv(myargs[0], myargs);  // runs “ls”
    } else {                // parent goes down this path (main)
        wait(NULL);
    }
}
```

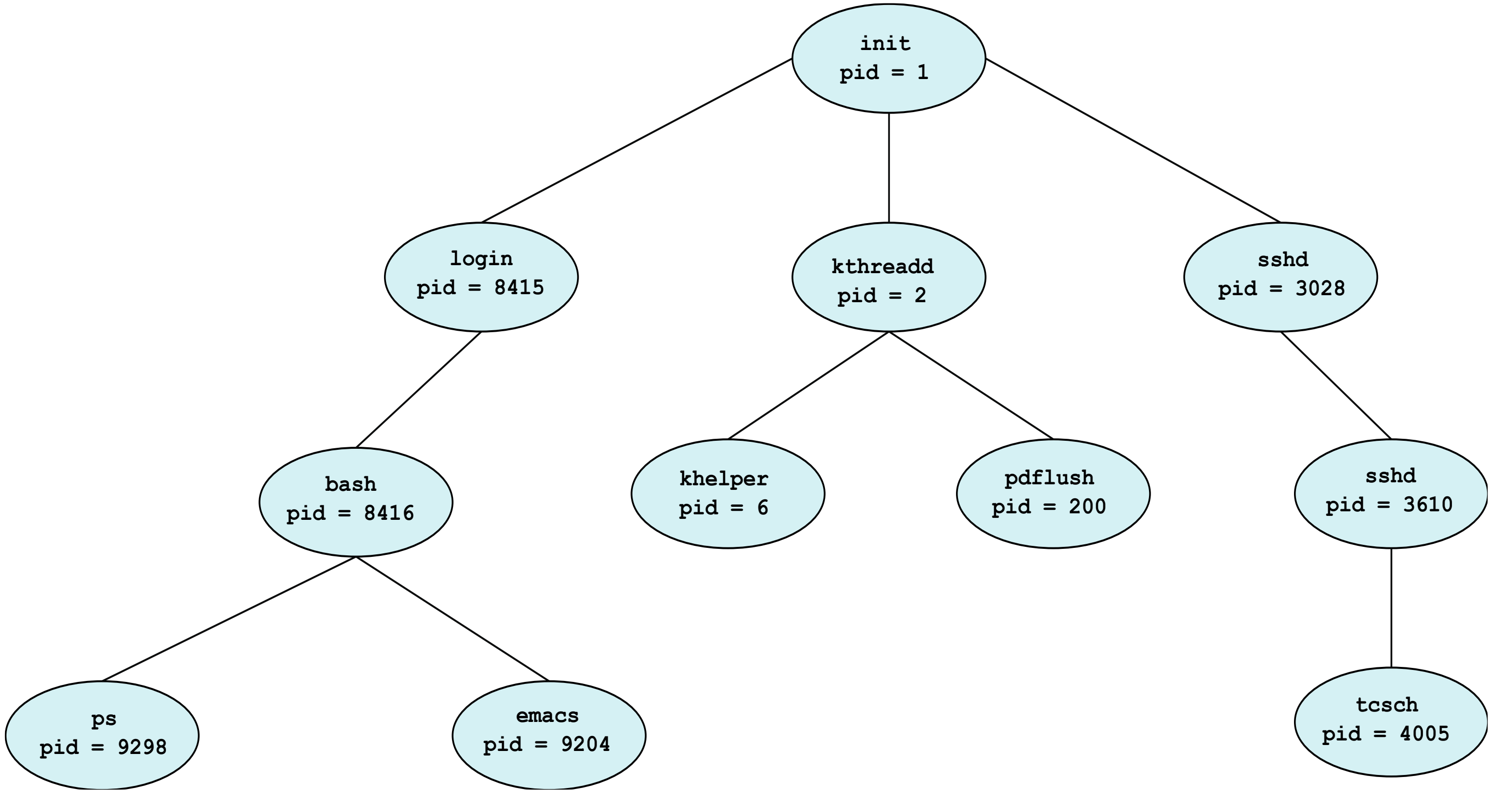
Redirection

```
main(int argc, char *argv[]){
    int rc = fork();
    if (rc == 0) { // child: redirect std output to a file
        close(STDOUT);
        open("output", O_CREAT|O_WRONLY|O_TRUNC);
        // now exec “ls"...
        char *myargs[2];
        myargs[0] = strdup(“ls"); // program: “ls"
        myargs[2] = NULL;          // marks end of array
        execvp(myargs[0], myargs); // runs ls
    } else {                 // parent goes down this path (main)
        wait(NULL);
    }
}
```

Outline for a shell program

```
While(1){
        Display prompt
        reads the command
        parse the command
        run the command
}
```

Processes tree



Orphan and Zombie process

- Child process becomes orphan if its parent exits before child

- A child process becomes zombie when it exits, and its PCB is not released.

  see https://en.wikipedia.org/wiki/Zombie_process

Disclaimer

- Some of the materials in this lecture slides are from the materials prepared by Prof. Arpaci, and Prof. Youjip. Thanks to all of them.