CS347 Lecture 7.

the system call mechanism

- abstraction for OS services.
- the system call interface
  list/set of all f^n's / entry points to invoke service
- fork, exec, wait, exit, read, write, open, close, lseek
  sleep getpid getppid pipe dup ~~printf~~

Homework

1. Lab 2
2. mycat.c
   mycat2.c
   mycat3.c
   which executes as a child process of mycat2 & mycat2 waits till mycat3 is done.

   mycat4.c (extend mycat3) which writes output to a file instead of to terminal.
3. Write a program (P1)
   - that overwrites itself. w/ program (P2).
   - Does anything after exec execute within the calling program?

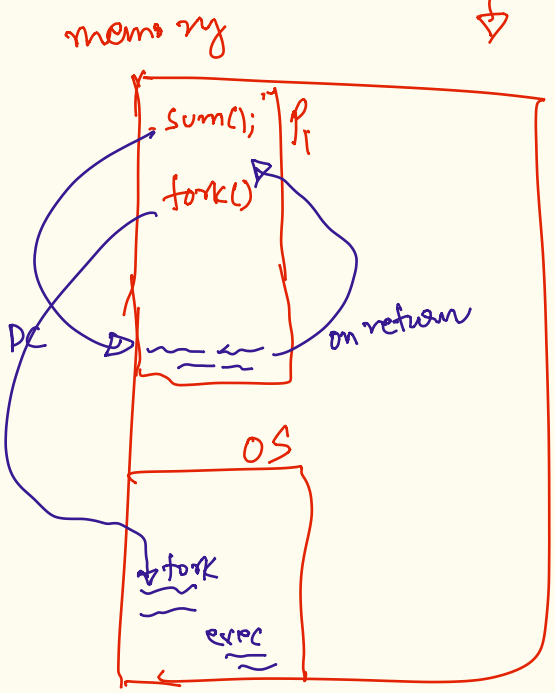
API vs ABI

(*) application programming interface
- source code level interaction

ABI
C appl^n. binary interface
~ runtime-used interface
~ interaction between compiled/binary programs

P1:

```
int a=23;
int b=57;
sum(a,b);
fork();
.
.
.
int sum (int a, int b)
    return (a+b);
```



(#) Qs for the system call mechanism

design

~ how to invoke a system call ? x86 instruction

~ mechanism to switch privilege levels? jmp <memory addr>

~ mechanism to save & restore context
- CPU state
- return addr.
- return value

└ PC is set to mem. addr.

~ —"— to handle/pass arguments

~ —"— to specify which system call & use to identify loc^n in OS program.

~ .
.
.

(#) how to invoke a system call?

- explicit interrupt
  - interrupts
    - hardware → external IO
    - software
      - implicit → div by zero, null ptr access
      - explicit → system call (execute an instruction explicitly)

x86 ISA

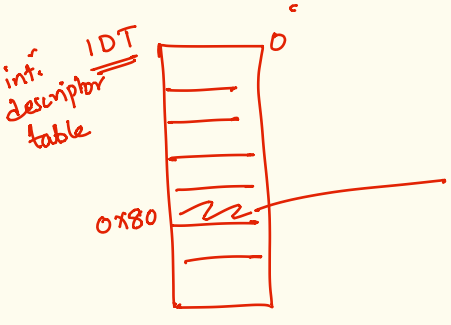
(*) `int 0x80`
- `int` — instruction to generate an interrupt
- `0x80` — interrupt vector

- switch CPU to highest PL
- saves context of running process (all regs. of CPU) → temporary buffer / Kernel stack
- switches to Kernel stack
- jumps to interrupt handler function

(*) `iret` — return from interrupt | instruction of the ISA

- restores context of CPU regs.
- switches to user stack
- changes PL to user-mode
- jmps to user-space return address

(#) interrupt handler / isr — interrupt service routine



IDT location is stored in a CPU register

IDTR ~ register storing address of IDT.

address of the function to handle interrupt 0x80