Introduction to Computer and Programming

Lecture 14

Yue Zhang
Westlake University

August 1, 2023

Chapter 14.

Operating System

WestlakeNLP

Motivation

- We have a bytecode program, written directly, assembled, or compiled from C code.
- We load it to a machine, start it, and wait for it to halt.
- We manage devices (harddrive, keyboard, monitor, $\cdots$) directly!

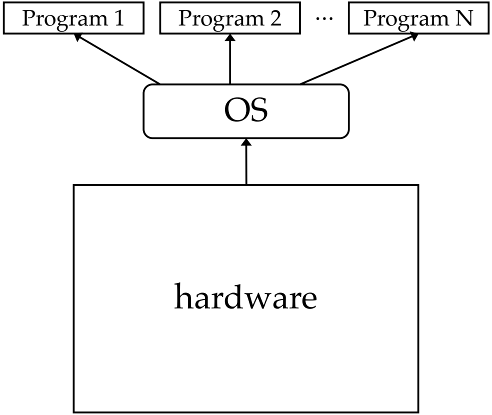
WestlakeNLP

Motivation

- We want to
  - Do not need to restart when launching a program.
  - Run a lot of different program simultaneously. (e.g., doing homework while listening to music)
  - No need to manage device code in every program.

Operating System

Basic idea

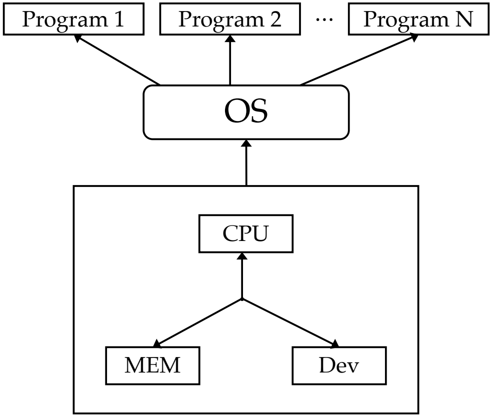


- Load OS as machine starts.
- Load other programs via OS.
- OS manages each program, providing it access to the hardware as if it occupies the hardware alone (**Virtual Machine**).

WestlakeNLP

Operating System

Basic idea



- What to virtualize?
  - CPU
  - Memory
  - Devices

WestlakeNLP

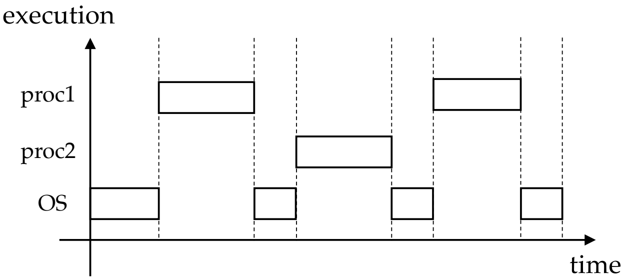
Virtualizing CPU

- In OS terminology, the runtime of a **bytecode** is a **process**.
- Bytecode is static; Process is dynamic.
- The concept of process is unnecessary if there is only one program being executed in a computer, but useful when multiple programs run simultaneously.

WestlakeNLP

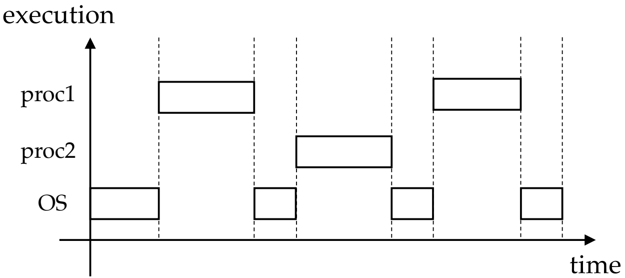
Virtualizing CPU

- How can we allow multiple processes to run simultaneously in a single CPU?
  - time-shared mechanism



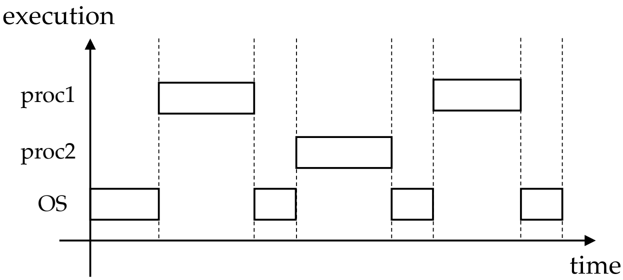
Virtualizing CPU

- time-shared mechanism



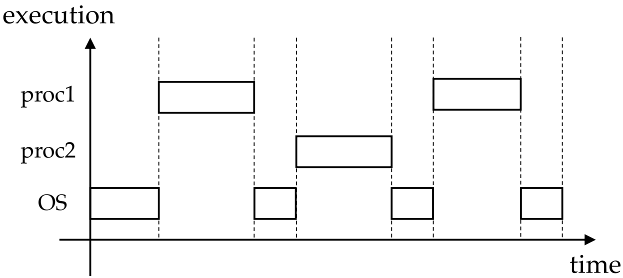
- Two issues to address
  - How to schedule processes?
  - How to make each process feel a) non-interrupted and b) owning the machine alone?

The Process Schedule



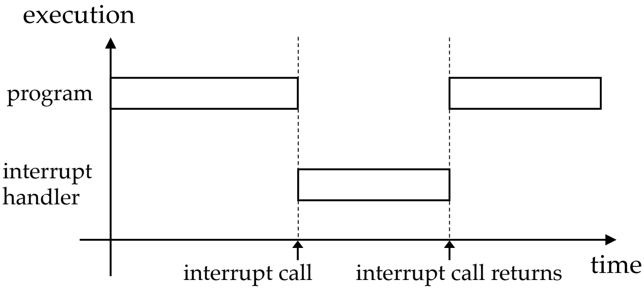
1. periodically executed by OS.
2. stops the current process P1.
3. save the current machine state into a memory record for P1.
4. decide a next process P2 to resume.
5. load its memory record for P2.
6. JMP to the code to resume P2.

The Process Schedule



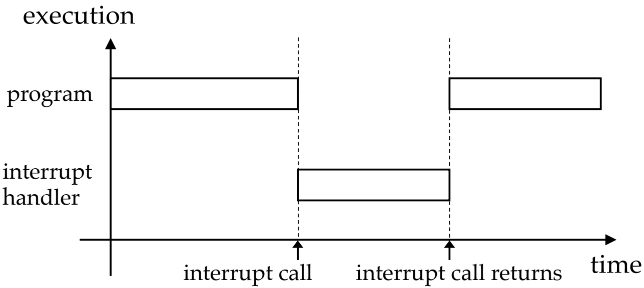
- How can the OS periodically kick in?
  - Need hardware support.
  - Special PC control called **interrupt**.

Interrupts



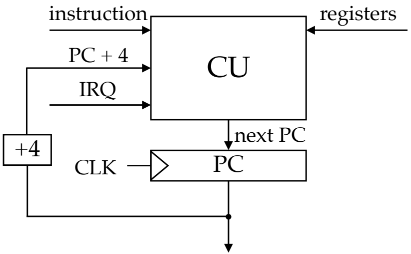
- Interrupts are a built-in mechanism for our computer.
- They are special function calls triggered by various devices, such as:
  - timer (periodical), keyboard press, mouse click

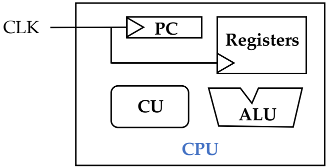
Interrupts



- The hardware part:
  1) The device sends a signal IRQ (interrupt request) to the CPU, together with its type.
  2) The CPU saves the current PC + 4 to a special register XP, and JMP to a designated address.

PC Control





- IRQ has higher priority.

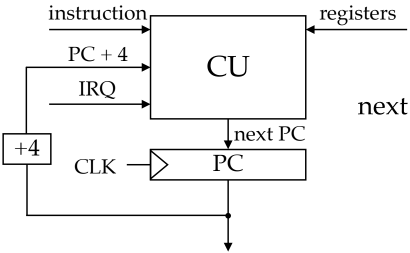
- ➢ The PC part of the CPU
- ➢ The CU (control unit) decides the value of the PC (program control) at the next step.

next PC =

1. sequential execution
   PC + 4
2. JMP instruction
   JMP address
3. interrupt request
   interrupt handler address
4. illegal instruction
   illegal instruction handler

Virtualizing CPU

PC Control



next PC =

1. sequential execution
   PC + 4
2. JMP instruction
   JMP address
3. interrupt request
   interrupt handler address
4. illegal instruction
   illegal instruction handler

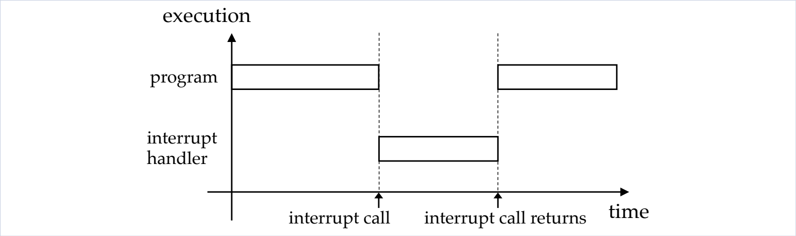
- What is illegal instruction?

e.g.,

ADD	R0	R1	R2	
000001	00000	00001	00010	00000000000

The first 6 bits of bytecode instruction must correspond to the CPU instruction set, if not, (e.g., 000000), the instruction is illegal.

Interrupts



- The hardware

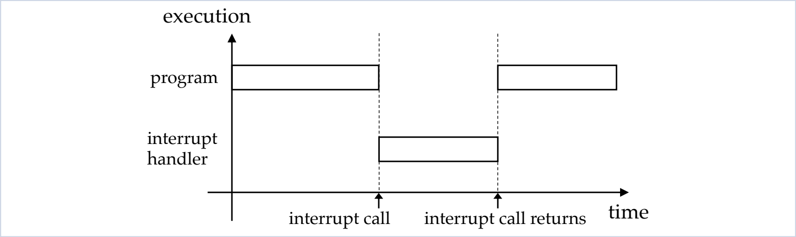
IRQ (device type) → { PC + 4 → XP ; interrupt handler → PC }

- Why store PC + 4 to XP ?
  For returning to process.
- The interrupt handler address are hardcoded.
  (e.g., 8 for timer, 12 for keyboard)

WestlakeNLP

Virtualizing CPU

Interrupts



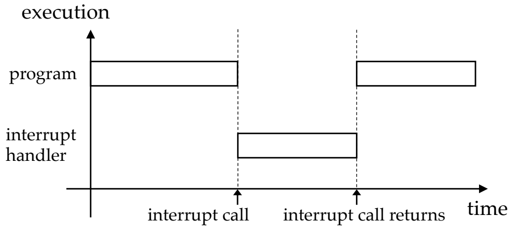
- The software (operating system code)

0	JMP	start	→ The address of a function for the start of OS code.
4	JMP	illegal	→ The address of illegal instruction handler.
8	JMP	timer	→ The address of timer handler.
12	JMP	keyboard	→ The address of keyboard interrupt handler.
16	JMP	mouse	→ The address of mouse interrupt handler.
⋮	⋮		

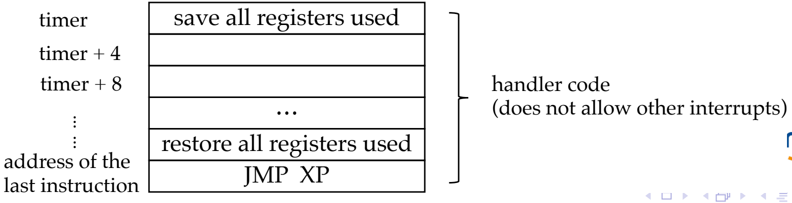
WestlakeNLP

Virtualizing CPU

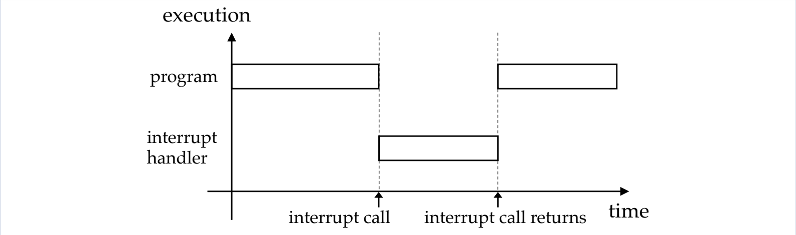
Interrupts



- The software (operating system code)



Interrupts



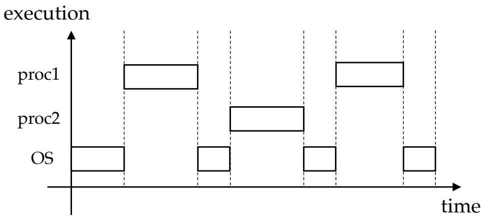
➢ The software (operating system code)

- The timer keeps track of system time.
- It is triggered periodically.
- Inside the timer, the OS can call the scheduler periodically.
- The scheduler cannot be called too frequent because it will introduce overhead.

WestlakeNLP

Virtualizing CPU

The Process Scheduler

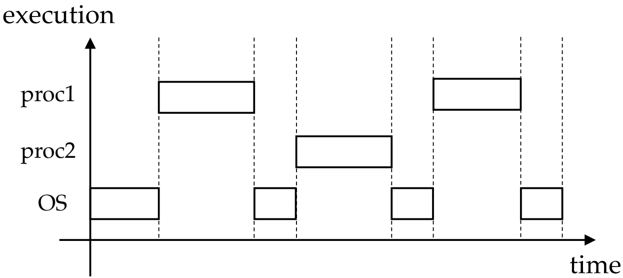


- The scheduler makes a record for each process, saving the content state of the machine.
  - The PC
  - The registers (all)
  - The memory (we will discuss this later.)
  - The virtual devices (some devices are shared)
- The record is saved to memory when a process is stopped, and reloaded when it resumes.

WestlakeNLP

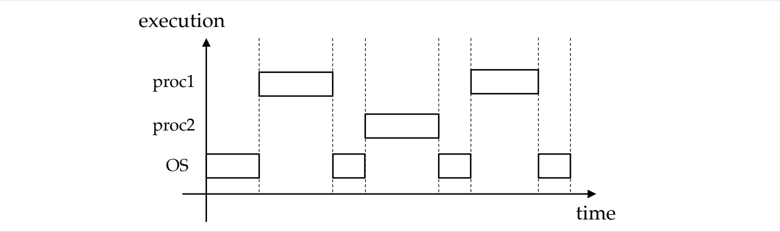
Virtualizing CPU

- time-shared mechanism



- Two issues to address
  - How to schedule processes?
  - How to make each process feel a) non-interrupted and b) owning the machine alone?

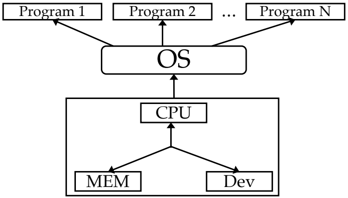
The Process Scheduler



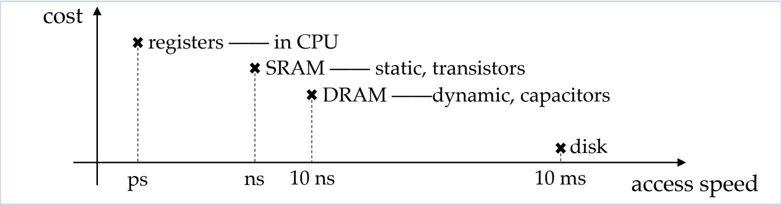
➢ When loading the next process, the OS can select process by considering priorities. e.g., the music being played should not be broken, while the document being edited can tolerate more delay without being noticed.

WestlakeNLP

Virtualizing Memory



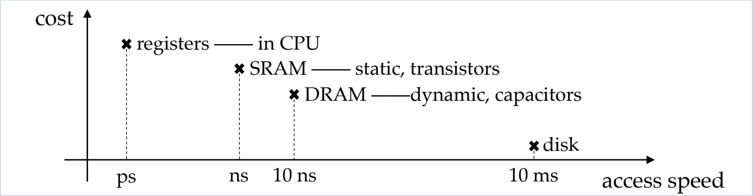
- Memory has two categories.
  - ROM —— read only memory
  - RAM —— read access memory
- RAM is the most commonly used.
- Different materials can be used for RAM.

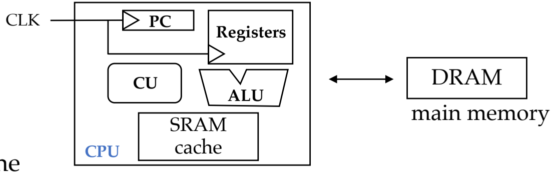


WestlakeNLP

Virtualizing Memory

- How to combine different materials?



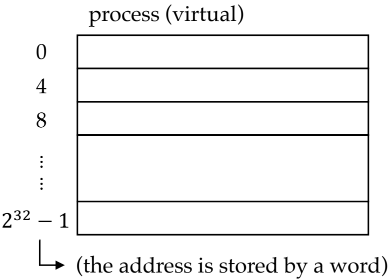


- Cache
  - small amount of SRAM in CPU.
  - store frequently accessed data.
  - allow overall memory speed ≈ SRAM
  - because of local access (e.g., loops)

WestlakeNLP

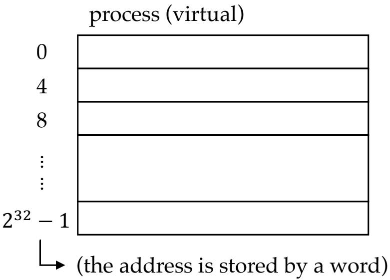
Virtualizing Memory

- Goal —— allow each process to see *maximum allowed* memory, *starting from 0* and *continuous*.



Virtualizing Memory

- Goal —— allow each process to see *maximum allowed* memory, *starting from 0* and *continuous*.

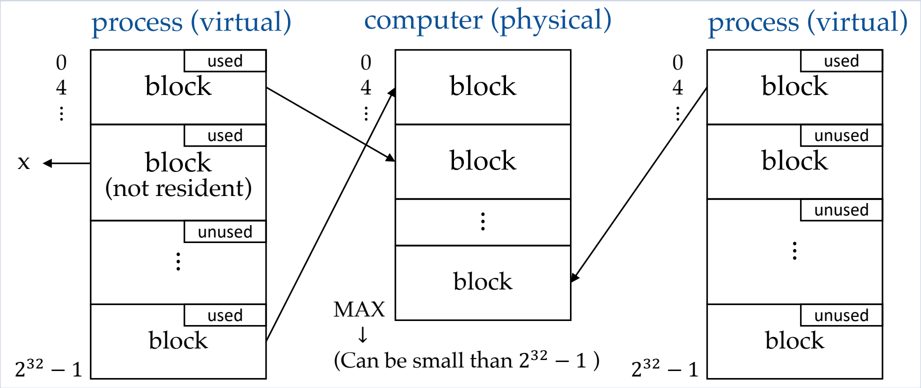


However, physical memory

- can be smaller than $2^{32}-1$
- can contain content for multiple processes

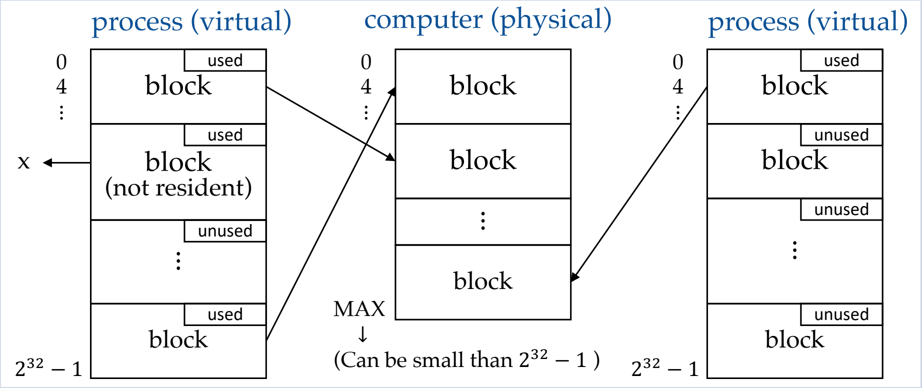
Virtualizing Memory

Solutions



- Organize memory in blocks (called *pages*).
- Use a *page table*, *page index* to map between process and physical page index.

Solutions



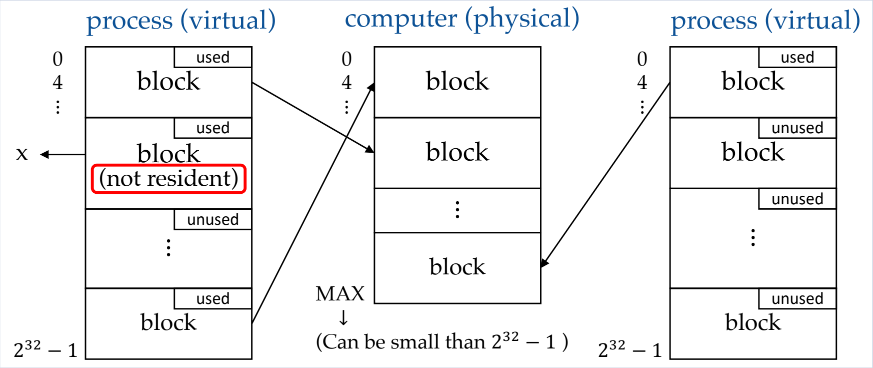
➢ Why not map word by word?

- Page table occupies memory.
- Data/code structure is continuous.

➢ How large can each page be?

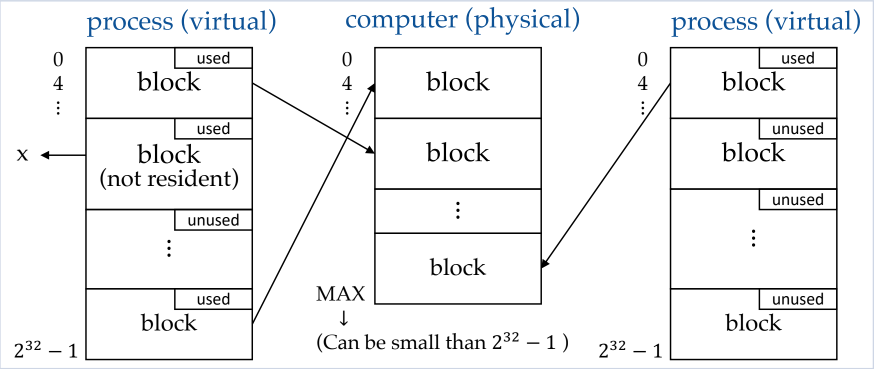
- Typically 4k(4096) bytes, or 1k(1024) words.
- Thus page index has $32 - log2^{4096} = 20$ bits.

Solutions



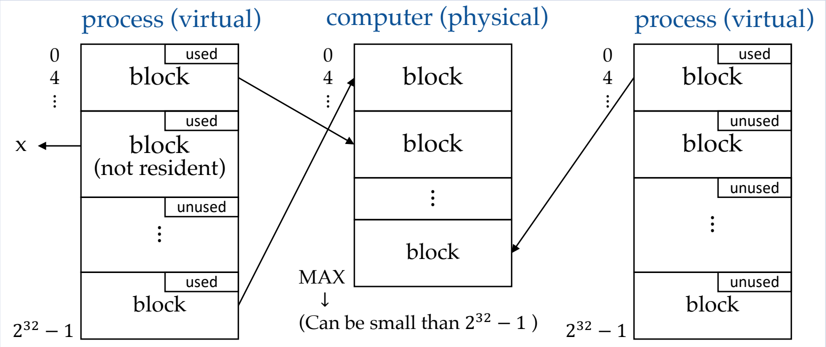
- What if the physical memory cannot hold **used** virtual memory by all processes?
  - Save less accessed pages to the disk.
  - In Linux, this file is called ***swapfile***.
  - Without ***swapfile***, a process will halt with "bad allocation" when out of memory.

Solutions



- What if the actually used memory is much more than the physical memory?
  - Frequently swapping between physical memory and disk.
  - The computer becomes slow.

Solutions

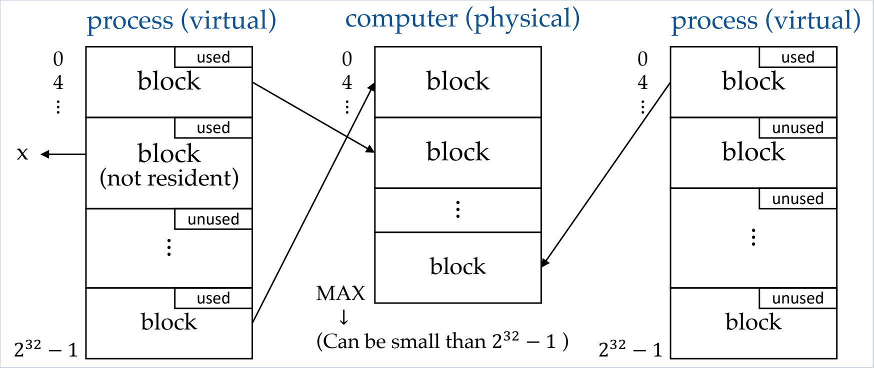


- How can we implement a page table?
  - It corresponds to the arrows in the figure.
  - One page for each process.
  - Roughly like this. →
  - When mapping a 32-bit address, the first 20-bits are **translated**, and the last 12-bits are **copied**.

Virtual address	Physical address	resident
00000⋯00000 (20 bits)	01010⋯10101 (20 bits)	1
⋮	⋮	⋮

Virtualizing Memory

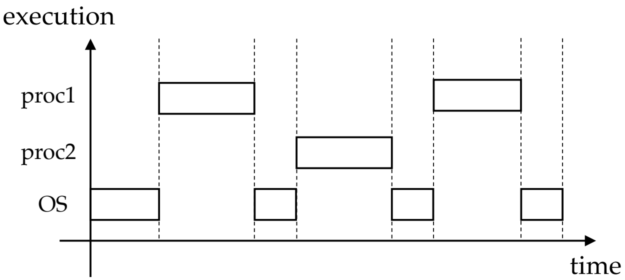
Solutions



➢ How can we implement a page table?

- Hardware: add a **memory management unit** (MMU) to CPU, which takes a page map address and 1) executes page mapping, 2) triggers OS when accessed memory is not resident.
- Software: OS maintains a page table for each process, and interrupts swap.

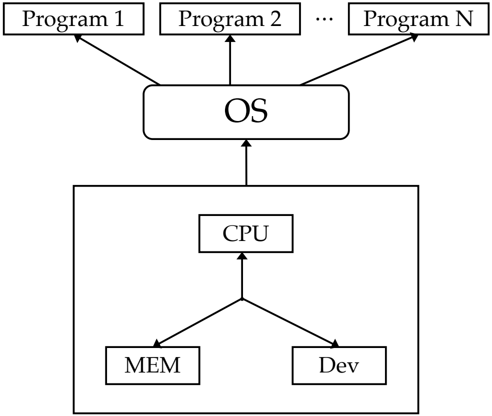
Virtualizing Memory



- The process record again
  - Current PC
  - All the registers
  - Page table address
  - State of relevant devices

WestlakeNLP

Devices

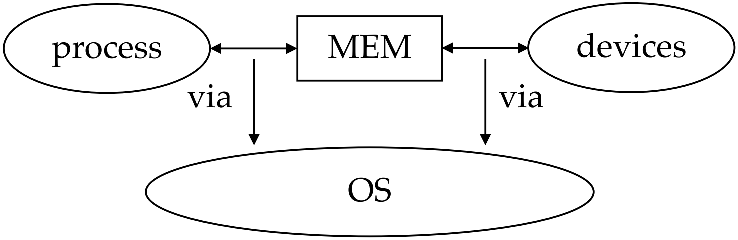


- OS manage devices through **driver** programs.
- Device can **call** OS through CPU via interrupts, can **be called** by OS.
- Devices **send** data and **receive** data from processes by writing them to memory.

WestlakeNLP

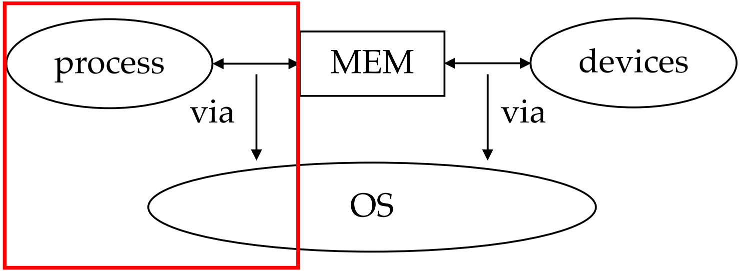
Devices

The data communication channel.



Devices

The data communication channel.

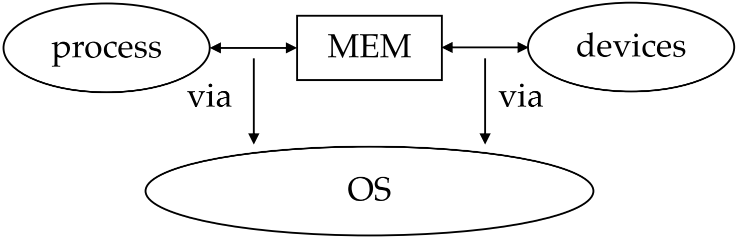


- Process can
  - send data to device. (e.g., *print*())
  - receive data from device. (e.g., *input*())
- Through supervisor calls.

WestlakeNLP

Devices

The data communication channel.



- How can a process call the OS?
  - Must be in bytecode.
  - Must be recognized by the OS.

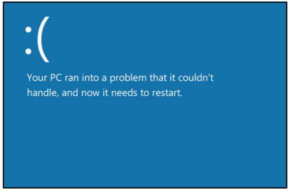
WestlakeNLP

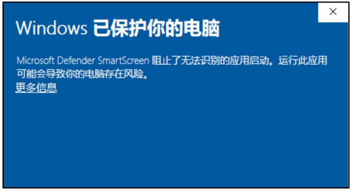
Devices

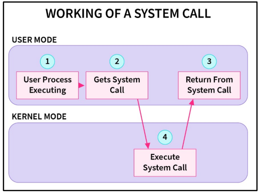
- How can a process call the OS?
  - Hardware supports for supervisor calls.

0	JMP	start
4	JMP	illegal
8	JMP	timer
12	JMP	keyboard
16	JMP	mouse
⋮	⋮	

- All instructions with the first 6 bits not in the instruction set trigger this.
- The **illegal** function is part of the OS.

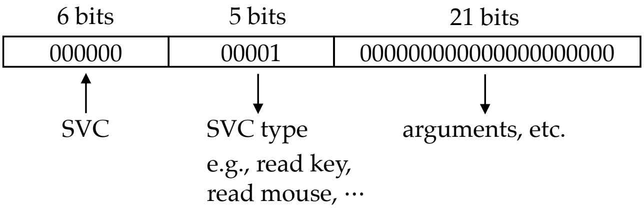






Devices

- How can a process call the OS?
  - Software supports for supervisor calls.
    - From the CPU perspective, supervisor call (SVC) can be illegal instructions, go to memory address 4.
    - From the OS perspective, it can tell the program to encode further information into a special instruction, and handle such calls at the illegal instruction handler.



Devices

- How can a process call the OS?
  - Software supports for supervisor calls.

OS pseudo code for ILLEGAL.

```
if instruction[0:5] == 0:
    read SVC type from instruction[6:10]
    read arguments from instruction[11:31]
    call SVC type with arguments
else:
    handle illegal instruction
```

Devices

Supervisor Call Examples

print("Hi.")

SVC display

↓

OS calls display function

↓

"Hi." being moved to console buffer (designated memory)

↓

Monitor display

↓

return to process

Supervisor Call Examples

input("a=")

SVC call display "a="

↓

SVC call keyboard

↓

sleeps the process until keystroke.

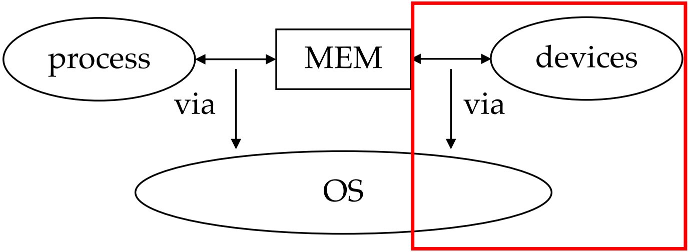
↓

read designated keyboard memory (designated memory).

sleep : a state that lets the OS know not to reschedule the process.

Devices

The data communication channel.



➢ Devices can

- send data to memory. (e.g., key press)
- receive data from memory via the bus.

Example of device request

Key Press

Device send IRQ to CPU

↓

CPU executes address = 12

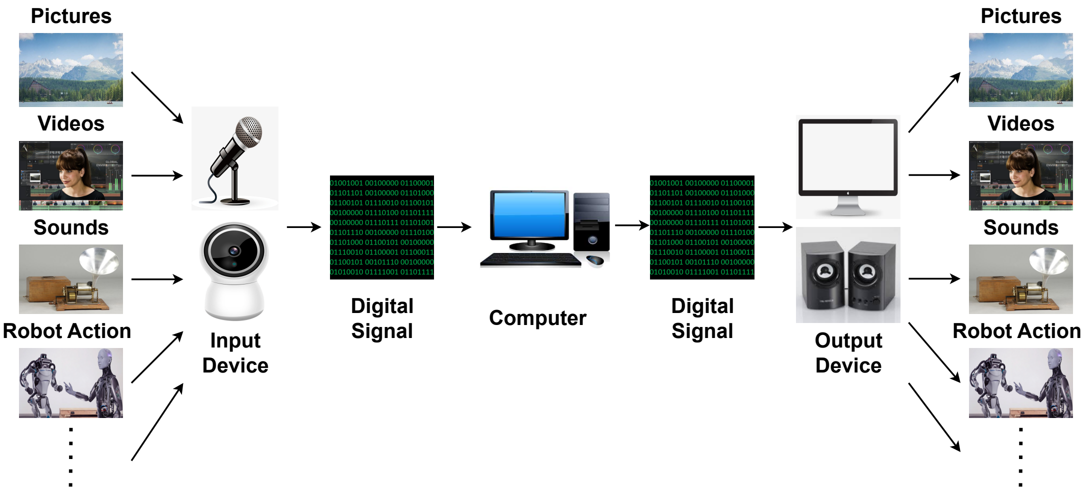
↓

OS code at address = 12 (JMP keyboard)

↓

OS code at KEYBOARD copy the pressed key from device buffer to designated memory.

Devices


Pictures
Videos
Sounds
Robot Action
Input Device
Digital Signal
Computer
Digital Signal
Output Device
Pictures
Videos
Sounds
Robot Action

OS Summary

0	JMP	start
4	JMP	illegal
8	JMP	timer
12	JMP	keyboard
16	JMP	mouse
⋮	⋮	
scheduler	The scheduler	
	Illegal instruction handler	
⋮	⋮	
display	Supervisor call functions	
⋮	⋮	
keypress	Interrupt handlers	