Operating Systems

4. CPU: Process

Hyunchan, Park

http://oslab.chonbuk.ac.kr

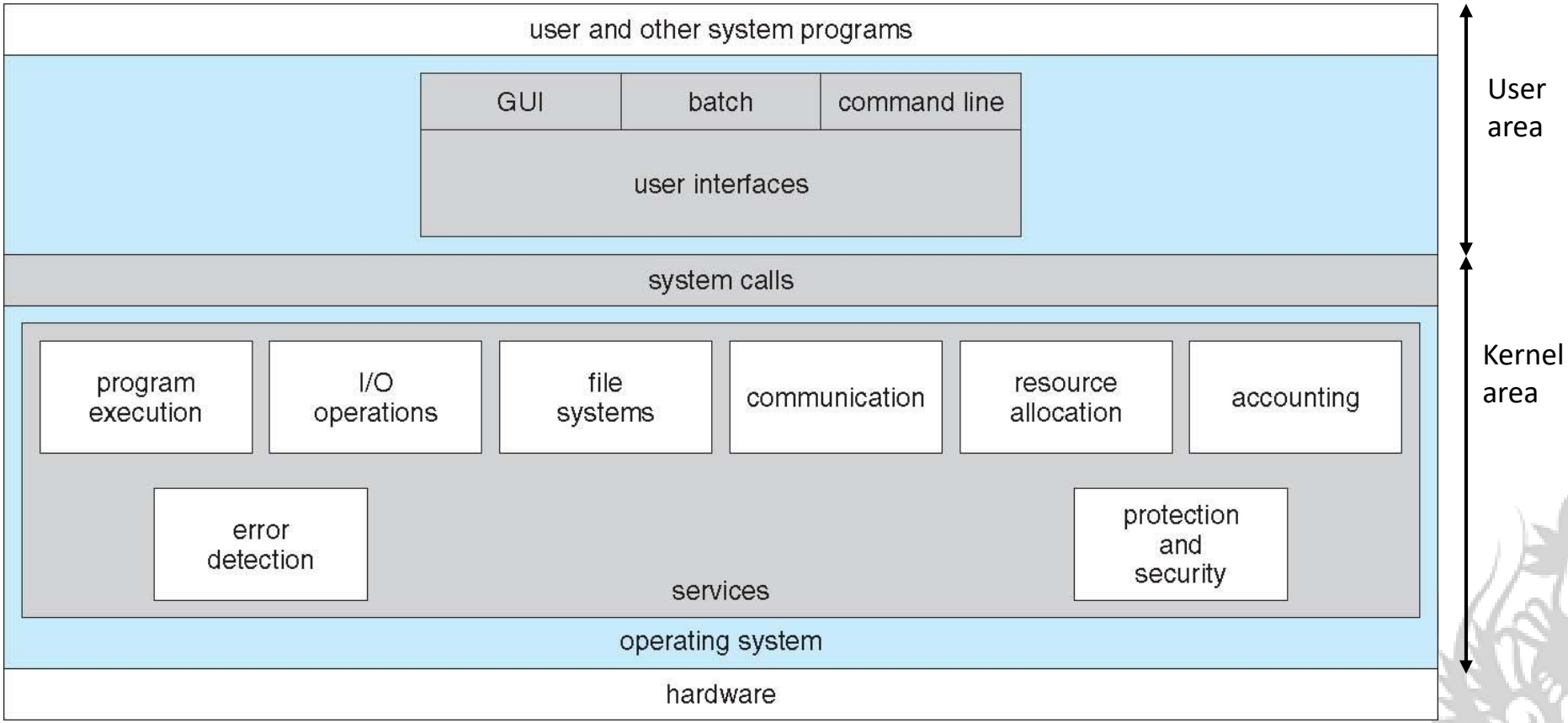
Division of Computer Science and Engineering

Chonbuk National University

Contents

- Process: concept
- Process: state diagram
- PCB: In kernel data-structure of process
- Context switching
- Creation and termination of process

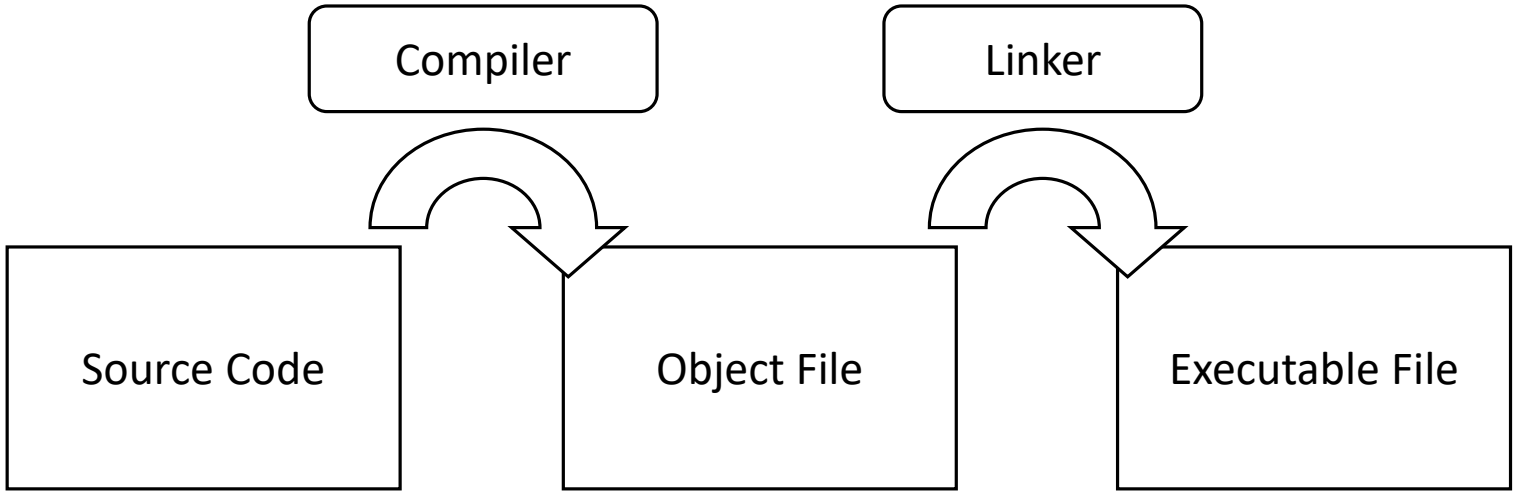
A View of Operating System Services



Scope of Operating System

- 프로세스 관리 및 스케줄링 (CPU)
- 가상 메모리 관리 (Memory)
- 파일 관리 (Secondary Storage)
- I/O 시스템 관리
- 네트워킹
- 보안

프로그램이 만들어지는 과정



프로그램이 만들어지는 과정(cont.)

- 소스코드 (.c file)
  - 프로그램이 수행하고자 하는 작업이 프로그래밍 언어로 표현
- 컴파일러 (Compiler)
  - 사람이 이해할 수 있는 프로그래밍 언어로 작성된 소스 코드를 컴퓨터(CPU)가 이해할 수 있는 기계어로 표현된 Object file로 변환
- 오브젝트 (.o file)
  - 컴퓨터가 이해할 수 있는 기계어로 구성된 파일. 자체로는 수행이 이루어지지 못함. 프로세스로 변환되기 위한 정보가 삽입되어야 함
  - Relocatable addresses (relative addresses) 로 표현
    - 심볼들의 주소가 상대적인 값으로 표현됨
    - 예) 시작 주소로부터 26바이트 지점

프로그램이 만들어지는 과정(cont.)

- 링커 (Linker, Linkage editor)
  - 관련된 여러 오브젝트 파일들과 라이브러리들을 연결하여 메모리로 로드 될 수 있는 하나의 실행파일로 작성
- 실행파일 (.exe와 같은 executable file)
  - 특정한 환경(OS)에서 수행될 수 있는 파일. 프로세스로의 변환을 위한 header, 작업 내용인 text, 필요한 데이터인 data를 포함한다
  - Absolute addresses로 표현
    - 심볼들의 주소가 절대값으로 표현됨
    - 예) 0x0100 3F0A
- 컴파일러와 링커는 결과물이 수행될 OS와 CPU에 따라 다른 형태의 파일을 만든다

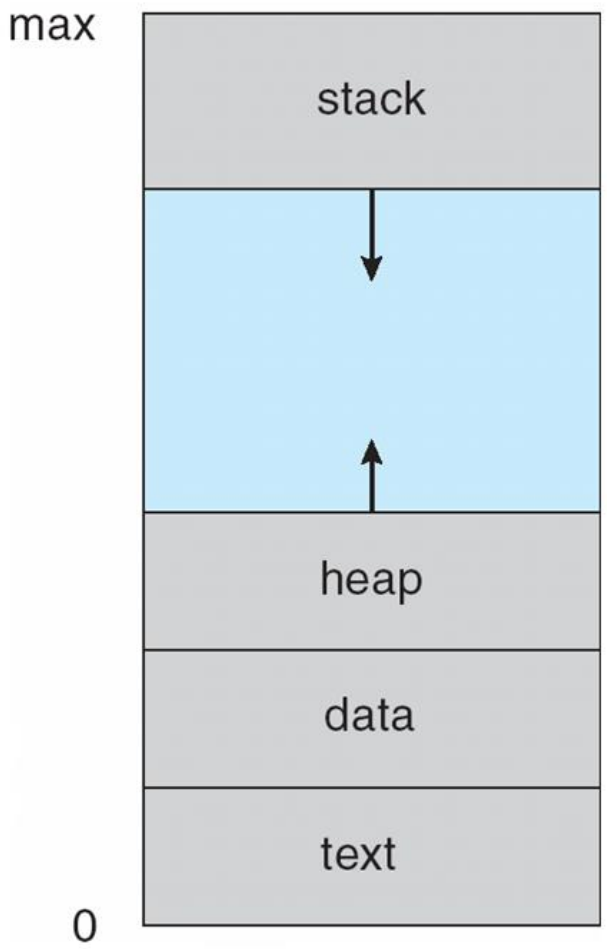
Process Concept

- 정의
  - A program in execution
  - Abstraction for CPU sharing
    - CPU를 공유하는 여러 작업 간의 구분을 위한 단위로 사용
    - 각 프로세스는 컴퓨터 시스템을 독점해서 사용하는 것으로 인식
    - 상호 간의 간섭, 침범 불가
- 구성
  - CPU 상태: registers including program counter
  - 메모리 (virtual address space)
    - Text section (프로그램 코드 저장)
    - Data section (프로그램 초기 데이터 저장)
    - Stack
    - Heap

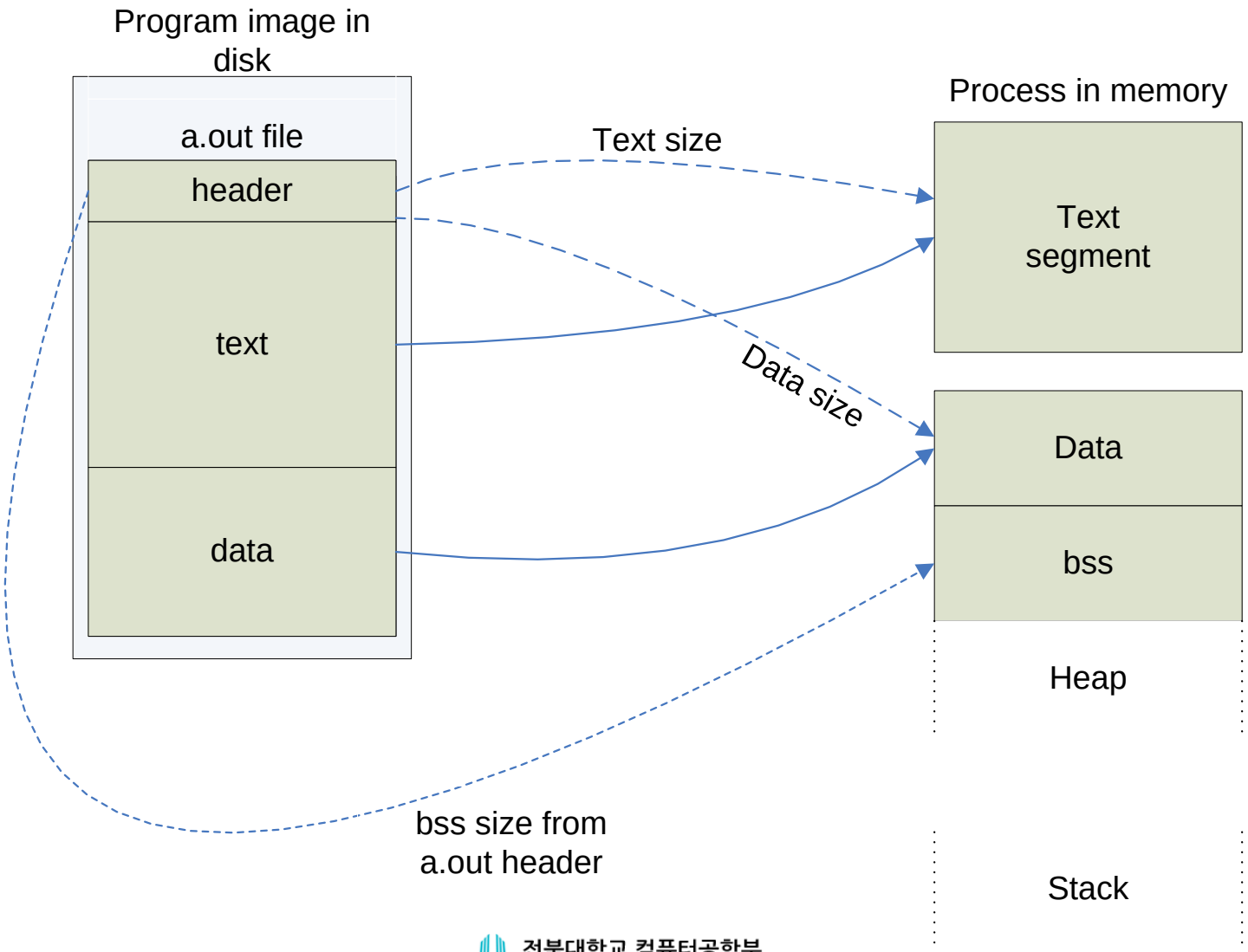
Process Concept

- 유사 용어 정리
  - Job: Batch system
  - Program: 실행되기 전의 프로그램 이미지
    - Passive state (program) vs. Active state (process)
    - 한 프로그램은 여러 프로세스로 구성될 수 있음
  - Task: 일반적 의미로 Job 과 유사.
    - CPU에서의 task: 하나의 실행 흐름. process and thread
  - Thread: Lightweight process. Process보다 가벼운 실행 흐름
  - Processor: CPU 프로세서!

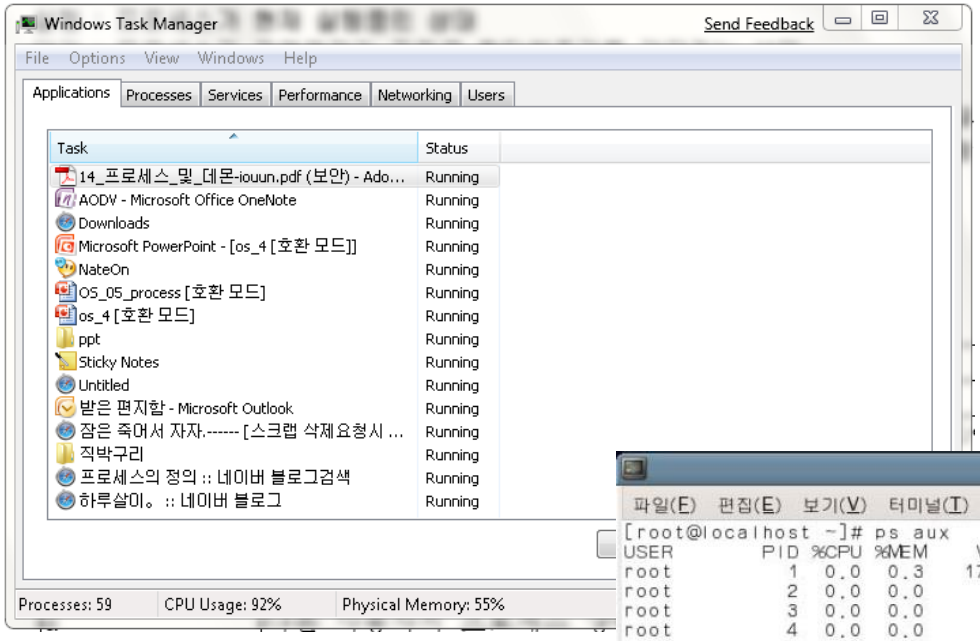
Process in memory



Process from Program

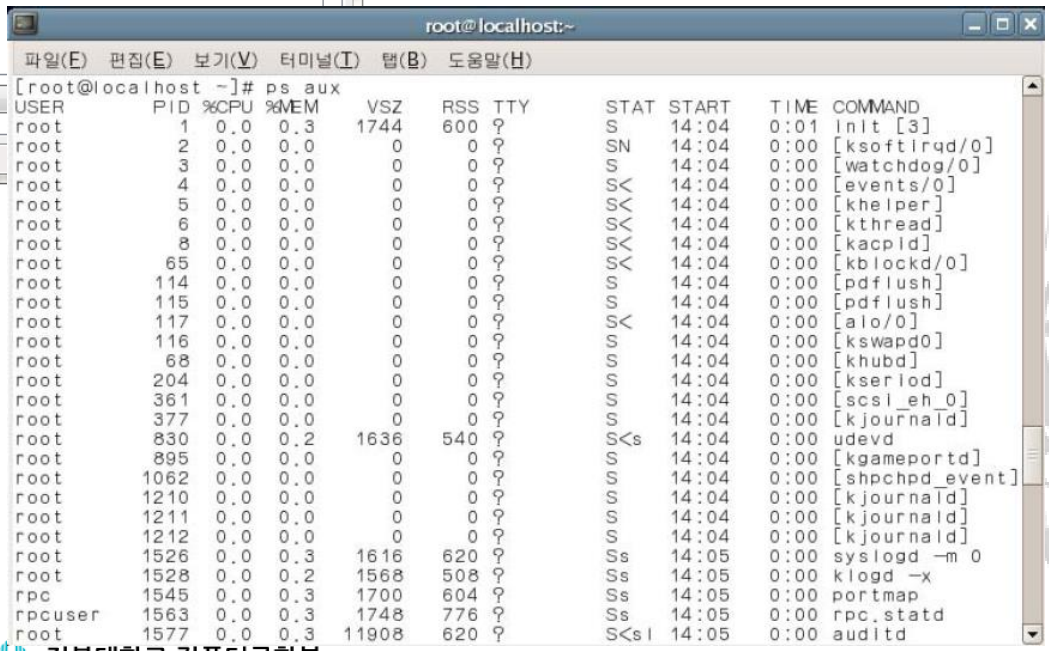


Process from Program



< Windows >

< Linux >

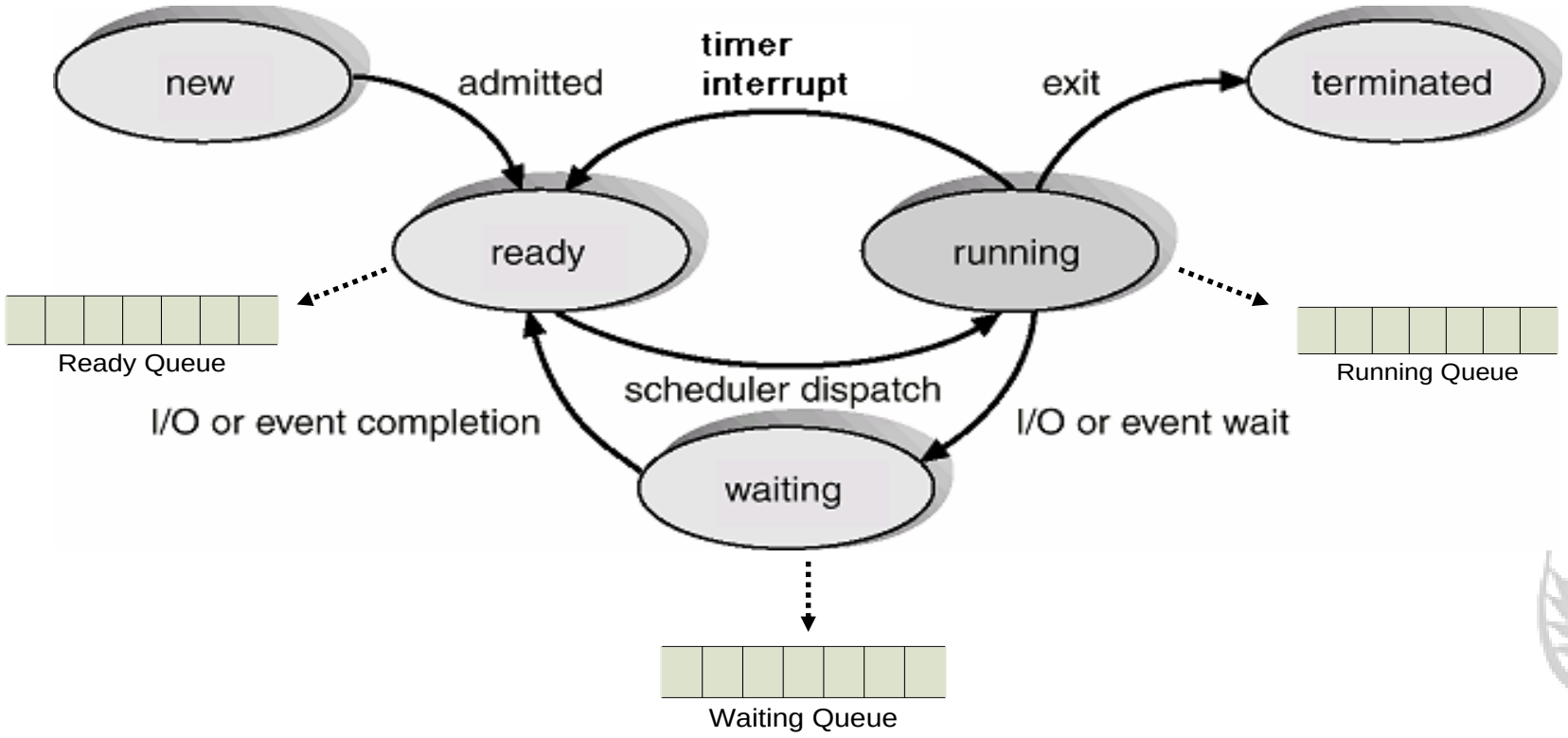


Process State

- New : The process is being created
- Running : Instruction are being executed
- Waiting : The process is waiting for some event to occur
  - (such as an I/O completion or reception of a signal)
- Ready : The process is waiting to be assigned to a processor
- Terminated: The process has finished execution

- 커널 내에 Ready queue, Waiting queue, Running queue를 두고 프로세스들을 상태에 따라 관리한다.

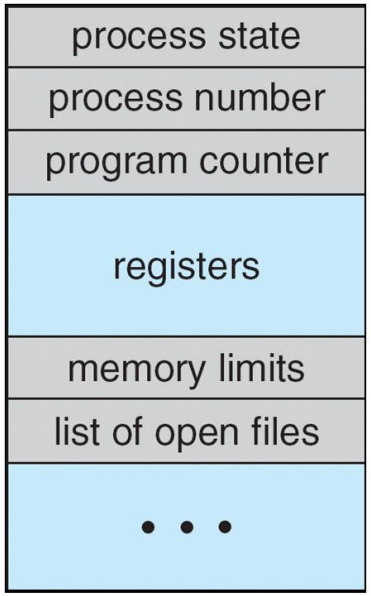
State diagram of process



Process control block

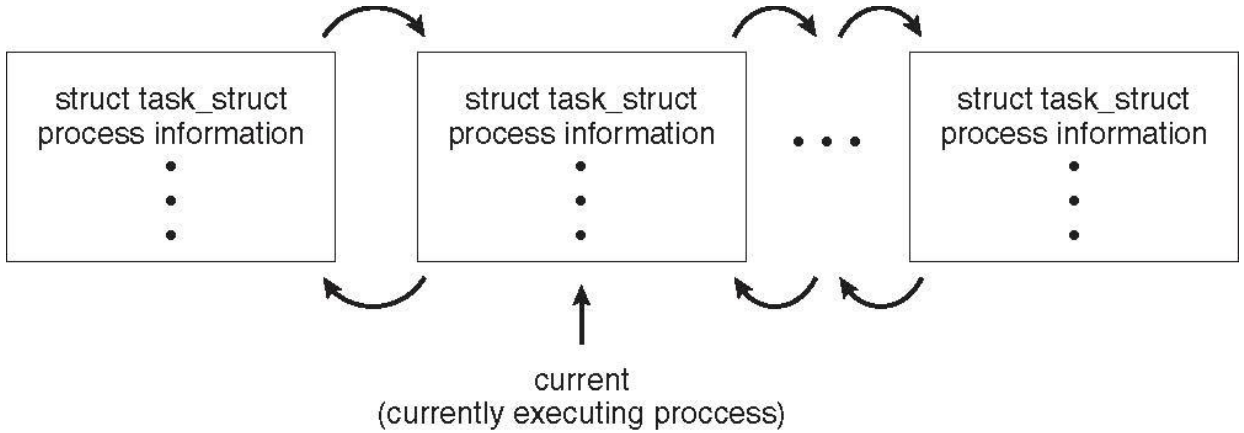
- Process control block (PCB)
  - Each process is represented in the OS by PCB.
- Information associated with each process

  (also called **task control block**)
  - Process state – running, waiting, etc
  - Program counter – location 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
  - And others



Process Representation in Linux

- Represented by the C structure task_struct
- pid t_pid; /* process identifier */
  long state; /* state of the process */
  unsigned int time_slice /* scheduling information */
  struct task_struct *parent; /* this process's parent */
  struct list_head children; /* this process's children */
  struct files_struct *files; /* list of open files */
  struct mm_struct *mm; /* address space of this process */



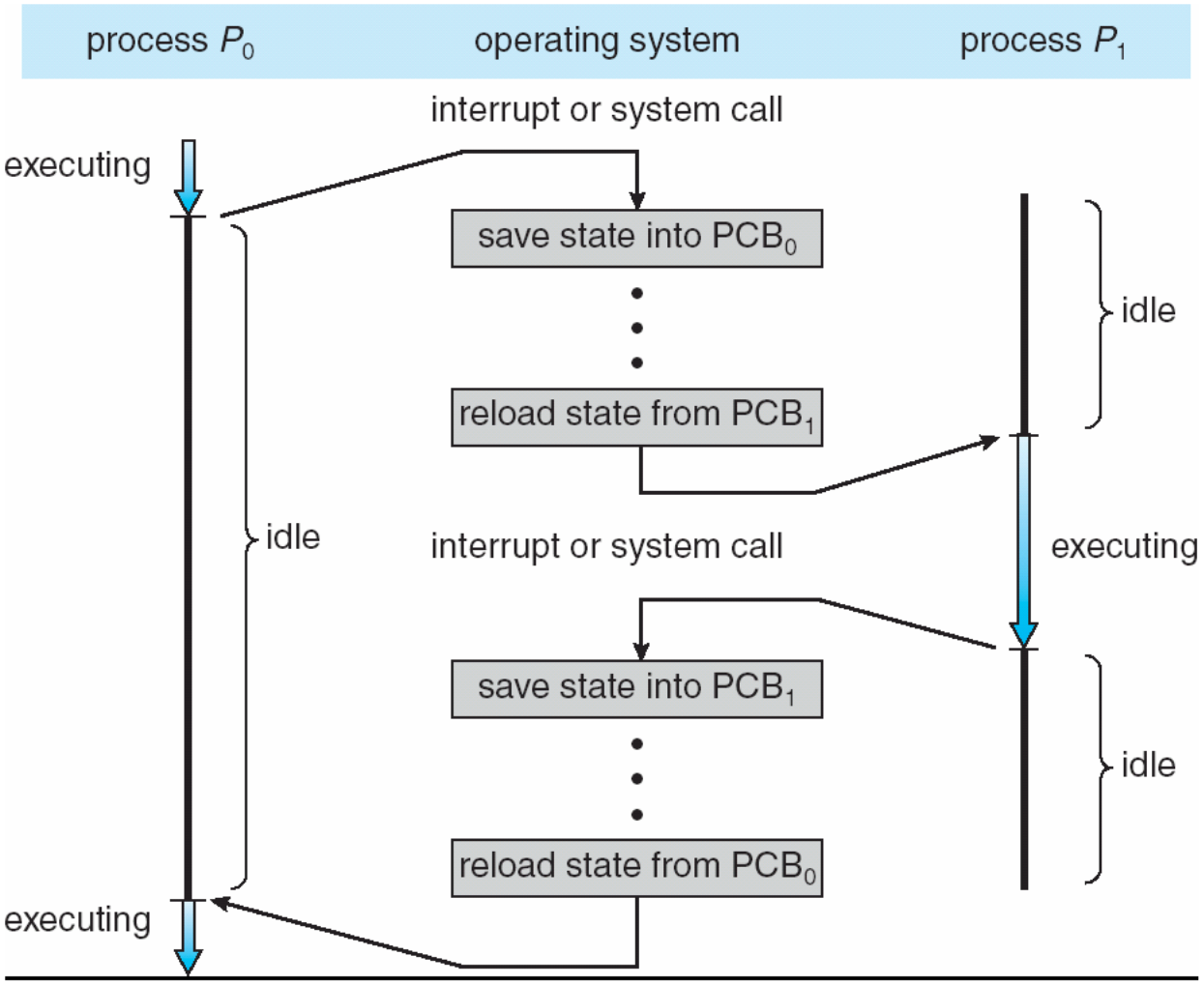
Context Switch (문맥 전환)

- 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

프로세서 구조에 따른 문맥전환의 차이

- CISC
  - 복잡한 명령어 셋 구성 -> 효율 높임, 클럭 속도 저하
  - 복잡한 회로 -> 물리적인 공간 차지 -> 레지스터 용량 저하
  - Ex) Intel Pentium processor
- RISC
  - 간단한 명령어 셋 구성 -> 클럭 속도 높임 -> 빠른 수행 속도
  - 절약된 물리적 공간에 보다 많은 레지스터 장착
    - 문맥 전환 시 레지스터 내용 변경에 보다 큰 오버헤드가 생김
  - 예) Sparc, Arm processor

Context Switch 도식



Operations on Processes

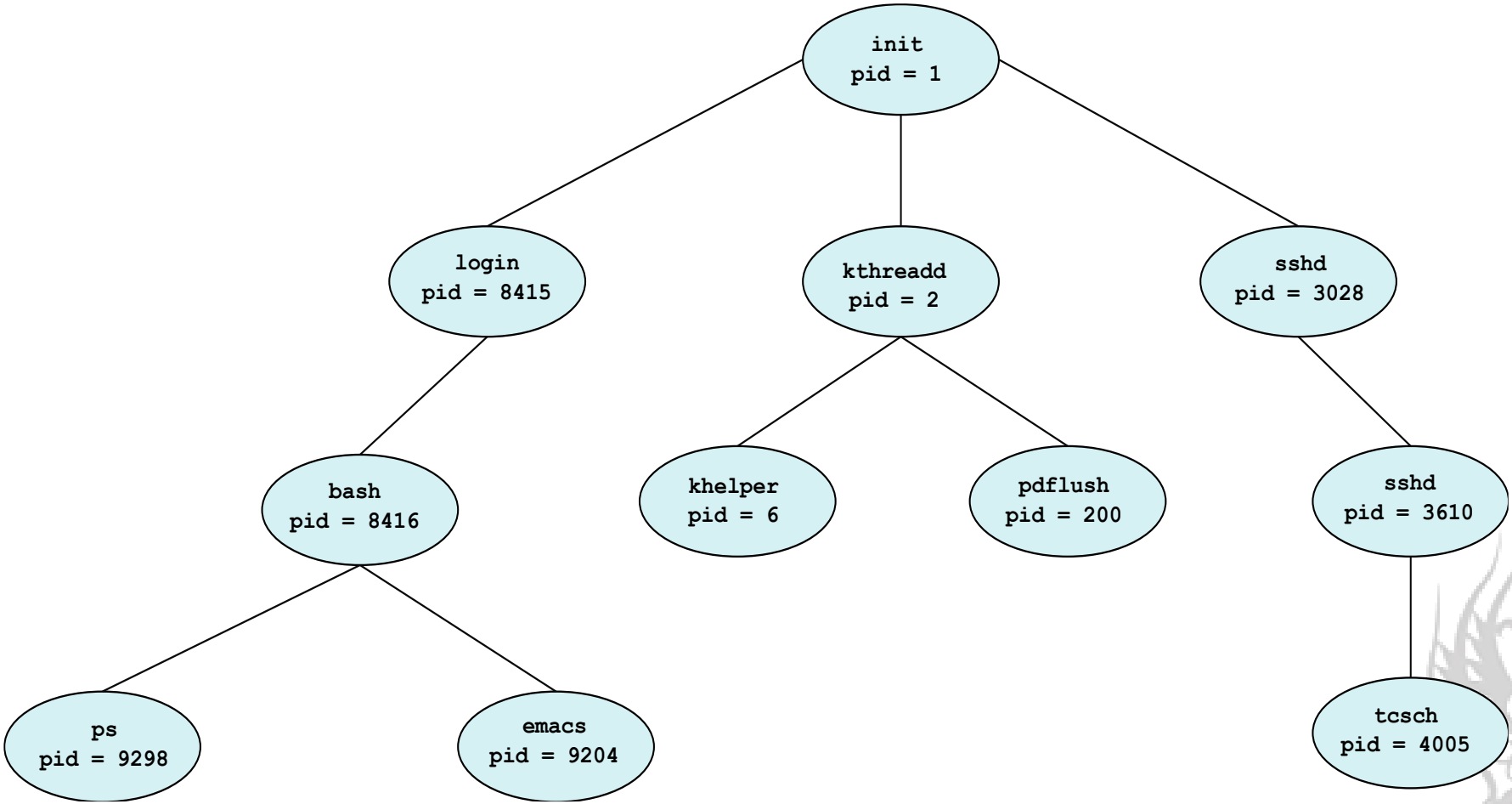
- The processes in the system
  - Can execute concurrently,
  - and they must be created and terminated dynamically.

- System must provide mechanisms for:
  - process creation
  - process termination

Process Creation

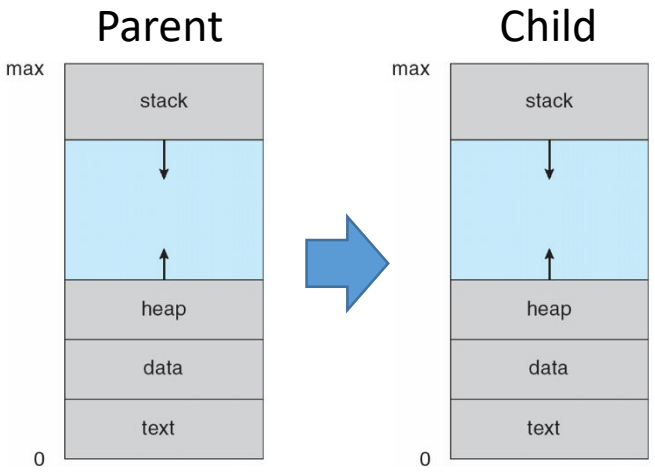
- Parent process create children processes, which, in turn create other process es, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources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

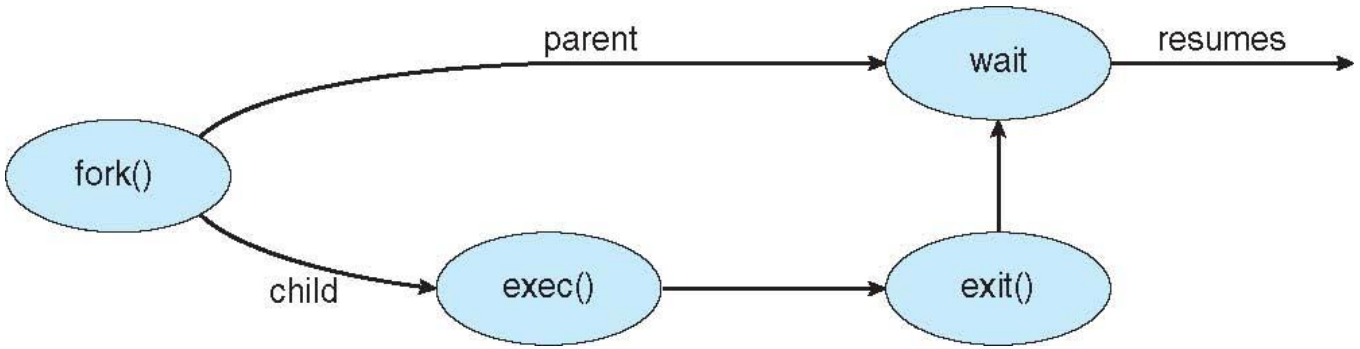
A Tree of Processes in Linux



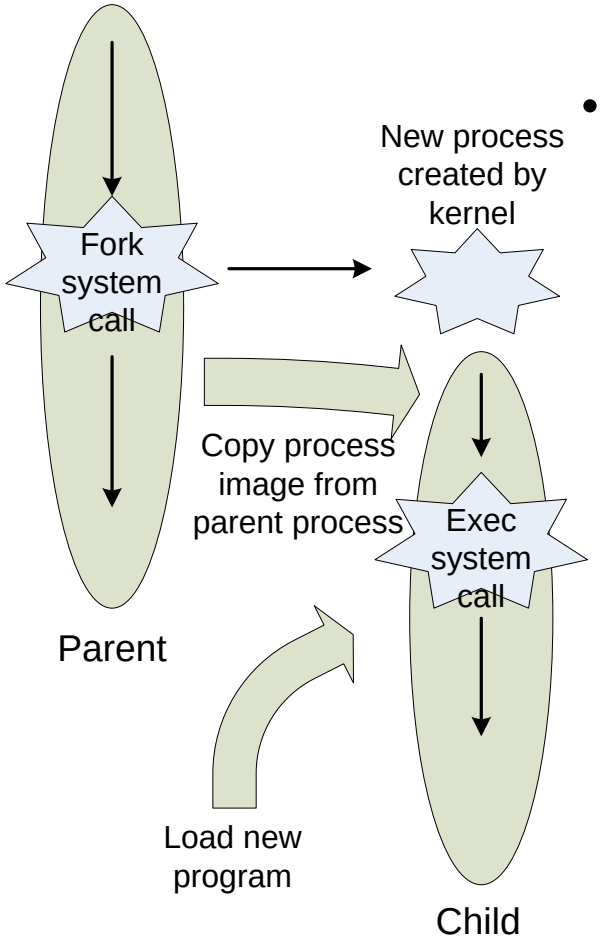
Process Creation (Cont.)

- 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





Process creation in memory view



- Memory (Text and data section)
  - Unix example
    - A new process created by *fork* system call.
    - A new process (child process) consists of a copy of the memory of the original process (parent process)
    - Exec(2): to replace the memory with a new program.

C Program Forking Separate Process

```
#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

Process Termination

- Some operating systems do not allow child to exists if its parent h as terminated. If a process terminates, then all its children must also be terminated.
  - Cascading termination. All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the wait()system call. The call returns status information an d 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