

Lec-1


Program → Process

- ① Compilation stages
- ② Loading the executable as a process

Linux

```
int main()  
{  
    printf(" ");  
}
```

\$> gcc main.c



a.out

\$> ./a.out

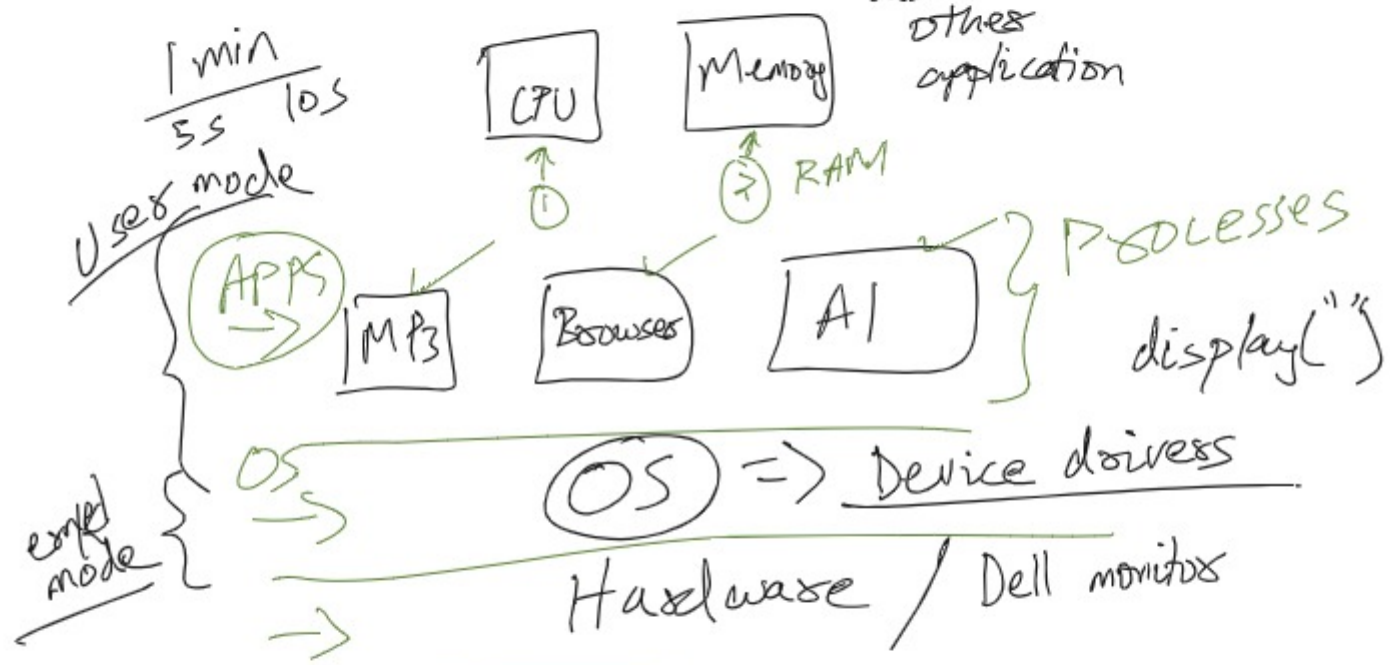
"Message"

OS Overview

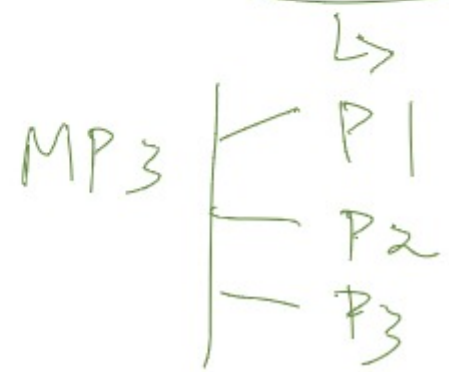
Operating system \Rightarrow mediator

$\left\{ \begin{array}{l} \hookrightarrow \text{music player} \\ \hookrightarrow \text{browsers} \\ \hookrightarrow \text{AI/ML} \end{array} \right. \Rightarrow \text{CPU, memory}$
 $\Rightarrow \text{GPU}$

\hookrightarrow application malware
none of the applications disturb the other application



> Processes X:
./a.out



OS

Process Block

MP3

- ↳ how much memory
- ↳ cpu usage
- ↳ scheduling

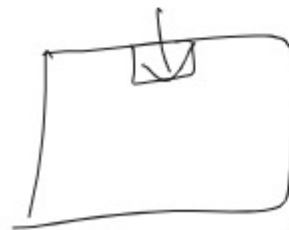
cannot be directly accessed by the application/process

Linux - 300 / read
system calls



Attacker \Rightarrow OS

Traps



$i = i + 1$
print(i)
i/o

Traps segmentation fault
to H/W

Virtual Memory

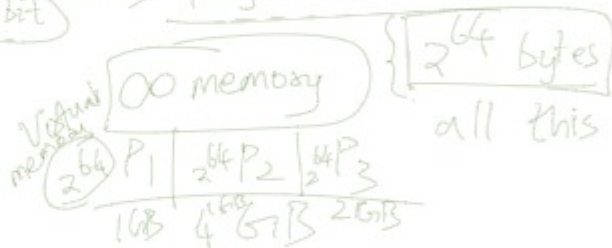
```
int main()
{
    int a;
    int *p;
}
```

Entire memory

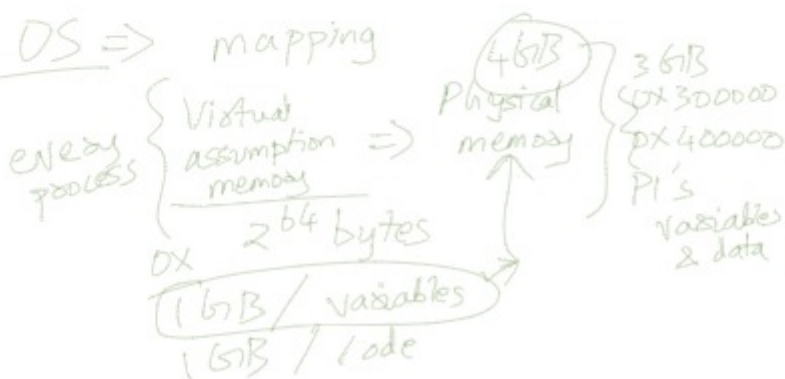
not 4 GB

64-bit

64 combinations 2^{64} Bytes of memory
 → addrs. (64 bit) ⇒ 1 byte of value



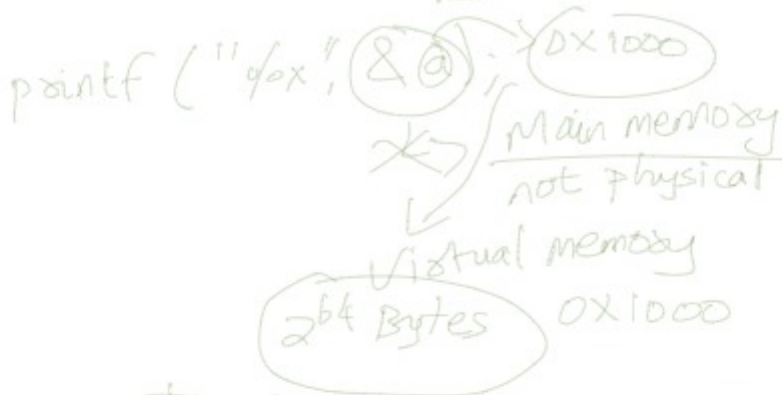
OS ⇒ mapping



① split memory

② map memory,

determine where data/code stored in each main memory



① Process does not get actual control of physical main memory

OS/H/W
 Paging Page tables



Lec 2

- ① compilation process
- ② binary executable, how is it executed

Linux

A diagram illustrating the compilation and execution process in a Linux terminal. It is enclosed in a large oval. At the top, the prompt '\$>' is followed by the command 'gcc' and the file 'main.c' (circled). An arrow points from 'main.c' to 'a.out' (circled). Below this, another prompt '\$>' is followed by the command './a.out' (circled). Underneath './a.out', the string '"Message"' is written.

```
$> gcc main.c  
      ↘  
      a.out  
  
$> ./a.out  
      "Message"
```

```
main()  
{  
    printf(" ");  
    fun1(); fun2();  
}
```

① Pre-processing

main.c int main #include <stdio.h> usr lib
{ printf(" ");
 ("Hello world %d", x);
}

② Compilation

5 arg, x, --
int dd (int a, (int) b)
 ① ②

③ Assembler

④ Linking

⑤ Loading

gcc
main.c

./a.out =>

② Compilation

src code

↳ compiler → assembly
file



Intel
is,
initial
com

Instruction Set Architecture

standard set of features/
instruction

+ con

that H/w is guaranteed
to support

64-
X86 ISA

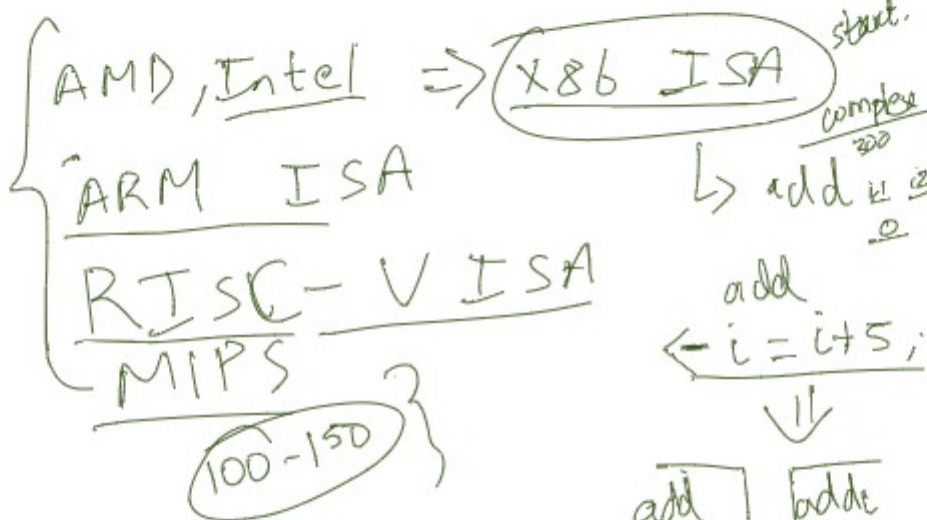
32-bit ISA

compilers engineers

$i = i + 5$

300
instructions

add, dv, a



① converts C code \Rightarrow X86 ISA
human
readable
form

② places each part of code,
data + g within its own
section

III

Assembly phase

C code $\xRightarrow{\text{compilation}}$ Human readable ISA format $\xrightarrow{\text{Assembler}}$ Machine readable ISA

main

Object Files

ELF \Rightarrow Executable & Linker Format
X
PE

Linking Phase

\Rightarrow standard

printf()

glibc

object file

\hookrightarrow Linker

resolves
all dependencies

printf()

a.out

a.out
executable