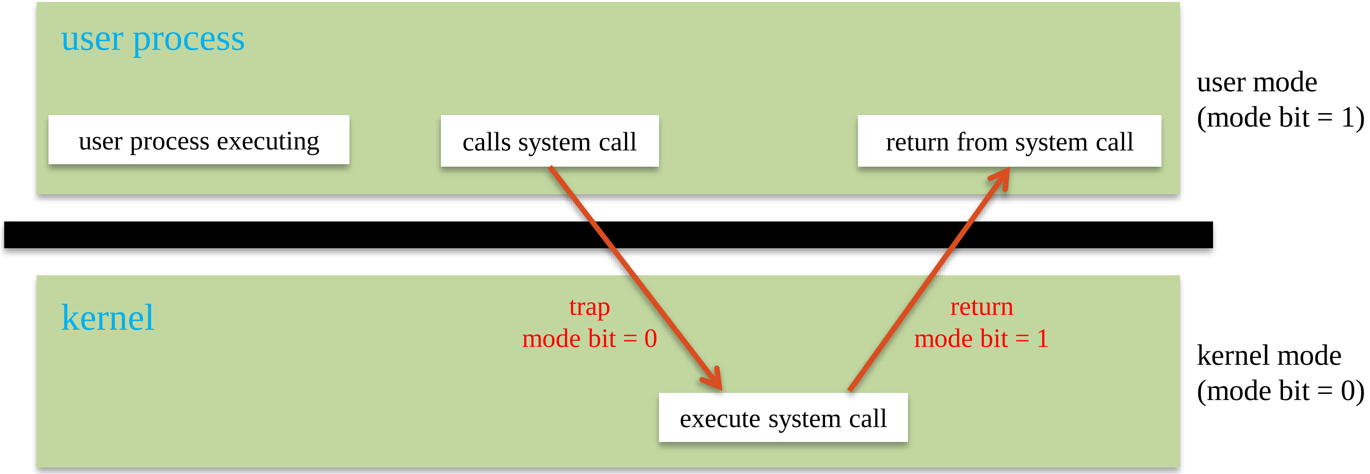
Operating Systems (OS)
Processes

Prof. Eun-Seok Ryu (esryu@skku.edu)
Multimedia Computing Systems Laboratory
http://mcsl.skku.edu
Department of Immersive Media Engineering
Department of Computer Education
Sungkyunkwan University (SKKU)

Review ‘System Calls’

용어 개념
확인



user process

user process executing

calls system call

return from system call

user mode
(mode bit = 1)

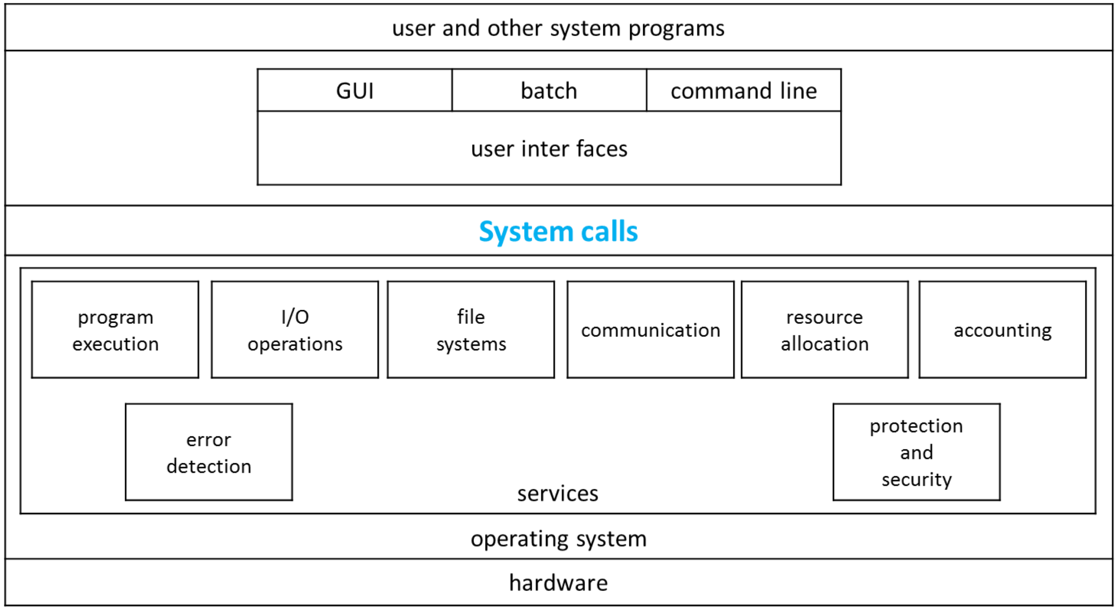
kernel

trap
mode bit = 0

return
mode bit = 1

execute system call

kernel mode
(mode bit = 0)



user and other system programs

GUI | batch | command line

user inter faces

System calls

program execution

I/O operations

file systems

communication

resource allocation

accounting

error detection

services

protection and security

operating system

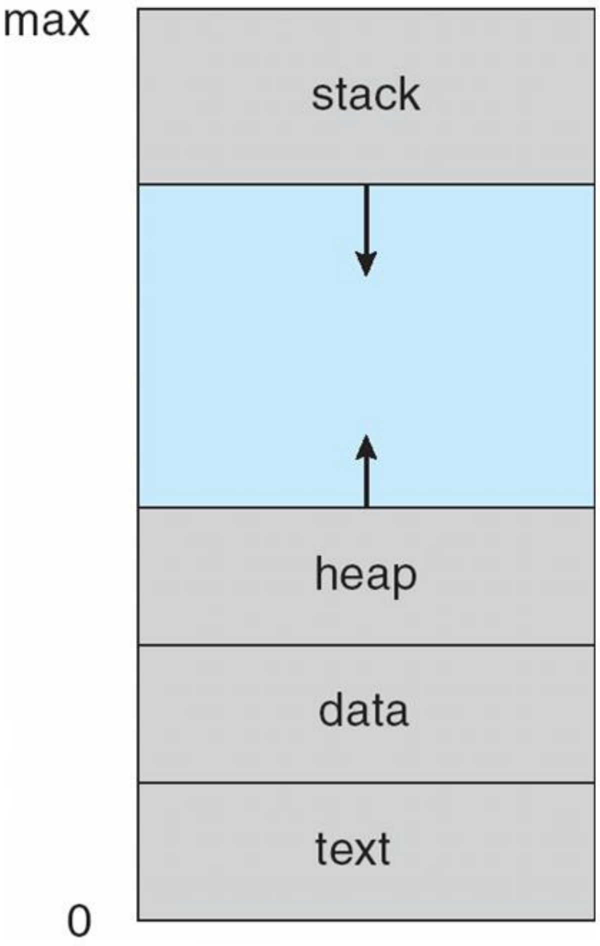
hardware

Process Concept

프로세스/프로그램
개념 중요

- **Process** – a program in execution; process execution must progress in sequential fashion

- Multiple parts
  - The program code, also called **text section**
  - Current activity including program counter, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

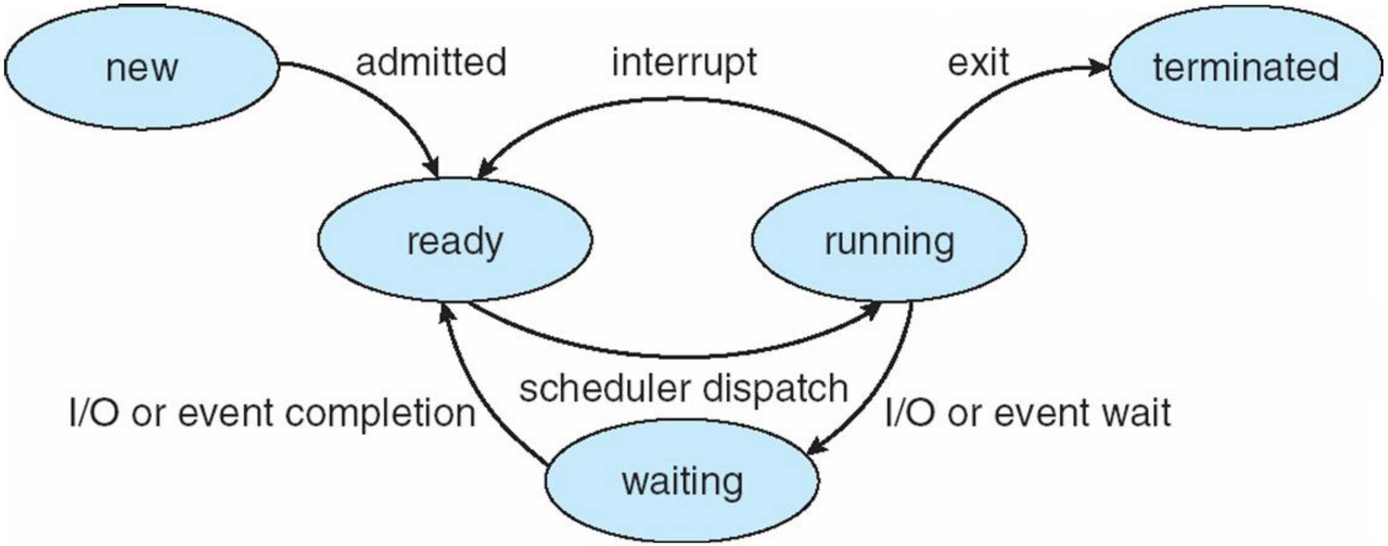


Process State

그릴 수 있는가?

As a process executes, it changes **state**

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **ready**: The process is waiting to be assigned to a processor
- **terminated**: The process has finished execution



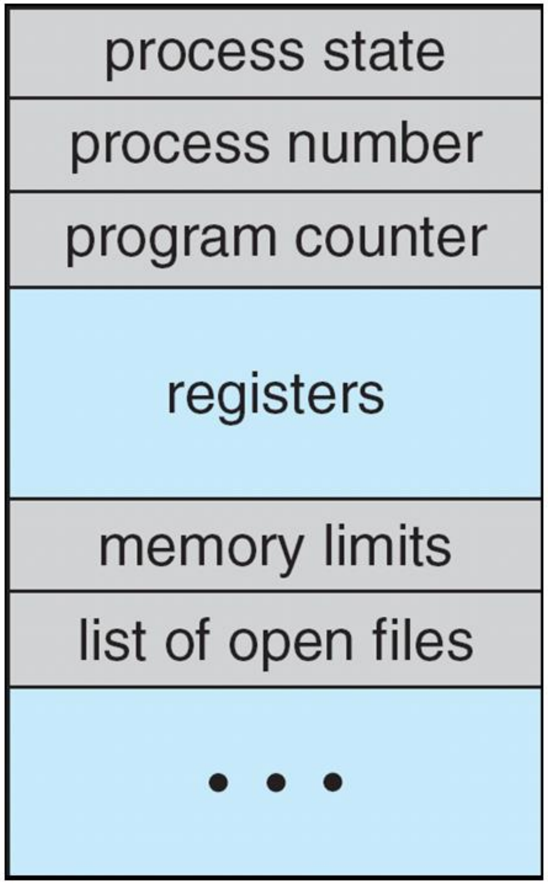
Process Control Block (PCB)

Scheduler를 만든다면 어떤 요소를 사용?

Information associated with each process

(also called **task control block (TCB)**)

- Process state – running, waiting, etc
- Program counter – location (address) of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



process state

process number

program counter

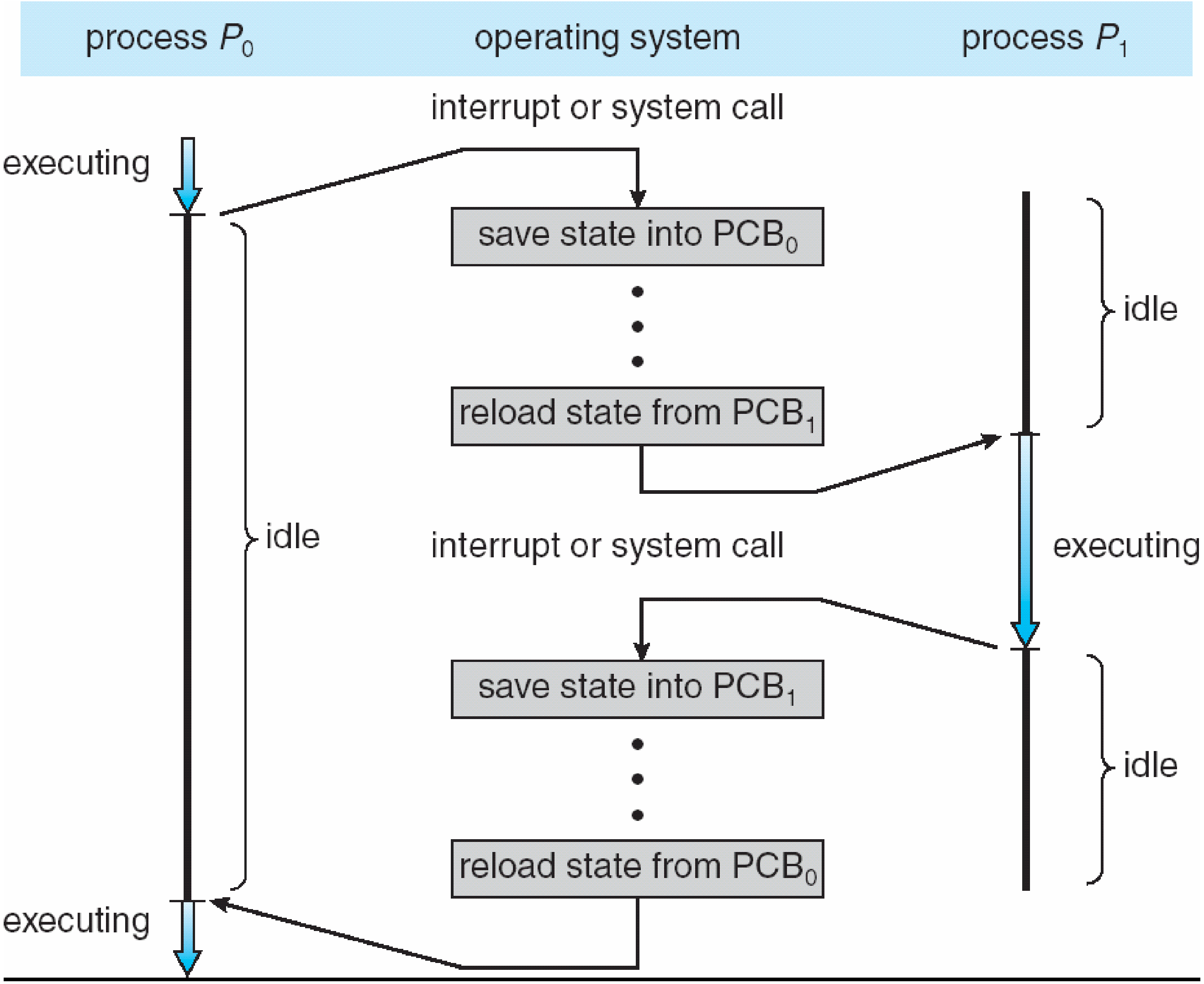
registers

memory limits

list of open files

• • •

CPU Switch From Process to Process



Threads

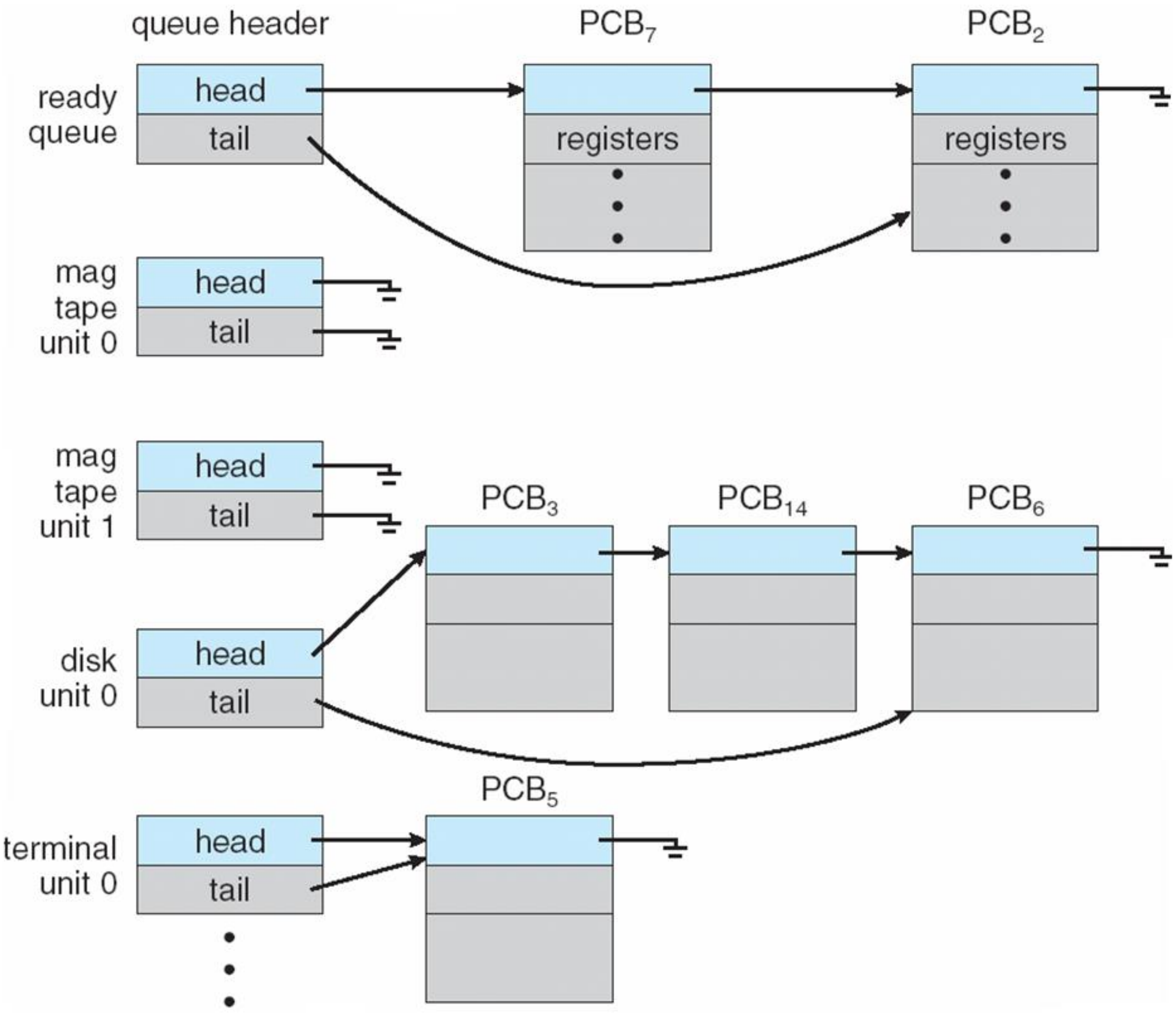
- So far, process has a single thread of execution
- Consider having multiple program counters per proce ss
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple pro gram counters in PCB

Process Scheduling

어떤 기능이 필요할지 미리 Q&A

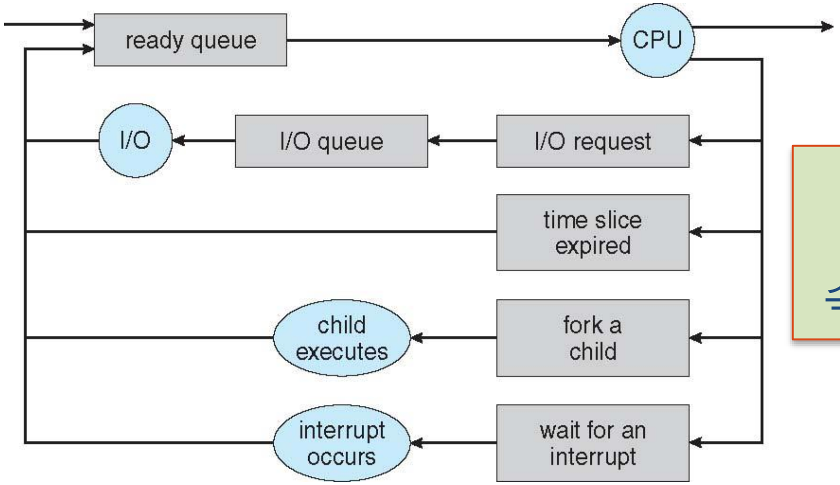
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

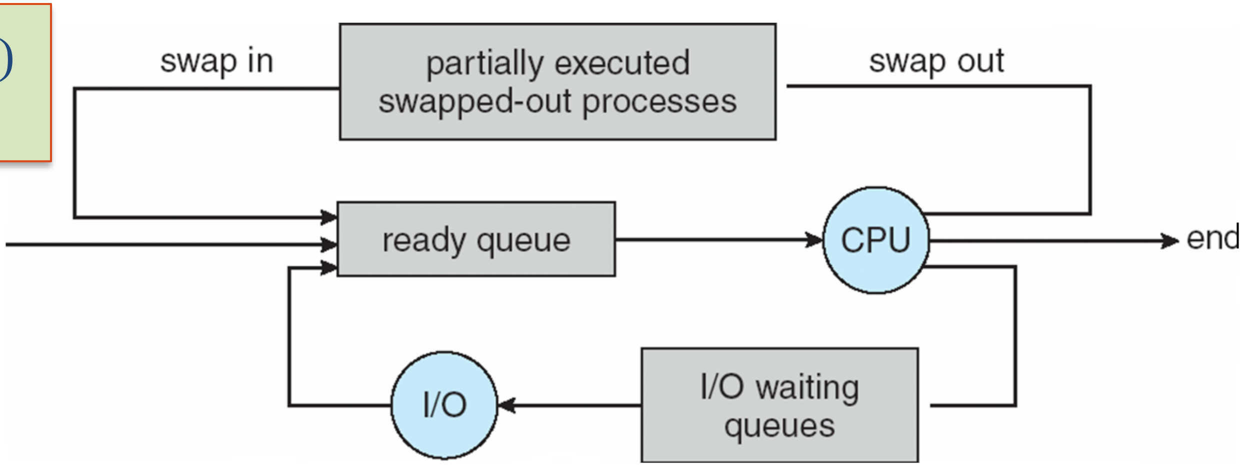
- **Queueing diagram** represents queues, resources, flows



결국, PCB 매니징
Scheduler의 처리
속도(타이머) 최저치 제시

Term Project: Process scheduling simulator implementation

Context(문맥)
Switching



Q&A

- Q1. PCB는 어디에 위치하는가? Kernel
- Q2. PCB 메모리 관리는?
  - Kernel의 Memory allocator인 Slab allocator가 Page(예: 4096 Bytes)단위로 메모리를 가져와서 PCB단위(예: 200 Bytes)로 분할하여 할당할지 말지를 판단 (자세한 것은 Googling 추천)
- Q3. PCB의 scheduling queue pointers는 구체적으로 어떤 형태?
  - Queue의 Next PCB의 주소 정도로 생각
- Q4. Multithread가 CPU 타임을 더 받는가? No
  - 효율이 높은 이유는 > Multi-cores를 활용
- Q5. Scheduler를 Double linked list로 구현하지 않고 Tree를 써도 되는가?
  - Yes, 최근 trend로 확인
  - 이 때, 프로그래밍 언어의 lib/SDK 활용은? 기본 지원 OK.

Reports

- Report #2:
  - Process와 Thread 생성 연습을 하고 이 과정을 정리하여 PDF 포맷 보고서 (3페이지 이내)로 제출 (다음수업일(금요일) 자정까지)

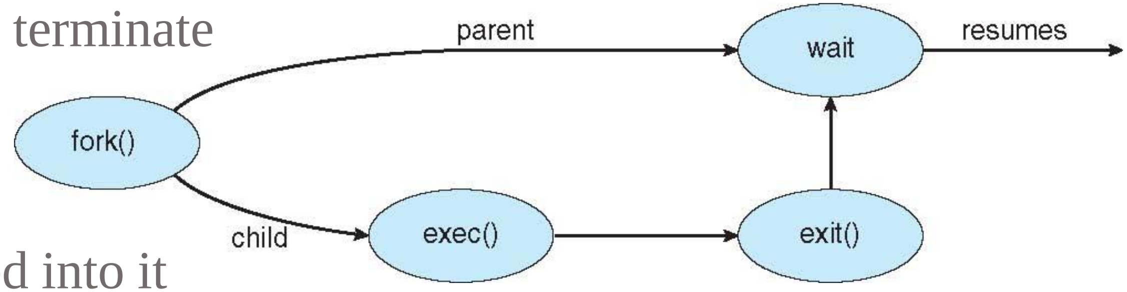
- 9월 22일(금) 수업 – 온라인 사전녹화
- Report #3: Message Passing 방식과 Shared Memory 방식 중 택1 이상으로 간단한 메시지 전송 코드 구현 및 PDF포맷 보고서 작성 (5페이지 이내) (다음수업일(금요일) 자정까지)

Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB → the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once

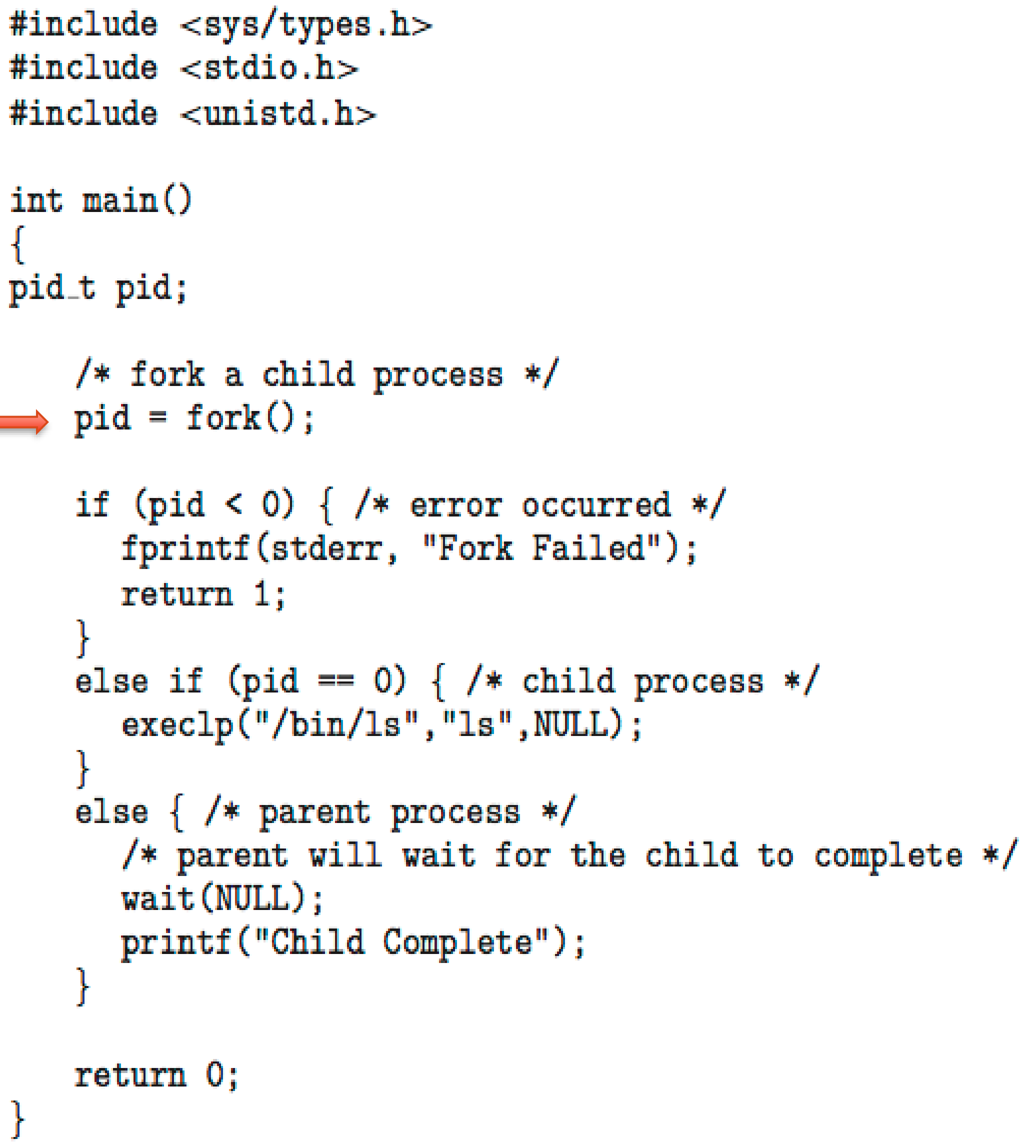
Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate



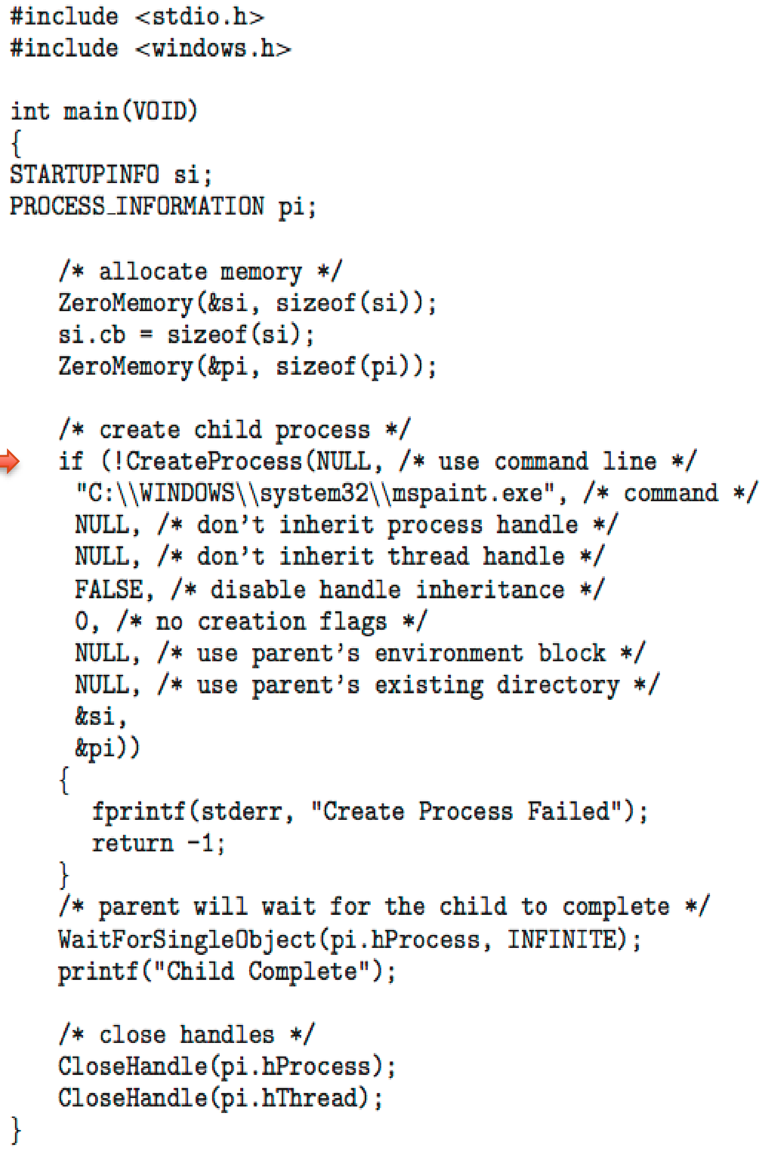
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork()** system call creates new process
  - **exec()** system call used after a **fork()** to replace the process' memory space with a new program

C Program Forking Separate Process



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

int main()
{
pid_t pid;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0) { /* child process */
      execlp("/bin/ls","ls",NULL);
   }
   else { /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
   }

   return 0;
}
```

Creating a Separate Process via Windows API



```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
      fprintf(stderr, "Create Process Failed");
      return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

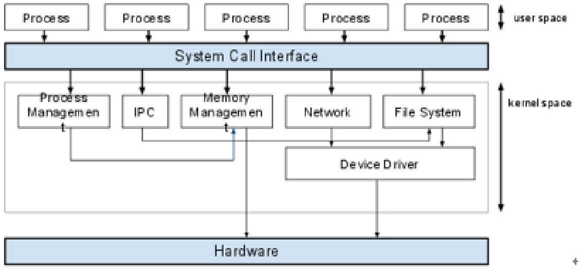
Process Termination

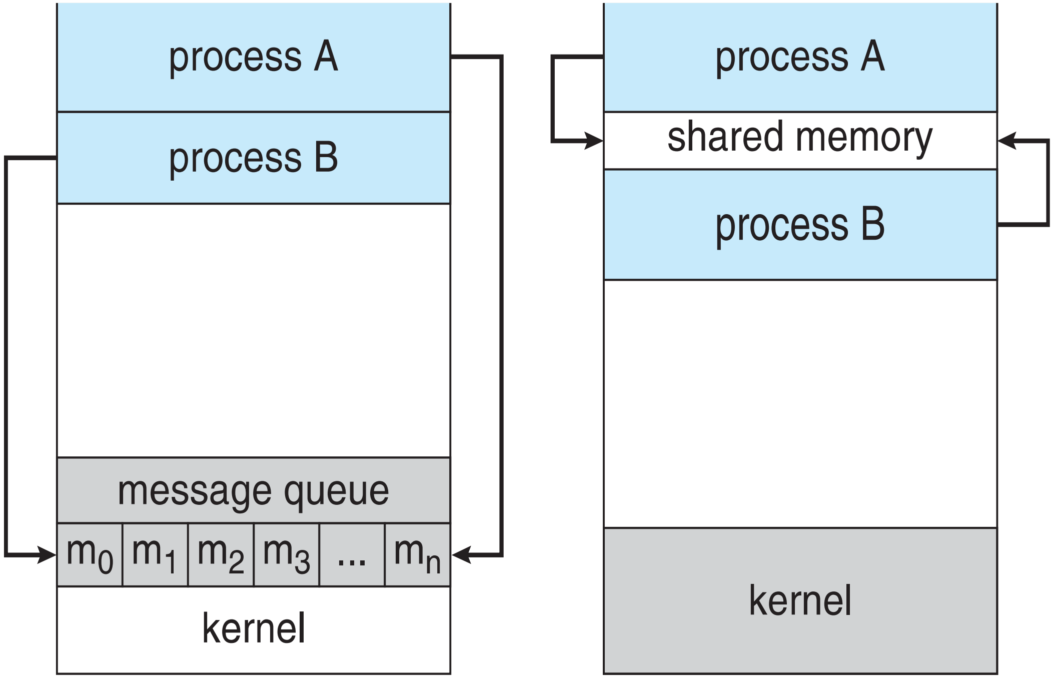
- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process′ resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates
- The parent process may wait for termination of a child process by using the **wait()**system call**.** The call returns status information and the pid of the terminated process

  pid = wait(&status);
- If no parent waiting (did not invoke **wait()**) process is a **zombie**
- If parent terminated without invoking **wait** , process is an **orphan**

Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, incl uding sharing data
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  a. **Message passing**
  b. **Shared memory**





(a)

(b)