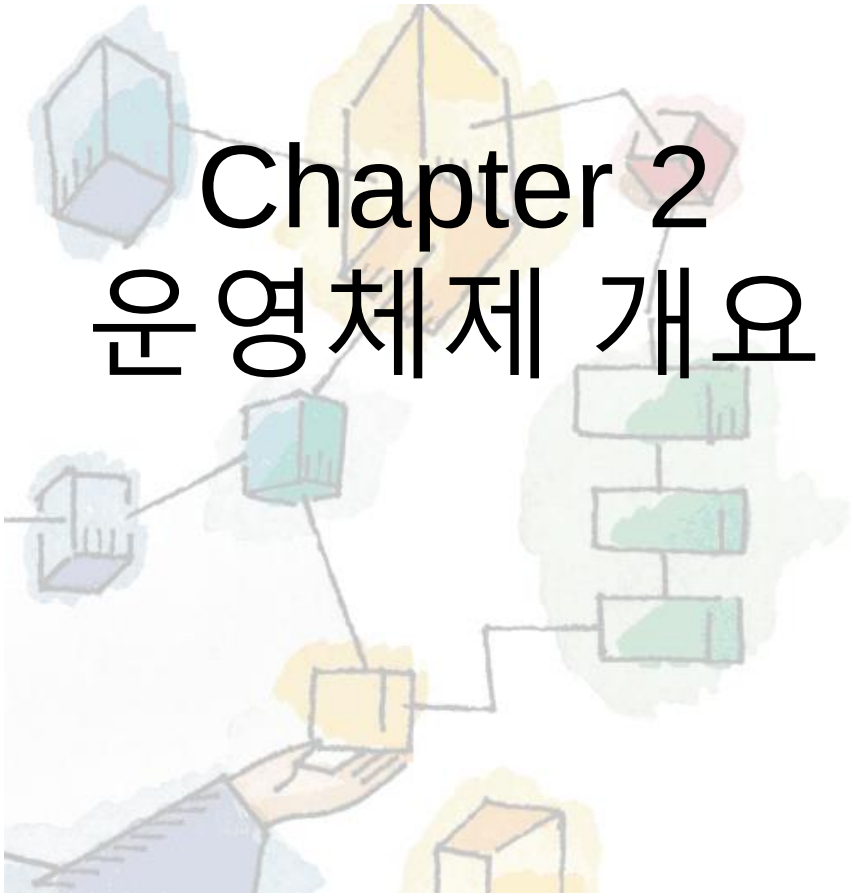
Chapter-2

운영 체제

정내훈

2023년 가을학기
게임공학과
한국공학대학교

Chapter 2
운영체제 개요



병행성

- 운영체제는 여러가지 일을 동시에 수행하는 곡예를 하고 있다.
  - 프로세스를 하나 실행하고, 다음 프로세스를 실행하고, 또 다음 프로세스를 실행하고…
- 현대의 멀티쓰레드 프로그램은 병행성 문제를 추가로 갖고 있다.

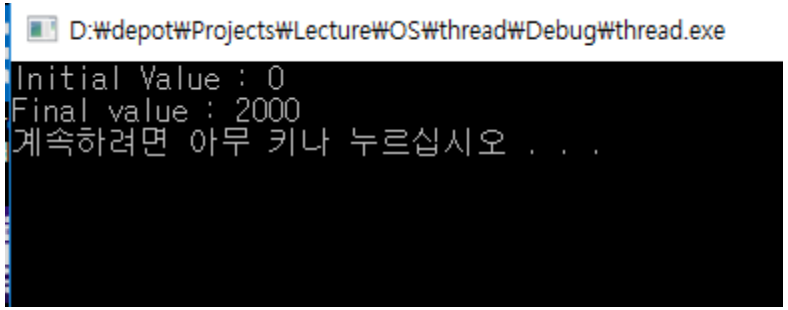
병행성

- 프로그램
  - 두개의 스레드를 만든다.
  - 스레드 : 같은 주소공간에서 동시에 실행되는 함수
  - 각 스레드는 worker()를 실행한다.
  - worker() : 카운터를 증가시킨다.

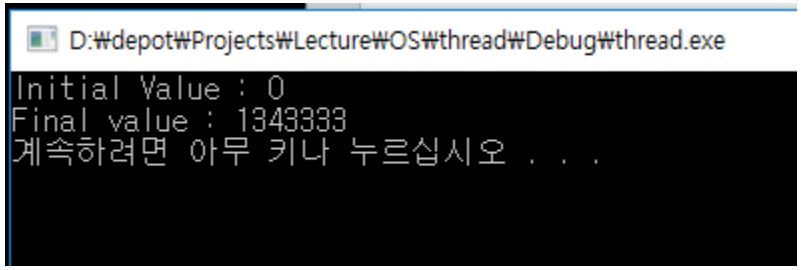
```
#include <iostream>
#include <thread>
using namespace std;

volatile int counter = 0;
int loops;

void worker()
{
    for (int i = 0; i < loops; i++) counter++;
}

int main()
{
    thread p1, p2;
    cout << "Initial Value : " << counter << endl;
    p1 = thread(worker);
    p2 = thread(worker);
    p1.join();
    p2.join();
    cout << "Final value : " << counter << endl;
}
```

병행성

- loop의 값 만큼 counter가 증가한다..
  - loop가 1000 일 경우



  - loop가 100’0000일 경우



병행성 문제

- 왜?
  - 공유 counter 증가 -> 3 개의 명령 필요
    - 메모리에서 counter값을 읽어서 레지스터로
    - 레지스터 증가
    - 레지스터의 값을 다시 counter로
  - 각각의 명령어들은 원자적(atomic)으로 수행되지 않는다 -> 병행성 문제의 발생

병행성 문제

- 앞으로 이러한 문제들을 어떻게 해결할 지 배우도록 한다.

영속성

- DRAM같은 메모리는 휘발성(volatile)이다.
- 영구적으로 데이터를 저장할 수 있는 하드웨어와 소프트웨어가 필요하다.
  - 하드웨어 : 하드디스크나 SSD같은 I/O장치
  - 소프트웨어 : 사용자가 생성한 파일들을 관리하는 파일시스템(file system).

영속성

- “hello world”를 파일에 저장하기

```
#include <fstream>
using namespace std;

int main()
{
   char mess[] = "Hello World";

   ofstream output("hw.txt", ios::out | ios::binary);
   output.write("Hello World", sizeof(mess));
   output.close();
}
```

  - …. C++는 너무 추상화 되어 있어서 운영체제의 느낌이 나지 않음…

영속성 (LINUX)

- “hello world”를 파일에 저장하기

```
1       #include <stdio.h>
2       #include <unistd.h>
3       #include <assert.h>
4       #include <fcntl.h>
5       #include <sys/types.h>
6
7       int
8       main(int argc, char *argv[])
9       {
10              int fd = open("/tmp/file", O_WRONLY | O_CREAT
                                | O_TRUNC, S_IRWXU);
11              assert(fd > -1);
12              int rc = write(fd, "hello world\n", 13);
13              assert(rc == 13);
14              close(fd);
15              return 0;
16      }
```

  - open(), write(), close()는 시스템호출로 운영체제에 있는 파일시스템에 요청을 전달한다.

영속성 (WINDOWS)

- “hello world”를 파일에 저장하기

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <share.h>
#include <sys/stat.h>

int main()
{
   int fd;
   _sopen_s(&fd, "hello_world.txt", _O_WRONLY | _O_CREAT, _SH_DENYNO, 0);
   int rc = _write(fd, "Hello worldWn", 13);
   _close(fd);
}
```

  - open(), write(), close()는 시스템호출로 운영체제에 있는 파일시스템에 요청을 전달한다.

영속성

- 디스크에 쓰기 위한 OS의 동작은?
  - 새 데이터가 들어갈 위치를 디스크에서 찾는다.
  - 저장장치에 I/O요청을 한다.
- 파일 시스템은 쓰는 도중 컴퓨터 크래시를 대비한다.
  - 저널링(Journaling), 쓰기-시-복사(copy on write)
  - 쓰기 순서 조정

설계 목표(1/2)

- **가상화** 구현
  - 시스템을 쉽고 편리하게 사용할 수 있도록 한다.
- 높은 **성능** 제공
  - 운영체제 오버헤드 최소화
  - 지나친 가상화 배제
- 응용 프로그램 간의 **보호**
  - 고립(isolation) : 프로그램의 오동작이 다른 응용프로그램이나 OS에 피해를 입히지 않도록 하는 것

설계 목표(2/2)

- 높은 신뢰성(reliability)
  - OS는 죽지 않아야 한다.
- 추가적인 목표
  - 에너지-효율성(energy-efficiency)
  - 보안(security)
  - 이동성(mobility)

역사 약간

- Main frame computer, IBM System/360, 1964

역사 약간

- 초창기 운영체제
  - 입출력 라이브러리에서 일괄 처리
  - 일괄처리 (batch)
    - 작업을 묶어서 한번 주는 것
  - 시스템 콜
    - 컴퓨터에 내장된 함수 실행
  - 하드웨어 특권 수준(hardware privilege level)
    - 사용자 모드
      - 사용자 프로그램 실행 (보호를 위한 실행 제한)
    - 커널 모드(kernel mode)
      - 컴퓨터에 내장된 OS실행 (모든 것을 컨트롤)

역사 약간

- PDP-11, 1970,
- UNIX

역사 약간

- 멀티프로그램 시대
  - 미니컴퓨터의 보급
  - 멀티프로그래밍 : 여러 프로그램을 동시에 메모리에 올려놓고 번갈아 가면서 실행
    - 목적 : CPU 이용률 증가
    - 메모리 보호(memory protection) 필요
  - 병행성 문제
    - 프로그램 사이의 공유 자원에서 발생
  - UNIX 등장
    - 시분할 시스템, shell, pipe, signal
    - 모든 현대 운영체제의 조상

역사 약간

- Personal Computer 1981,
- DOS, Windows, MAC-OS, 1981

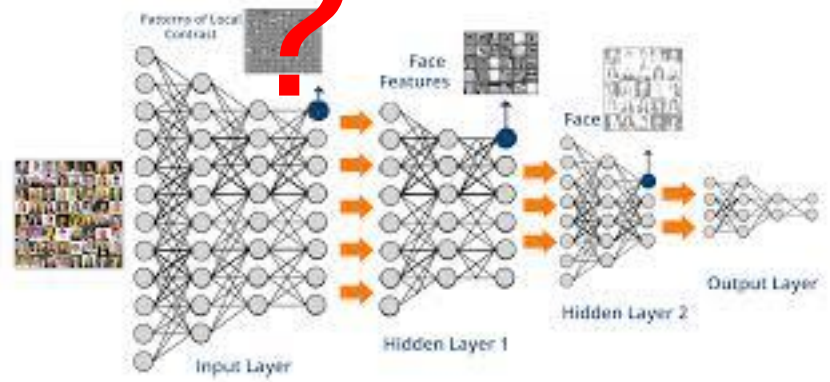
역사 약간 (1990)

- 개인에게 보급
  - 엄청난 보급율
- 운영체제의 암흑기
  - 배치 시절로 되돌아감
  - 낮은 가격, 느린 속도, 적은 용량
- 컴퓨터 그래픽의 도입
  - GUI : Graphic User Interface
  - Windows, 매킨토시

역사 약간 (2000 -)

- 모바일 운영체제의 대두
  - IOS, Android
- 64bit로의 전환
  - 메모리 공간 확장, 최신 서버 컴퓨터 6TB 메모리
- 멀티 CPU, 멀티 코어 CPU 지원
  - 96 Core CPU
- Open Source 운영체제의 대중화
  - 리눅스
- Cloud의 도입
  - 가상 머신
  - 아마존, Azure….

역사 약간


Patterns of Local Contrast
Face Features
Face
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer

숙제

- **실제 프로그램이 운영체제를 호출하는 순간을 포착하라.**
  - VS2022에서 샘플 프로그램을 실행하고, VS2022의 추적 기능을 사용해서 운영체제를 호출하는 순간의 스크린 샷을 찍어서 제출하라. 이때 RAX값을 통해 어떠한 기능을 호출하는지 살펴보자.
  - VS2022의 추적기능 사용법
    - 컴파일 모드를 x64비트, Release 모드로 변경한다.
      - Debug 모드는 추적할 분량이 많다.
    - C++ 프로그램에서 운영체제를 호출하는 문장에 break point를 건다.
    - 실행 시켜서 break point에서 멈추게 한다.
    - ALT+8을 눌러서 Disassembly창을 연다.
    - F11을 눌러서 명령어를 하나씩 실행시키면서 추적한다.
    - SYSCALL 명령이 나오면 스크린 샷을 찍고, RAX레지스터의 값을 기록한다.
      - SYSCALL명령어 실행 전의 RAX값을 기록.
      - RAX레지스터의 값은 ALT+5 키를 눌러서 확인한다.
      - RAX에는 운영체제에게 요청하는 명령어가 들어있다.
    - `while(true);`를 만나면 멈춘다.

숙제

제출

- 스크린샷, RAX의 값, 호출하는 운영체제의 기능 ID
  - http://j00ru.vexillium.org/ntapi_64/ 에 RAX값에 따른 기능들의 ID가 적혀 있음
  - 하나의 C 함수에서 운영체제 호출을 여러 번 할 수 있다. [모두 기록]
- 제출 : Eclass
- 샘플 프로그램

```
#include <stdio.h>

int main()
{
   printf( "Hello worldWn" );
   while(true);
}
```

학번 끝 : 0, 4, 7, 9

```
#include <stdio.h>
using namespace std;

int main()
{
   int a;
   scanf_s( "%d" , &a);
   while(true);
}
```

학번 끝 : 2, 3, 6

```
#include <malloc.h>

int main()
{
char *a = reinterpret_cast<char *>(malloc(1024 * 1024));
while(true);
}
```

학번 끝 : 1, 5, 8

숙제

- **정보**
  - 0008 NtWriteFile
  - 0006 NtReadFile
  - 0007 NtDeviceIoControlFile
  - 0018 NtAllocateVirtualMemory