Systems Programming

Prof. Shih-Wei Li

Department of Computer Science and Information Engineering
National Taiwan University

Unix Philosophy

- Introduced by Ken Thompson as a set of minimalist and modular software development approaches
- Emphasizes on building simple, modular, and extensible code that can be maintained and repurposed easily by developers other than its creators!
- Summarized Unix philosophy in the book: “A Quarter-Century of Unix” (in 1994):
  - Write programs that do one thing and do it well
  - Write programs that work together
  - Write programs that handle text streams, because that is the universal interface

Unix Philosophy: Case Study

- Consider two Unix programs (i.e., applications):
  - *head*: prints out the first lines of the input; "*head -5 $FILE*" prints out the first 5 lines of the *$FILE*
  - *tail*: prints out the last lines of the input; "*tail -5 $FILE*" prints out the last 5 lines of the *$FILE*
- Q: how can you use the two programs to print the 10th line of a file?

Why Unix?

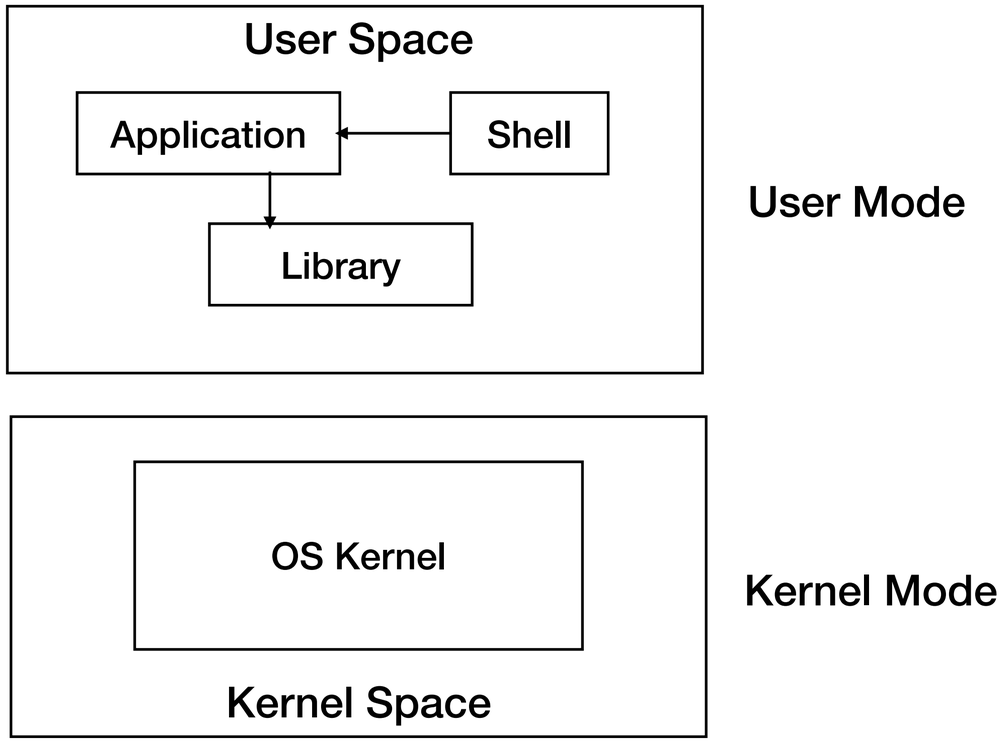
- Key to the success of the Unix Operating System: simplicity of the design and distribution of the source code
- Unix is simple (from “Linux Kernel Development 3rd Edition):
  - Unix implements only hundreds of system calls (whereas others could have thousands)
  - Modern systems like Linux use the Unix-based design, so learning Unix is crucial!

User and Kernel Split

- In a computer system like Unix-based systems, applications and the OS kernel run in different hardware environments
  - The environment defines the access to hardware resources
  - Why do we need such a split?
- For now, you only need to know that applications run in the *user space/mode* and the OS kernel runs in the *kernel space/mode*
  - You'll learn about what they are exactly in the Operating Systems and Computer Organization/Architecture course
  - We'll not explain them in details in today's course, but if you are interested, feel free to talk to me after the course

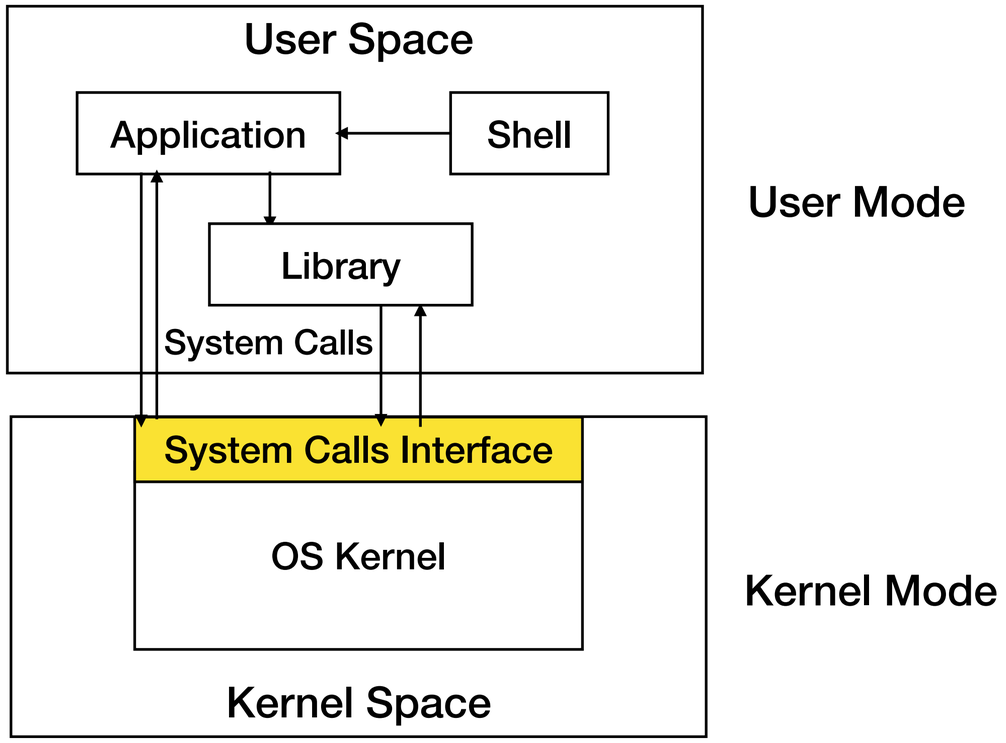
User and Kernel Split

- Applications run in the user space/mode
- The OS kernel runs in the kernel space/ mode
- Goal of the split: to isolate access to hardware resources
- Question: applications and the OS kernel are isolated, how do they interact?



User and Kernel Split

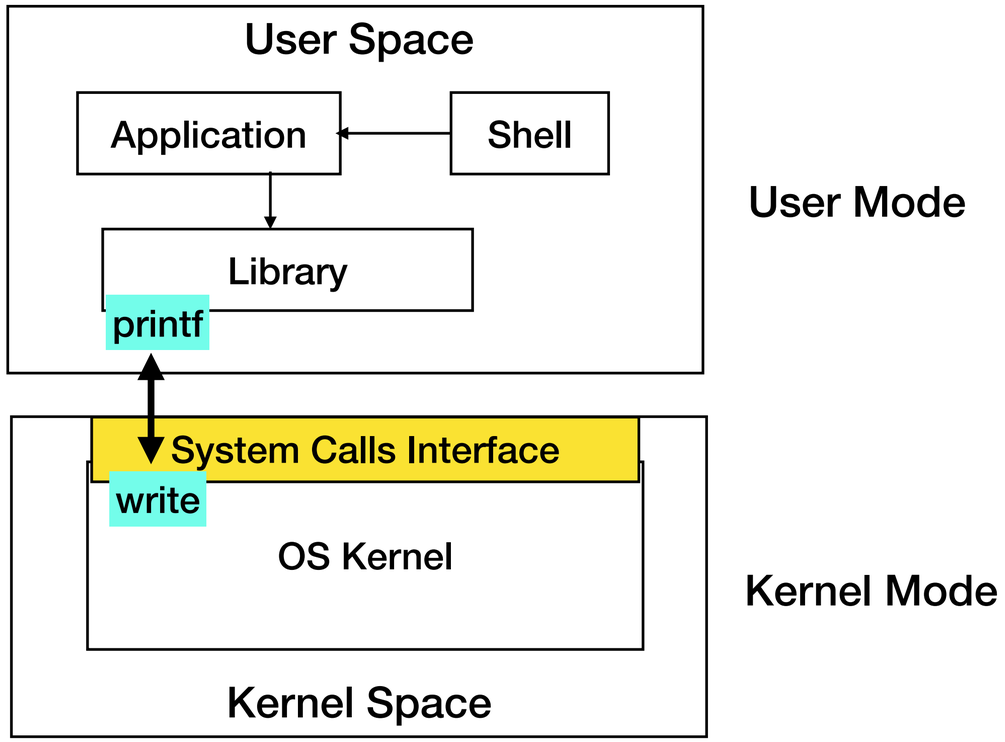
- Applications make *system calls* to use services/features provided by the OS kernel
  - The kernel defines the system calls interface, providing a set of system calls
  - System calls can be made by the applications directly or the library
  - Making a system call cause a *trap* from the user mode/space to the kernel mode/space
- The OS kernel handles the system call, and return to the caller in user space



System Calls v.s. Function Calls

	System Calls	Function Call
Decription	Application calls to the functions provided by the OS kernel	A program calls to predefined functions (ex: defined in a library)
Behavior	Cause switches from the user space/mode to kernel space/mode	Cause no space/mode switches
Performance	Slow	Fast

User and Kernel Split - Case Study

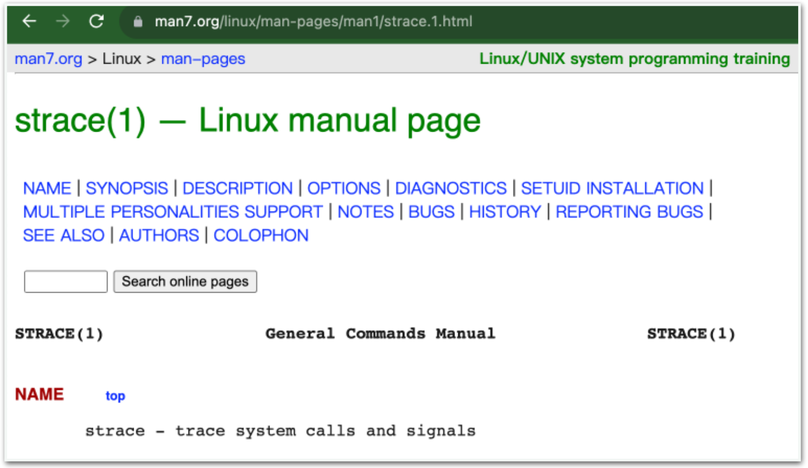


User and Kernel Split - Case Study

```
shihwei@linux1 [~/sp21] cat sp01-demo1.c
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <string.h>

int main()
{
        char *hello = "hello world\n";

        write(1, hello, strlen(hello));
        syscall(SYS_write, 1, hello, strlen(hello));
        printf("%s", hello);
}
```




```
shihwei@linux1 [~/sp21] strace ./syscall
execve("./syscall", ["./syscall"], 0x7fffa6ee2fe0 /* 42 vars */) = 0
brk(NULL)                               = 0x564b75bdc000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffec917d9a0) = -1 EINVAL (不適用的引數)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (沒有此一檔案或目錄)
openat(AT_FDCWD, "/opt/pin/intel64/runtime/glibc-hwcaps/x86-64-v3/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (沒有此一
newfstatat(AT_FDCWD, "/opt/pin/intel64/runtime/glibc-hwcaps/x86-64-v3/", 0x7ffec917cbd0, 0) = -1 ENOENT (沒有此一檔案或
openat(AT_FDCWD, "/opt/pin/intel64/runtime/glibc-hwcaps/x86-64-v2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (沒有此一
newfstatat(AT_FDCWD, "/opt/pin/intel64/runtime/glibc-hwcaps/x86-64-v2/", 0x7ffec917cbd0, 0) = -1 ENOENT (沒有此一檔案或
openat(AT_FDCWD, "/opt/pin/intel64/runtime/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (沒有此一檔案或目錄)
newfstatat(AT_FDCWD, "/opt/pin/intel64/runtime/", {st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=322011, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 322011, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f609cefc000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220~\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2366624, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f609cefa000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2411920, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f609cc00000
mmap(0x7f609cc26000, 1437696, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f609cc26000
mmap(0x7f609cd85000, 348160, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x185000) = 0x7f609cd85000
mmap(0x7f609cdda000, 417792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d9000) = 0x7f609cdda000
mmap(0x7f609ce40000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f609ce40000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f609cef8000
arch_prctl(ARCH_SET_FS, 0x7f609cefb640) = 0
set_tid_address(0x7f609cefb910)         = 257693
set_robust_list(0x7f609cefb920, 24)     = 0
rseq(0x7f609cefbf60, 0x20, 0, 0x53053053) = 0
mprotect(0x7f609cdda000, 409600, PROT_READ) = 0
mprotect(0x564b75b24000, 4096, PROT_READ) = 0
mprotect(0x7f609cf7c000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f609cefc000, 322011)          = 0
write(1, "hello world\n", 12hello world
)           = 12
write(1, "hello world\n", 12hello world
)           = 12
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x20), ...}, AT_EMPTY_PATH) = 0
getrandom("\x67\x93\xc0\xb2\xca\xd9\x04\xa7", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x564b75bdc000
brk(0x564b75bfd000)                     = 0x564b75bfd000
write(1, "hello world\n", 12hello world
)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
```

What you have learned so far

- What is an Operating System
- The Unix architecture
- Basic concepts of:
  - System call vs function call
  - The kernel and user split

What will be covered today?

- ***Services for application programs: (Chapter 1)***
- Unix standards (Chapter 2)
- File I/O (Chapter 3)

Services for Application Programs

- Chapter 1:
  - **User Identification**
  - Process Management
  - Memory Management
  - File/Directory Management

User Identification in Unix

- Logging in (check ch 1.3, 6.2, and 6.3):
  - The file /etc/passwd in your Unix System stores the system's user account information:
    - User ID, home directory, shell program, etc.
    - The user password is stored in /etc/shadow (you need root permission to access the file)
    - Good explanation of /etc/passwd: https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/
- Related shell command:
  - passwd: change user password

Output of shell command: cat /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
```

What does the "x" character specify here?

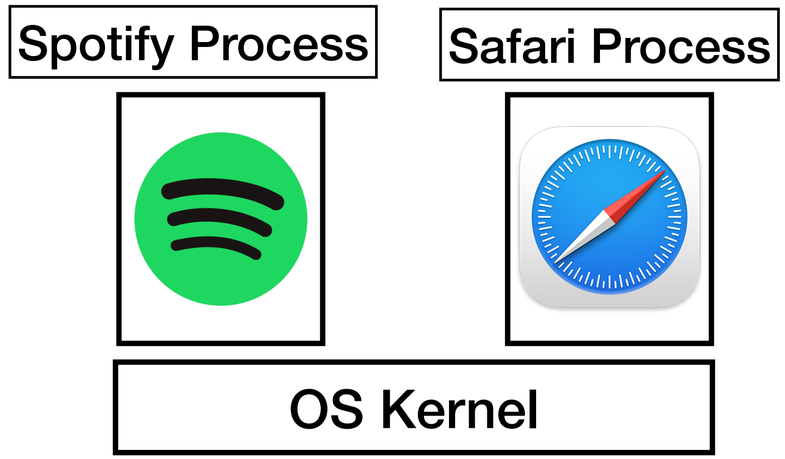
Services for Application Programs

- Chapter 1:
  - User Identification
  - **Process Management**
  - Memory Management
  - File/Directory Management

Programs and Processes

- Program:
  - An executable file stored on the disk
- Process:
  - An executing instance of a program, or a running program
  - The OS assigns a unique ID to each of the processes
    - The ID is called the process ID
- Related shell command:
  - ps: to report a snapshot of the current processes
  - top: display processes

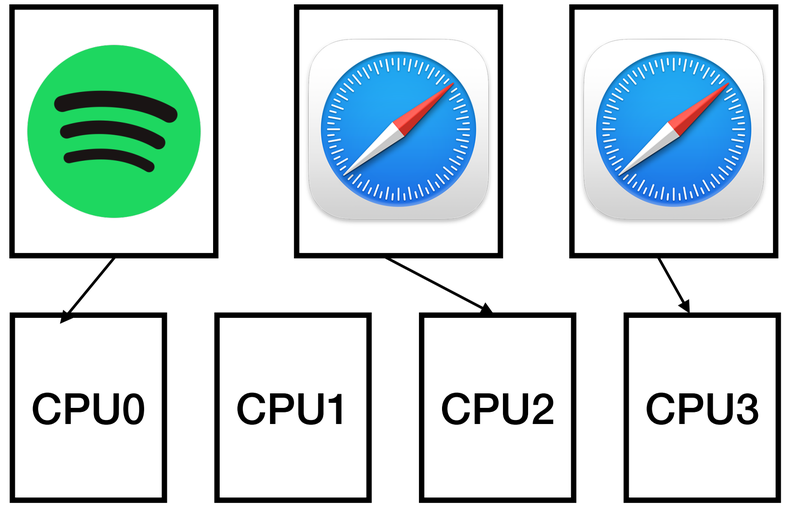
Disk



Process Scheduling

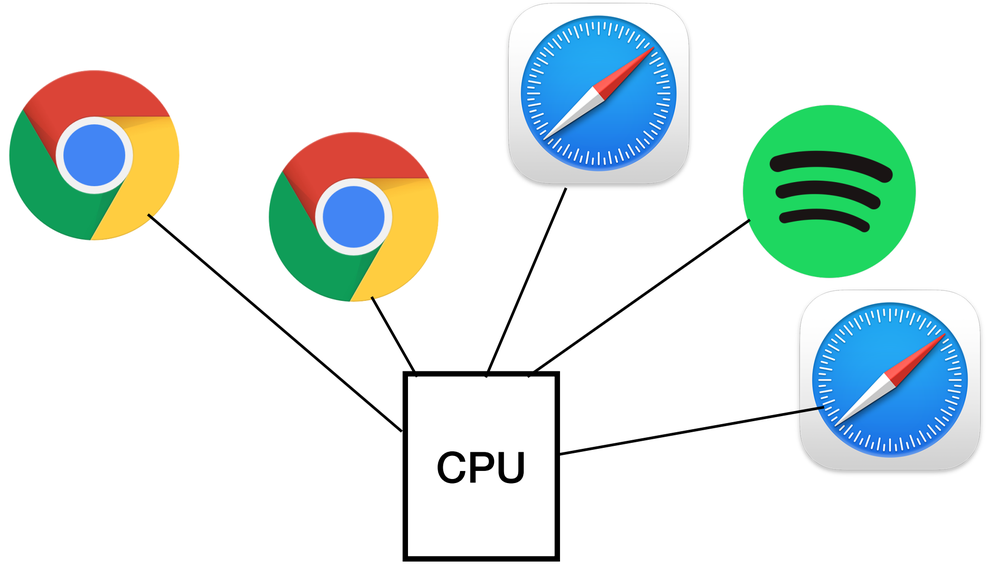
- Unix systems rely on the OS kernel to schedule processes to run on CPUs
  - A particular component called the “scheduler” in the kernel does the work
- Why process scheduling?

Who is next to run on the CPU?



Process Scheduling

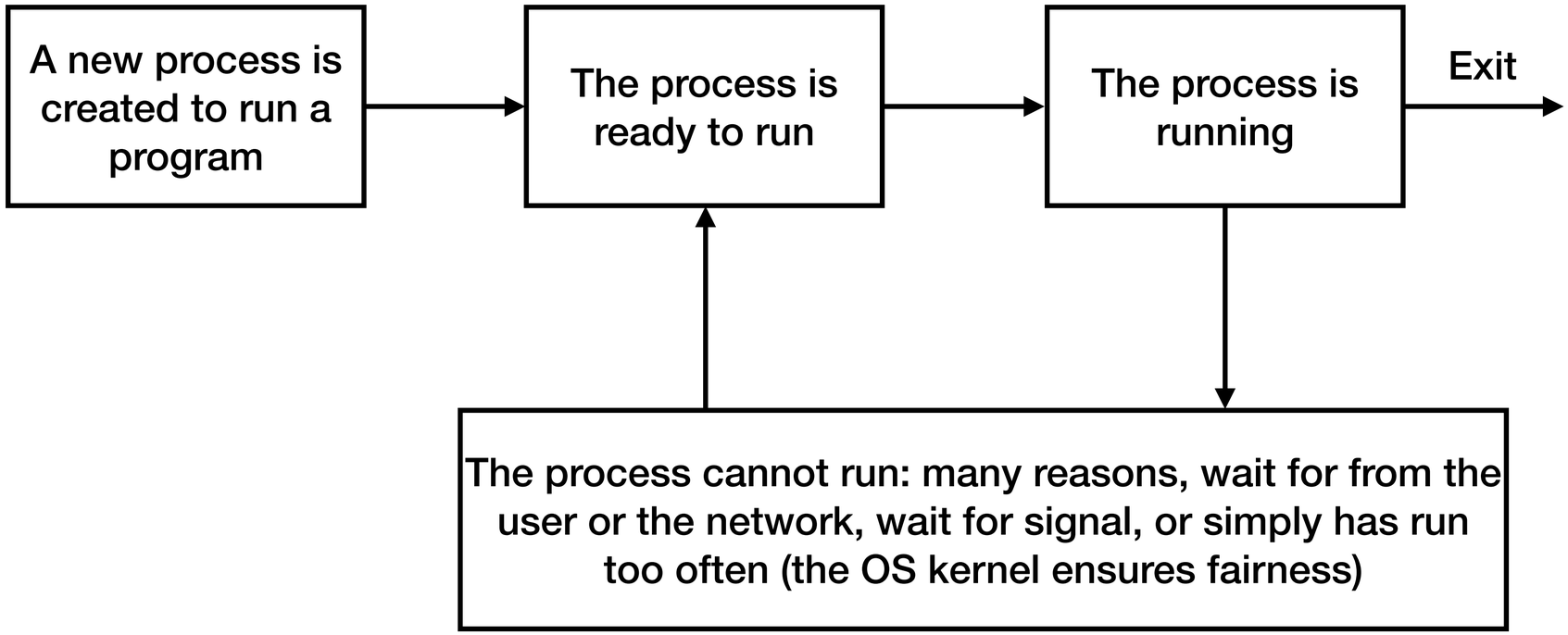
Time-sharing: multiplex processes to run on a given CPU; these processes "share" the CPU time



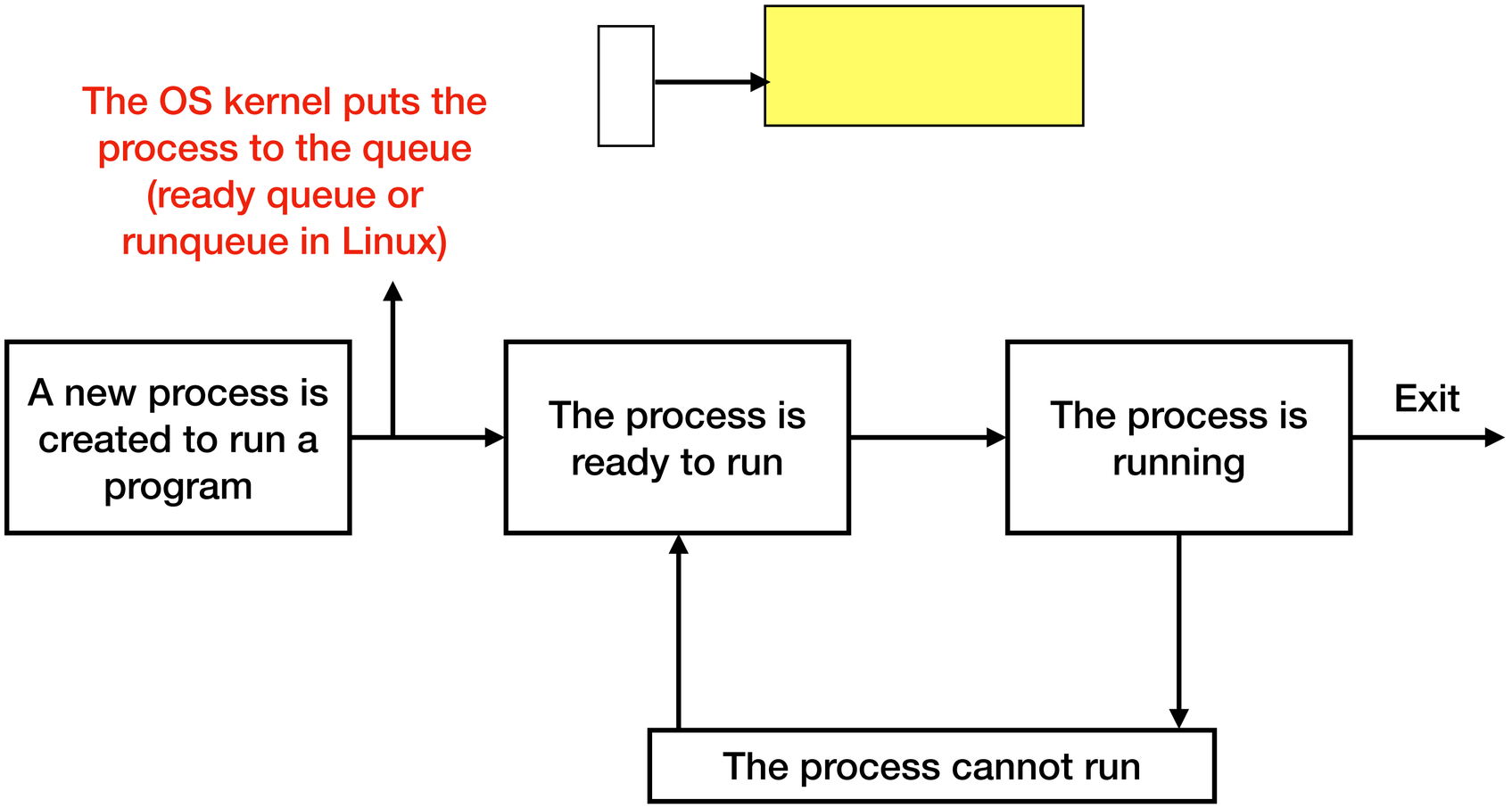
Process Scheduler:
(in the OS Kernel)

Process Scheduling

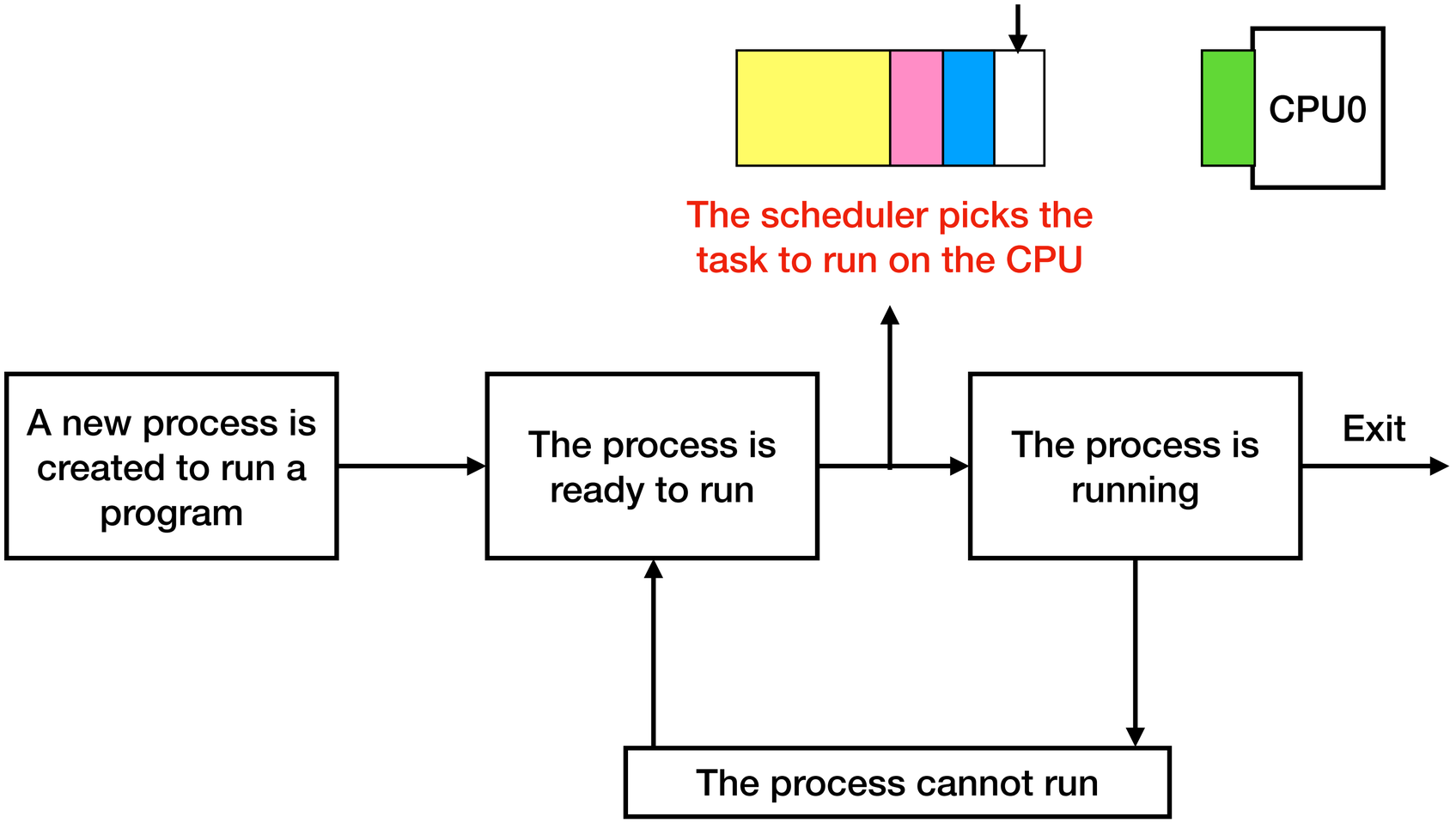
Process Life Cycle



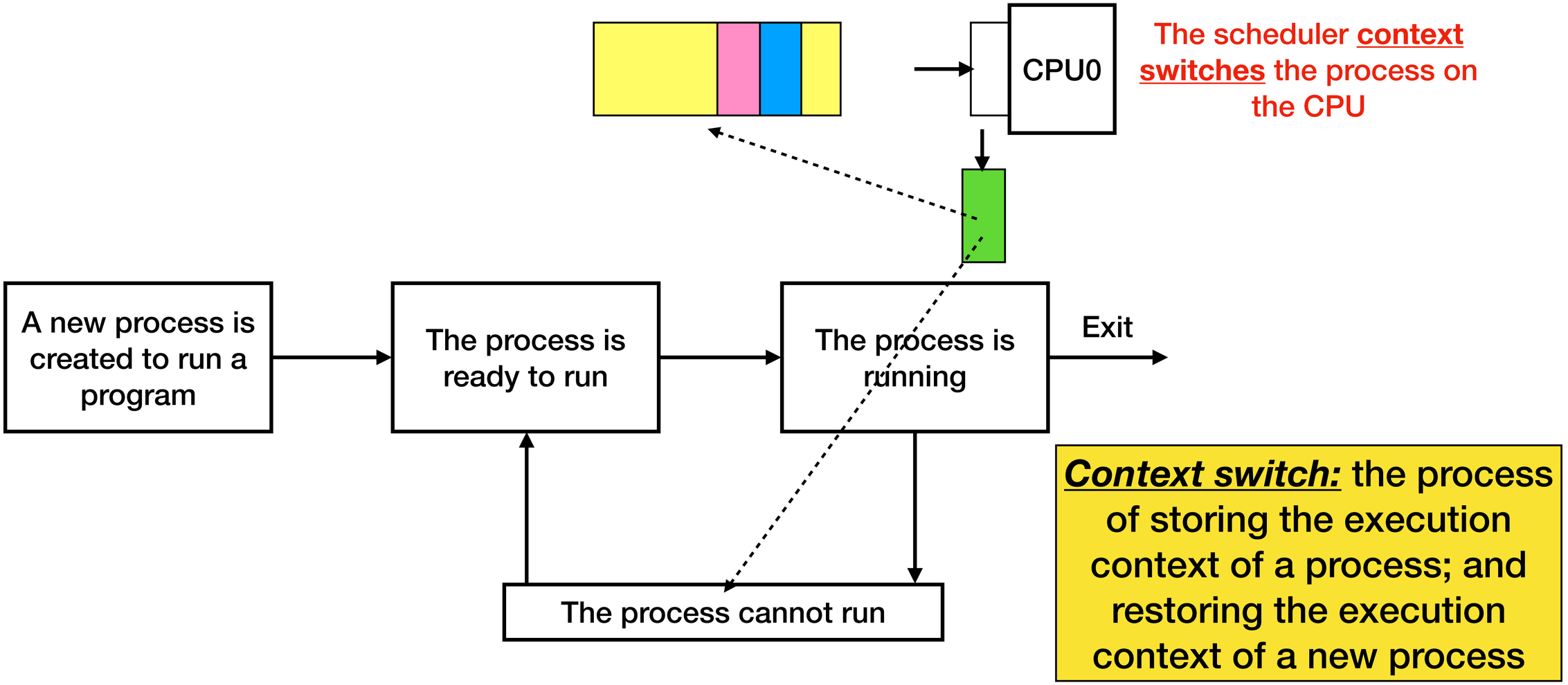
Process Scheduling



Process Scheduling

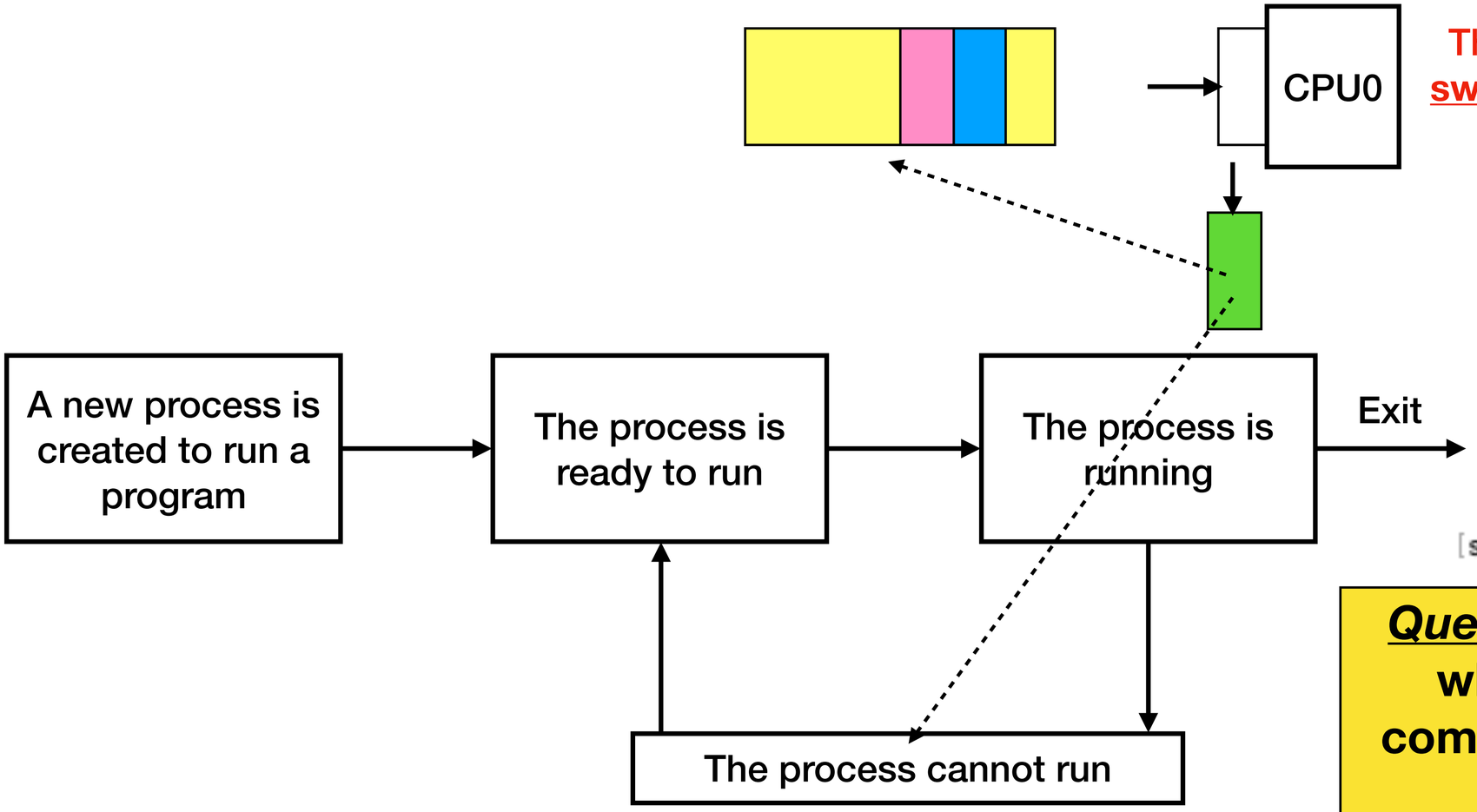


Process Scheduling



Context switch: the process of storing the execution context of a process; and restoring the execution context of a new process

Process Scheduling



The scheduler **context switches** the process on the CPU

```
[shihwei@linux1 [~] ls
```

Question: **what happens when you execute a command (like *ls*) in your shell?**

Process Input and Output

- Unix standardizes how a process inputs/outputs
  - Why? To interact with users or external events (e.g., network events)
  - Ex: the shell program takes user input from the keyboard (or a remote network connection) and outputs to your terminal
  - We'll talk more about it when we introduce file I/Os (Chap. 3)

Process Input and Output

- The input and output of Unix programs can be redirected:
  - Pipe (“|”): sends the output of one process to the input of another
    - *e.g., “head file 5 | tail 1”*
  - Filter: a program that takes the output from another process as input and transforms the input in some way
    - *e.g., “ps aux | grep hello”*

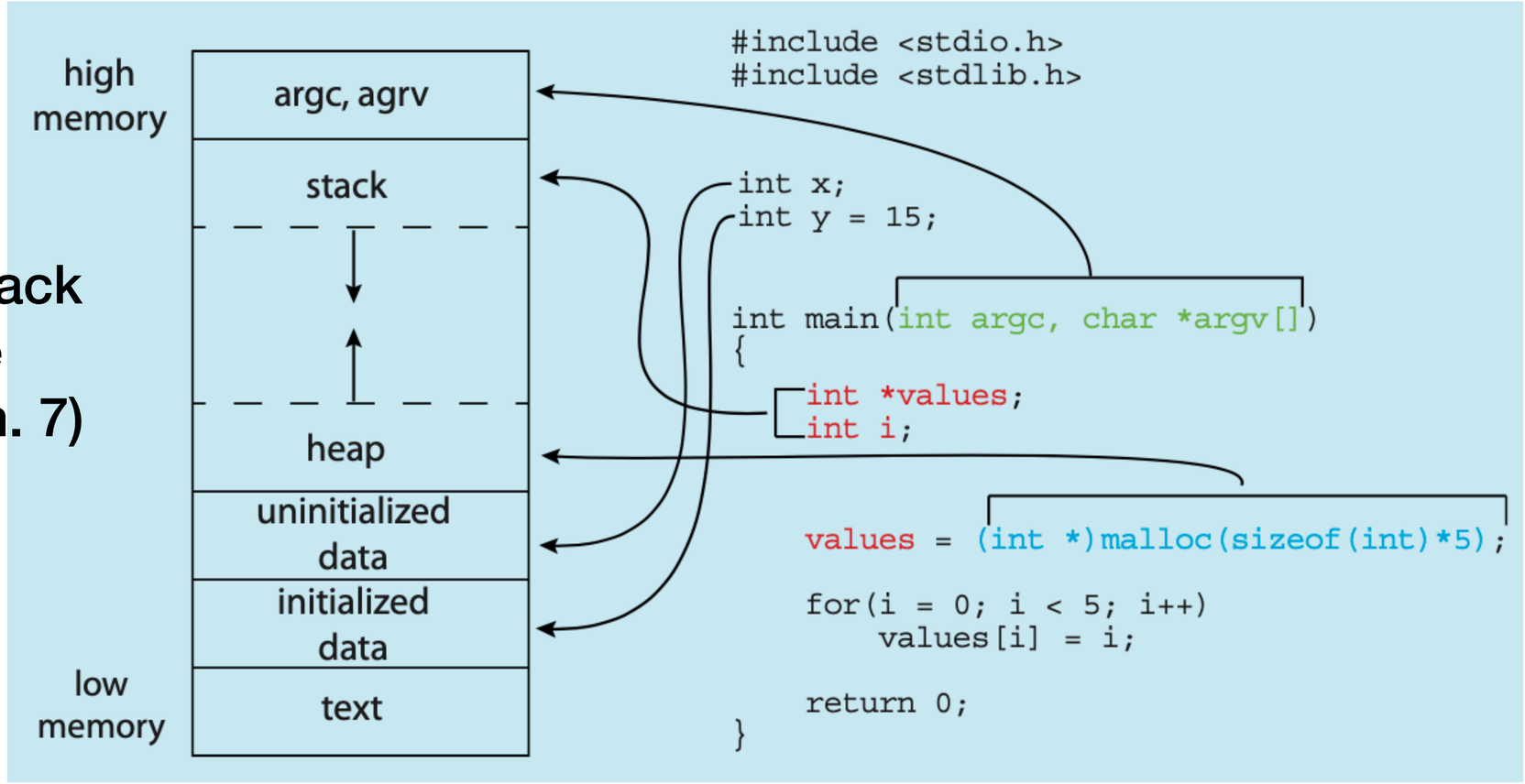
```
$> ls -la | more
$> cat file | wc
$> man ksh | grep “history”
$> ls -l | grep “bowman” | wc
$> who | sort > current_users
```

Services for Application Programs

- Chapter 1:
  - User Identification
  - Process Management
  - **Memory Management**
  - File/Directory Management

Process Memory Layout

We will come back to this in the course later (Ch. 7)



From: Operating Systems Concept 3rd Edition: Memory Layout of a C Program

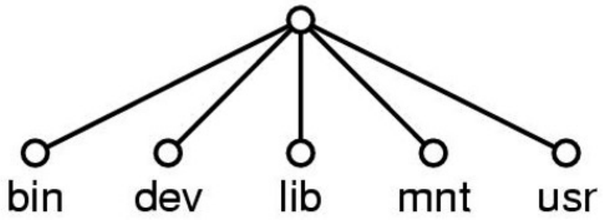
Services for Application Programs

- Chapter 1:
  - User Identification
  - Process Management
  - Memory Management
  - **File/Directory Management**

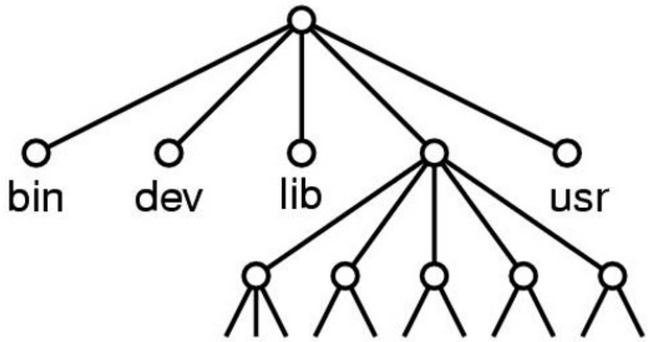
Unix File and Directory

- File:
  - A sequence of bytes; with names, so you know how to refer to files
- Directory:
  - A file that includes info on how to find other files
  - Directories form a hierarchy in Unix; contains files and other directories
  - Root directory: the top-most directory in a hierarchy (also called “/“)
  - Home directory: the directory you will get to when you log in
- Pathnames:
  - Absolute pathname: “/x/y/z”
  - Relative pathname: “x/y/z” — the path is relative to the current directory

Mount File Systems



(a)



(b)

- (a) file system before the mount in /mnt
- (b) file system after the mount in /mnt

Use command “mount” to show all mounted file systems!

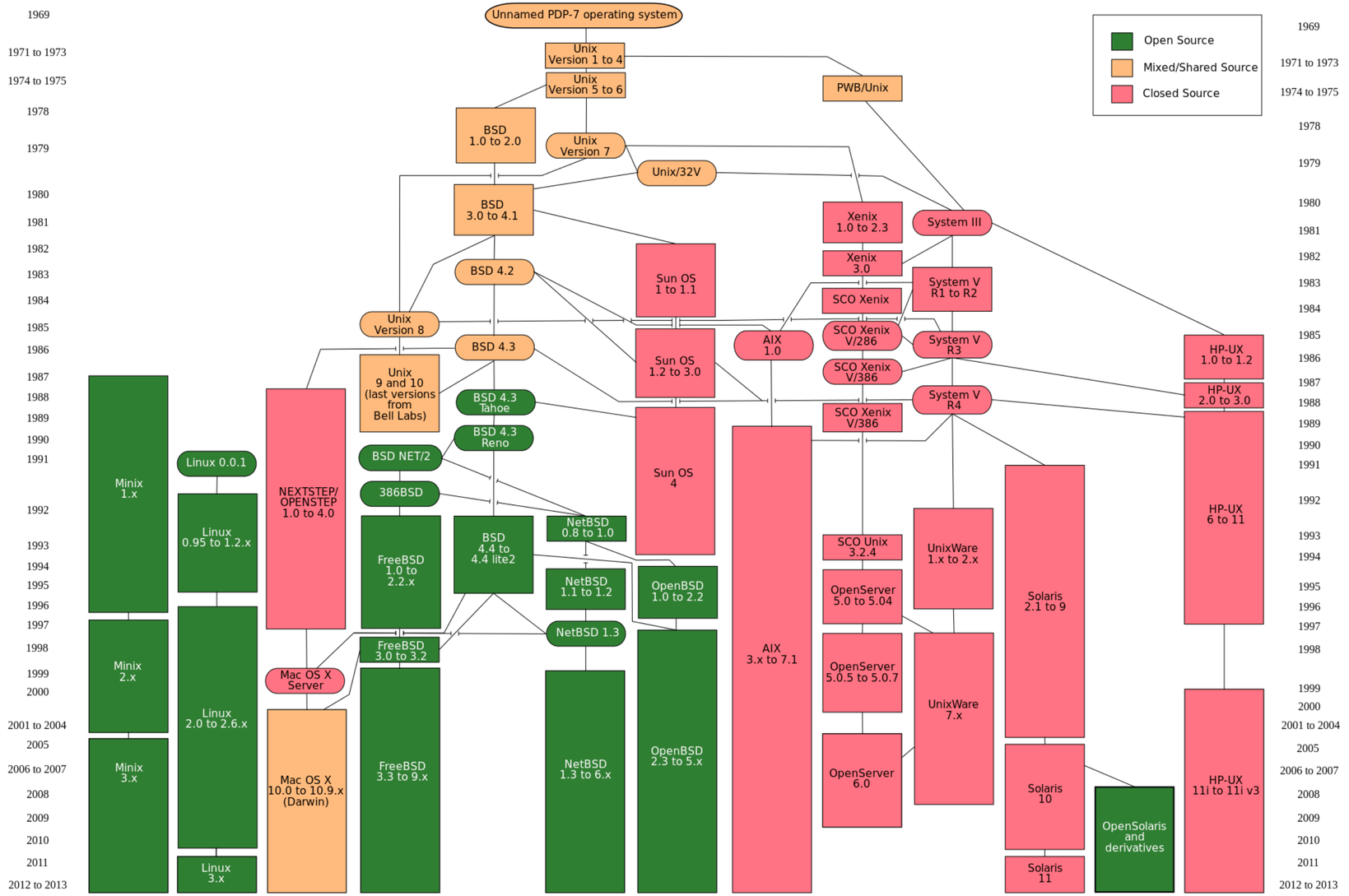
What will be covered today?

- Services for application programs: (Chapter 1)
- ***Unix standards (Chapter 2)***
- File I/O (Chapter 3)

Versions of Unix

- Unix was intended to be small, flexible, and portable
  - Ken Thompson built Unix when porting of the game “Spec Travel” to the PDP-7 machine
  - Unix was simpler Multics, a time-sharing operating system developed by Bell Labs — Ken Thompson used to work on Multics
- Most of the later Unix versions based on either System V and BSD

Versions of Unix



Unix譜系 - from wikipedia

Standardization

- Much work has gone into standardizing the Unix programming environment
  - Why standardization? To define the specification that implementations must meet; ex: the standardization for different Unix and compiler implementations
  - Reduce porting efforts: improves portability and reusability
- Standards for Unix programming environment:
  - IEEE POSIX
  - The Single Unix Specification (SUS)

Standardization - IEEE POSIX

- POSIX (Portable Operating System Interface) is a family of standards initially developed by the IEEE (Institute of Electrical and Electronics Engineers)
- POSIX initially consists of a single document that defines: the core programming interface (includes process creation/control, signals, file operations, etc.)
  - This version is IEEE Standard 1003.1, also called POSIX.1
- Later versions of POSIX grows into 19 separate documents, that include: shells and utilities (command interpreter, utility programs), and facilities for threading, networking, etc.
  - This version is IEEE Standard 1003.2, also called POSIX.2
- An OS that implements the POSIX interface is “POSIX compliant”
  - Programs and libraries that compile and run one POSIX compliant system should compile and run on other POSIX compliant systems

Standardization - IEEE POSIX

- The POSIX specification includes both the required and optional headers
  - Headers include: unistd.h, pwd.h, dirent.h, grp.h, fcntl.h, …

Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries (Section 4.22)
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control (Section 3.14)
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file (Section 6.4)
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function (Section 14.4.2)
<pthread.h>	•	•	•	•	threads (Chapters 11 and 12)
<pwd.h>	•	•	•	•	password file (Section 6.2)
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O (Chapter 18)
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions
<arpa/inet.h>	•	•	•	•	Internet definitions (Chapter 16)
<net/if.h>	•	•	•	•	socket local interfaces (Chapter 16)
<netinet/in.h>	•	•	•	•	Internet address family (Section 16.3)
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function (Section 14.4.1)
<sys/socket.h>	•	•	•	•	sockets interface (Chapter 16)
<sys/stat.h>	•	•	•	•	file status (Chapter 4)
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times (Section 8.17)
<sys/types.h>	•	•	•	•	primitive system data types (Section 2.8)
<sys/un.h>	•	•	•	•	UNIX domain socket definitions (Section 17.2)
<sys/utsname.h>	•	•	•	•	system name (Section 6.9)
<sys/wait.h>	•	•	•	•	process control (Section 8.6)

From: Figure 2.2 (Required headers defined by the POSIX standard)
- Advanced Programming in the UNIX Environment

Standardization - Single Unix Specification (SUS)

- The superset of the POSIX.1 standard, consists of:
  - Base specifications (equivalent to POSIX.1)
  - X/Open System Interfaces (XSI): defines the optional interfaces, including file synchronization, thread attributes
- The Open Group owns the SUS, which was released in 1994; the latest is version 4
- Formal definition of ***Unix systems:*** OS implementations that support the SUS
  - macOS (since 10.5) was certified as a Unix system
  - Most of the Linux distributions are not registered Unix systems (one notable exception is Huawei's Euler OS); most of the Linux distributions are POSIX-compliant

What will be covered today?

- Services for application programs: (Chapter 1)
- Unix standards (Chapter 2)
- ***File I/O (Chapter 3)***

Key Unix Design Principle

- In Unix, everything is a file: documents, executables, directories, keyboards, monitors, printers, network connections, etc.
- Why is this useful?
  - The same functions can be used to access different types of “files”
  - Simplify the manipulation of data and devices into a set of core system calls (open/read/write/lseek/close)

Unix functions for file management

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

We will come back to this in the course later (Ch. 3)