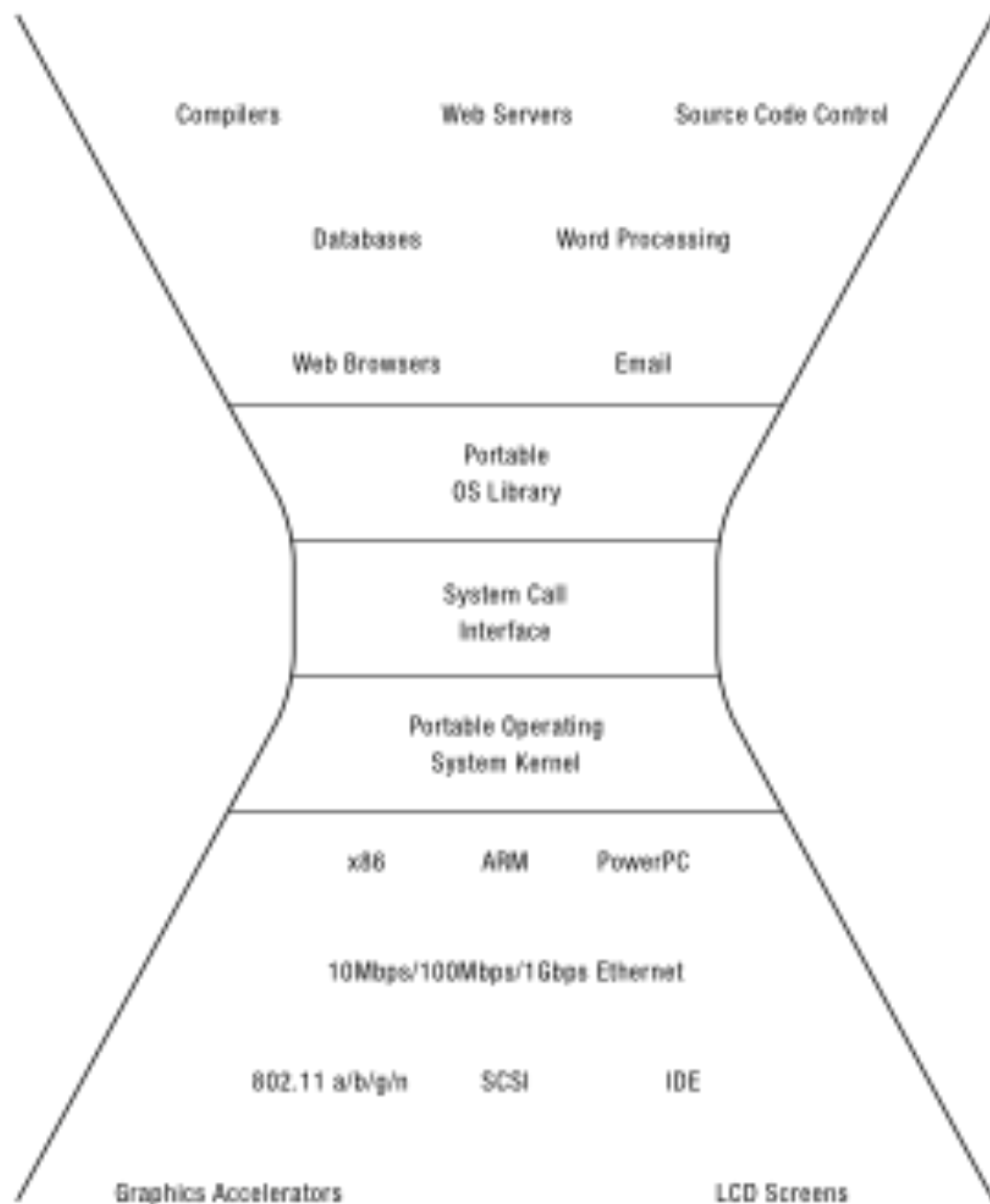


The Process and Programming Interface

- สร้างวิธีไหน
- สร้างยังไง





Main Points

- Creating and managing processes
 - fork, exec, wait
- Performing I/O
 - open, read, write, close
- Communicating between processes
 - pipe, dup, select, connect
- Example: implementing a shell

การสื่อสาร (ส่งข้อมูล)

kernel = เวมส์ด

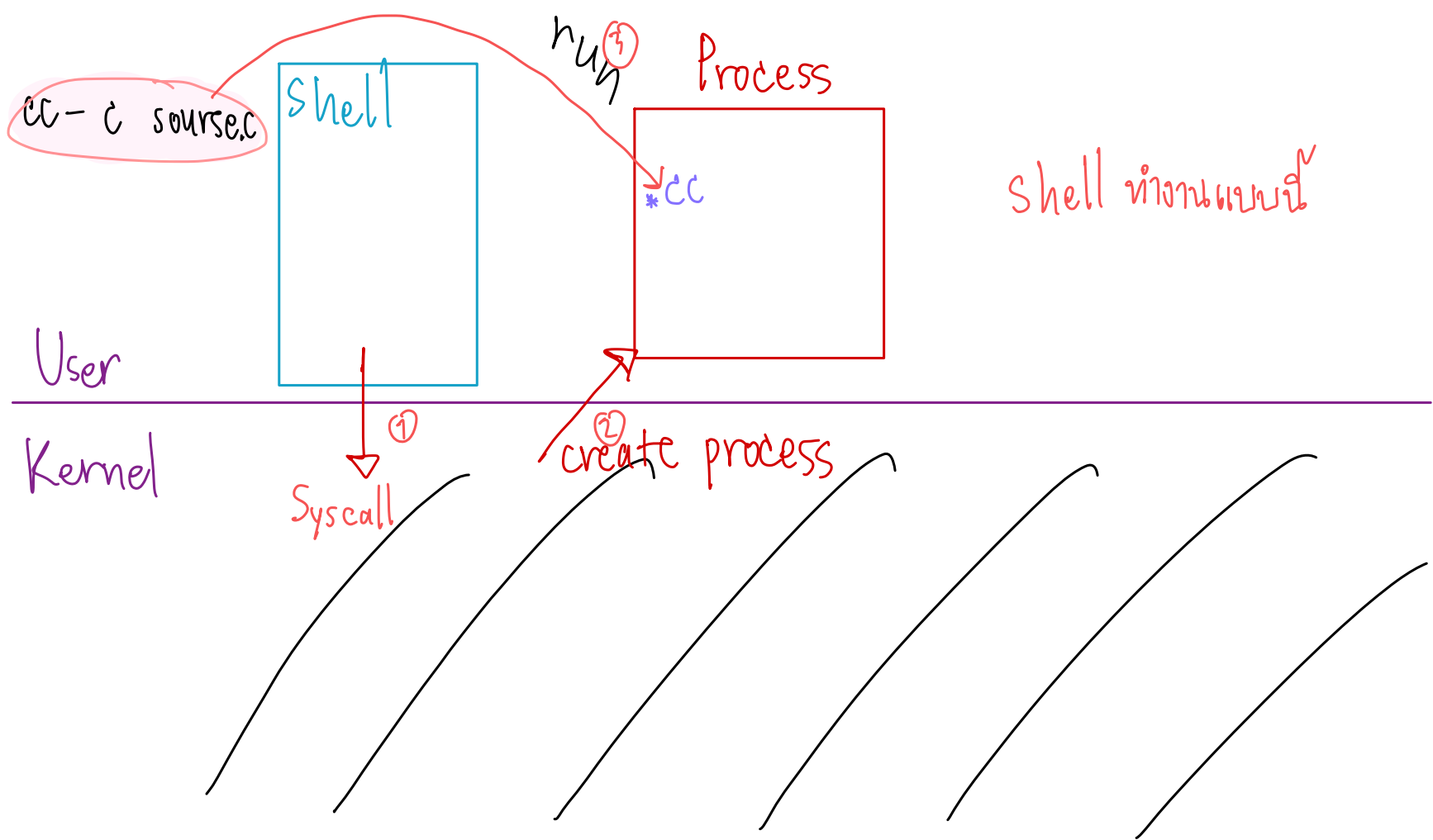
Shell (เฟสอาก) ปรึบเสสวึอน UI

ติดท้อuser กับ kernel

↗ จัดลําดับงาน

- 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, Andriod ^{หน้าทํานานที่ติดท้อกับ user} luncher
- Example: to compile a C program

```
cc -c sourcefile1.c
cc -c sourcefile2.c
ln -o program sourcefile1.o sourcefile2.o
```



Activity #1

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

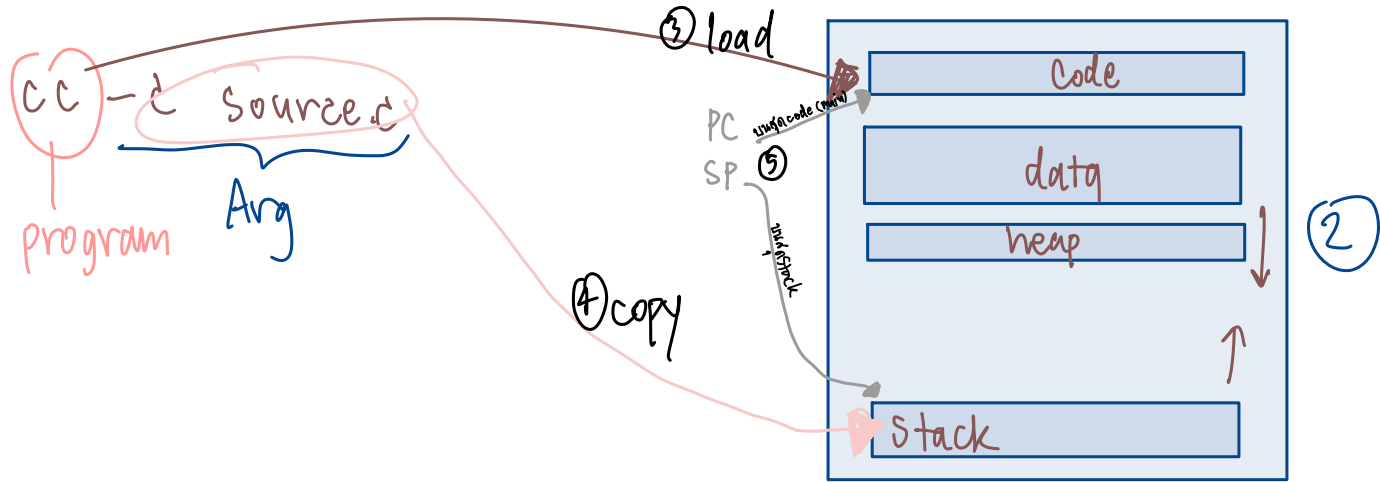
Unix २ fork(), exec(), wait()

Windows २ Create process

Windows 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

6 steps ที่
เกิดขึ้นโดย System Calls
"Create process"



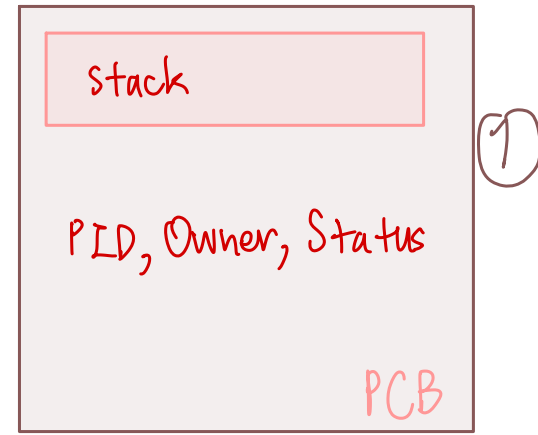
U

K



Scheduler

จัดสรรว่า process ไหนจะเข้าใช้ kernel
ได้เมื่อไหร่ ตามความสำคัญ



Windows CreateProcess API

(simplified) ของจริงเยอะกว่านี้

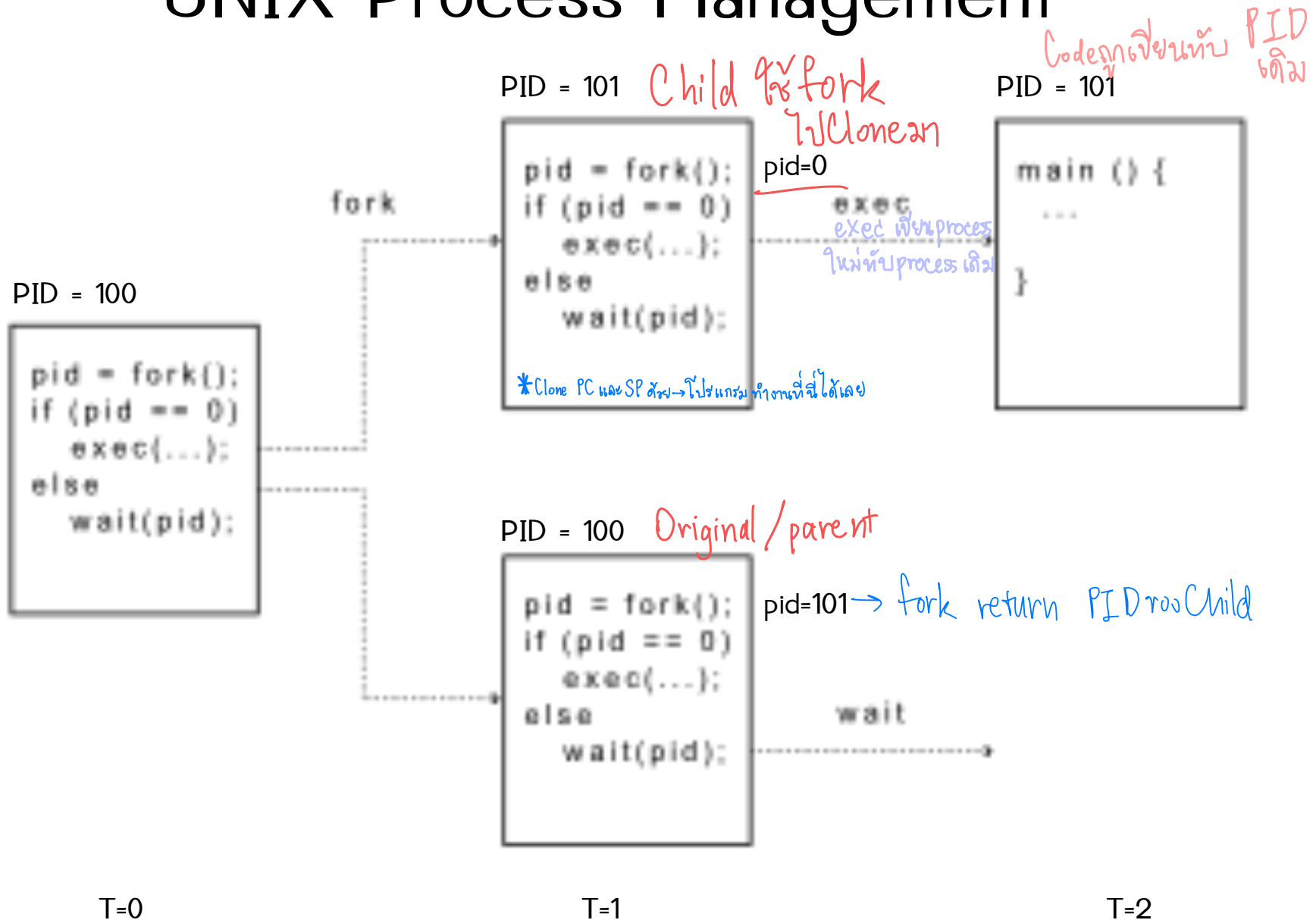
พรี
รัน
สคริปต์

```
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  
)
```

UNIX Process Management

- UNIX ^(RUN) fork – ^{fork = Clone (Cloning) process นั้น} system call to create a copy of the current process, and start it running
 - No arguments!
- UNIX ^(RUN) exec – ^{exec = process เดิมถูกเปลี่ยนไป} system call to change the program being run by the current process
- UNIX ^{(RUN) กับ shell} wait – ^{wait = ไม่ทำงานจนกว่าเงื่อนไขจะตรงหรือจบ} system call to wait for a process to finish
- UNIX ^{ส่ง signal Process อื่น} signal – system call to send a notification to another process

UNIX Process Management



Activity #2: What does this code print?

สับสน 2 ข้อ 1) parent 2) child

int child_pid = fork();

ถ้า fork ด้ PID=100 ถ้า else
คือ 92 child

if (child_pid == 0) { // I'm the child process

printf("I am process 107 \n", getpid());

return 0;

win fork

} else { // I'm the parent process

printf("I am parent of process 100 \n", child_pid);

return 0;

}

Activity #3

- Can UNIX fork() return an error? Why?

Yes เพราะอาจเกิดกับ Resource เช่น RAM ใกล้เคียง RAM เต็มแล้ว

- Can UNIX exec() return an error? Why?

Yes เพราะ Load ใกล้เคียง < ลบทิ้ง โปรแกรมจากดิสก์มาใส่หน่วย

- Can UNIX wait() ever return immediately? Why?

No หากไม่มีโปรแกรมทำงาน

Implementing UNIX fork

Steps to implement UNIX fork

- Create and initialize the process control block (PCB) in the kernel ที่ใน RAM memory)
- Create a new address space ก็จะได้ถ้า RAM ไม่พอ/มีน้อย
ก็เกี่ยวข้องกับ Resource
- Initialize the address space with a copy of the entire contents of the address space of the parent ไม่ผิดพลาด
- Inherit the execution context of the parent (e.g., any open files) ไปมา
- Inform the scheduler that the new process is ready to run ทำได้ถ้าสร้าง process + user space ได้

Implementing UNIX exec



- Steps to implement UNIX exec

- Load the program into the current address space *เกิด error เวลา Load เช่น รันหมด / สิทธิไม่พอ execute ไปตรงไหน*
- Copy arguments into memory in the address space *ไม่ผิด error ยกเว้น RAM สิ้นสุด*
- Initialize the hardware context to start execution at "start" *ใน code ให้อ่านให้*

UNIX I/O



- Uniformity ชุดคำสั่งเดียวกันทั้งหมด ยกเว้น Syscalls
 - All operations on all files, devices use the same set of system calls: ^{ทำงานแบบนี้} open, close, read, write
- Open before use ระบบตรวจสอบ (block ได้)
 - Open returns a handle (file descriptor) for use in later calls on the file ^{เปิดได้ทีละ 1 program หากโปรแกรมอื่นมาเปิดระบบจะทำการ block จนที่เปิดอีกคนหนึ่งที่จะกว่าคนนั้นจะปิด}
- Byte-oriented Byte array 1 บิต
- Kernel-buffered ^{ตัวอ่านกลาง} read/write ^{ไม่ได้ทำให้ file โดยตรง ทำ buffer ก่อนจน}
- Explicit close ^{ให้ kernel รู้ว่าต้องเอา buffer ไปทิ้งไปไว้ที่นั่นหมดก่อนล้าง buffer} ^{buffer เต็ม ถึงจะบันทึกให้}
 - To garbage collect the open file descriptor

UNIX File System Interface

- 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
 - ...

Activity #4

Interface Design Question

- Why not ^{666777 666777}separate syscalls for open/create/exists?

หากเป็นแบบ environment single user / single task ไม่ซับซ้อน
 ถ้าเป็น multi user อาจจะมีปัญหานิดๆ เช่น create พร้อมกัน 2 คน ตามชื่อซ้ำ
 if (!exists(name))
 อีกคนไม่สำเร็จ


```
create(name);    // can create fail?
```

```
fd = open(name); // does the file exist?
```

→ Syscall ตัวอื่นไม่มากนัก

→ จบก่อน ค่อยทำในบทต่อไป
น้ำฝน

→ ลดการทำงานซ้ำซ้อน

ทำ open 3 ฟังก์ชัน ซึ่งผลปรากฏ หน้า 
สลับกันไป

Implementing a Shell

```
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;
    }
}
```

In Unix patch file

- A program can be a file of commands
- A program can send its output to a file
- A program can read its input from a file
- The output of one program can be the input to another program

ให้อำสั่ง

dir > file.txt

redirect

xxx < file.txt

Input

inter process

อวทฤษฎีการส่งไฟล์

dir | more

โปรแกรม โปรแกรม

หนึ่ง process → process หนึ่ง

Interprocess Communication

ส่งข้อมูลระหว่าง process

- Producer-consumer ส่งข้อมูลไม่ได้ (2 program ต้องทำงานพร้อมกัน / ส่ง - รับพร้อมกัน)
 - Output of one program is accepted as input of another program
 - One-way communication
 - Pipe
- Client-server ส่งข้อมูลผ่าน Socket ทำงานพร้อมกัน
 - Two-way communication Client \rightleftharpoons server
 - Server implements specialize task ใช้ services ที่งานอาจมีหรือรับ - พร้อมส่ง
 - Print serve
- File system เก็บเป็น file ใช้ไฟล์เป็นสื่อกลางในการรับ-ส่งข้อมูล
 - Write data to a file then read file as an input
 - Reader and writer are not need to running at the same time

2 ways commu

Operating system structure

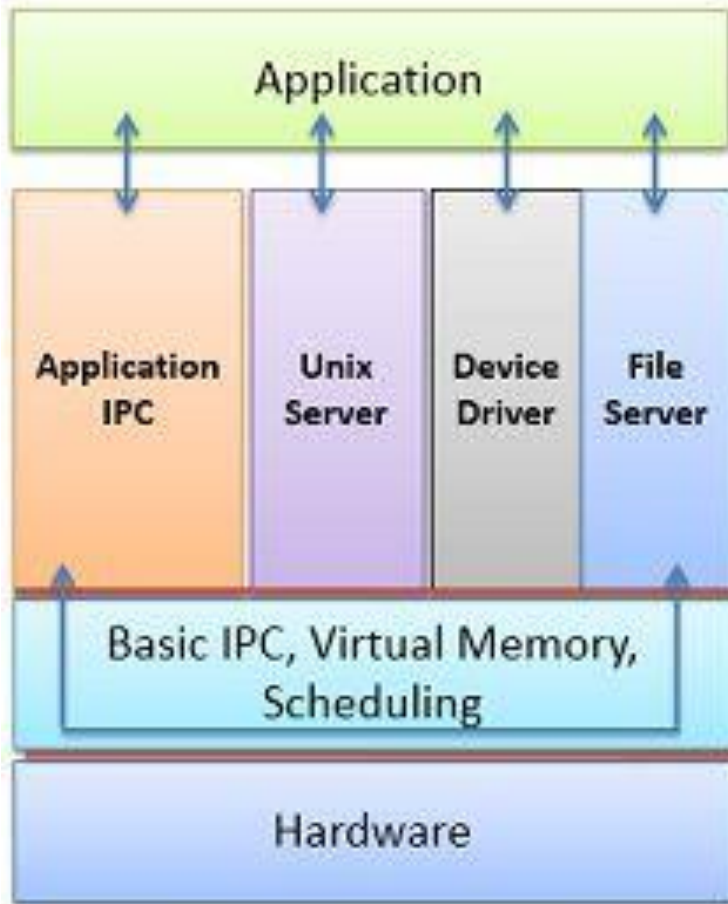
- Monolithic kernel
- Microkernel

```
if  
{  
  if  
  {  
    }  
  }  
}
```

```
print (" + " , i , end = " ")
```

```
lt  
ln
```

ข้อเสียคือ ใช้ service มากมายอยู่ใน user mode หรือ switch mode บ่อยๆ

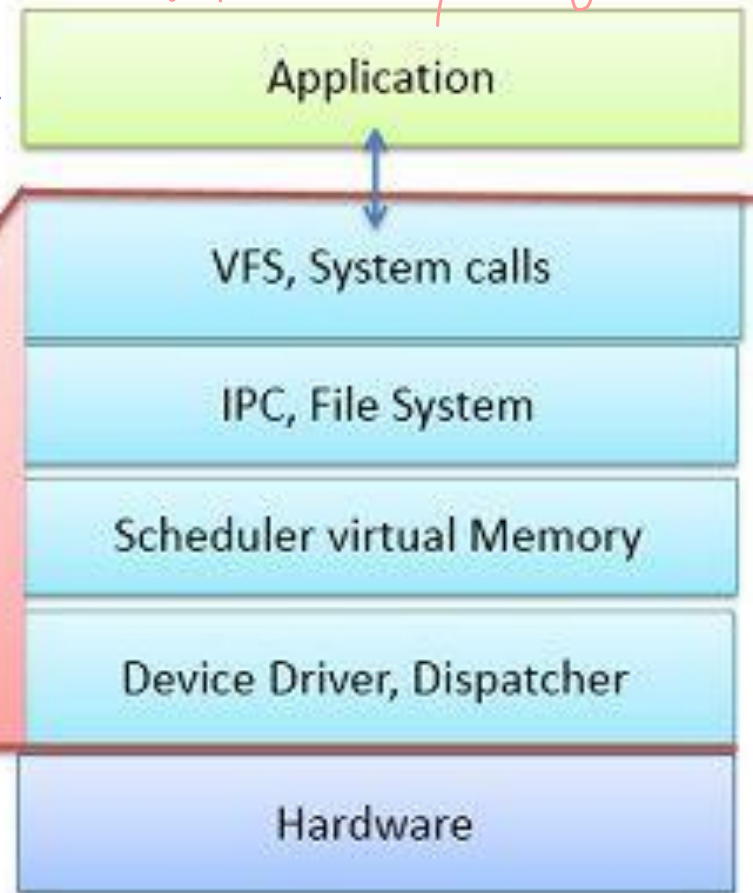


Microkernel

ไม่ใช่ kernel เอง

หลักเป็น module ประกอบกันได้อีก

ปัจจุบันมีโอเอสที่ผสม
ใช้ kernel hybrid



Monolithic Kernel

ใหญ่/อวตารทั้งก้อน