ILLINOIS TECH

College of Computing

CS 450 Operating Systems
OS and XV6 Overview

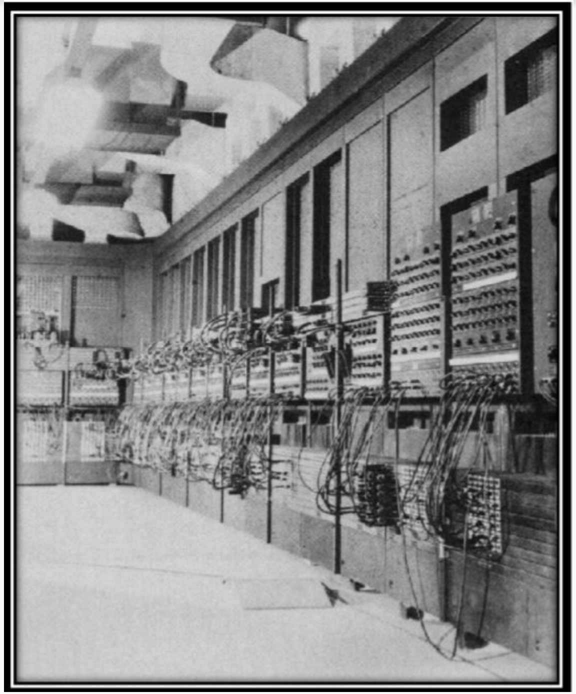
Yue Duan

PHASE 1 (1945 – 1975)

COMPUTERS EXPENSIVE, HUMANS CHEAP

Early Era (1945 – 1955):

- ENIAC
  - UPenn, 30 tons
  - Vacuum tubes
  - card reader/puncher
  - 100 word memory added in 1953
- Single User Systems
  - one app, then reboot
- “O.S” = loader + libraries
- Problem: Low utilization

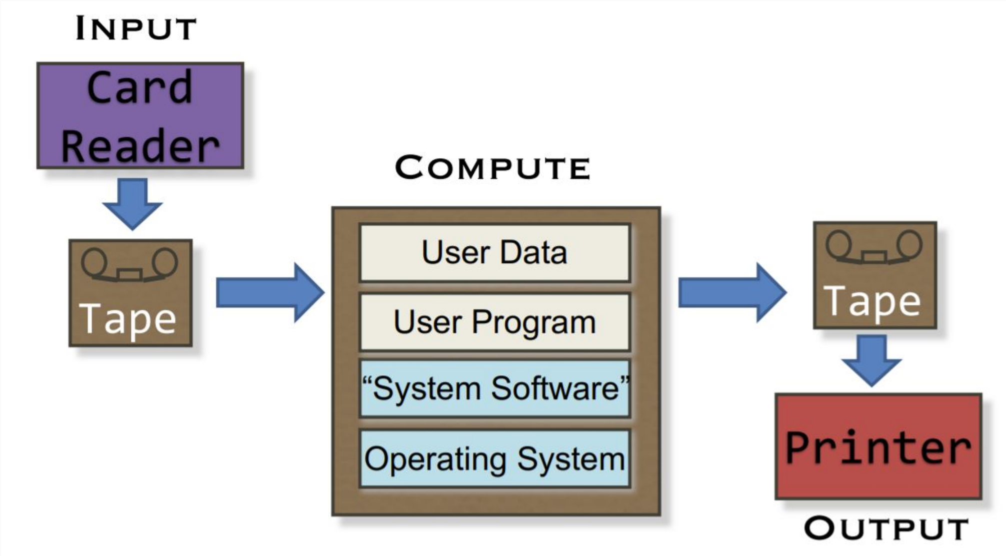


Batch Processing (1955 – 1960):

- First Operating System: GM-NAA-I/O
  - General Motors research division
  - North American Aviation
  - Input/Output
- Written for IBM 704 computer
  - 10 tons
  - Transistors
  - 4K word memory (about 18 Kbyte)

Batch Processing (1955 – 1960):

- O.S = loader + libraries + sequencer
- Problem: CPU unused during I/O



Time-Sharing (1960 –):

- Multiplex CPU
- CTSS: first time-sharing O.S.
  - Compatible Time Sharing System
  - MIT Computation Center
  - predecessor of all modern OS's
- IBM 7090 computer
- 32K word memory

Time-Sharing + Security (1965 –):

- Multics (MIT)
  - security rings
- GE-645 computer
  - hw-protected virtual memory
- Multics predecessor of
  - Unix (1970)
  - Linux (1990)
  - Android (2008)

PHASE 2 (1975 – TODAY)

COMPUTERS CHEAP, HUMANS EXPENSIVE

Personal Computers (1975 –):

- 1975: IBM 5100 first “portable” computer
  - 55 pounds
- 1977: RadioShack/Tandy TRS-80
  - first “home” desktop
- 1981: Osborne 1 first “laptop”
  - 24.5 pounds, 5” display

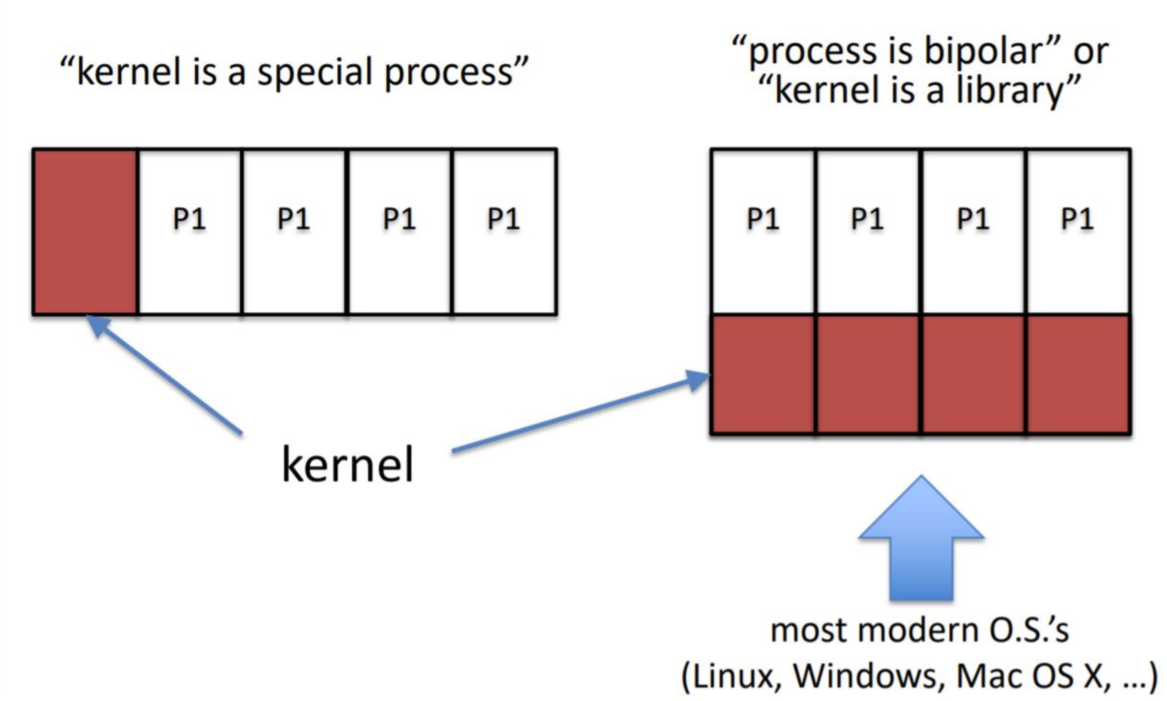
Modern Era (1990 –)

- Ubiquitous Computing / Internet-of-Things
  - 1988-ish
- Personal Computing
  - PDA (“PalmPilot”) introduced in 1992
  - #computers / human >> 1
- Cloud Computing
  - Amazon EC2, 2006

Hardware Support for Processes

- **Supervisor mode**
- manage and isolate multiple processes
  - Kernel runs in **supervisor mode** (aka kernel mode)
    - unrestricted access to all hardware
  - Processes run in **user mode**
    - restricted access to memory, devices, certain machine instructions

Two architectures of OS kernels



Two architectures of OS kernels

Kernel is a process	Kernel is a library
Kernel has one interrupt stack. Each process has a user stack	Each process has a user stack and an interrupt stack (part of Process Control Block)
Kernel implemented using “event-based” programming (programmer saves/restores context explicitly)	Kernel implemented using “thread-based programming” (context handled by language run-time through “blocking”)
Kernel has to translate between virtual and physical addresses when accessing user memory	Kernel can access user memory directly (through page table)

How does the kernel get control?

- Boot (reset, power cycle, …)
  - kernel initializes devices, etc.
- Interrupts
  - user mode -> supervisor mode
- There is no “main loop”
  - again: kernel more like a library than a process

Types of interrupts

Exceptions (aka Faults)

- Synchronous / Non-maskable
- Process missteps (*e.g.*, div-by-zero)
- Privileged instructions

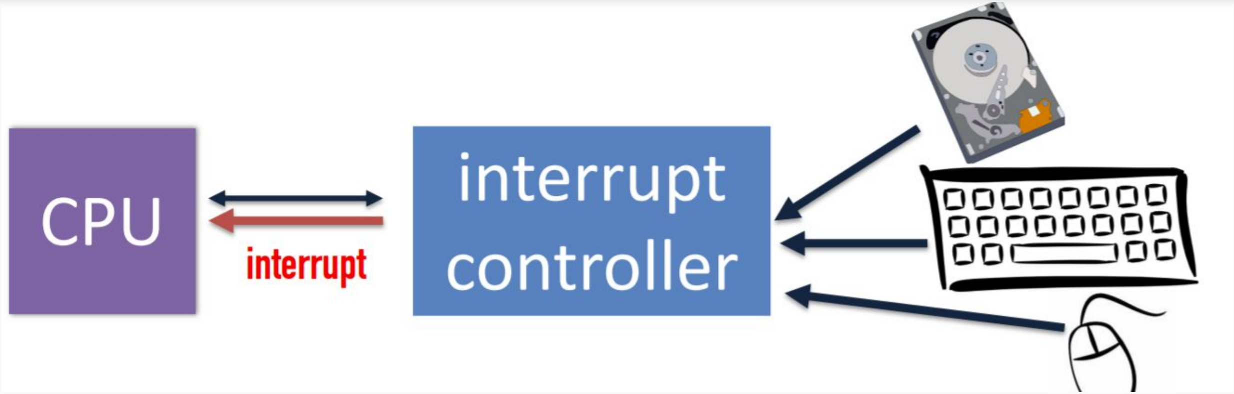
System Calls

- Synchronous / Non-maskable
- User program requests OS service

(Device) Interrupts

- Asynchronous / Maskable
- HW device requires OS service
  - timer, I/O device, inter-processor, …

H/W Interrupt Management



- A CPU has only one device interrupt input
- An Interrupt controller manages interrupts from multiple devices:
  - Interrupts have descriptor of interrupting device

Support for Devices

- Another primary objective of an O.S. kernel is to manage and multiplex devices
- Example devices:
  - screen
  - keyboard
  - mouse
  - camera
  - microphone
  - printer

Device Registers

- A device presents itself to the CPU as (pseudo)memory
- Simple example:
  - each pixel on the screen is a word in memory that can be written
- Devices define a range of device registers
  - accessible through LOAD and STORE operations

Device Drivers

- Device Driver: a code module that deals with a particular brand/model of hardware device
  - usually provided by device manufacturer
  - initialization
  - starting operations
  - interrupt handling
  - error handling

Device Drivers

- kernels provide many functions for drivers:
  - interrupt management
  - memory allocation
  - queues
  - copying between user space/kernel space
  - error logging

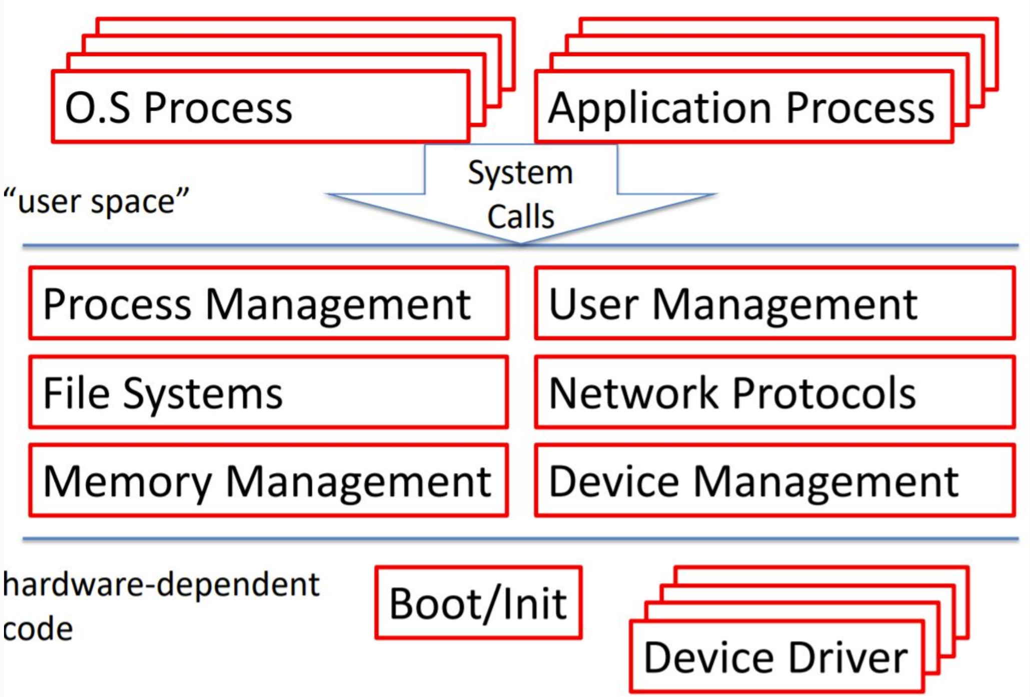
Booting an OS

- Steps in booting an OS:
  - 1. CPU starts at fixed address
    - in supervisor mode with interrupts disabled
  - 2. BIOS (in ROM) loads “boot loader” code from specified storage or network device into memory and runs it
  - 3. boot loader loads OS kernel code into memory and runs it

OS Initialization

- 1. determine location/size of physical memory
- 2. set up initial MMU / page tables
- 3. initialize the interrupt vector
- 4. determine which devices the computer has
    - invoke device driver initialization code for each
- 5. initialize file system code
- 6. load first process from file system
- 7. start first process

OS Code Architecture



XV6 Overview

- Xv6, a simple Unix-like teaching operating system

THANK YOU!