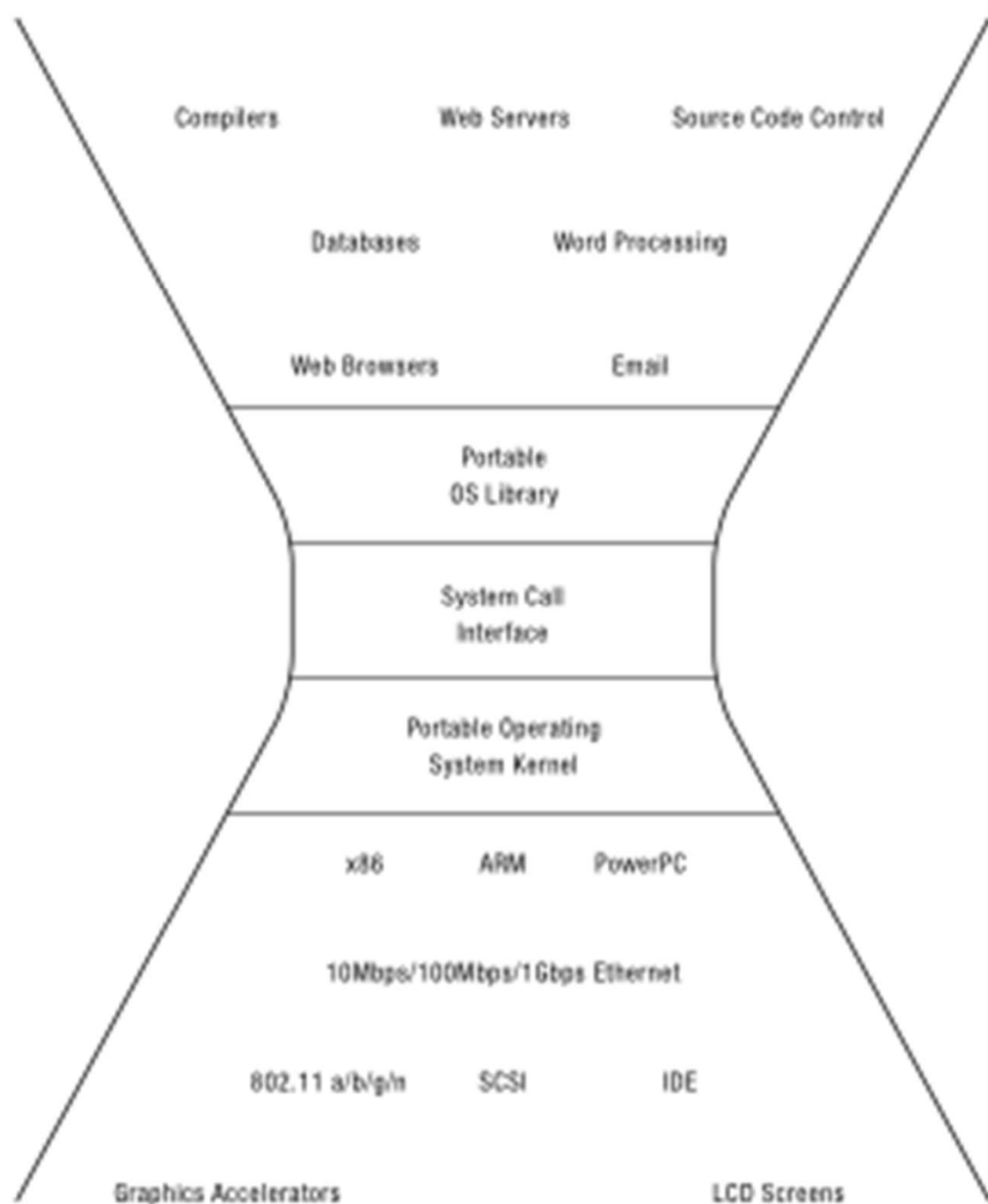


# The Programming Interface



# Main Points *function of shell*

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

# Shell

ตอบ, จัดการระบบ app

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

- Example: to compile a C program

```
cc -c sourcefile1.c
```

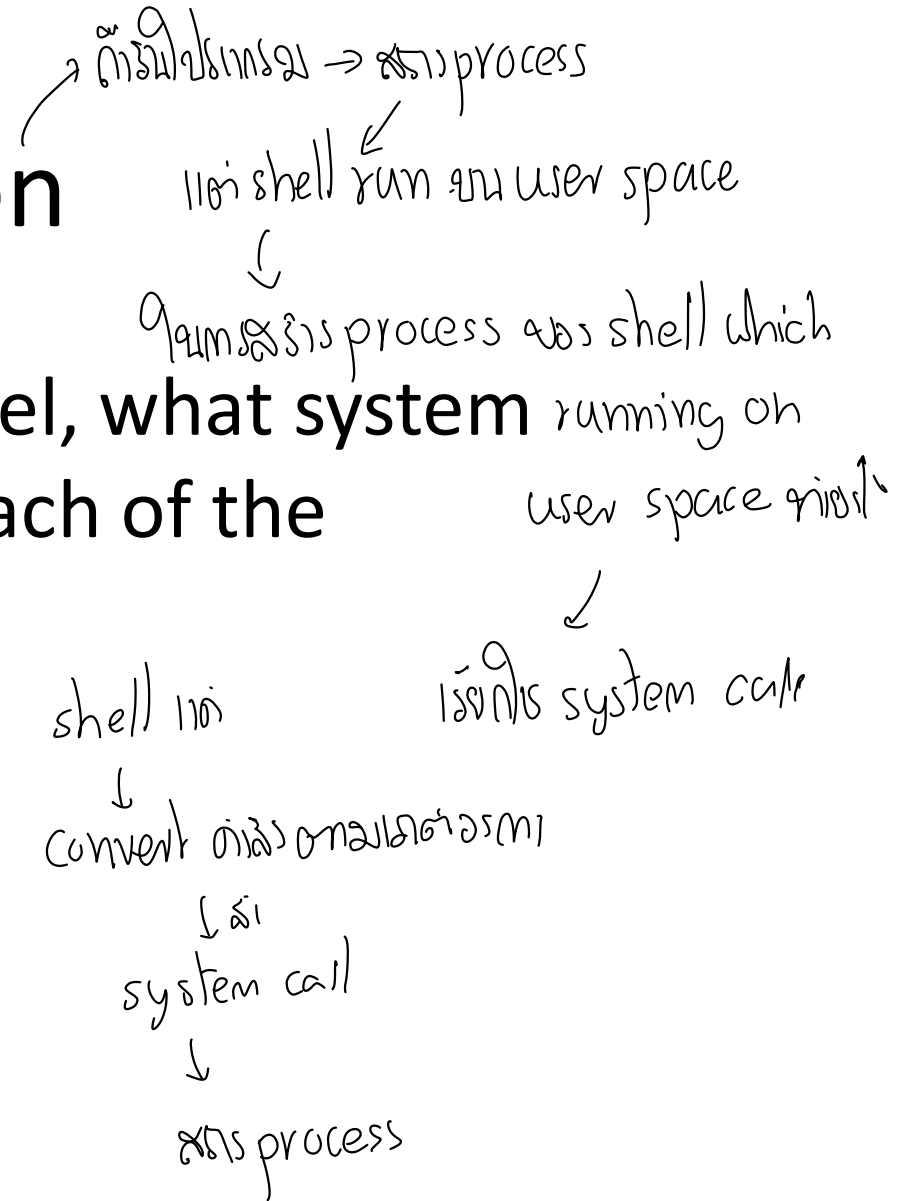
```
cc -c sourcefile2.c
```

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

} พอสั่งงาน shell ไปได้แล้ว  
ก็มาสั่งงาน

# Question

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



# 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 → *ချက်ချင်း*
  - Copy arguments into memory in the address space
  - Initialize the hardware context to start execution at "start"
    - ↳ *pc စီမံကိန်း main, sp ခလံာ်*
  - Inform the scheduler that the new process is ready to run
    - ↳ *h i ၀ ၁ ၂ ၃ ၄ ၅ ၆ ၇ ၈ ၉ ၁၀ ၁၁ ၁၂ ၁၃ ၁၄ ၁၅ ၁၆ ၁၇ ၁၈ ၁၉ ၂၀ ၂၁ ၂၂ ၂၃ ၂၄ ၂၅ ၂၆ ၂၇ ၂၈ ၂၉ ၃၀ ၃၁ ၃၂ ၃၃ ၃၄ ၃၅ ၃၆ ၃၇ ၃၈ ၃၉ ၄၀ ၄၁ ၄၂ ၄၃ ၄၄ ၄၅ ၄၆ ၄၇ ၄၈ ၄၉ ၅၀ ၅၁ ၅၂ ၅၃ ၅၄ ၅၅ ၅၆ ၅၇ ၅၈ ၅၉ ၆၀ ၆၁ ၆၂ ၆၃ ၆၄ ၆၅ ၆၆ ၆၇ ၆၈ ၆၉ ၇၀ ၇၁ ၇၂ ၇၃ ၇၄ ၇၅ ၇၆ ၇၇ ၇၈ ၇၉ ၈၀ ၈၁ ၈၂ ၈၃ ၈၄ ၈၅ ၈၆ ၈၇ ၈၈ ၈၉ ၉၀ ၉၁ ၉၂ ၉၃ ၉၄ ၉၅ ၉၆ ၉၇ ၉၈ ၉၉ ၁၀၀ ၁၀၁ ၁၀၂ ၁၀၃ ၁၀၄ ၁၀၅ ၁၀၆ ၁၀၇ ၁၀၈ ၁၀၉ ၁၁၀ ၁၁၁ ၁၁၂ ၁၁၃ ၁၁၄ ၁၁၅ ၁၁၆ ၁၁၇ ၁၁၈ ၁၁၉ ၁၂၀ ၁၂၁ ၁၂၂ ၁၂၃ ၁၂၄ ၁၂၅ ၁၂၆ ၁၂၇ ၁၂၈ ၁၂၉ ၁၃၀ ၁၃၁ ၁၃၂ ၁၃၃ ၁၃၄ ၁၃၅ ၁၃၆ ၁၃၇ ၁၃၈ ၁၃၉ ၁၄၀ ၁၄၁ ၁၄၂ ၁၄၃ ၁၄၄ ၁၄၅ ၁၄၆ ၁၄၇ ၁၄၈ ၁၄၉ ၁၅၀ ၁၅၁ ၁၅၂ ၁၅၃ ၁၅၄ ၁၅၅ ၁၅၆ ၁၅၇ ၁၅၈ ၁၅၉ ၁၆၀ ၁၆၁ ၁၆၂ ၁၆၃ ၁၆၄ ၁၆၅ ၁၆၆ ၁၆၇ ၁၆၈ ၁၆၉ ၁၇၀ ၁၇၁ ၁၇၂ ၁၇၃ ၁၇၄ ၁၇၅ ၁၇၆ ၁၇၇ ၁၇၈ ၁၇၉ ၁၈၀ ၁၈၁ ၁၈၂ ၁၈၃ ၁၈၄ ၁၈၅ ၁၈၆ ၁၈၇ ၁၈၈ ၁၈၉ ၁၉၀ ၁၉၁ ၁၉၂ ၁၉၃ ၁၉၄ ၁၉၅ ၁၉၆ ၁၉၇ ၁၉၈ ၁၉၉ ၂၀၀ ၂၀၁ ၂၀၂ ၂၀၃ ၂၀၄ ၂၀၅ ၂၀၆ ၂၀၇ ၂၀၈ ၂၀၉ ၂၁၀ ၂၁၁ ၂၁၂ ၂၁၃ ၂၁၄ ၂၁၅ ၂၁၆ ၂၁၇ ၂၁၈ ၂၁၉ ၂၂၀ ၂၂၁ ၂၂၂ ၂၂၃ ၂၂၄ ၂၂၅ ၂၂၆ ၂၂၇ ၂၂၈ ၂၂၉ ၂၃၀ ၂၃၁ ၂၃၂ ၂၃၃ ၂၃၄ ၂၃၅ ၂၃၆ ၂၃၇ ၂၃၈ ၂၃၉ ၂၄၀ ၂၄၁ ၂၄၂ ၂၄၃ ၂၄၄ ၂၄၅ ၂၄၆ ၂၄၇ ၂၄၈ ၂၄၉ ၂၅၀ ၂၅၁ ၂၅၂ ၂၅၃ ၂၅၄ ၂၅၅ ၂၅၆ ၂၅၇ ၂၅၈ ၂၅၉ ၂၆၀ ၂၆၁ ၂၆၂ ၂၆၃ ၂၆၄ ၂၆၅ ၂၆၆ ၂၆၇ ၂၆၈ ၂၆၉ ၂၇၀ ၂၇၁ ၂၇၂ ၂၇၃ ၂၇၄ ၂၇၅ ၂၇၆ ၂၇၇ ၂၇၈ ၂၇၉ ၂၈၀ ၂၈၁ ၂၈၂ ၂၈၃ ၂၈၄ ၂၈၅ ၂၈၆ ၂၈၇ ၂၈၈ ၂၈၉ ၂၉၀ ၂၉၁ ၂၉၂ ၂၉၃ ၂၉၄ ၂၉၅ ၂၉၆ ၂၉၇ ၂၉၈ ၂၉၉ ၃၀၀ ၃၀၁ ၃၀၂ ၃၀၃ ၃၀၄ ၃၀၅ ၃၀၆ ၃၀၇ ၃၀၈ ၃၀၉ ၃၁၀ ၃၁၁ ၃၁၂ ၃၁၃ ၃၁၄ ၃၁၅ ၃၁၆ ၃၁၇ ၃၁၈ ၃၁၉ ၃၂၀ ၃၂၁ ၃၂၂ ၃၂၃ ၃၂၄ ၃၂၅ ၃၂၆ ၃၂၇ ၃၂၈ ၃၂၉ ၃၃၀ ၃၃၁ ၃၃၂ ၃၃၃ ၃၃၄ ၃၃၅ ၃၃၆ ၃၃၇ ၃၃၈ ၃၃၉ ၃၄၀ ၃၄၁ ၃၄၂ ၃၄၃ ၃၄၄ ၃၄၅ ၃၄၆ ၃၄၇ ၃၄၈ ၃၄၉ ၃၅၀ ၃၅၁ ၃၅၂ ၃၅၃ ၃၅၄ ၃၅၅ ၃၅၆ ၃၅၇ ၃၅၈ ၃၅၉ ၃၆၀ ၃၆၁ ၃၆၂ ၃၆၃ ၃၆၄ ၃၆၅ ၃၆၆ ၃၆၇ ၃၆၈ ၃၆၉ ၃၇၀ ၃၇၁ ၃၇၂ ၃၇၃ ၃၇၄ ၃၇၅ ၃၇၆ ၃၇၇ ၃၇၈ ၃၇၉ ၃၈၀ ၃၈၁ ၃၈၂ ၃၈၃ ၃၈၄ ၃၈၅ ၃၈၆ ၃၈၇ ၃၈၈ ၃၈၉ ၃၉၀ ၃၉၁ ၃၉၂ ၃၉၃ ၃၉၄ ၃၉၅ ၃၉၆ ၃၉၇ ၃၉၈ ၃၉၉ ၄၀၀ ၄၀၁ ၄၀၂ ၄၀၃ ၄၀၄ ၄၀၅ ၄၀၆ ၄၀၇ ၄၀၈ ၄၀၉ ၄၁၀ ၄၁၁ ၄၁၂ ၄၁၃ ၄၁၄ ၄၁၅ ၄၁၆ ၄၁၇ ၄၁၈ ၄၁၉ ၄၂၀ ၄၂၁ ၄၂၂ ၄၂၃ ၄၂၄ ၄၂၅ ၄၂၆ ၄၂၇ ၄၂၈ ၄၂၉ ၄၃၀ ၄၃၁ ၄၃၂ ၄၃၃ ၄၃၄ ၄၃၅ ၄၃၆ ၄၃၇ ၄၃၈ ၄၃၉ ၄၄၀ ၄၄၁ ၄၄၂ ၄၄၃ ၄၄၄ ၄၄၅ ၄၄၆ ၄၄၇ ၄၄၈ ၄၄၉ ၄၅၀ ၄၅၁ ၄၅၂ ၄၅၃ ၄၅၄ ၄၅၅ ၄၅၆ ၄၅၇ ၄၅၈ ၄၅၉ ၄၆၀ ၄၆၁ ၄၆၂ ၄၆၃ ၄၆၄ ၄၆၅ ၄၆၆ ၄၆၇ ၄၆၈ ၄၆၉ ၄၇၀ ၄၇၁ ၄၇၂ ၄၇၃ ၄၇၄ ၄၇၅ ၄၇၆ ၄၇၇ ၄၇၈ ၄၇၉ ၄၈၀ ၄၈၁ ၄၈၂ ၄၈၃ ၄၈၄ ၄၈၅ ၄၈၆ ၄၈၇ ၄၈၈ ၄၈၉ ၄၉၀ ၄၉၁ ၄၉၂ ၄၉၃ ၄၉၄ ၄၉၅ ၄၉၆ ၄၉၇ ၄၉၈ ၄၉၉ ၅၀၀ ၅၀၁ ၅၀၂ ၅၀၃ ၅၀၄ ၅၀၅ ၅၀၆ ၅၀၇ ၅၀၈ ၅၀၉ ၅၁၀ ၅၁၁ ၅၁၂ ၅၁၃ ၅၁၄ ၅၁၅ ၅၁၆ ၅၁၇ ၅၁၈ ၅၁၉ ၅၂၀ ၅၂၁ ၅၂၂ ၅၂၃ ၅၂၄ ၅၂၅ ၅၂၆ ၅၂၇ ၅၂၈ ၅၂၉ ၅၃၀ ၅၃၁ ၅၃၂ ၅၃၃ ၅၃၄ ၅၃၅ ၅၃၆ ၅၃၇ ၅၃၈ ၅၃၉ ၅၄၀ ၅၄၁ ၅၄၂ ၅၄၃ ၅၄၄ ၅၄၅ ၅၄၆ ၅၄၇ ၅၄၈ ၅၄၉ ၅၅၀ ၅၅၁ ၅၅၂ ၅၅၃ ၅၅၄ ၅၅၅ ၅၅၆ ၅၅၇ ၅၅၈ ၅၅၉ ၅၆၀ ၅၆၁ ၅၆၂ ၅၆၃ ၅၆၄ ၅၆၅ ၅၆၆ ၅၆၇ ၅၆၈ ၅၆၉ ၅၇၀ ၅၇၁ ၅၇၂ ၅၇၃ ၅၇၄ ၅၇၅ ၅၇၆ ၅၇၇ ၅၇၈ ၅၇၉ ၅၈၀ ၅၈၁ ၅၈၂ ၅၈၃ ၅၈၄ ၅၈၅ ၅၈၆ ၅၈၇ ၅၈၈ ၅၈*

၂၄၆၇၈

↳ PC စီမံကိန်း, SP ချခံ

↓ กลไกที่สมองใช้ประมวลผลความรู้สึก  
↓ สมองได้ข้อมูล

↓  
अवस्था status → ready

↓  
1. task scheduler  
2. algorithms  
3. priority

# Windows CreateProcess API (simplified)

→ ตัวนี้สร้าง window

↓  
เปิด double click

↓  
กดปุ่ม start button sys call

```
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  
)
```

creat process

↓

สร้าง

↓

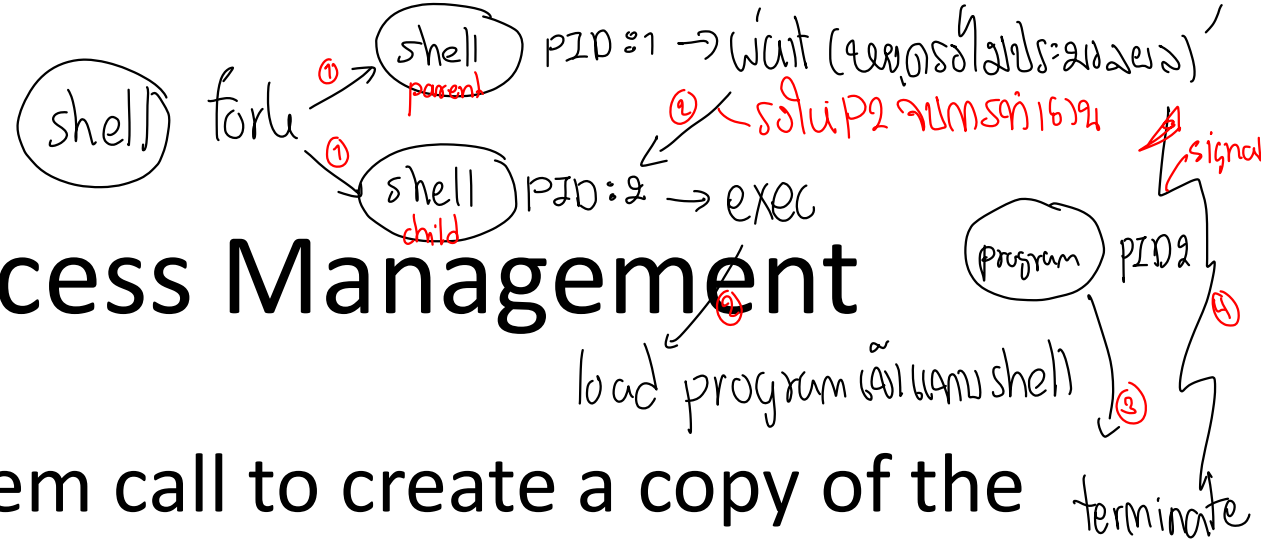
จบ

scheduler ready

↓  
scheduler จะเลือก process ใดมา run  
→ want to run process

5

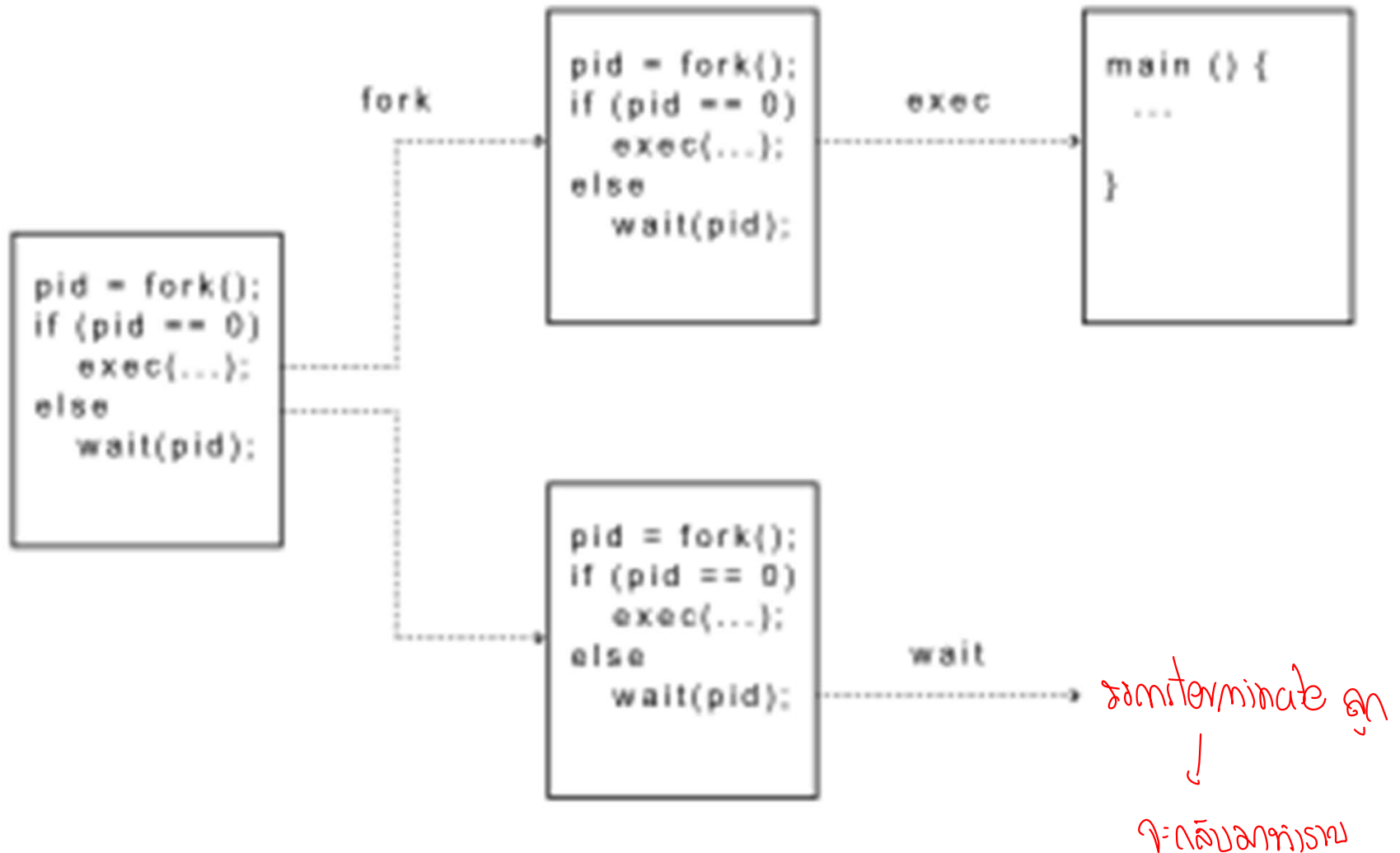
# UNIX Process Management



- UNIX fork – system call to create a copy of the current process, and start it running
  - No arguments!
- UNIX exec – system call to change the program being run by the current process
- UNIX wait – system call to wait for a process to finish
- UNIX signal – system call to send a notification to another process



# UNIX Process Management



Question: What does this code print?

```
int child_pid = fork();  
if (child_pid == 0) {  
    printf("I am process # %d\n", getpid());  
    return 0;  
} else {  
    printf("I am parent of process # %d\n", child_pid);  
    return 0;  
}
```

Handwritten annotations:

- return child pid ของกระบวนการ (return child pid of the process)
- process ใหม่, ลูก → process ที่ยัง 2 แยกกัน (code) (new process, child → process that is still 2 separate (code))
- return child pid = 0 ของลูก (return child pid = 0 of child)
- process ลูก → exec → (child process → exec →)
- process ใหม่ (new process)
- waitpid ลูก (waitpid child)
- same concept always (same concept always)

# Questions

- Can UNIX fork() return an error? Why?  
*↳ resource limit → no more*
- Can UNIX exec() return an error? Why?  
*↳*
- Can UNIX wait() ever return immediately? Why?

# 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 fork
  - 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''

# UNIX I/O

- Uniformity
  - All operations on all files, devices use the same set of system calls: open, close, read, write
- Open before use
  - Open returns a handle (file descriptor) for use in later calls on the file
- Byte-oriented
- Kernel-buffered read/write
- Explicit close
  - 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
    - ...

# Interface Design Question

- Why not separate syscalls for open/create/exists?

```
if (!exists(name))
```

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

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

A  
1952



# 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

- 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

# Interprocess Communication

- ① • **Producer-consumer**
  - Output of one program is accepted as input of another program
    - One-way communication
    - Pipe
- ② – **Client-server**
  - Two-way communication
  - Server implements specialize task
    - Print serve
- ③ – **File system**
  - Write data to a file then read file as an input
  - Reader and writer are not need to running at the same time

# Operating system structure

- Monolithic kernel
- Microkernel

