Processes

Kartik Gopalan

References:

- Chapter 2 of the Tanenbaum's book
- Chapter 4 of OSTEP book
- man pages in any UNIX/Linux system

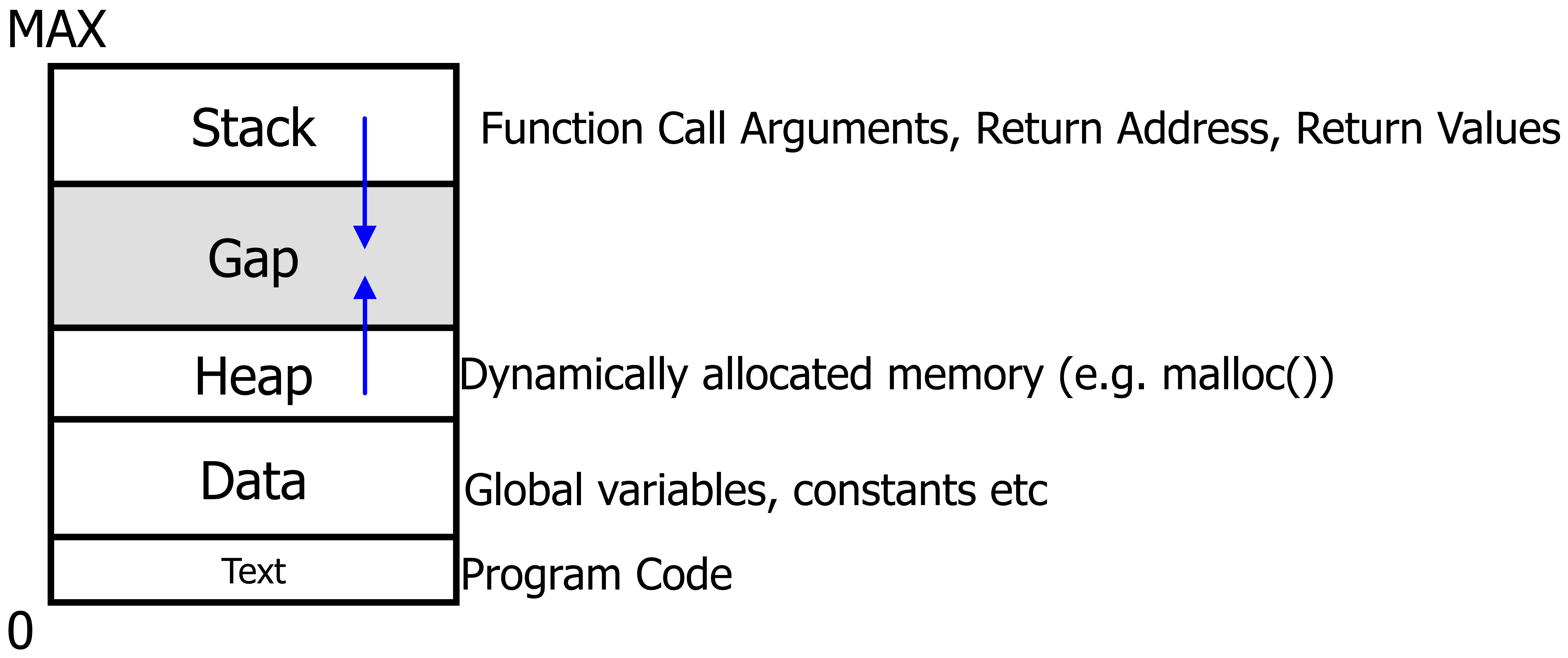
Process versus Program

- Program
  - Program is a passive executable file stored in the disk
  - Contains static code and static data

- Process: A program in execution. A process contains:
  - Code
  - Procedure call stack
  - Memory (static and dynamic data)
  - Registers: Program counter, Stack pointer, General purpose registers
  - Program is just one component of a process.
  - Open files, connections

- There can be multiple processs running the same program
  - Example: many users can run “ls” at the same time

Memory Layout of a typical process



- Stack and heap grow towards each other

System calls to control process lifetime

- fork()
  - Create a process

- exec()
  - Run a new program
  - More accurately: Replace the current process with a new program image

- wait() or waitpid()
  - wait for a child process to terminate

- exit()
  - Terminate the calling process

Example : fork() and waitpid()

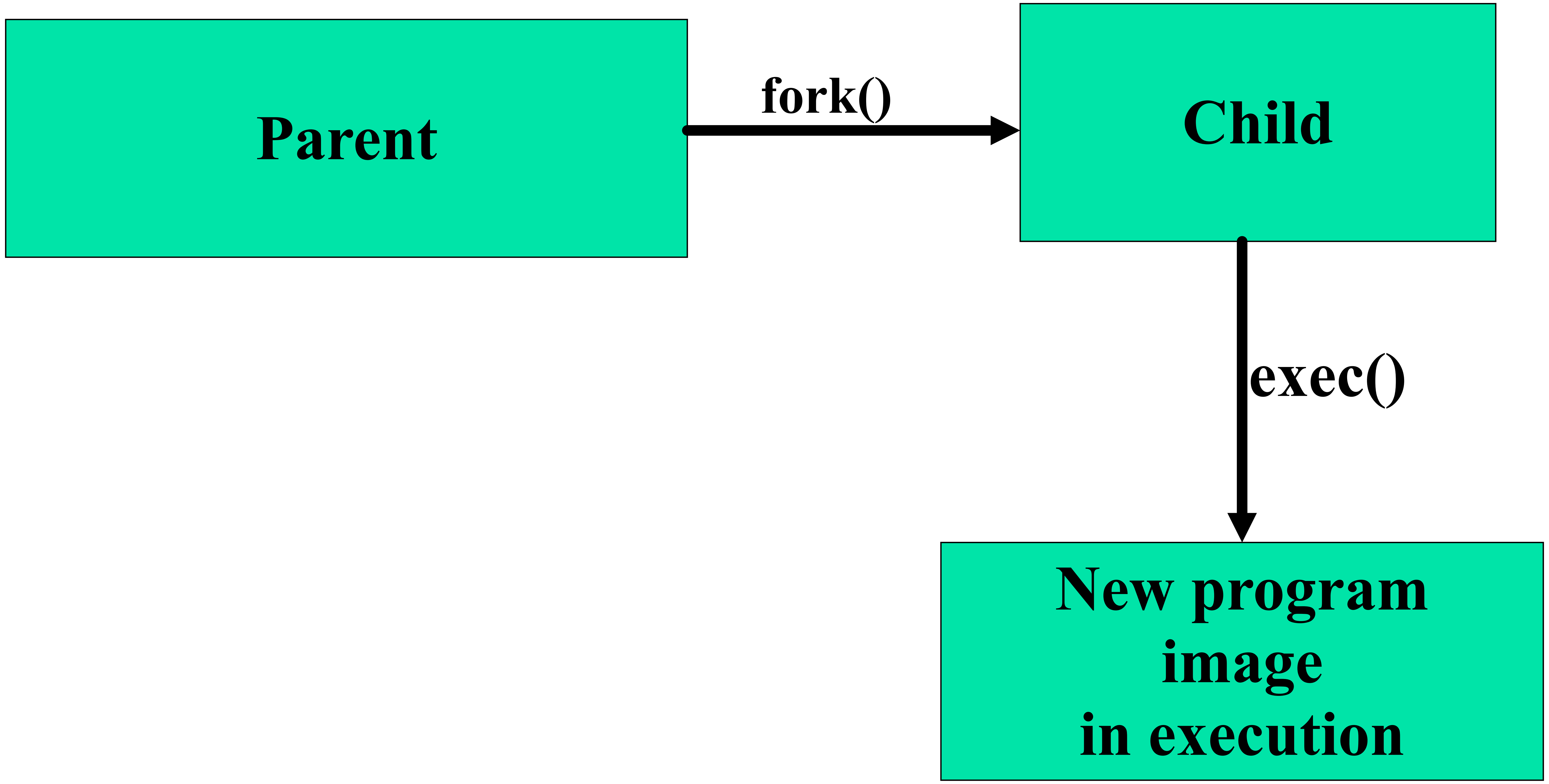
https://oscourse.github.io/examples/fork_ex.c

```
pid = fork();

if (pid < 0) {
        perror("fork failed:");
        exit(1);
}

if (pid == 0) { // Child executes this block
        printf("This is the child\n");
        exit(0);
}

if (pid > 0) { //Parent executes this block

  printf("This is parent. The child is %d\n", pid);

ret = waitpid(pid, &status, 0);
if (ret < 0) {
   perror("waitpid failed:")
   exit(2);
}

printf("Child exited with status %d\n", status);
exit(0);
```

- fork() is called once …
  - but it returns twice!!
  - once in the parent and
  - once in the child
- Parent and child are two processes.
- Child is an exact "copy" of parent.
- Return value of fork in child = 0
- Return value of fork in parent = [process ID of child]
- By checking fork's return value, the parent and the child can take different code paths.

Running a new program using exec()

exec() replaces the caller's memory with a new program image.



exec() - Example code

https://oscourse.github.io/examples/exec_ex.c

```
if ((pid = fork()) < 0) {
 fprintf(stderr, "fork failed\n");
 exit(1);
}

if (pid == 0) {
 if( execlp("echo",
       "echo",
       "Hello from the child",
       (char *) NULL) == -1)
  fprintf(stderr, "execl failed\n");

 exit(2);
}

printf("parent carries on\n");
```

- exec() is called once
  - But doesn't return!!
- All I/O descriptors that were open before exec stay open after exec.
  - I/O descriptors = file descriptors, socket descriptors, pipe descriptors etc.
  - This property is very useful for implementing filters.

Different Types of exec()

- `int execl(char * pathname, char * arg0, … , (char *)0);`
  - Full pathname + long listing of arguments

- `int execv(char * pathname, char * argv[]);`
  - Full pathname + arguments in an array

- `int execle(char * pathname, char * arg0, … , (char *)0, char envp[]);`
  - Full pathname + long listing of arguments + environment variables

- `int execve(char * pathname, char * argv[], char envp[]);`
  - Full pathname + arguments in an array + environment variables

- `int execlp(char * filename, char * arg0, … , (char *)0);`
  - Short pathname + long listing of arguments

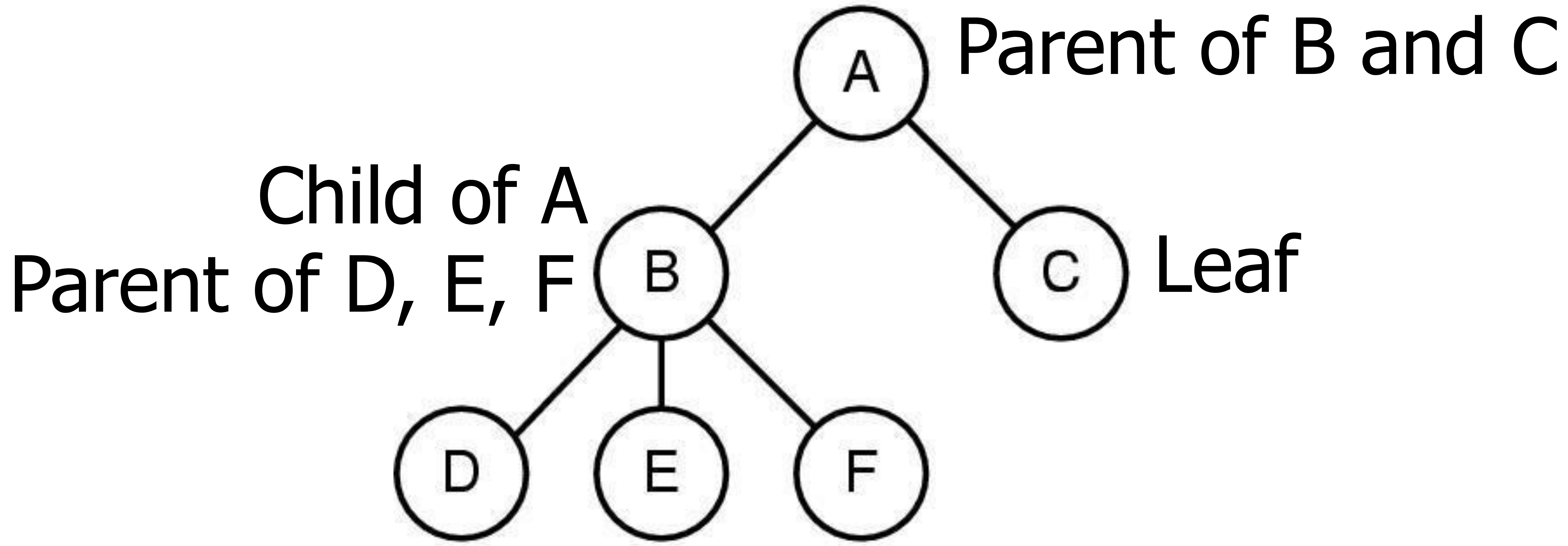
- `int execvp(char * filename, char * argv[]);`
  - Short pathname + arguments in an array

- More info: check “man 3 exec”

Terminating a process

- Return from main()
- Call exit(status)
  - Exit the program.
  - Status is retrieved by the parent using wait().
  - 0 for normal status, non-zero for error

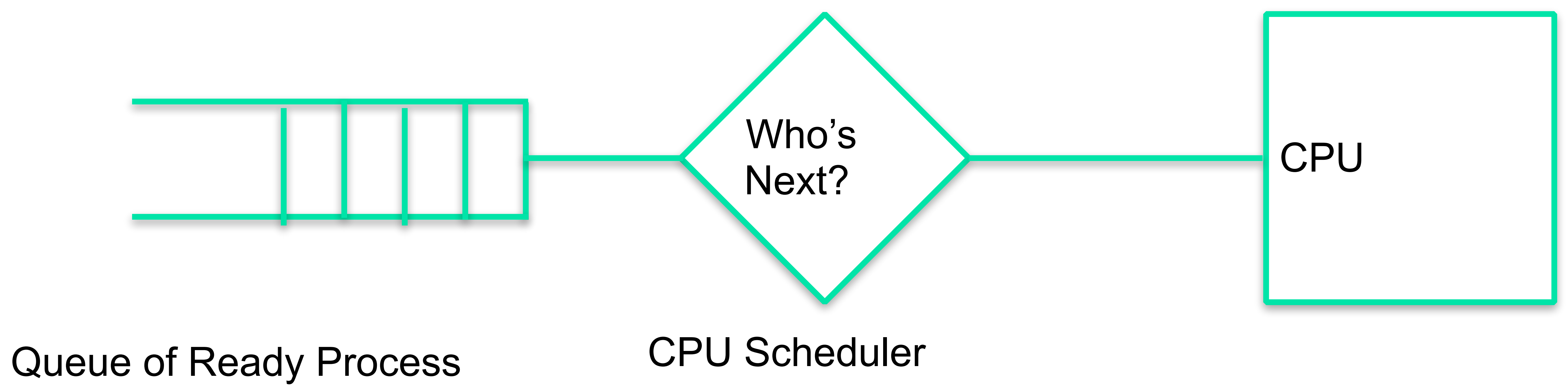
Process Hierarchy Tree



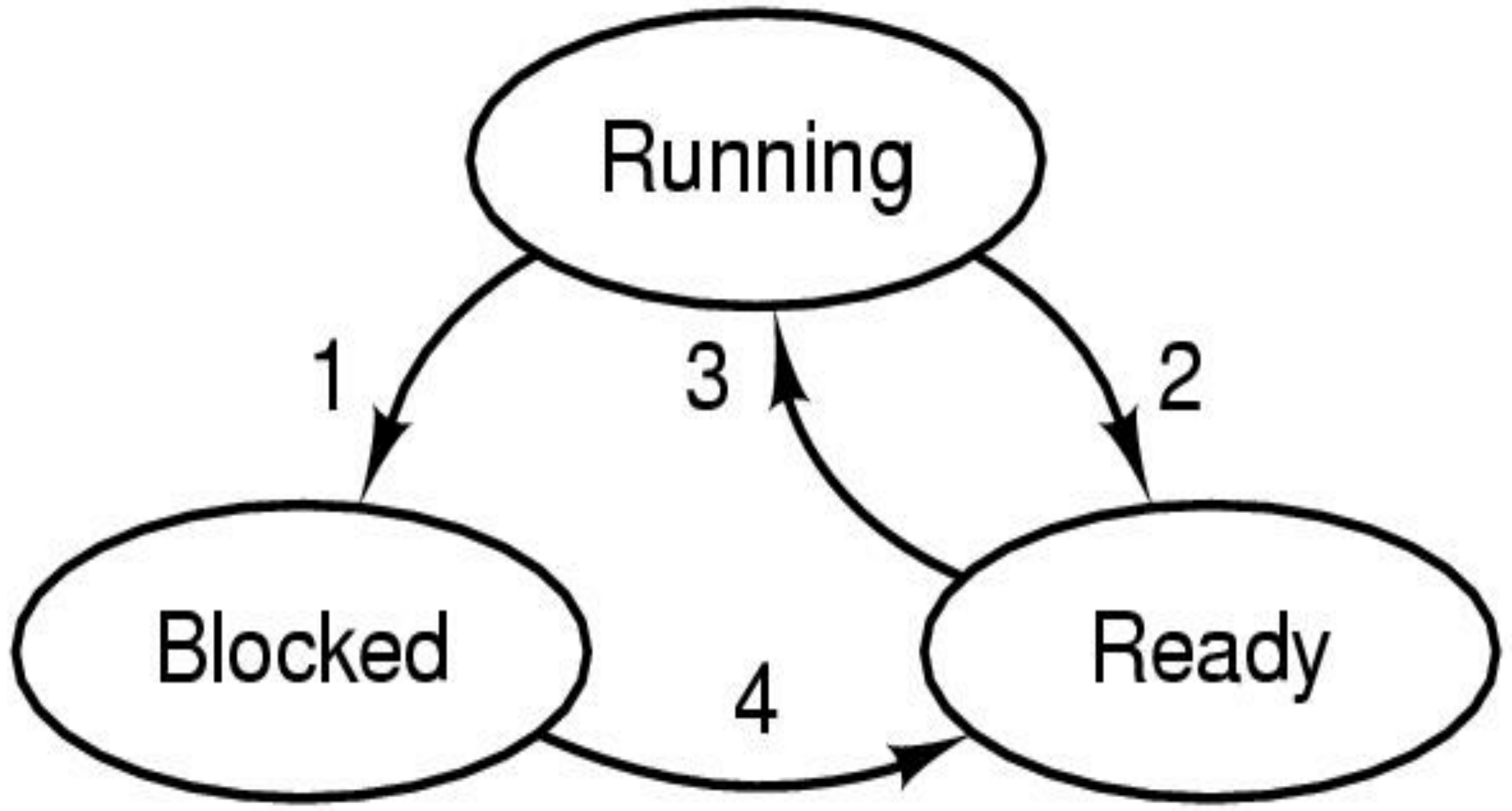
- A created two child processes, B and C
- B created three child processes, D, E, and F

CPU scheduler

- Time-shares many processes on one CPU



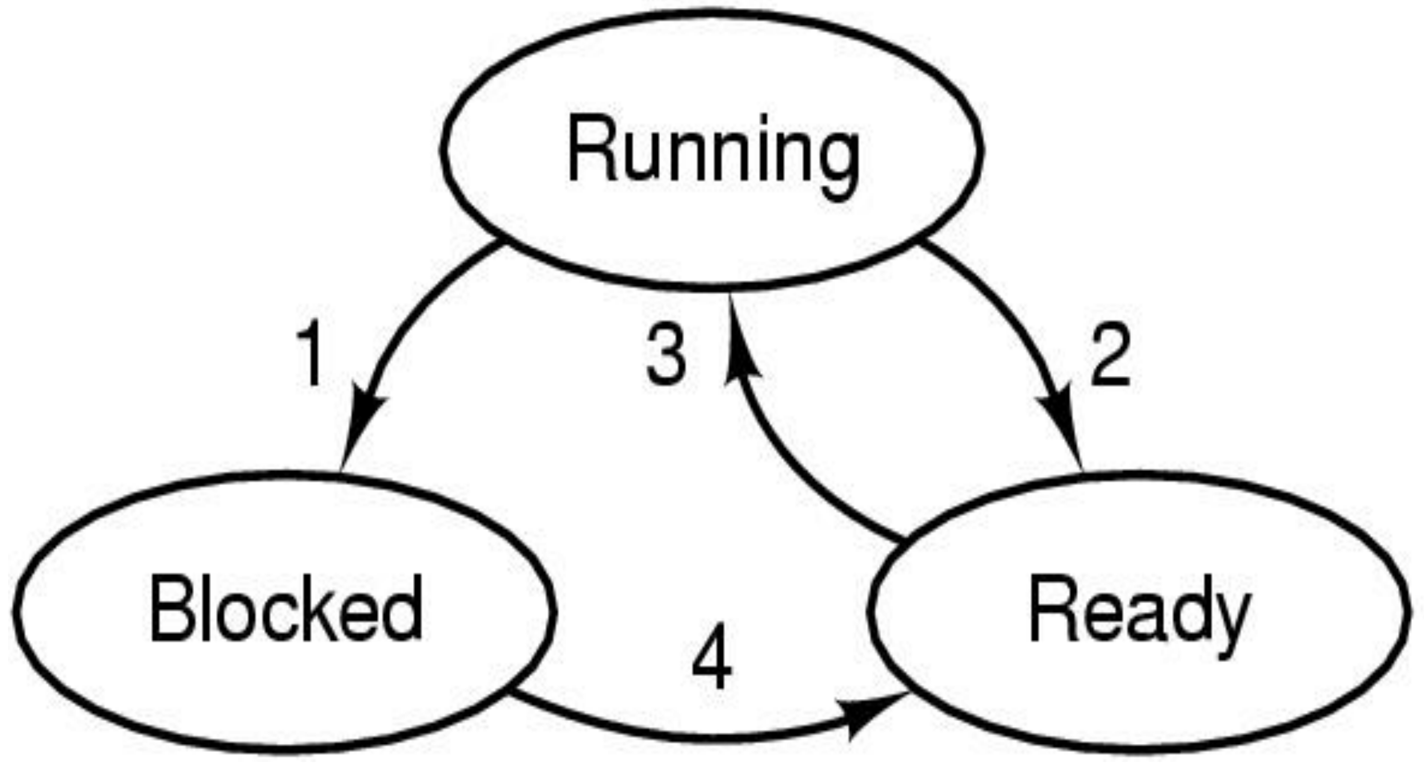
Process Lifecycle



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Ready
  - Process is ready to execute, but not yet executing
  - Its waiting in the scheduling queue for the CPU scheduler to pick it up.
- Running
  - Process is executing on the CPU
- Blocked
  - Process is waiting (sleeping) for some event to occur.
  - Once the event occurs, process will be woken up, and placed on the scheduling queue.

How do multiple processes share CPU?

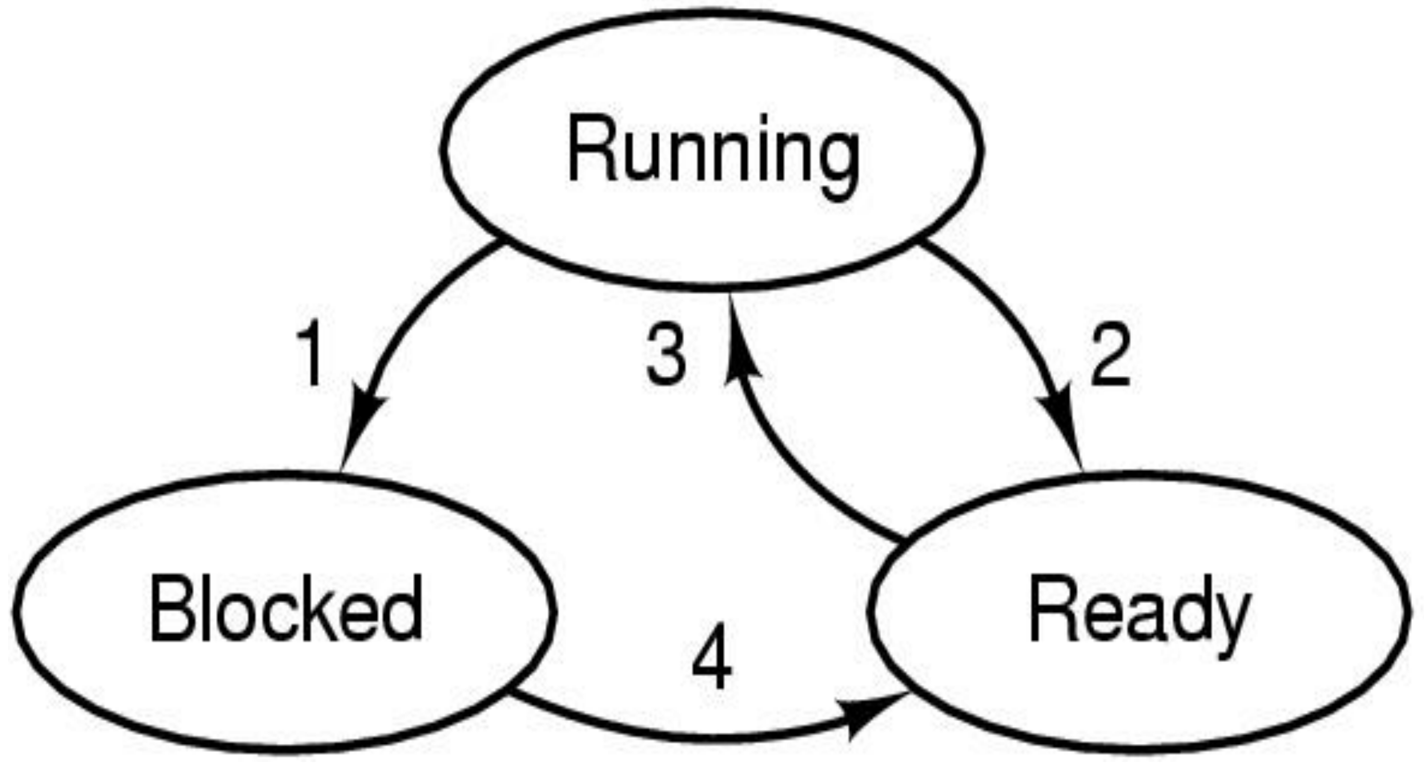


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process_0 now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process_1 now done

Figure 4.3: **Tracing Process State: CPU Only**

How do multiple processes share CPU?



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

Figure 4.4: **Tracing Process State: CPU and I/O**

Examining Processes in Unix/Linux

- ps command
  - Standard process attributes

- /proc directory
  - More interesting information if you are the root.

- top command
  - Examining CPU and memory usage statistics.

Orphan process

- When a parent process dies, child process becomes an orphan process.
- The init process (pid = 1) becomes the parent of the orphan processes.
- Here's an example:
  - https://oscourse.github.io/examples/orphan.c
  - Do a 'ps –l' after running the above program and check parent's PID of the orphan process.
  - After you are done remember to kill the orphan process 'kill –9 <pid>'

Zombie Process

- When a child dies, a SIGCHLD signal is sent to the parent.
- If parent doesn’t wait()on the child, and child exit()s, it becomes a zombie (status “Z” seen with ps).
- Zombies hang around till parent calls wait() or waitpid().
- But they don’t take up any system resources.
  - Just an integer status is kept in the OS.
  - All other resources are freed up.