CS 423
Operating System Design:
The Programming Interface

Jongyul Kim

* Thanks for Prof. Adam Bates for the slides.

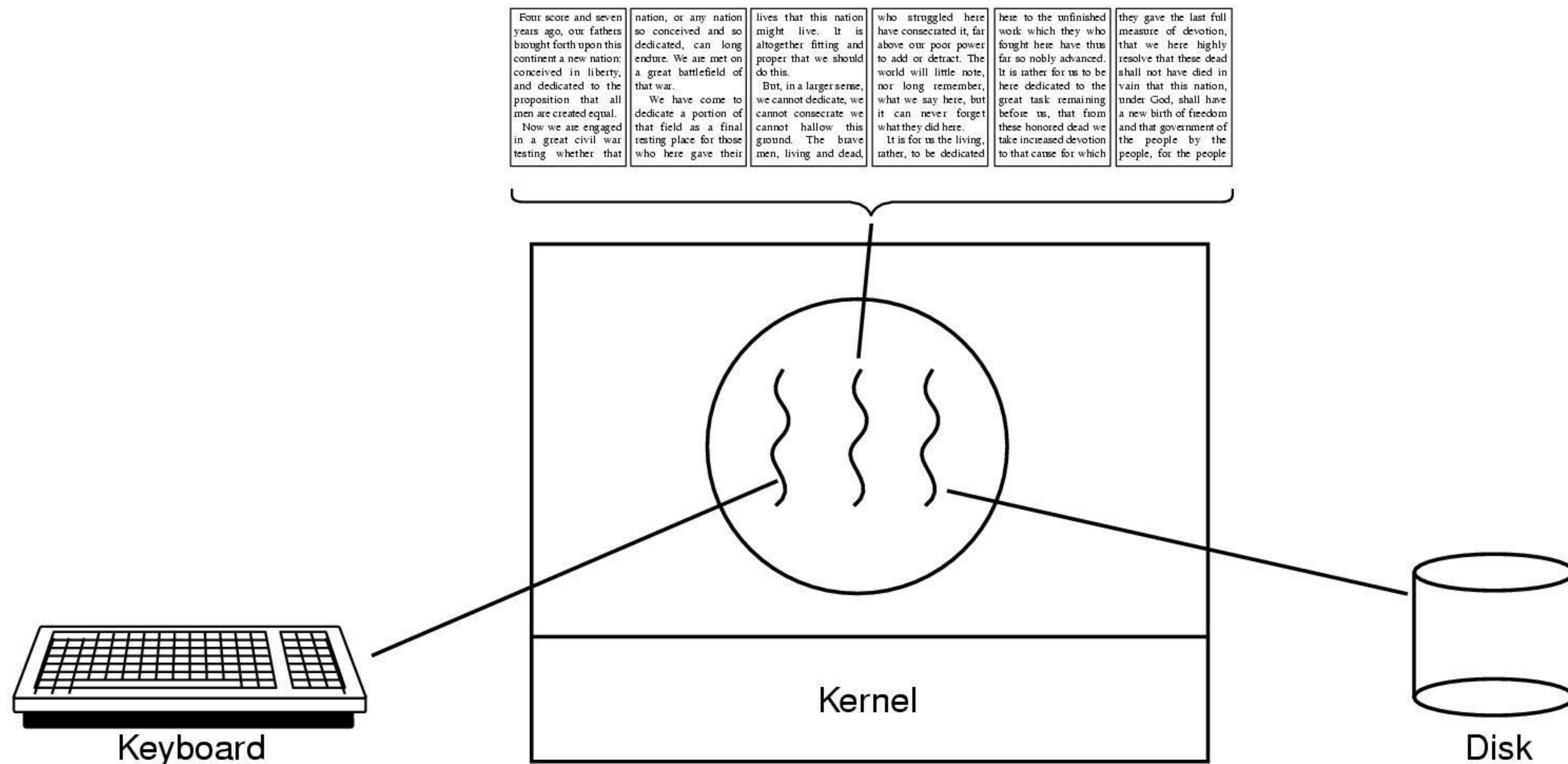
What We will Learn Today

- Threading
- Programming interface – Software layers
- System calls

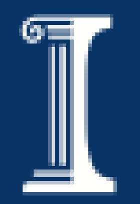
A Brief note on Threading

- Why should an application use multiple threads?
- Things suitable for threading
  - Block for potentially long waits
  - Use many CPU cycles
  - Respond to asynchronous events
  - Execute functions of different importance
  - Execute parallel code

Example: Word Processor



What if the application was single-threaded?



Example: Web Server

Web server process

Dispatcher thread

Worker thread

Web page cache

User space

Kernel

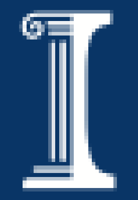
Kernel space

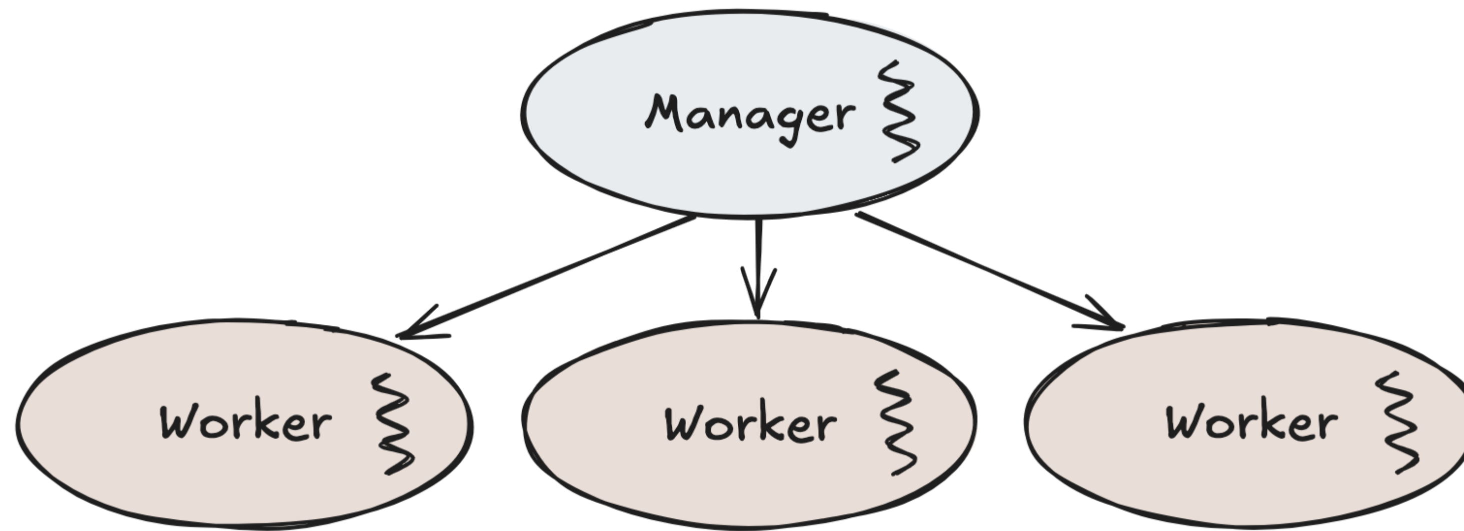
Network connection

What if it the application was single-threaded?

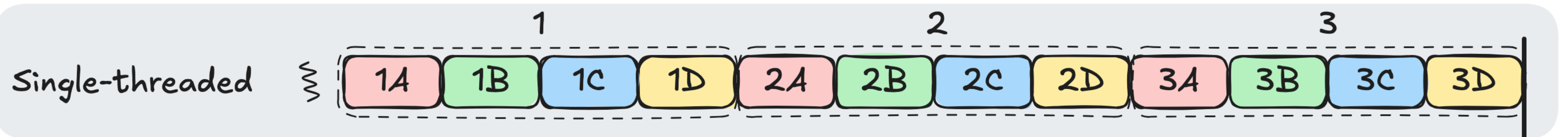
- **Manager/worker**
  - A single thread, the manager assigns work to other threads, the workers. Typically, the manager handles all input and parcels out work to the other tasks
- **Peer**
  - Similar to the manager/worker model, but after the main thread creates other threads, it participates in the work.
- **Pipeline**
  - A task is broken into a series of sub-operations, each of which is handled by a different thread. An automobile assembly line best describes this model

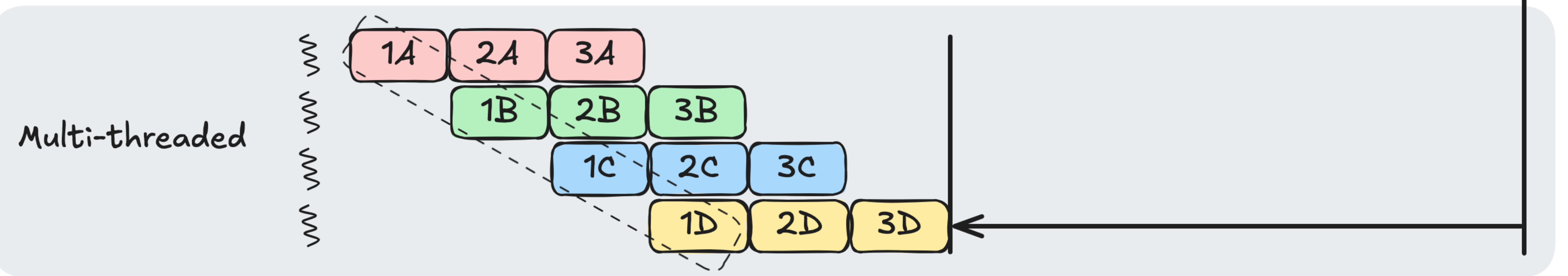
Common Multi-thread Software Architectures





Manager / worker



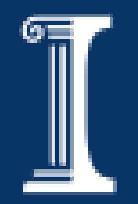


Pipeline

User-level Threads

- **Advantages**
  - Fast Context Switching:
    - User level threads are implemented using user level thread libraries, rather than system calls, hence no call to OS and no interrupts to kernel
    - When a thread is finished running for the moment, it can call **thread_yield.** This instruction (a) saves the thread information in the thread table, and (b) calls the thread scheduler to pick another thread to run.
    - The procedure that saves the local thread state and the scheduler are local procedures, hence no trap to kernel, no context switch, no memory switch, and this makes the thread scheduling very fast.
  - Customized Scheduling

The Programming Interface!



OS Runs on Multiple Platforms while presenting the same Interface:

Application Software

Web Server | Browser | Slack | Pop Mail

The POSIX Standard Specifies UNIX Interface

Operating System (machine independent part)

Read/Write | **Standard Output** | **Device Control** | **File System** | **Communication**

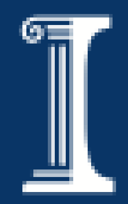
Machine specific part

Portable

Hardware

Network

API is IP of OS



Compilers Web Servers Source Code Control

Databases Word Processing

Web Browsers Email

Portable
OS Library

System Call
Interface

Portable Operating
System Kernel

x86 ARM PowerPC

10Mbps/100Mbps/1Gbps Ethernet

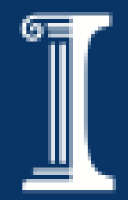
802.11 a/b/g/n SCSI IDE

Graphics Accelerators LCD Screens

Syscall API bridges diverse applications and hardware in the system stack.

Similar to the Internet Protocol (IP)'s role in the network stack!

Software Layers



Application call libraries…

Application

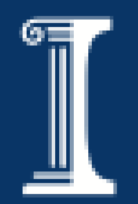
Libraries (e.g., stdio.h)

Portable OS Layer

Machine-dependent layer

Provided pre-compiled
Defined in headers
Input to linker (compiler)
Invoked like functions
May be "resolved" when program is loaded

Software Layers



… libraries make OS system calls…

Software Layers

… system calls access drivers, machine-specific code, etc.

Application

Libraries (e.g., stdio.h)

Portable OS Layer

Machine-dependent layer

- Bootstrap
- System initialization
- Interrupt and exception
- I/O device driver
- Memory management
- Kernel/user mode switching
- Processor management

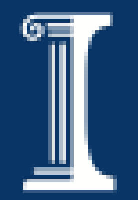
Some Important Syscall Families

- Performing I/O
  `open`, `read`, `write`, `close`

- Creating and managing processes
  `fork`, `exec`, `wait`

- Communicating between processes
  `pipe`, `dup`, `select`, `connect`

Example Syscall Workflow



read (fd, buffer, nbytes)

Address
0xFFFFFFFF

User space

Return to caller
Trap to the kernel
5 Put code for read in register

Library procedure read

10
4

Increment SP 11
Call read
3 Push fd
2 Push &buffer
1 Push nbytes

User program calling read

6
9

Kernel space (Operating system)

Dispatch 7 8 Sys call handler

0

Question

open, read, fork, wait, ...

Is it possible to invoke a syscall without libc wrappers?

yes.

```
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <signal.h>

int
main(int argc, char *argv[])
{
    pid_t tid;

    tid = syscall(SYS_gettid);          // gettid()
    syscall(SYS_tgkill, getpid(), tid, SIGHUP);   // tgkill()
}
```

POSIX Syscalls 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

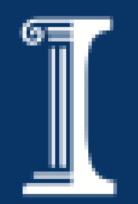
POSIX Syscalls for…

… directory management:

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Open: more than meets the eye



- UNIX file open is a Swiss Army knife:
  - Open the file, return file descriptor
  - Options:
    - if file doesn't exist, return an error
    - If file doesn't exist, create file and open it
    - If file does exist, return an error
    - If file does exist, open file
    - If file exists but isn't empty, nix it then open
    - If file exists but isn't empty, return an error
    - …

Shells… how do they work?

A shell is a job control system
- Allows programmer to create and manage a set of programs to do some task
- Windows, MacOS, Linux all have shells

Example: Shell cmds to compile a C program

```
cc –c sourcefile1.c
cc –c sourcefile2.c
ld –o program sourcefile1.o sourcefile2.o
```

```
helloos@jyw1:~$ ls /
bin    cdrom  dev   home  lib32  libx32      media  nfsroot  proc  run   snap  swapfile  tmp  var
boot   data   etc   lib   lib64  lost+found  mnt    opt      root  sbin  srv   sys       usr
helloos@jyw1:~$ mkdir myfirstdir
helloos@jyw1:~$ ls
myfirstdir
helloos@jyw1:~$ cd myfirstdir/
helloos@jyw1:~/myfirstdir$ pwd
/home/helloos/myfirstdir
helloos@jyw1:~/myfirstdir$
```

Shell Question

If the shell runs at user-level, what system calls does it make to run each of the programs?

POSIX Syscalls for…

… process management:

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

UNIX fork – system call to create a copy of the current process, and start it running

No arguments!

UNIX Process Mgmt

```
pid = fork();
if (pid == 0)
   exec(...);
else
   wait(pid);
```

fork

```
pid = fork();
if (pid == 0)
   exec(...);
else
   wait(pid);
```

exec

```
main () {
  ...

}
```

```
pid = fork();
if (pid == 0)
   exec(...);
else
   wait(pid);
```

wait

Implementing UNIX Fork

Steps to implement UNIX `fork`

- Create and initialize the process control block (PCB) in the kernel
- Create a new address space
- Initialize the address space with a copy of the entire contents of the address space of the parent
- Inherit the execution context of the parent (e.g., any open files)
- Inform the scheduler that the new process is ready to run

Implementing UNIX Exec

- Steps to implement UNIX `exec`
  - Load the program into the current address space
  - Copy arguments into memory in the address space
  - Initialize the hardware context to start execution at ``start''

Simple Shell Implementation

```
char *prog, **args;
int child_pid;

// Read and parse the input a line at a time
while (readAndParseCmdLine(&prog, &args)) {
   child_pid = fork();      // create a child process
    if (child_pid == 0) {
       exec(prog, args);    // I'm the child process. Run program
      // NOT REACHED
    } else {
       wait(child_pid);     // I'm the parent, wait for child
       return 0;
    }
}
```

Process Mgmt Questions

- Can UNIX fork() return an error?
- Can UNIX exec() return an error?
- Can UNIX wait() ever return immediately?

What about Windows?

Windows has `CreateProcess`

- System call to create a new process to run a program
  - Create and initialize the process control block (PCB) in the kernel
  - Create and initialize a new address space
  - Load the program into the address space
  - Copy arguments into memory in the address space
  - Initialize the hardware context to start execution at ``start''
  - Inform the scheduler that the new process is ready to run

What about Windows?

Windows has `CreateProcess`

```
if (!CreateProcess(
    NULL,           // No module name (use command line)
    argv[1],        // Command line
    NULL,           // Process handle not inheritable
    NULL,           // Thread handle not inheritable
    FALSE,          // Set handle inheritance to FALSE
    0,              // No creation flags
    NULL,           // Use parent's environment block
    NULL,           // Use parent's starting directory
    &si,            // Pointer to STARTUPINFO structure
    &pi )           // Pointer to PROCESS_INFORMATION structure
)
```

POSIX Syscalls for…

… miscellaneous tasks:

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970