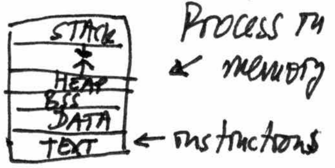
COMSW4118-5-1 9/19/2017

- Last time - talked about computer hardware, The hardware view:
- CPU runs instructions one after another, in b/n instructions checks for interrupts and, if found, executes them immediately
- OS runs programs as processes (programs in execution):
- Interrupt handler handles interrupts = code to handle irq's.
  ← part of the OS!



- Thus, normally the OS is doing nothing and user processes are running. The OS gets involved once an interrupt goes off. OS is interrupt (event) driven.

- What does it mean to run an interrupt? It is a special thing that runs in the context of the current process by interrupting it.
  Thus, the OS does not "run" but gets triggered by interrupts (events).
- OS code is on disk, when it is booted, the hardware knows to find it and load it into memory – this is the bootstrap process that also creates the first process in the process tree.
- As part of the bootstrap process, the interrupt handler code is loaded by the OS and registered in hardware via an:

  Interrupt Table (Interrupt Descriptor Table / IDT)

- Which is pointed to by an IDTRegister ← OS saves the IDT address there.
- IDT has pointers from interrupt number to the handler code. for it
- The Programmable Interrupt Controller knows the interrupt number and dispatches it to the appropriate handler code for this number.
- Thus, the OS implements the handler code, loads it, sets up the IDT and points the IDT register to it.
- The PIC (Progr. Inter. Contr.) is a chip, i.e. hardware. It communicates with the hardware (mouse, keyboard, etc.) and generates a number of pre-defined interrupts for each hardware event – this is specified by the computer architecture.
- The interrupt handler runs in kernel mode, so a context switch is made, if necessary, when handling the interrupt.

- The service view perspective is:
  How does a process do a system call like fork() or exec()?
- A software interrupt is generated by a CPU instruction that causes an interrupt. It goes through the same IDT.
- This is the way to issue a system call – it is always via a software interrupt.
- To identify which system call to invoke, the process that generates the software interrupt provides the system call number which the software interrupt handler uses to look into a system call table to determine which function to call.

- The process passes arguments to system calls in a pre-defined set of registers (say 1-6), and gets results back from another set of pre-defined registers. Those are in the architecture specific code of the OS - the C functions that implement the system calls do not care about them much.

● **Protection Mechanisms:** ← NOTE that those are provided by hardware and used by OS

▶ CPU modes - kernel can exec privileged instructions, user cannot.

- Most of the time, you are running a user process. If it has no system calls and there are no hardware interrupts, then the OS will not run.
- The way to give the OS a chance to run is through a timer interrupt that goes off periodically - this runs the OS and it can check on the status of running processes, even if they try to usurp the CPU.

▶ Timer interrupt - is a protection mechanism in this way (if no timer interrupt supported by hardware, then processes need to yield)

▶ Memory Protection via base and limit registers (set by the OS) for the process that runs:

base ≤ memory user tries to access < base + limit

this is not checked by the OS but by hardware, the OS just sets up the registers

● **Examples from Linux** (via Linux Source Code navigator) for kernel 3.10

interrupts are hardware-specific:

arch/arm/kernel/entry/entry-common.S

to see how syscalls are triggered by an interrupt

sys_call_table is the system call ~~tab~~le

calls.S - defines the stuff that is in the system calls table

NOTE instructions "adr tbl, sys_call_table", "scno" for syscall #, "cmp scno, #NR_syscalls", "ldrcc pc, [tbl, scno, lsl #2]"

↖ load PC with tbl + scno

include/linux/syscalls.h - declarations of syscall functions

kernel//fork.c - definition of fork syscall

SYSCALL_DEFINE0(fork) to define it w/ 0 arguments.