IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

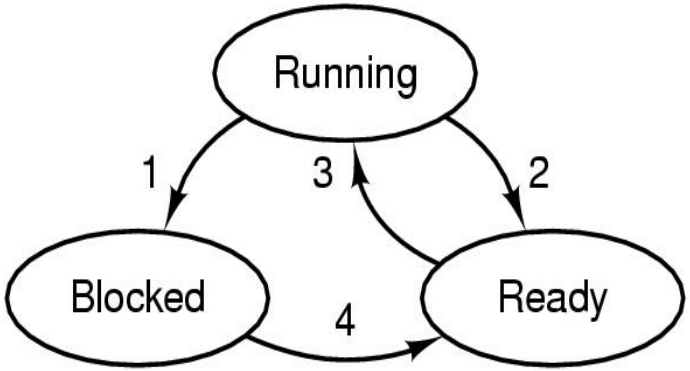
Lecture 07: Processes III

Agenda

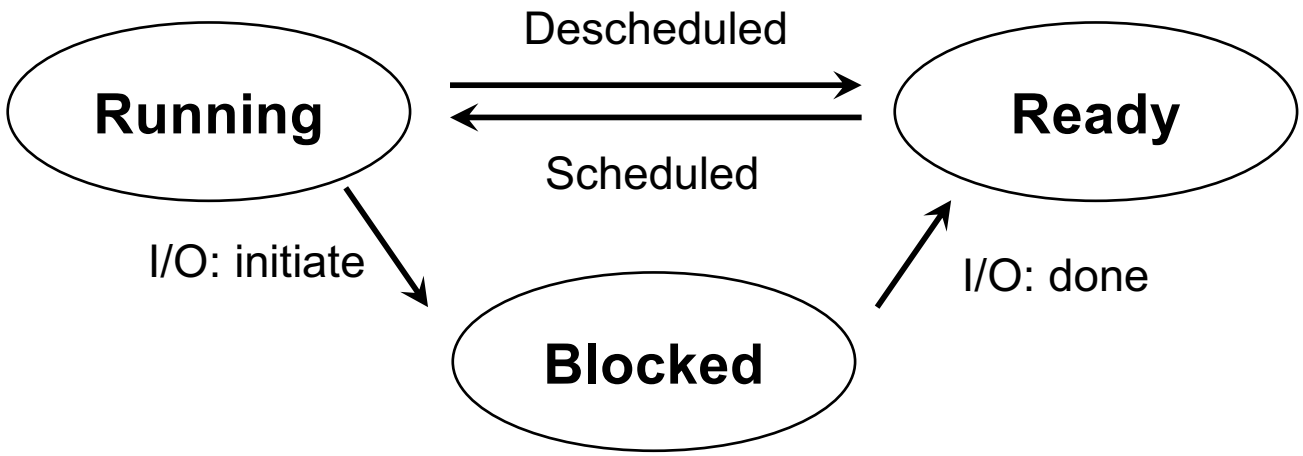
- Recap
- Processes III
  - Process API: fork(), wait(), exec()

Recap

- Process states & transition



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



Recap

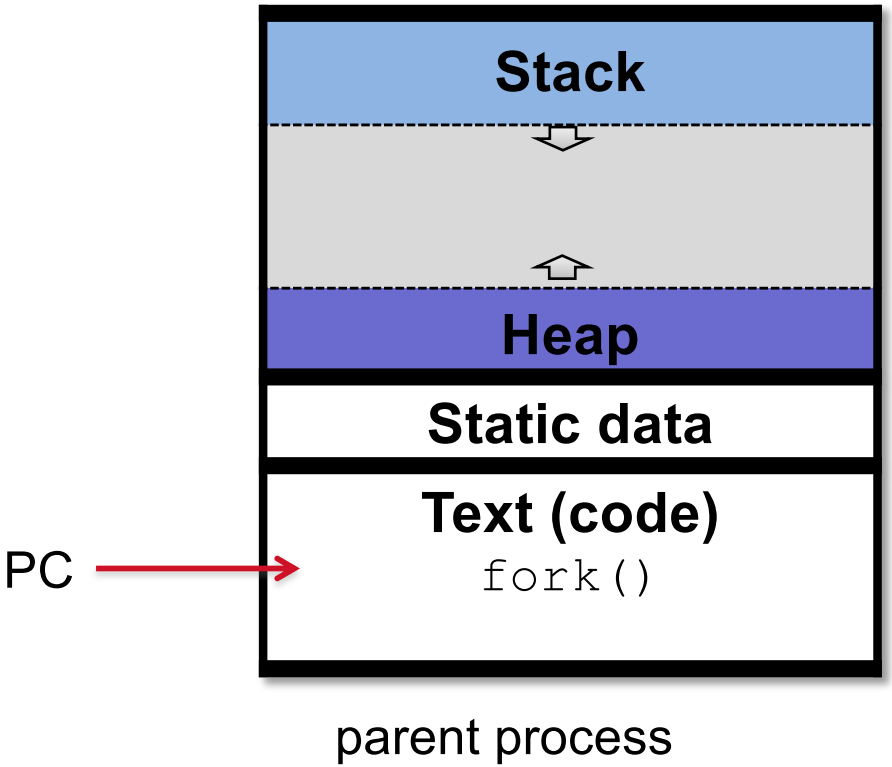
- Process Context
  - Process table: one entry per process
    - each entry is a “Process Control Block (PCB)” containing all information about a process, i.e. “**process context**”
      - registers, program counter, process ID, process state, open files, ...
- Context switch
  - switching the CPU to another process by
    - saving the register values of an old process
      - from CPU to kernel memory (PCB_old)
    - loading the register values of a new process
      - from kernel memory (PCB_new) to CPU

Recap

- Process API: fork()
  - creates a new process by duplicating the calling process
    - The new process is referred to as the **child** process
    - The calling process is referred to as the **parent** process
    - The child process is a copy of the parent process
      - Same core image
      - Same context (except process id): registers, open files, ...
  - On success
    - the **process ID** of the child is returned in the parent
    - **0** is returned in the child
  - On failure
    - -1 is returned in the parent, no child process is created

Recap

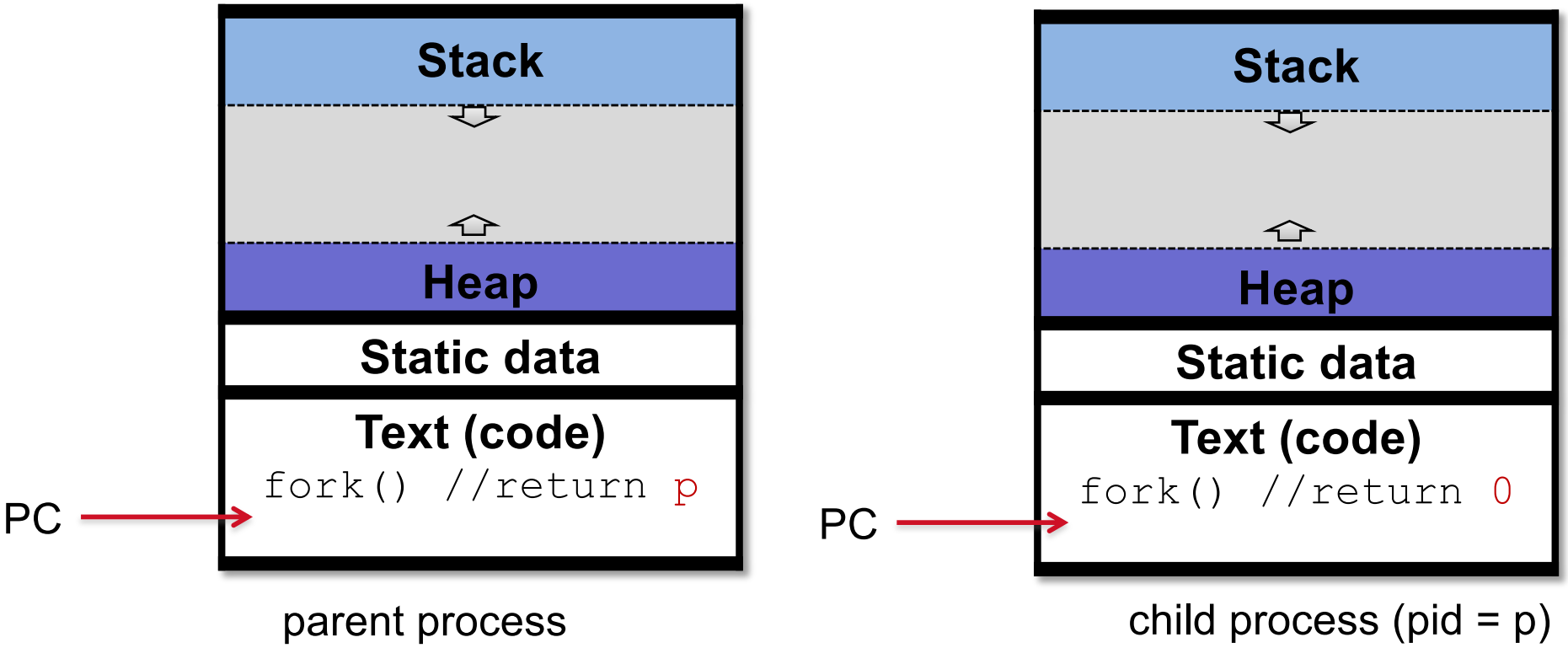
- Process API: fork()
  - before fork():



parent process

Recap

- Process API: fork()
  - after fork():



Agenda

- ~~Recap~~
- Processes III
  - Process API: fork(), wait(), exec()

Process API

- Example: fork()

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

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid()); //get process ID
    int rc = fork();       // create a child process
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

Process API

- Example: fork()
  - results (non-deterministic)
    - assume the code is compiled to an executable named “p1”

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

Process API

- Example: wait()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        int wc = wait(NULL); // wait for child to terminate
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
        rc, wc, (int) getpid());
    }
    return 0;
}
```

Process API

- Example: wait()
  - result (deterministic)
    - assume the code is compiled to an executable named “p2”

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

Process API

- Example: exec()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {                    // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {            // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");           // program: "wc" (word count)
        myargs[1] = strdup("p3.c");         // argument: file to count
        myargs[2] = NULL;                   // marks end of array
        // continue...
```

Process API

- Example: exec()

```
        // continue...
        execvp(myargs[0], myargs); // a variant of exec()
        printf("this shouldn't print out");
    } else {                          // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

- result (deterministic)

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

“Zombie” Process

- A special process state

https://pdos.csail.mit.edu/6.828/2019/xv6.html

```
//xv6 process states
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE};
```

- A process that has completed execution but still has an entry in the process table
  - allow the parent process to read its child's exit status
  - once the exit status is read (via the `wait` system call), the zombie's entry is removed from the process table and it is said to be "reaped”

Agenda

- ~~Recap~~
- ~~Processes III~~
  - ~~Process API: fork(), wait(), exec()~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.