Operating Systems

0107451

Chapter 2 Processes and Threads

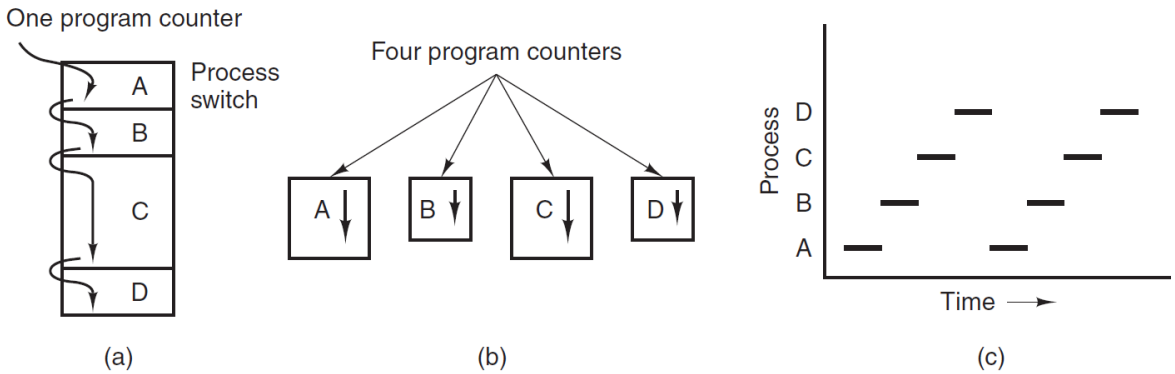
Dr. Naeem Odat

Tafial Technical University
Department of Computer and Communications Engineering

2.1 Processes

The Process Model

Each software runnable on computer is organized as processes including the operating system.



a) 4 processes in memory,
b)each process has its own control flow (logical program counter, registers,...),
c) actual run of each process (it has a small share on CPU)

Processes

Process Creation

Processes are created by 4 principal events:

1. System initialization. Foreground processes to interact with users or background processes (**daemons**) to accept the incoming requisites or respond to incoming notification as email or web server.
2. Execution of a process-creation system call by a running process. Using fork system call in UNIX or CreateProcess with 10 parameters in windows.
3. A user request to create a new process. By running his own program (typing on shell or clicking on icon)
4. Initiation of a batch job. To run the next job in the batch.

In UNIX a parent copy is created when a new child is created (in some implementation copy is not created until one of them, parent or child, needs to write something in its address space, **copy-on-write**).
In windows both images, parent and child, are different from the start.

Processes

Process Creation

```
#include <unistd.h>
pid_t fork(void); //prototype for fork
```

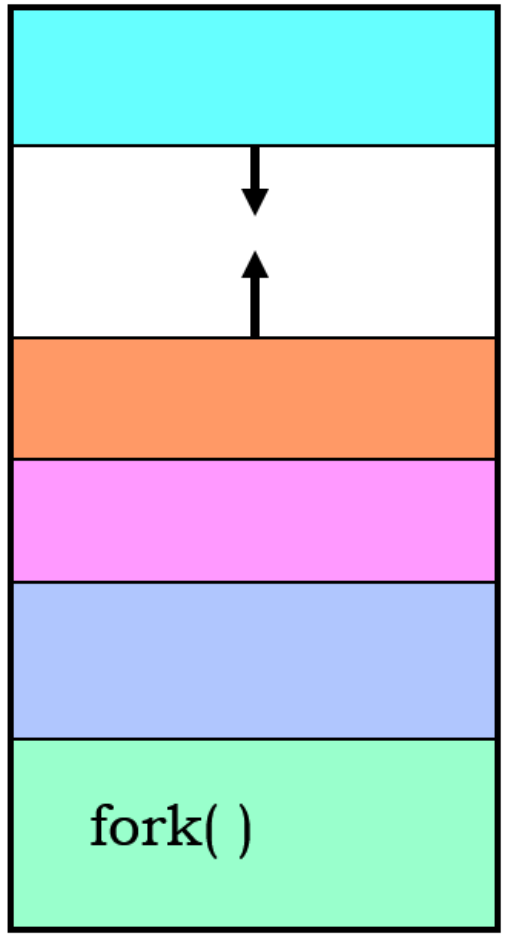
- *fork()* returns a process id (a small integer)
- *fork()* returns twice!
  1. In the parent fork returns the id of the child process
  2. In the child fork returns a 0

Process Creation Example

```
#include <unistd.h>
#include <stdio.h>
void main(void) {
   pid_t pid = fork();
   if (pid > 0)
      printf("I am the parent\n");
   else if (pid == 0)
      printf("I am the child\n");
   else
     printf("ERROR!\n");
}
```

Processes

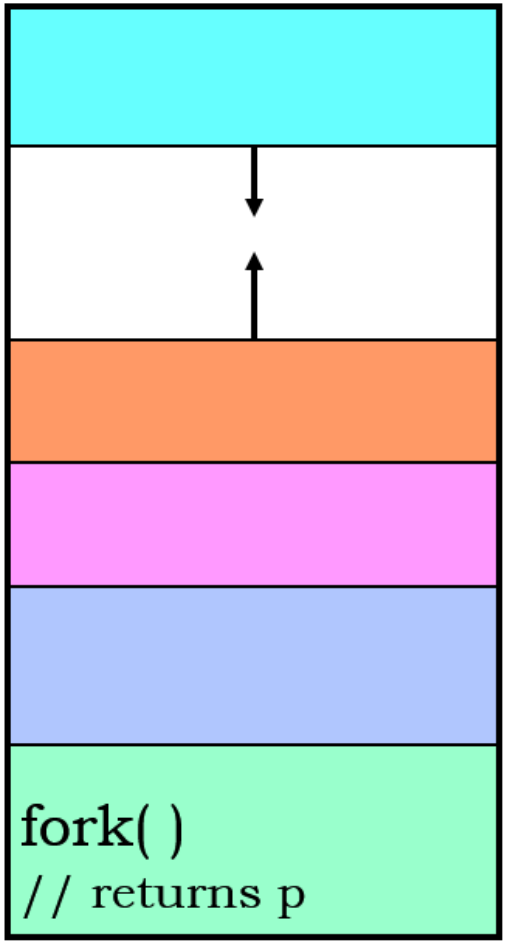
Process Creation - Before *fork()*



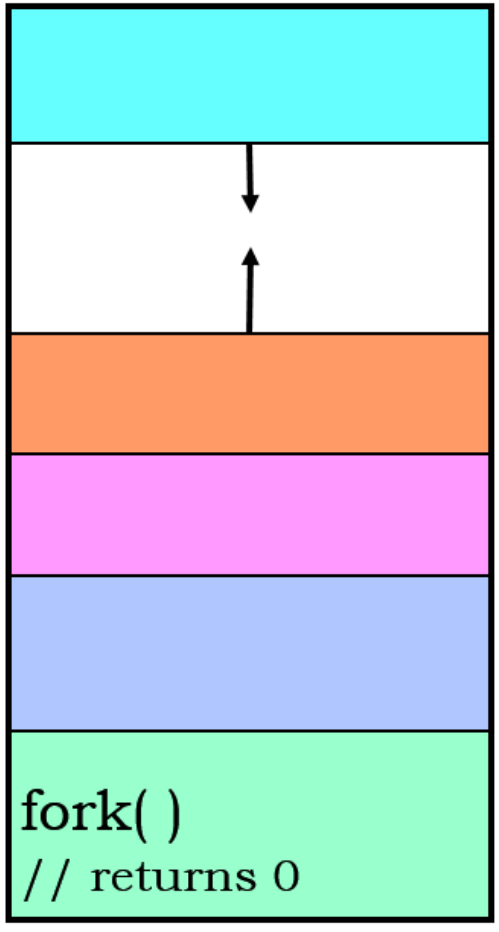
Parent process

Processes

Process Creation - After *fork()*



Parent process



Child process

Processes

Parent and child processes

- The child process is a *copy* of the parent process
  - It is running the same program.
  - Same memory contents
  - It has its own process ID
- The child process inherits many *attributes* from the parent, including:
  - Current working directory
  - User id
  - Group id

Processes

Process Hierarchies

- UNIX parent creates a child process, child can create more processes
  - The first process created when system booted is "init".
  - Forms a hierarchy rooted at init.
  - UNIX calls this a **process group**. All processes within a group are logically related
- Windows has no concept of process hierarchy
  - all processes are equal and identified by a handler.

Process Termination

- A process usually terminated due to:
  1. Normal exit (voluntary). The process completed its work.
  2. Error exit (voluntary). When a process is given bad parameter (gcc na.c, na.c doesn't exist).
  3. Fatal exit (involuntary). The process accessed nonexistent memory, or divide something by zero.
  4. Killed by another process (involuntary). By issuing kill system call in UNIX or TerminateProcess in Windows.
- When a process exits, the OS delivers a termination status to the parent process.

More on *fork()*

How many processes does this piece of code create?

```
int main(){
    fork();
    fork();
}
```

Processes

Bad example, don't try

```
#include<unistd.h>
#include<stdio.h>
void main(void){
    while(!fork())
        printf("I am the child %d\n",getpid());
    printf("I am the parent %d\n", getpid());

}
```

fork()

What is the output of this?

```
int main(){
    int i;
    for(i=0; i<3; i++){
        fork();
        printf("%d\n", i);
    }
    return 0;
}
```

Waiting

- Parent processes often wait for their child processes to end.
- This waiting is done via *wait* system call:
  - *pid_t wait(int* status);*
  - *pid_t waitpid(pid_t pid, int* status, ...);*

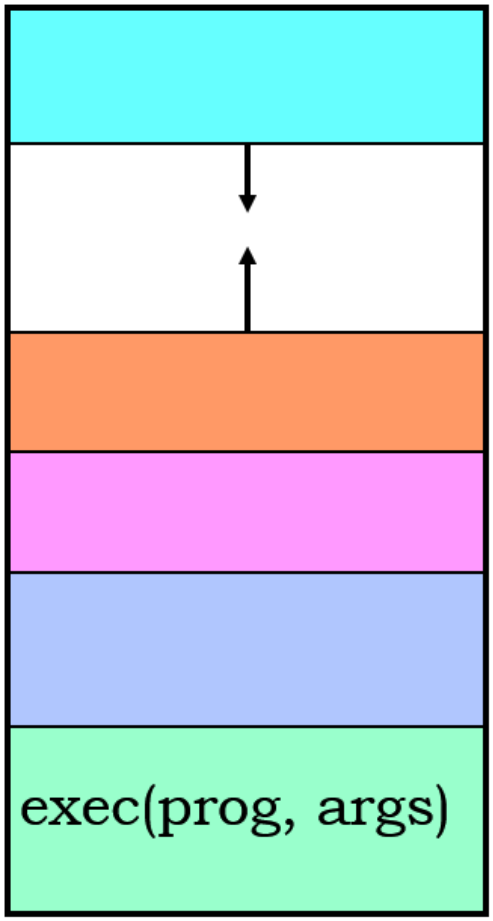
Processes

Switching programs

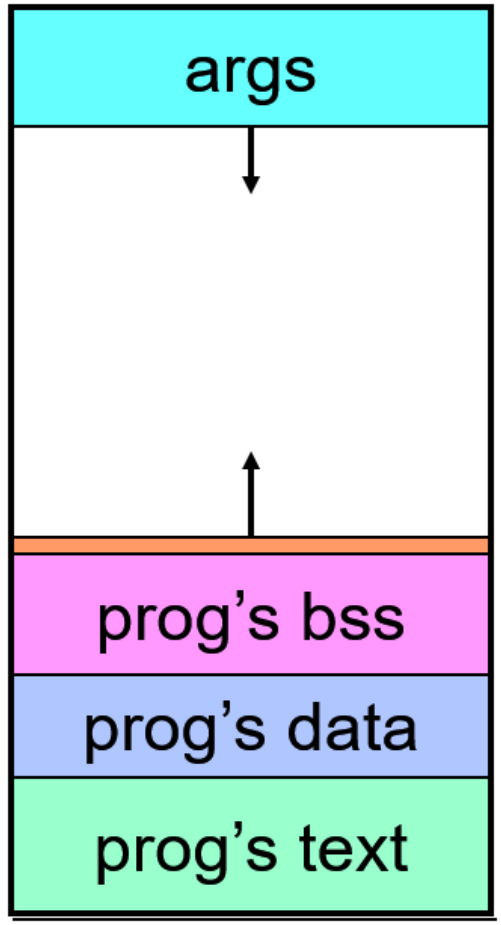
- *fork()* creates new process.
- There should be a way to switch to a program within the new created process.
- *exec* system calls family is used to load new program to an existing process.

Processes

exec(): Loading a new image



Before



After

exec() example

```
#include<unistd.h>
main(){
    printf("executing ls\n");
    execl("/bin/ls", "ls", "-l", (char*)0);
    /*if execl returns, the call failed*/
    perror("execl failed to run ls");
    exit(1);
}
```

Processes

Process Creation-Stripped Down Shell

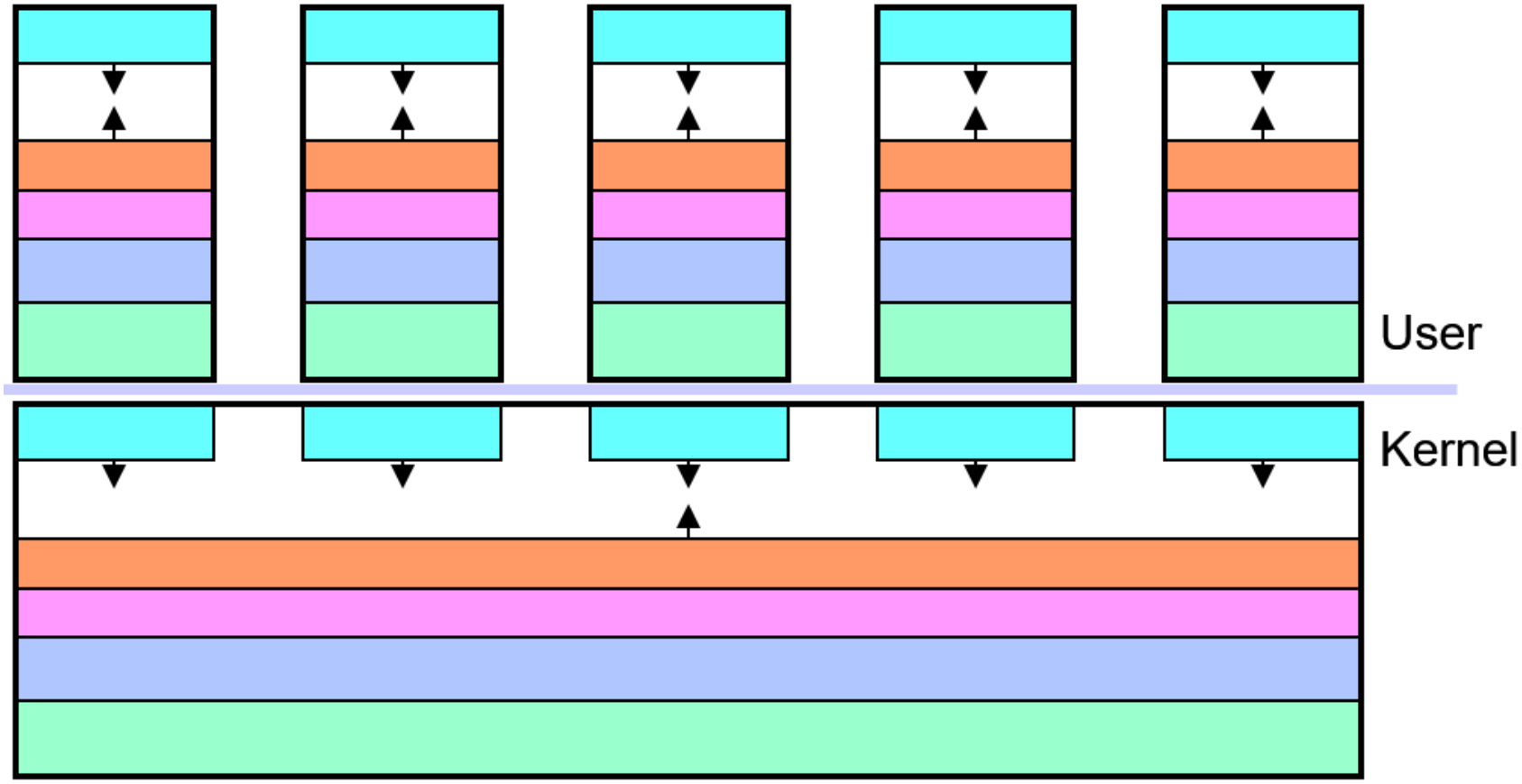
```
while (TRUE) { /*repeat forever*/
    type_prompt( ); /* display prompt */
    /* input from terminal */
    read_command (command, parameters);

    if (fork() != 0) {/* fork off child process */
        /*   Parent code */
        /* wait for child to exit */
        waitpid( -1, &status, 0);
    } else {
        /*  Child code */
        /* execute command */
        execve (command, parameters, 0);
    }
}
```

Processes and Multiprogramming

- Multiprogramming: Many processes executing in parallel.
- Imagine not having multiprogramming (e.g., DOS)
  - Type in command
  - Wait for results
  - Cannot surf Internet in the meanwhile

Processes

Processes and Multiprogramming



Processes and Multiprogramming

On single processor, will the following get speeded up with multiprogramming?

- Four compilations running in parallel.
- A compilation and a text editor
- Four parallel Internet downloads.

Processes

Process context

- Context switch: Resources (CPU) taken from one process and given to another.
- Context switch: Save the context of previous process and restore the context of new process.
- Process context:

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

- Operating system stores process contexts in the process table one entry (process control block) per process.

Processes

When to switch context?

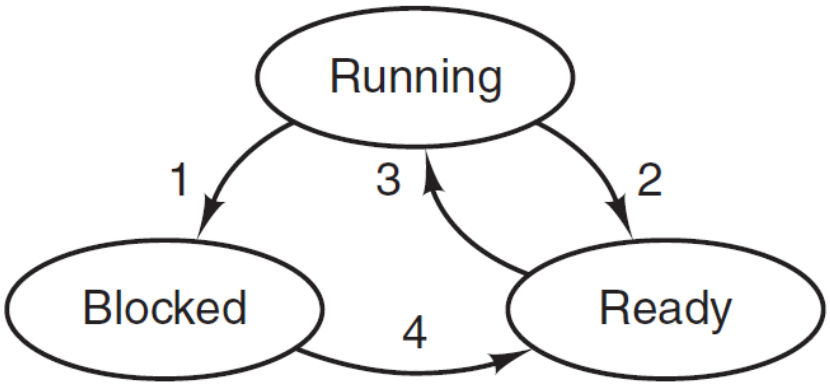
- When a process waits for I/O
- When a process has got its quota of time.
- When a previously started I/O has completed.

How to switch context?

- How would you force a process to give up a processor to another process?
- Interrupt driven context switch:
  1. Interrupt occurs (timer or I/O)
  2. Each interrupt has its own service procedure (address given by interrupt vector)
  3. Save some context and then jump to interrupt service procedure.
  4. Scheduler might context switch after the interrupt is served.

Processes

Process States



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

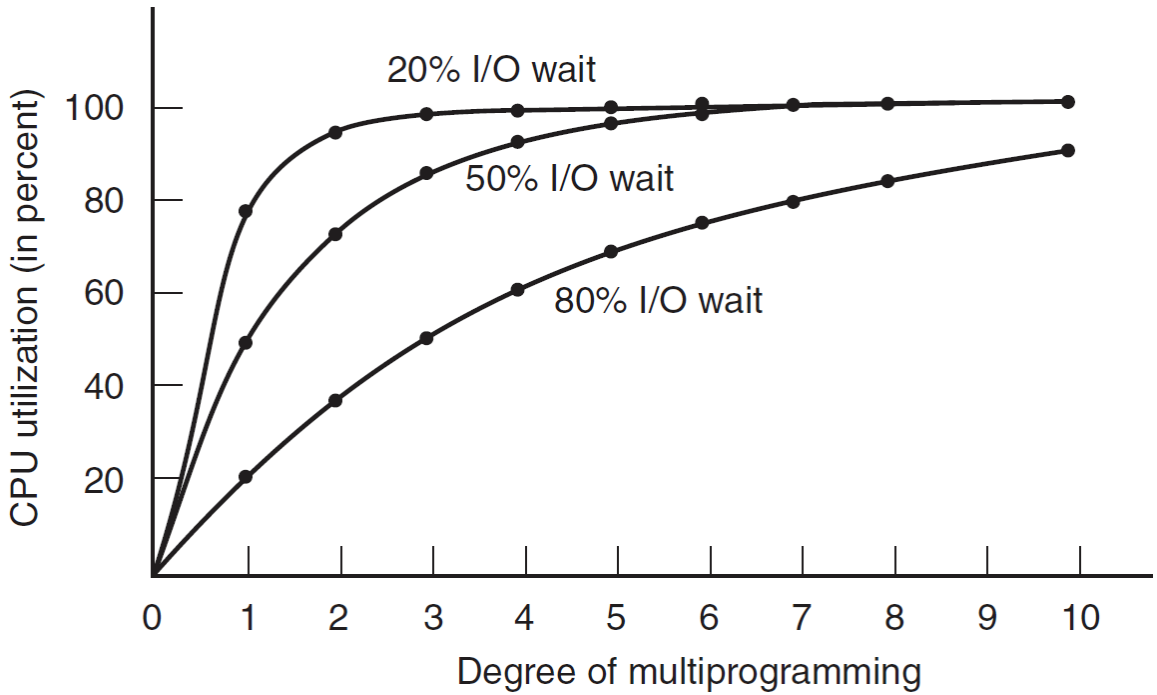
Implementation of Processes

Every I/O has an interrupt vector in a vector table located usually in a low memory area. When an interrupt occurs several steps are taken:

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly-language procedure saves registers.
4. Assembly-language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly-language procedure starts up new current process.

Modeling Multiprogramming

From a probabilistic viewpoint. Suppose that a process spends a fraction p of its time waiting for I/O to complete. With n processes in memory at once, the probability that all n processes are waiting for I/O (in which case the CPU will be idle) is p^n. The CPU utilization is then given by the formula: **CPU utilization = 1 - p^n**



Processes

Questions

- Why can't we voluntarily change to privileged (kernel) space?
- Why do we need a separate kernel stack for each process?
- Is a scheduler a separate process?