Structure of Processes

CSI 309: Operating System Concepts
United International University

PROCESSES AND PROGRAMS

Users, Programs, Processes

- Users have accounts on the system
- Users launch programs
  – Many users may launch same program
  – One user may launch many instances of the same program
- Program: an algorithm expressed in some suitable notation.
- Process: a program in execution.

What is a process?

- A task created by the OS, running in a restricted virtual machine environment –a virtual CPU, virtual memory environment, interface to the OS via system calls
- The unit of execution
- Operating system provided ***abstraction*** to represent what is needed to run a single program
- The same as “job” or “task” or “sequential process”.

Example: Windows Task Manager

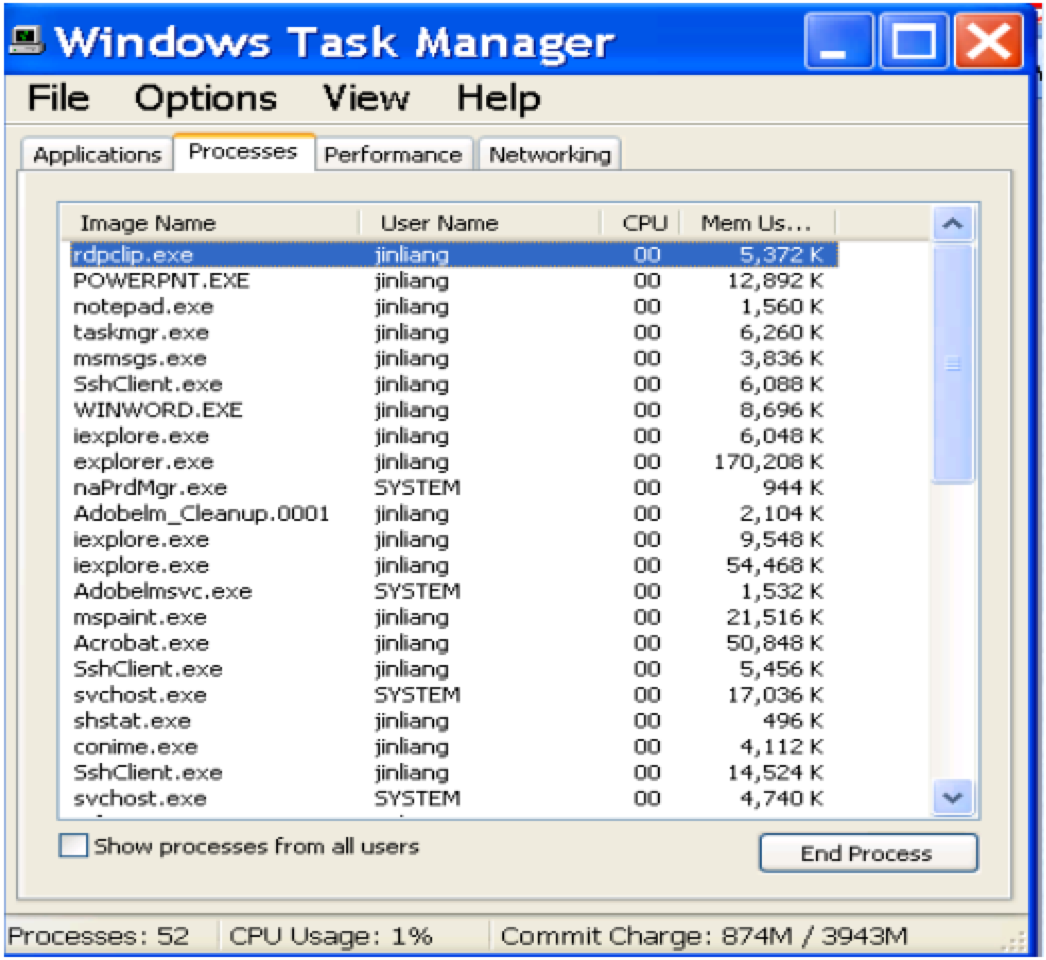
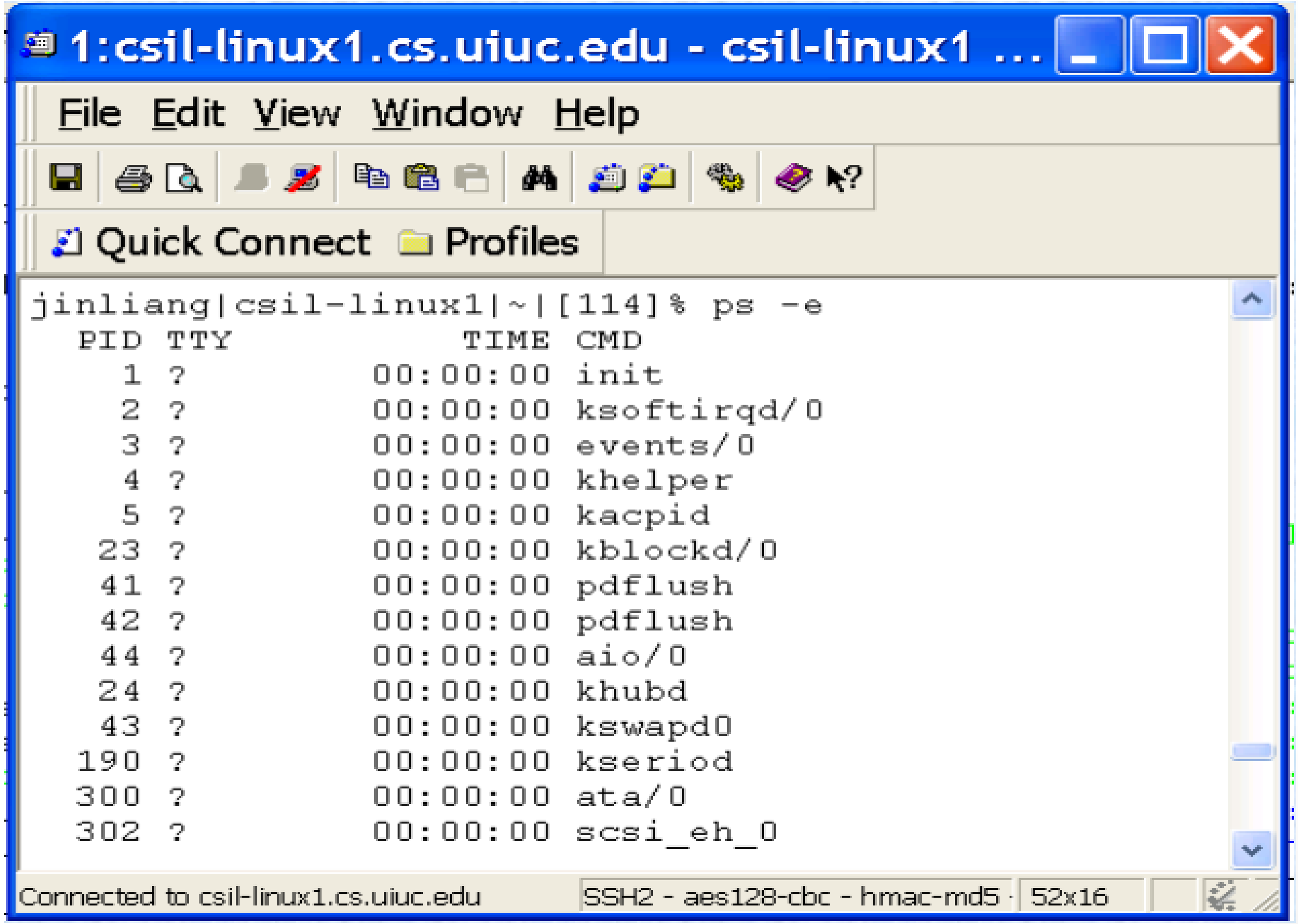


Image Name	User Name	CPU	Mem Us...
rdpclip.exe	jinliang	00	5,372 K
POWERPNT.EXE	jinliang	00	12,892 K
notepad.exe	jinliang	00	1,560 K
taskmgr.exe	jinliang	00	6,260 K
msmsgs.exe	jinliang	00	3,836 K
SshClient.exe	jinliang	00	6,088 K
WINWORD.EXE	jinliang	00	8,696 K
iexplore.exe	jinliang	00	6,048 K
explorer.exe	jinliang	00	170,208 K
naPrdMgr.exe	SYSTEM	00	944 K
Adobelm_Cleanup.0001	jinliang	00	2,104 K
iexplore.exe	jinliang	00	9,548 K
iexplore.exe	jinliang	00	54,468 K
Adobelmsvc.exe	SYSTEM	00	1,532 K
mspaint.exe	jinliang	00	21,516 K
Acrobat.exe	jinliang	00	50,848 K
SshClient.exe	jinliang	00	5,456 K
svchost.exe	SYSTEM	00	17,036 K
shstat.exe	jinliang	00	496 K
conime.exe	jinliang	00	4,112 K
SshClient.exe	jinliang	00	14,524 K
svchost.exe	SYSTEM	00	4,740 K

□ Show processes from all users

Processes: 52 | CPU Usage: 1% | Commit Charge: 874M / 3943M

Example: ps in Unix


1:csil-linux1.cs.uiuc.edu - csil-linux1 ...
File Edit View Window Help
Quick Connect
Profiles

```
jinliang|csil-linux1|~|[114]% ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 ksoftirqd/0
    3 ?        00:00:00 events/0
    4 ?        00:00:00 khelper
    5 ?        00:00:00 kacpid
   23 ?        00:00:00 kblockd/0
   41 ?        00:00:00 pdflush
   42 ?        00:00:00 pdflush
   44 ?        00:00:00 aio/0
   24 ?        00:00:00 khubd
   43 ?        00:00:00 kswapd0
  190 ?        00:00:00 kseriod
  300 ?        00:00:00 ata/0
  302 ?        00:00:00 scsi_eh_0
```

Connected to csil-linux1.cs.uiuc.edu SSH2 - aes128-cbc - hmac-md5 52x16

What is a program?

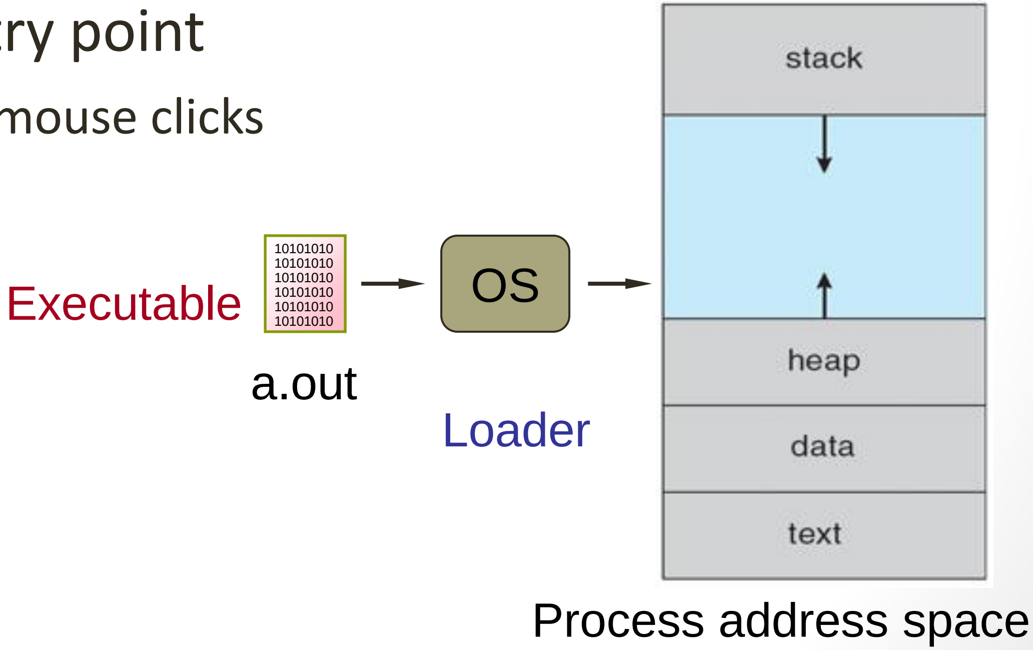
- A Program is an **executable file** that contains:
  - Code: Machine instructions
  - Data: Variables stored and manipulated in memory
    - initialized variables (globals)
    - dynamically allocated variables (malloc, new)
    - stack variables (C automatic variables, function arguments)
  - DLL: libraries that were not compiled or linked with the program
    - containing code & data, possibly shared with other programs

Process != Program

- A process is an executing program.
- Example:
  - We can run 2 instances of Mozilla Firefox:
    - Same program
    - Separate processes
- Program is passive: Code + data
- Process is running program: stack, registers, program counter

How Program Becomes a Process

- When a program is launched
  - OS loads program into memory
  - Creates kernel data structure for the process
  - Initializes data
  - Starts from an entry point
    - e.g., main(), GUI mouse clicks



Executable

a.out

OS

Loader

stack

heap

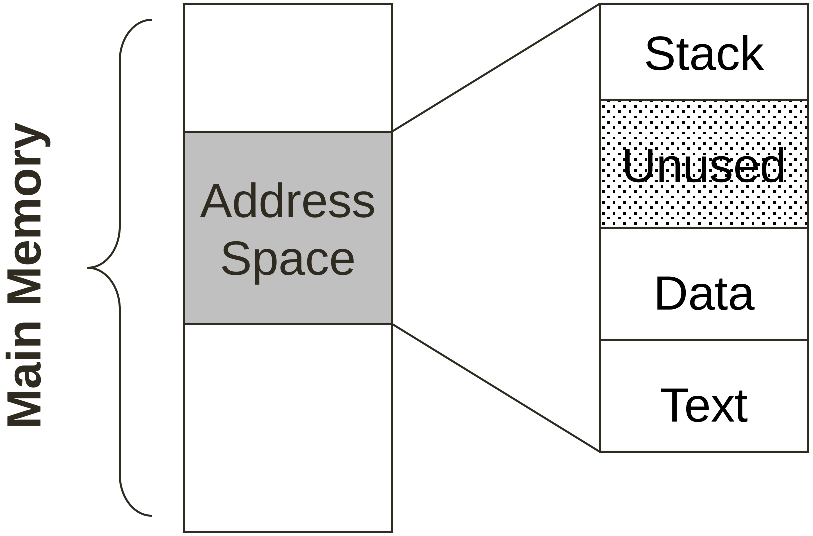
data

text

Process address space

Process address space

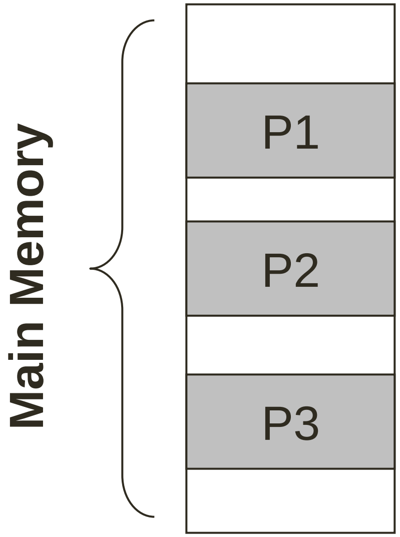
- set of all memory addresses accessible by a process



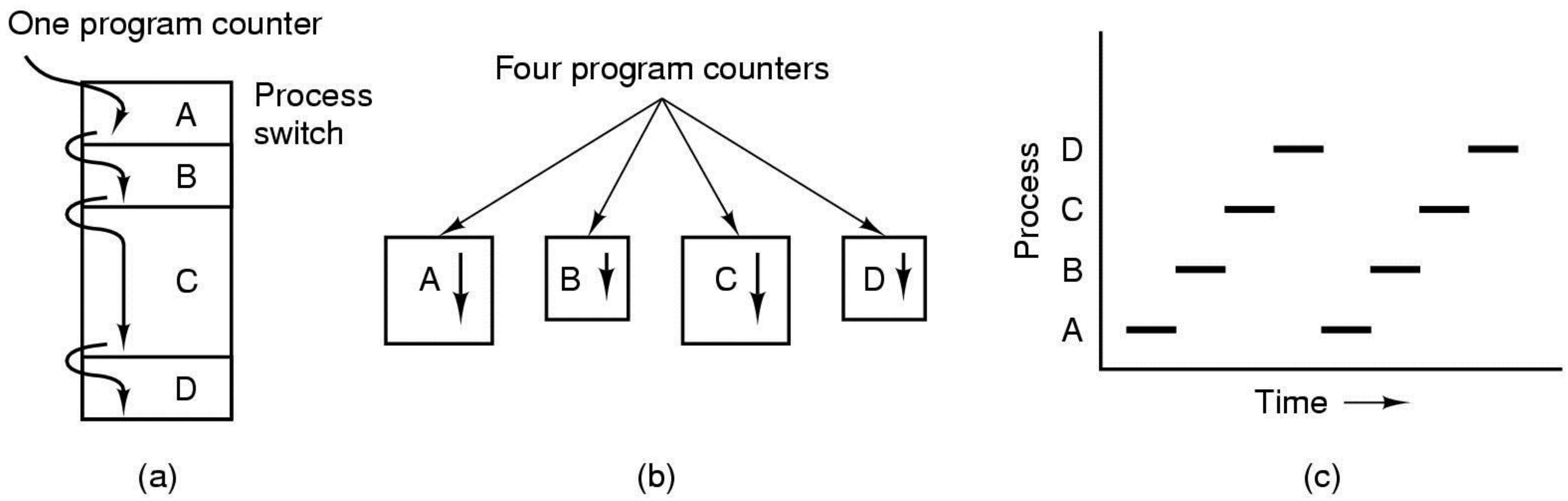
MULTIPROGRAMMING

Multiprogramming

- Each process has its own address space (virtual memory address)
- Even if two processes are running the same program, they have their own address space
- OS schedules processes



Multiprogramming



- Multiprogramming of 4 processes in a single CPU

  CPU switches from one process to other process
- Only 1 physical program counter

  4 logical program counters
- Conceptual model of 4 independent, sequential processes
- Only 1 program active at any instant.
- Real life analogy?

  A daycare teacher trying to feed 4 infants.

PROCESS OPERATIONS

Process Creation

Process Termination

Process State Transitions

Process Creation

- System initialization
  – Boot, reboot.
- Execution of a process creation system call
  – fork()
- User request to create a new process
  – Command line or click an icon.
- Initiation of a batch job

Process Termination

- Normal exit (voluntary)
  - End of main()
- Error exit (voluntary)
  - exit(2)
- Fatal error (involuntary)
  - Divide by 0, core dump
- Killed by another process (involuntary)
  - kill procID, end task

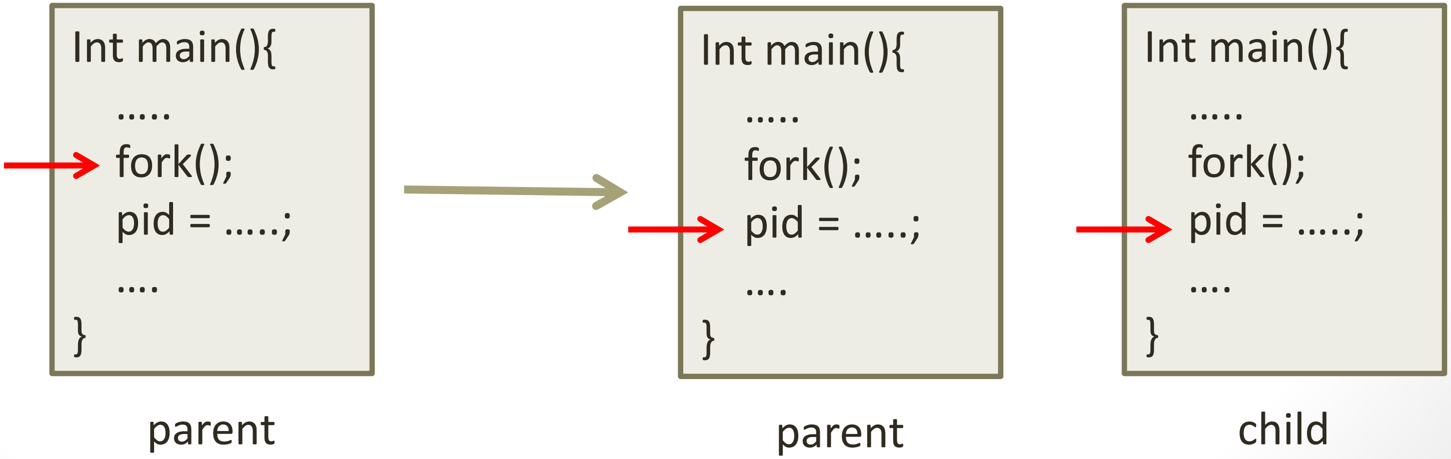
Example: fork() in Unix

- int childpid;

-
  ```
  if ((childpid = fork()) == -1) {
          perror(can’t create a new process);
          exit(1);
  } else if (childpid == 0) {
          // executes child process code
          exit(0);
  } else {
          // executes parent process code
          exit(0);
  }
  ```

The fork() System Call

- When fork() is called in process A
  - Control is switched to kernel
  - Kernel creates new process B by copying A's
    - Address space
    - Kernel data structure (process descriptor)
  - OS now has two identical processes to run.
    Both resume from after fork().
    However, return value of fork() will be different.



```
Int main(){
    …..
    fork();
    pid = …..;
    ….
}
```

parent

```
Int main(){
    …..
    fork();
    pid = …..;
    ….
}
```

parent

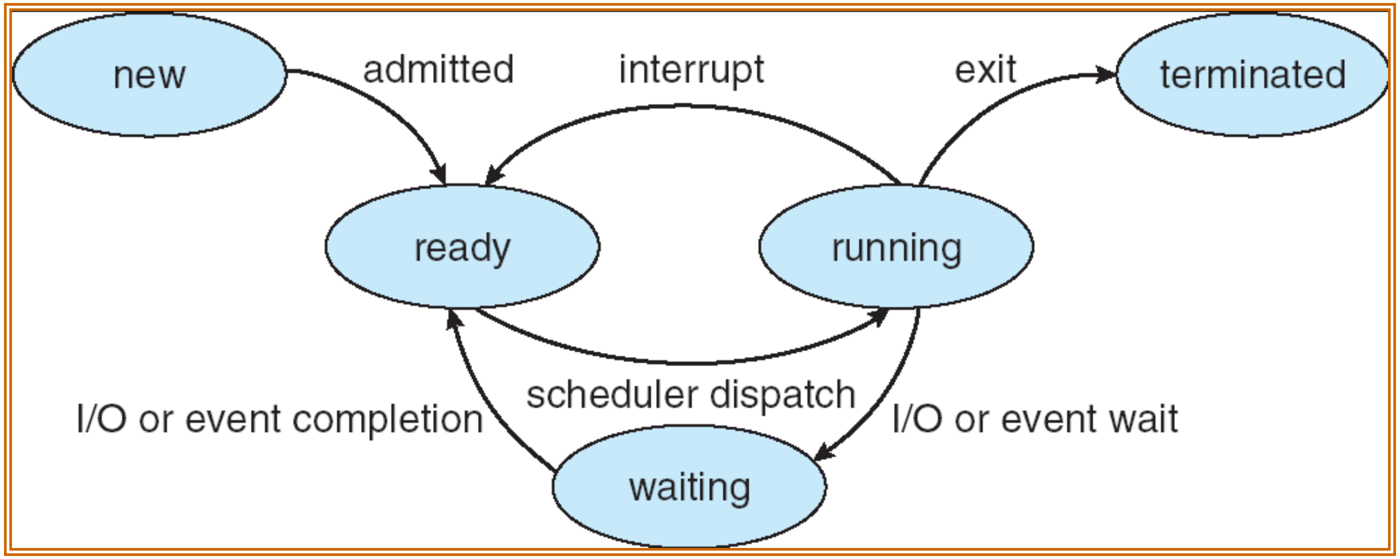
```
Int main(){
    …..
    fork();
    pid = …..;
    ….
}
```

child

Process States

- Many processes in system, only one on CPU
- “Execution State” of a process:
  - Indicates what it is doing
  - Basically 3 states:
    1. Ready: waiting to be assigned to the CPU
    2. Running: executing instructions on the CPU
    3. Waiting: waiting for an event, e.g. I/O completion
- Process moves across different states

Process State Transitions



- As a process executes, it changes *state*
    1. new: The process is being created
    2. ready: The process is waiting to run
    3. running: Instructions are being executed
    4. waiting: Process waiting for some event to occur
    5. terminated: The process has finished execution

Process State Transitions

Processes hop across states as a result of:

- Actions they perform, e.g. system calls
- Actions performed by OS, e.g. rescheduling
- External actions, e.g. I/O

Process State Transitions

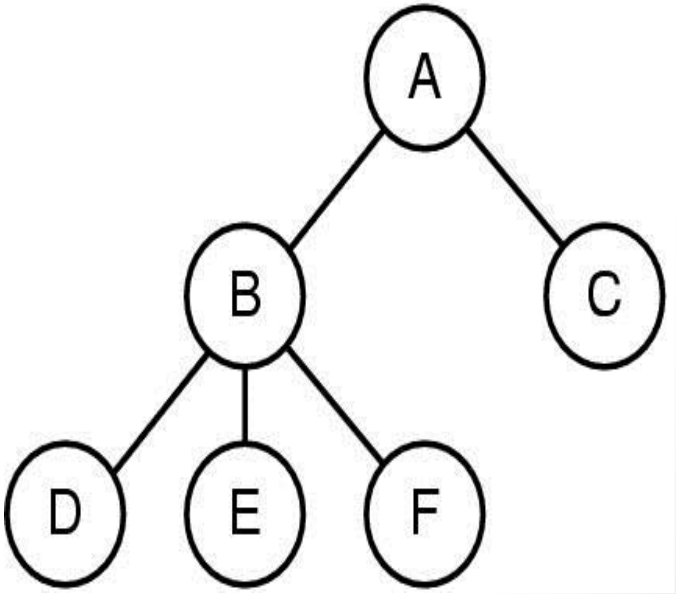
- Running → Block**:** process discovers that it cannot continue. If running process initiates an I/O operation before its allotted time expires, the running process voluntarily relinquishes the CPU.
- Running → Ready: scheduler decides that the running process has run long enough and it is time to let another process have CPU time.
- Ready → Running: all other processes have had their share and it is time for the first process to run again
- Blocked → Ready: external event for which a process was waiting (such as arrival of input) happens.
- New → Ready : process is created.
- Running → Terminated : process has finished execution.

PROCESS HIERARCHIES
AND
PROCESS DATA STRUCTURES

Process Hierarchies

- When process A creates process B, A is called the “parent” process, B is the “child”
- Forms a hierarchy
  - – UNIX calls this a "process group“
  - – A special process present in boot image is init
- Windows has no process hierarchy
  - – Processes are independent

    after creation



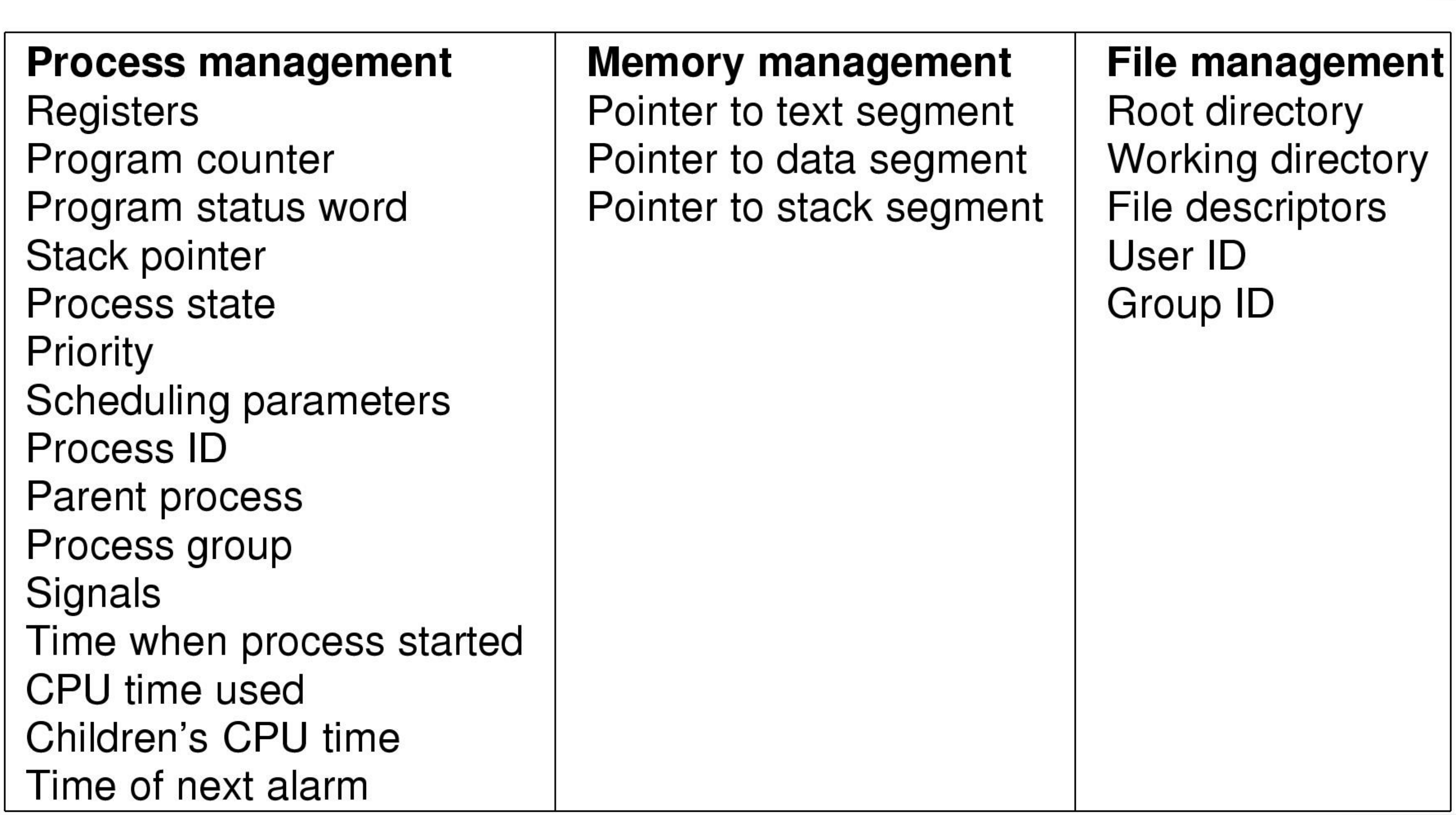
Process Data Structures

- OS represents a process using a PCB
  - – Process Control Block
  - – Has all the details of a process

Process Id	**Security Credentials**
Process State	**Username of owner**
General Purpose Registers	**Queue Pointers**
Stack Pointer	**Signal Masks**
Program Counter	**Memory Management**
Accounting Info	**…**

Process Control Block (PCB)

Fields of a Process Table Entry



Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	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		

Reference

Modern Operating Systems

Section 2.1