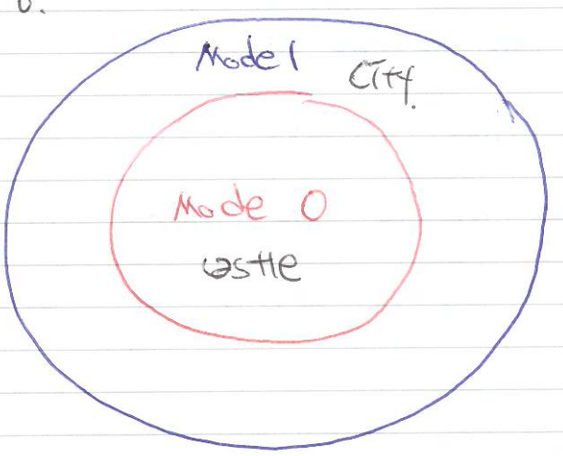
Lec 6.



Kernel Mode
Can modify page table : allocate frames to process, handle page fault
Mode 0 = full access to the hardware
there are some security

Mode 1 = limited things to do with computer.
It can't directly use hardware only
Can access memory via page table.

System call =
Interrupt =
Trap =

User Process

user process executing → calls system call → return from system call

Kernel: trap, mode bit = 0 → execute system call → return. → mode bit = 1

System call parameters (value of read from, info you want to store)
How many bytes you want to read.

1)

2)

3)

① Resisters — but if there are more parameters than registers

② Memory — to use member variables

③ Stack — push them on to the stack.

④ Can mix and match — e.g) Resister + stack.

Strace — cat hello.~~txt~~ txt (1)

mmap() = It is unix-system call that maps file or devices ~~memory~~ in to memory

logical address (code, Data, stack)

- code: 읽기전용으로 프로그램 명령어 포함
- Data: 읽기, 쓰기 가능
- stack: 읽기, 쓰기 가능

* malloc() 으로 확장가능.

MMap() = 메모리 특정 영역을 파일로 나누어 주는 역할을 한다. 파일은 시스템 전역적인 객체 이므로 다른 프로세스에서 접근 가능하다.

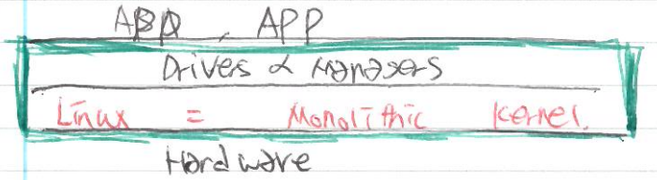
활용 = ① 프로세스간 데이터 교환

② 비용을 줄일 수 있다. e.g. read, write 와 같은 함수를 사용하지 않기 때문에 성능이 향상된다.

I/O Interlock. : file 복사에 쓰는 page는 page replacement algorithm 이 돌아가는 동안 선택되지 ~~않아야 한다~~ 않게 lock 되어 있어야 한다.

DMA : Direct memory Access

Ct : round up to next page for consqutive pages to I/O interlock //



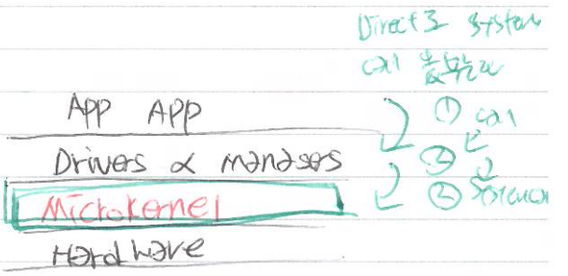
반드시 필요한 기능들을

Monolithic kernel : 운영체제의 ~~핵심적인 기능~~ 커널과 동일한 메모리 공간에 적재, 실행하는 기법.

Drivers, Process scheduler, Page fault handler live inside of the kernel and interface ~~them~~ it making by ~~the~~ ~~Page~~ system call

Everything learning with O

Microkernel



① Resisters

③

④

Stra

MMap(

logical

읽기전용으로
열려서 ㅍ

MMap() = on
파
가능

활용 = ① 프로세스간
② 비용을
않기

Interlock : file
이 줄
되어

Stack programming

```
int main() {
    get password();              // 1
    if ( isRoot) {               // 2
        givethemsuperpowers();   // 3
    }                            // 4
    return 0;                    // 5.
}
```

Stack.

1:03:50초 쯤에 위치

buffer overflow attack = gets() this don't check limit literally reads smt.

Stack은 여러가지 중요정보를 관리 하지만 그 중 return address를 가지고 있다. 공격자가 return address를 위조해서 위조한 return address로 돌아가서 계속 실행된다.

Stack Protection.

아이디가 카나리를 데리고 다닌다. 근데 카나리는 공기 예민해서 금방 죽는다.

Canary word : before the return address, after a method
스택에서 리턴 주소 앞에 특정한 값을 삽입해서 이 값의 변경 여부를 이용해 버퍼 오버플로우의 발생을 알아내는 법 프로그램 그냥 종료 시키기!

단점 : 리턴주소 이외의 값을 이용하는 공격은 방어가 어려운 단점

Other protection.

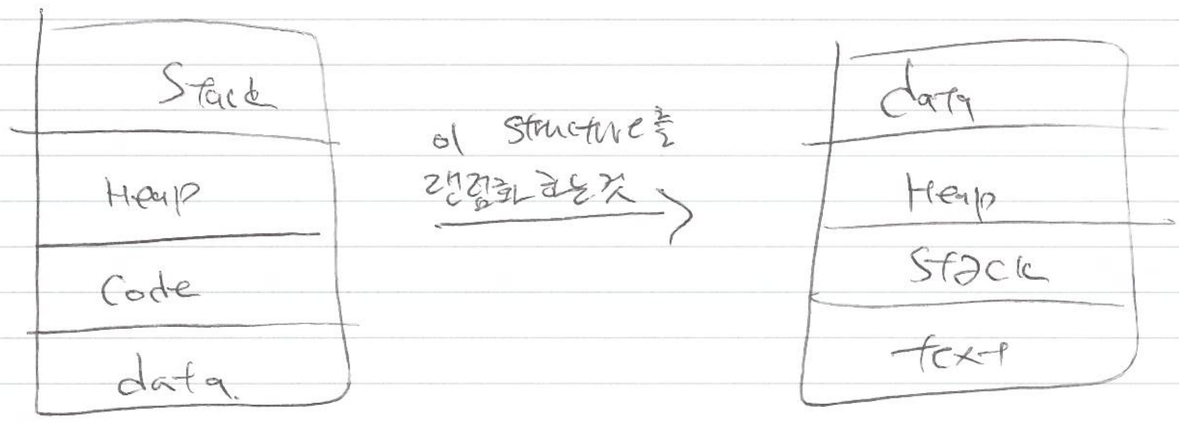
Librarylood-order - randomisation.

=> Libraries loaded in a random order.
Limitations -> Sometimes it's work.
라이브러리가 적으면 확률이 높다 때려 맞추기, Probability
Let's say ABC in library

순서 경우. abc
acb
bac

cat 명령어 사용

Address space layout randomization.



e.g)

- Every time a process is loaded randomise the memory address of the program code, shared libraries, data