Processes

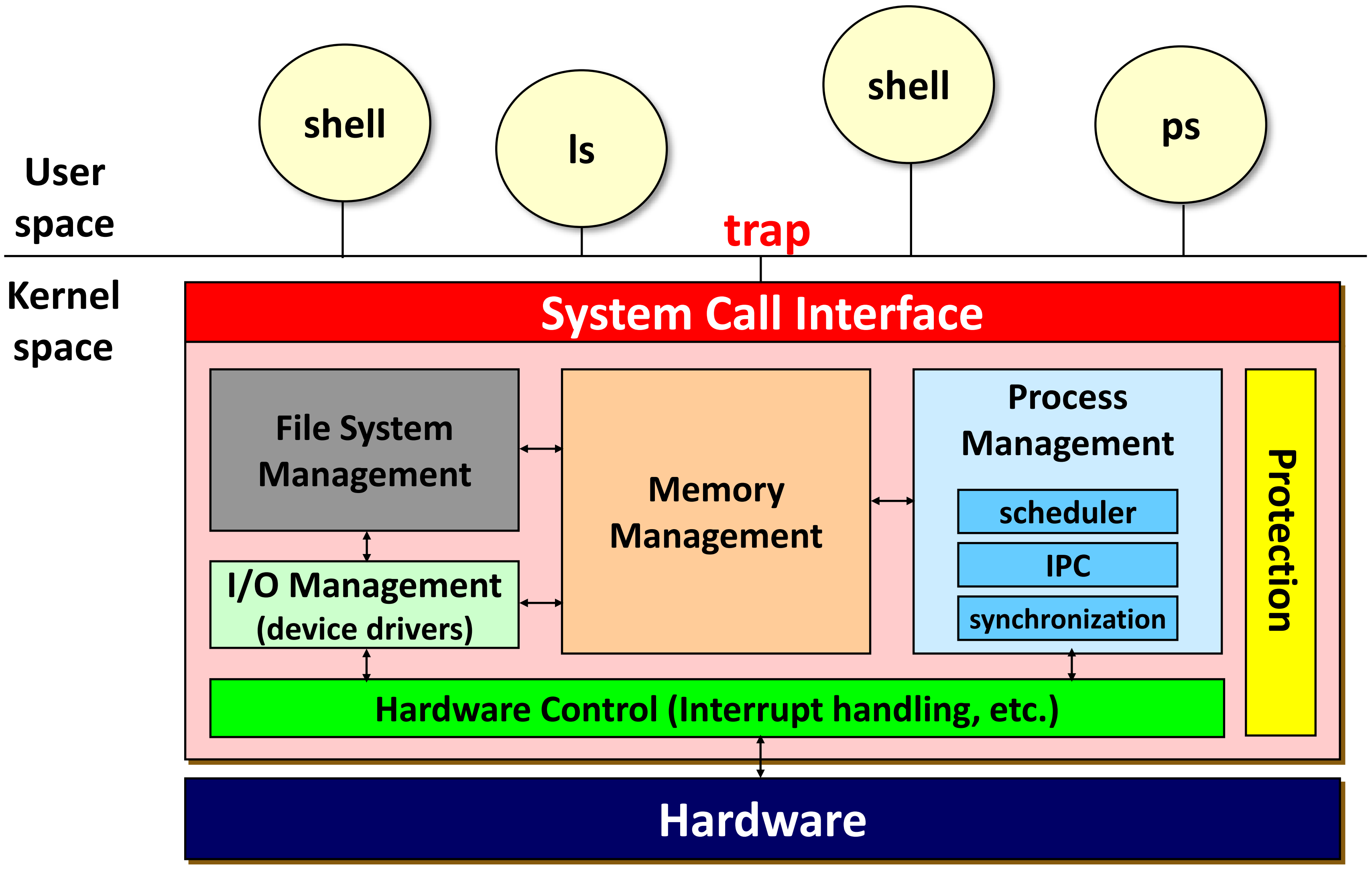
Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2024

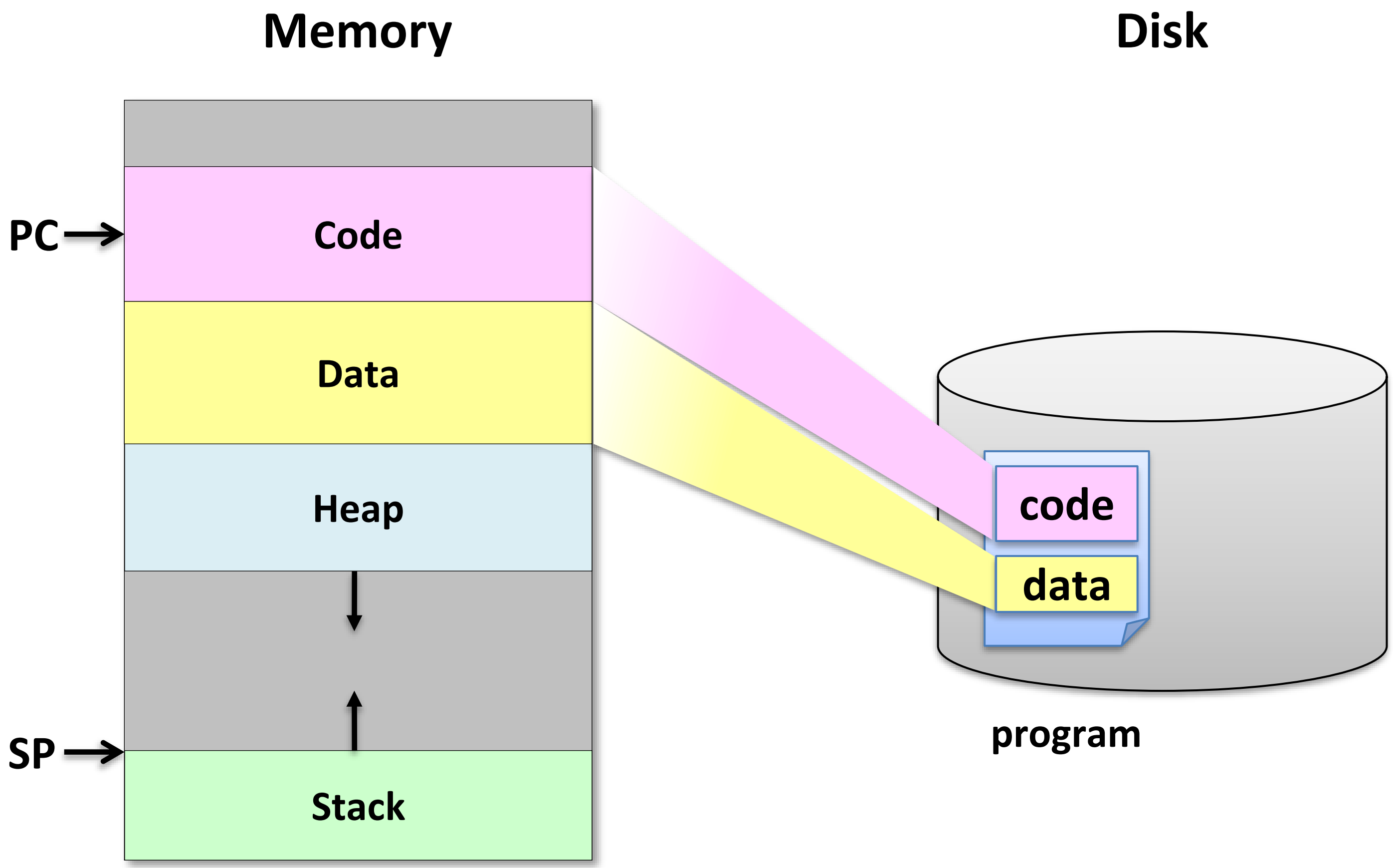
OS Internals



What is a Process?

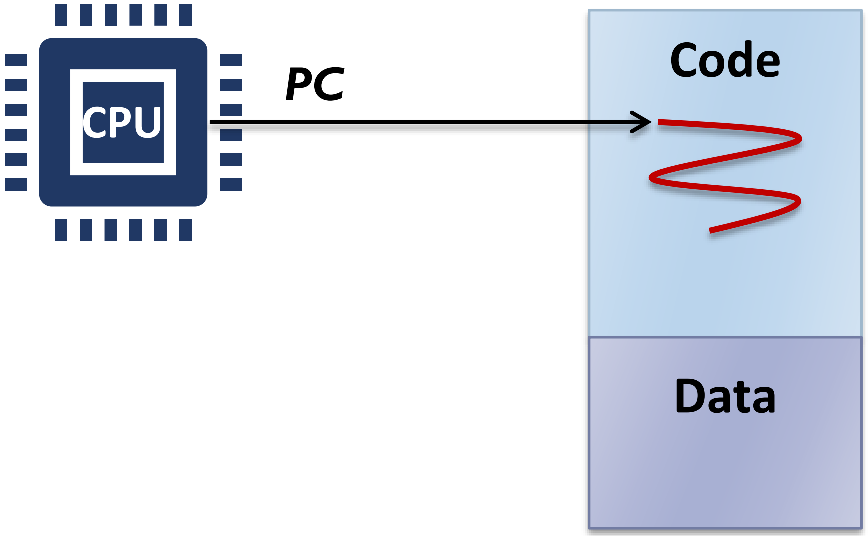
- A(An) _________ of a program in execution
- Java analogy:
  - Class → "program" (static)
  - Object → "process" (dynamic)
- The basic unit of protection
- A process is identified using its process ID (PID)
- A process includes
  - CPU context (registers)
  - OS resources (address space, open files, etc.)
  - Other information (PID, state, owner, etc.)

From Program to Process

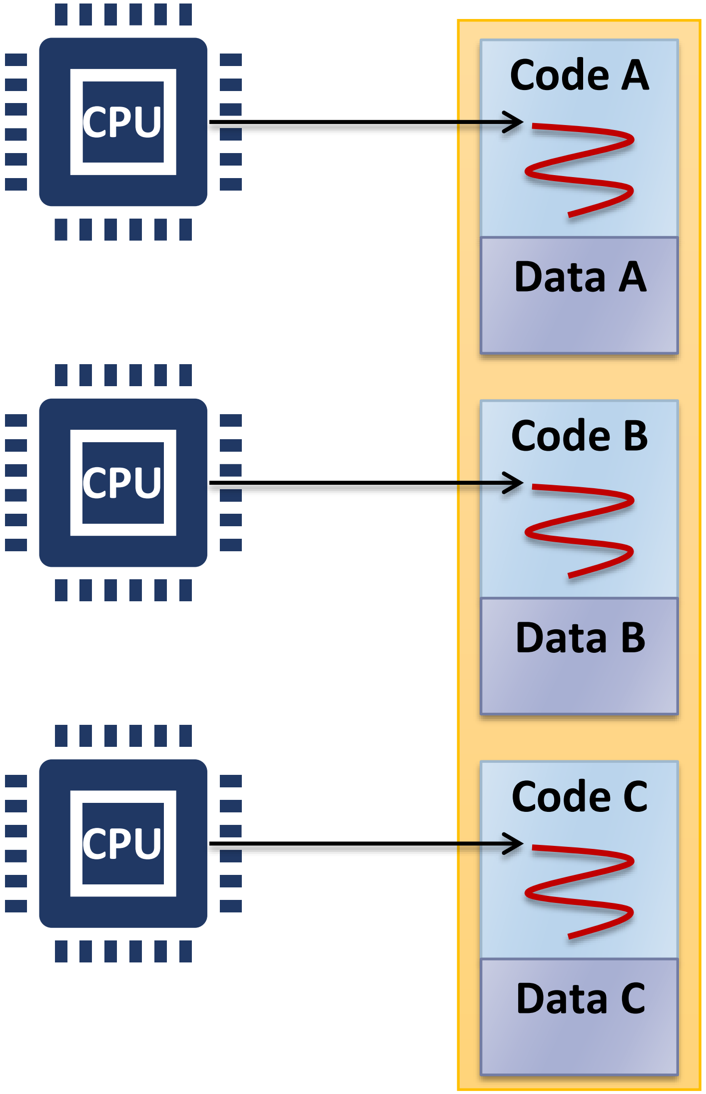


Running a Process

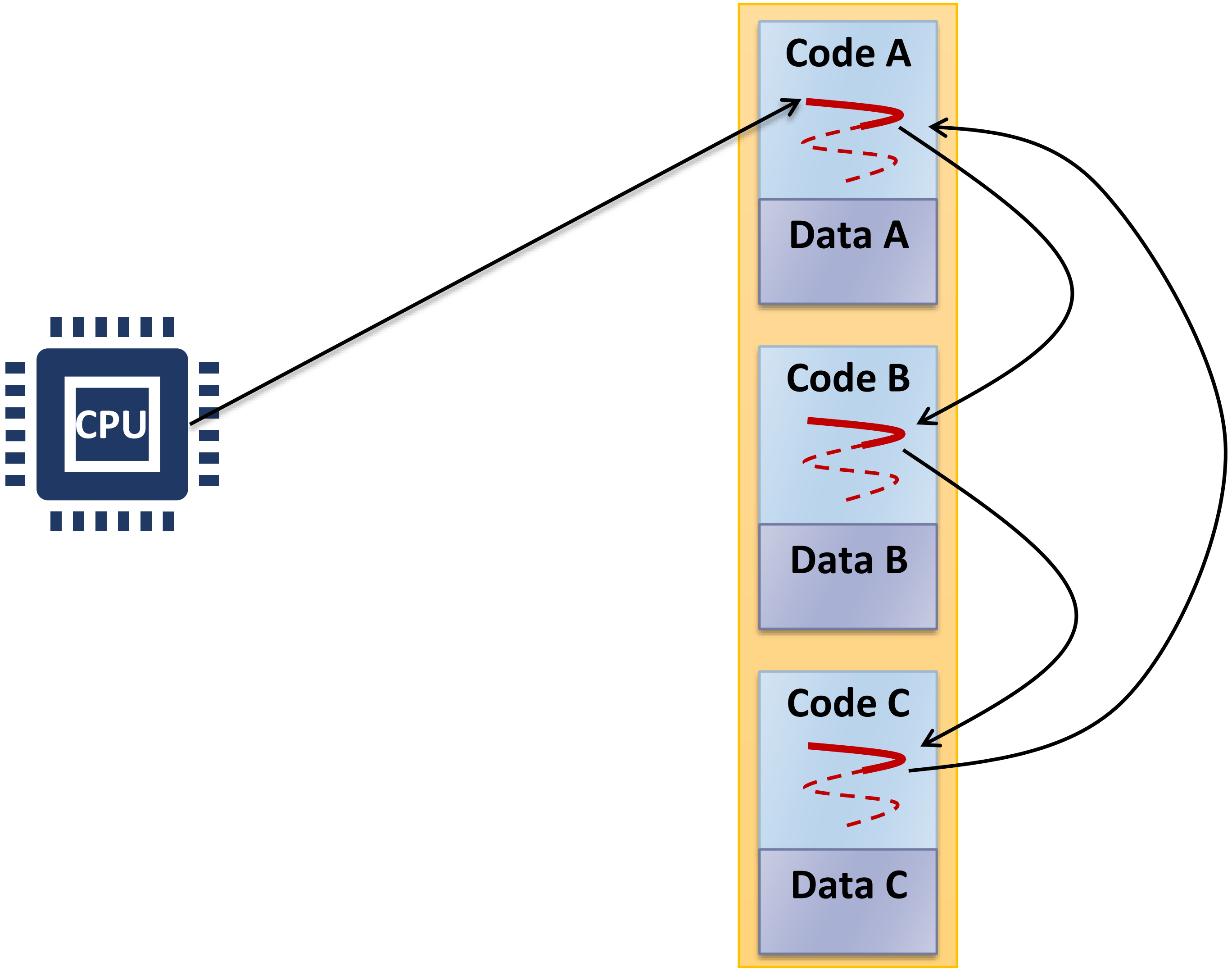
Fetch ***I*** ← Mem[***PC***]
Decode ***I***
Execute ***I***
Update ***PC***



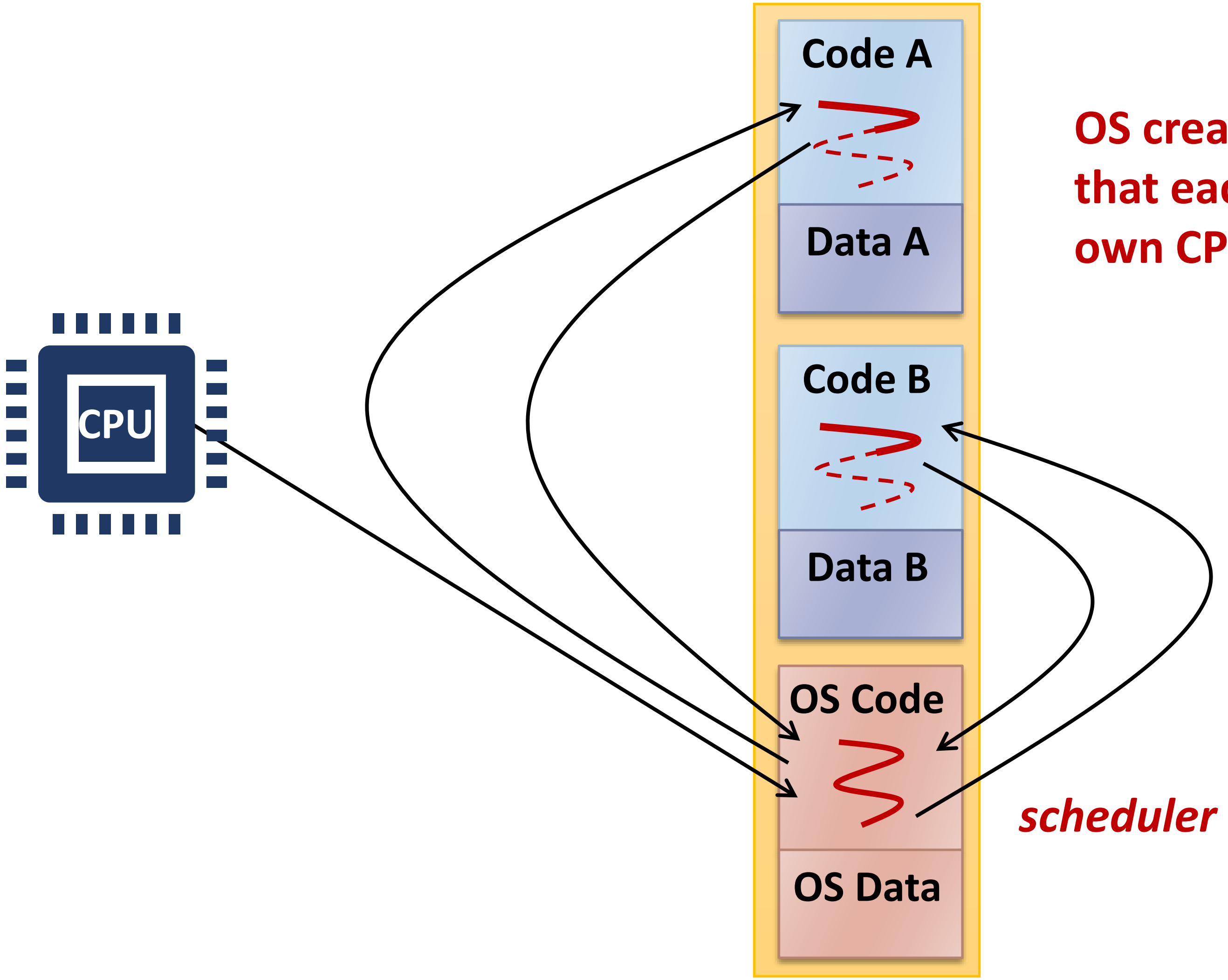
Running Multiple Processes



Interleaving Multiple Processes



Virtualizing the CPU



OS creates an illusion that each process has its own CPU (and memory)

Example: Creating a Process

```
#include <sys/types.h>
#include <unistd.h>

int main() {
    int pid;

    if ((pid = fork()) == 0)
        printf (“Child of %d is %d\n”, getppid(), getpid());  /* child */
    else
        printf (“I am %d. My child is %d\n”, getpid(), pid);  /* parent */
}
```

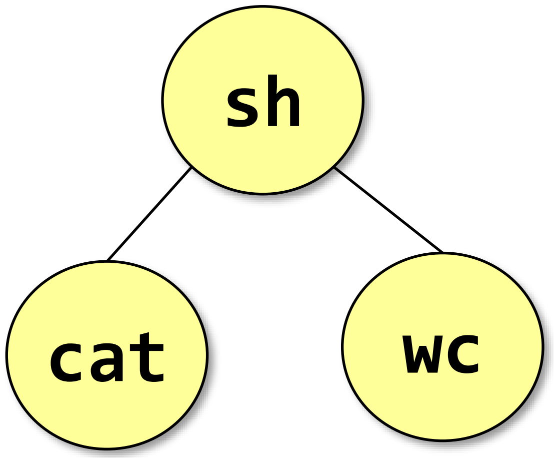
```
$ ./a.out
I am 31098. My child is 31099.
Child of 31098 is 31099.
$ ./a.out
Child of 31100 is 31101.
I am 31100. My child is 31101.
```

Process Hierarchy

- Parent-child relationship
  - One process can create another process
  - Unix calls the hierarchy a "process group"
  - Windows has no concept of process hierarchy

- Browsing a list of processes:
  - ps in Unix
  - Task Manager (taskmgr) in Windows

```
$ cat file1 | wc
```



Process Creation

- `fork()`
  - Creates a new process cloning the parent process
    - Parent inherits most of resources and privileges: open files, UID, etc.
    - Child also duplicates the parent's address space
  - Parent may either wait for the child to finish (using `wait()`), or it may continue in parallel
  - Shells or GUIs use this system call internally
  - Called once, returned twice
- `exec()`
  - Replaces the current process image with a new program
  - Windows: `CreateProcess()` = `fork()` + `exec()`
  - Called once, never returns

Process Termination

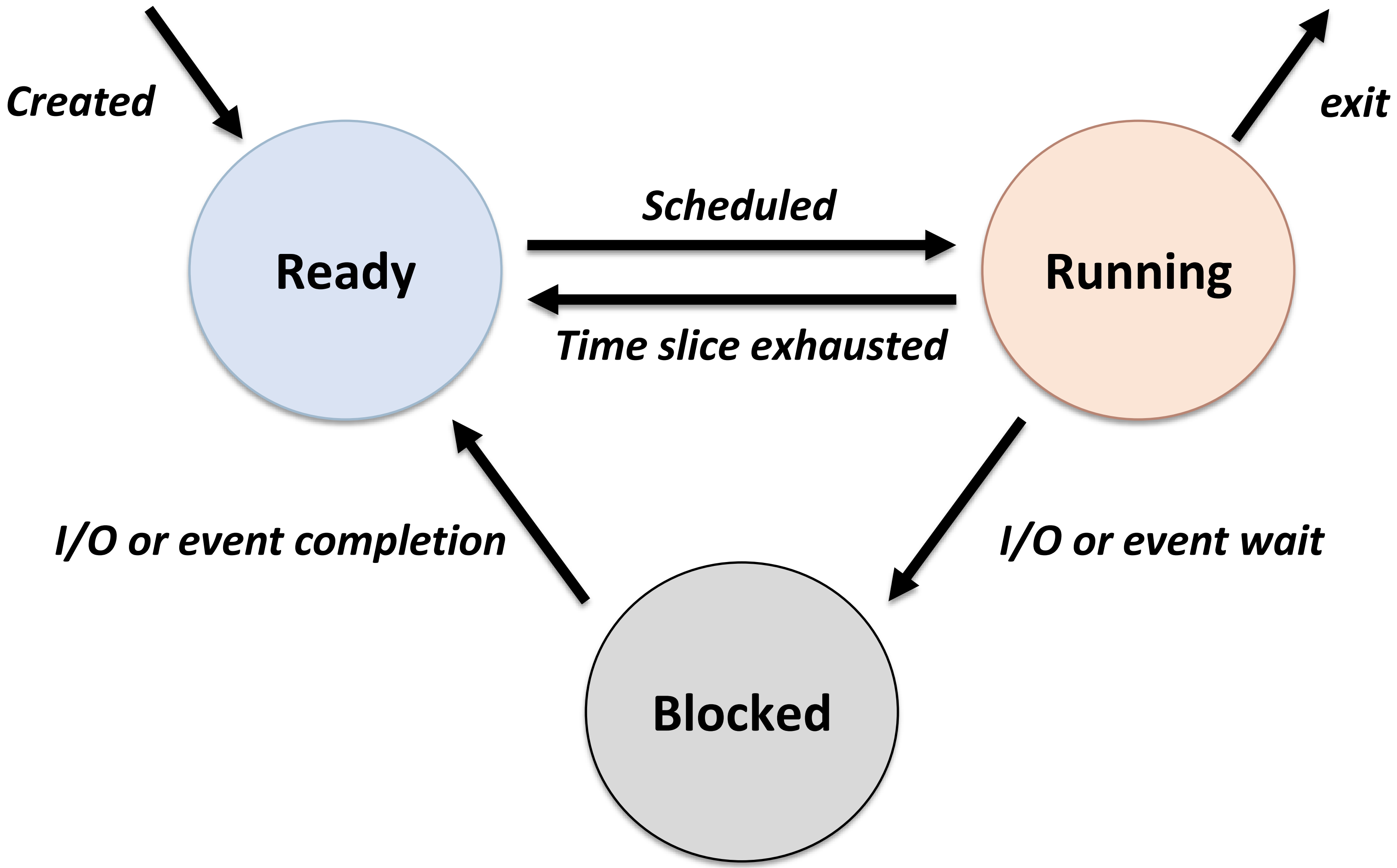
- Normal exit (voluntary)
- Error exit (voluntary)
- Fatal error (involuntary)
  - Segmentation fault – illegal memory access
  - Protection fault
  - Exceed allocated resources, etc.
- Killed by another process (involuntary)
  - By receiving a signal

- ________ process: terminated, but not removed

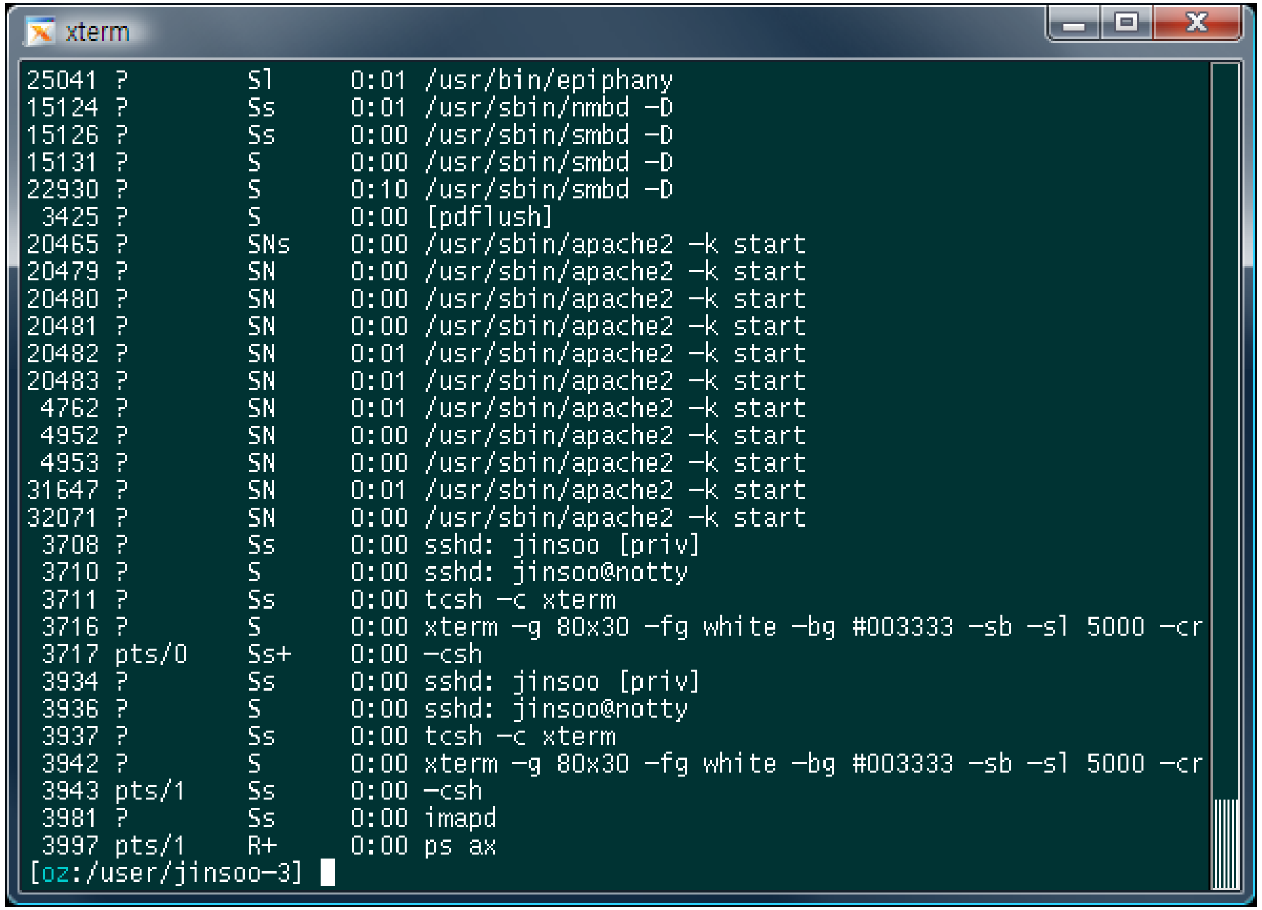
Simplified Shell

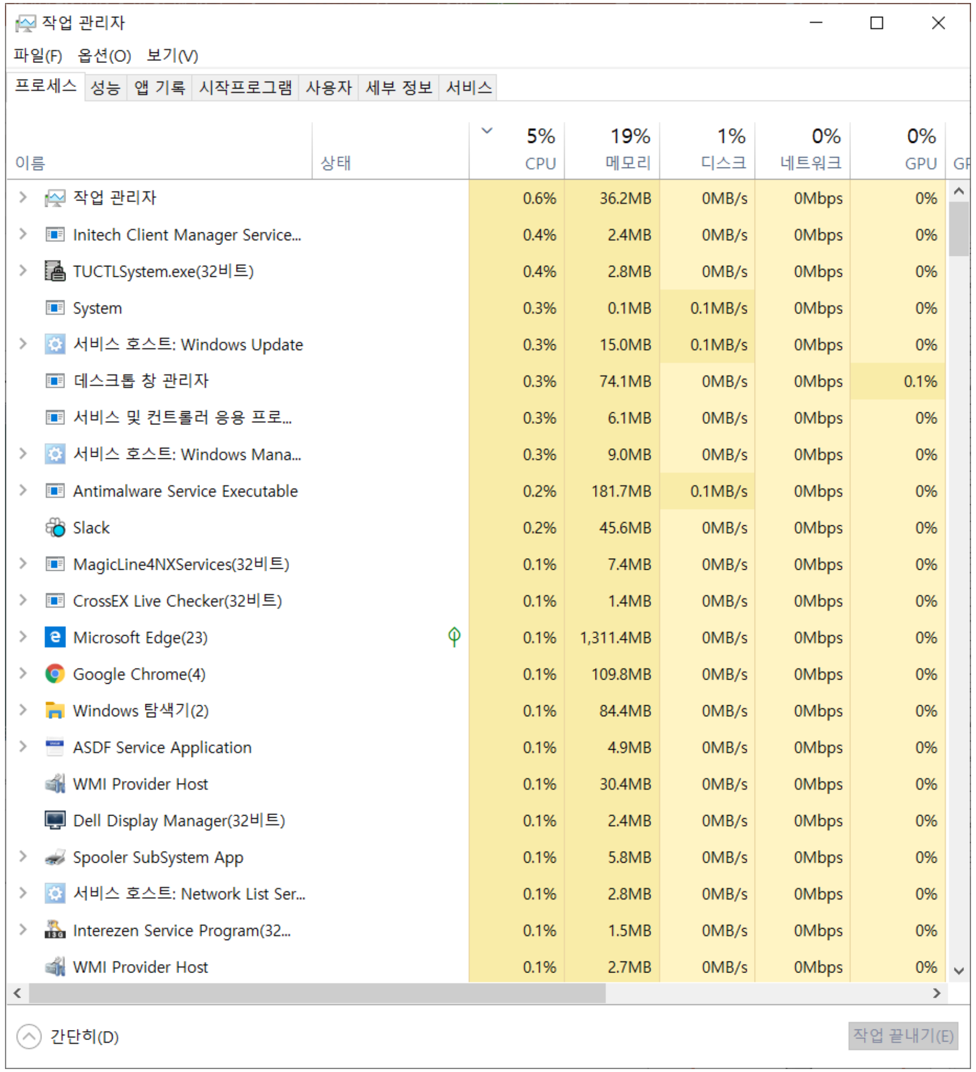
```
int main(void)
{
    char cmdline[MAXLINE];
    char *argv[MAXARGS];
    pid_t pid;
    int status;

    while (getcmd(cmdline, MAXLINE) >= 0) {
        parsecmd(cmdline, argv);
        if (!builtin_command(argv)) {
            if ((pid = fork()) == 0) {
                if (execv(argv[0], argv) < 0) {
                    printf("%s: command not found\n", argv[0]);
                    exit(0);
                }
            }
            waitpid(pid, &status, 0);
        }
    }
}
```

Process State Transitions



Processes


xterm
25041 ? Sl 0:01 /usr/bin/epiphany
15124 ? Ss 0:01 /usr/sbin/nmbd -D
15126 ? Ss 0:00 /usr/sbin/smbd -D
15131 ? S 0:00 /usr/sbin/smbd -D
22930 ? S 0:10 /usr/sbin/smbd -D
3425 ? S 0:00 [pdflush]
20465 ? SNs 0:00 /usr/sbin/apache2 -k start
20479 ? SN 0:00 /usr/sbin/apache2 -k start
20480 ? SN 0:00 /usr/sbin/apache2 -k start
20481 ? SN 0:00 /usr/sbin/apache2 -k start
20482 ? SN 0:01 /usr/sbin/apache2 -k start
20483 ? SN 0:01 /usr/sbin/apache2 -k start
4762 ? SN 0:01 /usr/sbin/apache2 -k start
4952 ? SN 0:00 /usr/sbin/apache2 -k start
4953 ? SN 0:00 /usr/sbin/apache2 -k start
31647 ? SN 0:01 /usr/sbin/apache2 -k start
32071 ? SN 0:00 /usr/sbin/apache2 -k start
3708 ? Ss 0:00 sshd: jinsoo [priv]
3710 ? S 0:00 sshd: jinsoo@notty
3711 ? Ss 0:00 tcsh -c xterm
3716 ? S 0:00 xterm -g 80x30 -fg white -bg #003333 -sb -sl 5000 -cr
3717 pts/0 Ss+ 0:00 -csh
3934 ? Ss 0:00 sshd: jinsoo [priv]
3936 ? S 0:00 sshd: jinsoo@notty
3937 ? Ss 0:00 tcsh -c xterm
3942 ? S 0:00 xterm -g 80x30 -fg white -bg #003333 -sb -sl 5000 -cr
3943 pts/1 Ss 0:00 -csh
3981 ? Ss 0:00 imapd
3997 pts/1 R+ 0:00 ps ax
[oz:/user/jinsoo-3]


작업 관리자
파일(F) 옵션(O) 보기(V)
프로세스 성능 앱 기록 시작프로그램 사용자 세부 정보 서비스
이름 상태 5% CPU 19% 메모리 1% 디스크 0% 네트워크 0% GPU
작업 관리자 0.6% 36.2MB 0MB/s 0Mbps 0%
Intech Client Manager Service... 0.4% 2.4MB 0MB/s 0Mbps 0%
TUCTLSystem.exe(32비트) 0.4% 2.8MB 0MB/s 0Mbps 0%
System 0.3% 0.1MB 0.1MB/s 0Mbps 0%
서비스 호스트: Windows Update 0.3% 15.0MB 0.1MB/s 0Mbps 0%
데스크톱 창 관리자 0.3% 74.1MB 0MB/s 0Mbps 0.1%
서비스 및 컨트롤러 응용 프로... 0.3% 6.1MB 0MB/s 0Mbps 0%
서비스 호스트: Windows Mana... 0.3% 9.0MB 0MB/s 0Mbps 0%
Antimalware Service Executable 0.2% 181.7MB 0.1MB/s 0Mbps 0%
Slack 0.2% 45.6MB 0MB/s 0Mbps 0%
MagicLine4NXServices(32비트) 0.1% 7.4MB 0MB/s 0Mbps 0%
CrossEX Live Checker(32비트) 0.1% 1.4MB 0MB/s 0Mbps 0%
Microsoft Edge(23) 0.1% 1,311.4MB 0MB/s 0Mbps 0%
Google Chrome(4) 0.1% 109.8MB 0MB/s 0Mbps 0%
Windows 탐색기(2) 0.1% 84.4MB 0MB/s 0Mbps 0%
ASDF Service Application 0.1% 4.9MB 0MB/s 0Mbps 0%
WMI Provider Host 0.1% 30.4MB 0MB/s 0Mbps 0%
Dell Display Manager(32비트) 0.1% 2.4MB 0MB/s 0Mbps 0%
Spooler SubSystem App 0.1% 5.8MB 0MB/s 0Mbps 0%
서비스 호스트: Network List Ser... 0.1% 2.8MB 0MB/s 0Mbps 0%
Interezen Service Program(32... 0.1% 1.5MB 0MB/s 0Mbps 0%
WMI Provider Host 0.1% 2.7MB 0MB/s 0Mbps 0%
간단히(D)
작업 끝내기(E)

Implementing Processes

- PCB (Process Control Block) or Process Descriptor
  - Each PCB represents a process
  - Contains all the information about a process
    - CPU registers
    - PID, PPID, process group, priority, process state, signals
    - CPU scheduling information
    - Memory management information
    - Accounting information
    - File management information
    - I/O status information
    - Credentials
  - `struct task_struct` in Linux: 6592 bytes as of Linux 6.2.0
  - `struct proc` in xv6: 360 bytes

Context Switch

- The act of switching CPU from one process to another
- Administrative overhead
  - Saving and restoring registers and memory maps
  - Flushing and reloading the memory cache
  - Updating various tables and lists, etc.
- The overhead depends on hardware support
  - Multiple register sets in UltraSPARC
  - Advanced memory management techniques may require extra data to be switched with each context (e.g., page tables, TLB, etc.)
- 100s or 1000s of switches/sec typically

Example: Context Switches in Linux

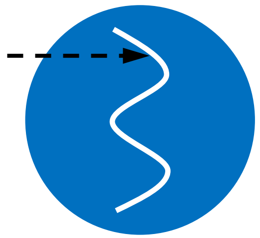
- Total uptime: 10,738,129.85 sec (124 days) `/proc/uptime`
- Total 6,770,575,007 context switches `/proc/stat`
- Average 630.5 context switches / sec (for all 4 cores)
- Roughly 158 context switches / sec / core

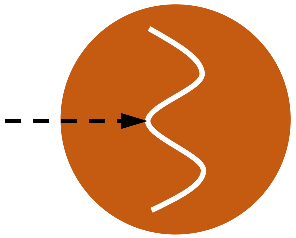
sys

```
[sys:~-2016] cat /proc/uptime
10738129.85 42606450.99
[sys:~-2017] grep ctxt /proc/stat
ctxt 6770575007
[sys:~-2018] echo "6770575007 / 10738129.85" | bc -l
630.51714791845248546701
[sys:~-2019] echo "630.51714791845248546701 / 4" | bc -l
157.62928697961312136675
[sys:~-2020]
```

Performing Context Switch in xv6

Process	RISC-V	Kernel
❶ Timer interrupt	❷ Set `sepc` ← `pc, scause` Disable interrupt Change to kernel mode Jump to trap handler @ `stvec`	
		❸ Save user regs to trapframe(A) Change to kernel page table Make A's state RUNNABLE Save A's context to PCB(A) Run scheduler() Make B's state RUNNING Restore B's context from PCB(B)
		❹ Change to user page table Restore user regs from trapframe(B) return-from-trap
	❺ Move back to user mode Enable interrupt Set pc ← sepc	


Process A


Process B

Process State Queues

- The OS maintains a collection of queues that represent the state of all processes in the system
  - Ready queue (or run queue)
  - Wait queue(s): one queue for each type of event (device, timer, message, …)

- Each PCB is queued onto a state queue according to its current state
  - As a process changes state, its PCB is migrated between the various queues

Implementing fork()

```
int fork()
```

- Creates and initializes a new PCB
- Creates and initializes a new address space
- Initializes the address space with a copy of the entire contents of the address space of the parent
- Initializes the kernel resources to point to the resources used by the parent (e.g., open files)
- Places the PCB on the ready queue
- Returns the child's PID to the parent, and zero to the child

Implementing `exec()`

```
int execv(char *prog, char *argv[])
```

- Stops the current process
- Loads the program "`prog`" into the process's address space
- Initializes hardware context and "`args`" for the new program
- Places the PCB on the ready queue

- `exec()` does not create a new process
- What does it mean for `exec()` to return?

Policy vs. Mechanism

- Policy
  - *What* should be done?
  - Policy decisions must be made for all resource allocation and scheduling problems
  - e.g., What is the next process to run?

- Mechanism
  - *How* to do something?
  - The tool for implementing a set of policies
  - e.g., How to make multiple processes run at once?

Separating Policy from Mechanism

- A key principle in operating system design
- Policies are likely to change depending on workloads and also across places or over time
- A general mechanism, separated from policy, is more desirable
- Allows to build a more modular OS
- Enables extensible systems – User-specific policies?