Runtime systems

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Admin

- Mid-term has been scheduled:
  - Mon 03/04, 2019
  - 8:00p - 10:00p
  - LWSN B155
- Lab 0 is due next Monday
  - Make sure to attend the PSO
  - Online students should contact your TA

Previous lecture

- OS Organization:
  - Operating systems supply a set of services
  - System calls provide interface between OS and application
  - Concurrency is a fundamental concept
    - Between IO devices and processor
    - Between multiple computations
  - Process an OS abstraction for concurrency
  - Process differs from program and function
  - Queues and lists

Previous lecture

- XINU:
  - Process creation and execution
  - XINU types
  - Unified list storage

Goals for today

- What is the OS view of memory and the rest of the hardware?
- How to implement processes?

Key hardware features from OS perspective

- Processor
- Memory system
- Bus and address space
- Individual I/O devices
- Interrupt structure

Processor

- Instruction set
- Registers
- Optional hardware facilities:
  - Firmware (e.g., BIOD in ROM)
  - Co-processor

Instruction sets on the example platform

- Galileo board:
  - CISC design
  - Well-known Intel x86 instruction set
  - The one used for the labs
- BeagleBone black:
  - RISC design
  - Well-known ARM instruction set

What are the important properties of the memory system?

Memory system

- How to address?
  - Addressing unit: size of a byte
- How to store a value?
  - Endianness
- How to layout the address space?
  - Monolithic
  - Linear (but not necessarily contiguous)
- How to cache (address, value) pairs

Byte order terminology

- Order of bytes that make an integer in memory:
    - Little endian: stores least-significant byte at lowest address
    - Big endian: stores most-significant byte at lowest address

Memory caches

- Special-purpose HW units
- Speed memory access
- Conceptually placed between processor and memory

Conceptual placement of memory cache

- All references (including instruction fetch) go through the cache
- Multi-level memory cache:
  - L1, L2, and L3
  - Instruction cache, data cache
- Cache policies:
  - Write-through, write-back
  - Non-inclusive cache, inclusive cache

Conceptual placement of memory cache

- Q: Do caches use virtual or physical addresses?
- A: Depends on the processor and cache
- Q: What is the impact of cache addressing?
- A:
  - Performance
  - OS needs to ensure correct semantic by invalidating cache

I/O devices

- Wide-variety of peripheral devices available:
  - Displays
  - Keyboards / mice
  - Disks
  - Wired and wireless network interfaces
  - Printer and scanners
  - Cameras
  - Sensors
- Multiple transfer paradigms:
  - Character, block, packet, stream

Communication between device and processor

- Communication with I/O devices is memory-mapped
- Processor can:
  - Interrogate the device status
  - Control a device (e.g., shutdown or awaken)
  - Start a data transfer
- Device uses bus to:
  - Reply to commands from the processor
  - Transfer data to/from memory
  - Interrupt when it needs attention from the processor (e.g., operation complete)

Bus operation

- Only two basic operations supported:
  - Fetch
  - Store

Bus operation

- Fetch:
  - Processor places address on bus
  - Processor uses control line to signal fetch request
  - Device senses its address
  - Device puts specified data on bus
  - Device uses control line to signal response

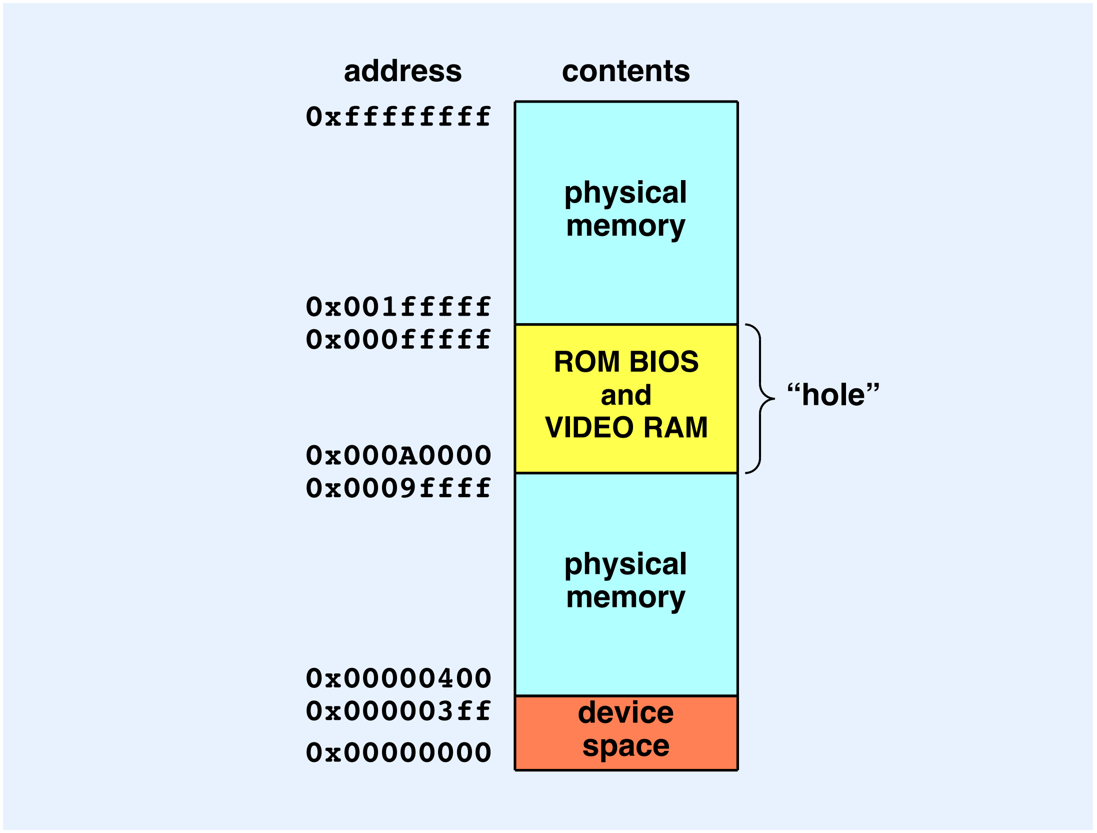
Bus operation

- Store:
  - Processor places data and address on bus
  - Processor uses control line to signal store request
  - Device senses its address
  - Device extracts data from the bus
  - Device uses control line to signal data extracted

Two basic approaches for I/O

- Port mapped I/O
    - Special instruction used to access devices (*in* and *out* in x86)
    - Used on earlier Intel machines
- Memory-mapped I/O
    - Devices placed beyound physical memory
    - Processor uses normal fetch/store memory instructions
    - Most common approach (e.g., recent Intel machines)

Address space on Intel System

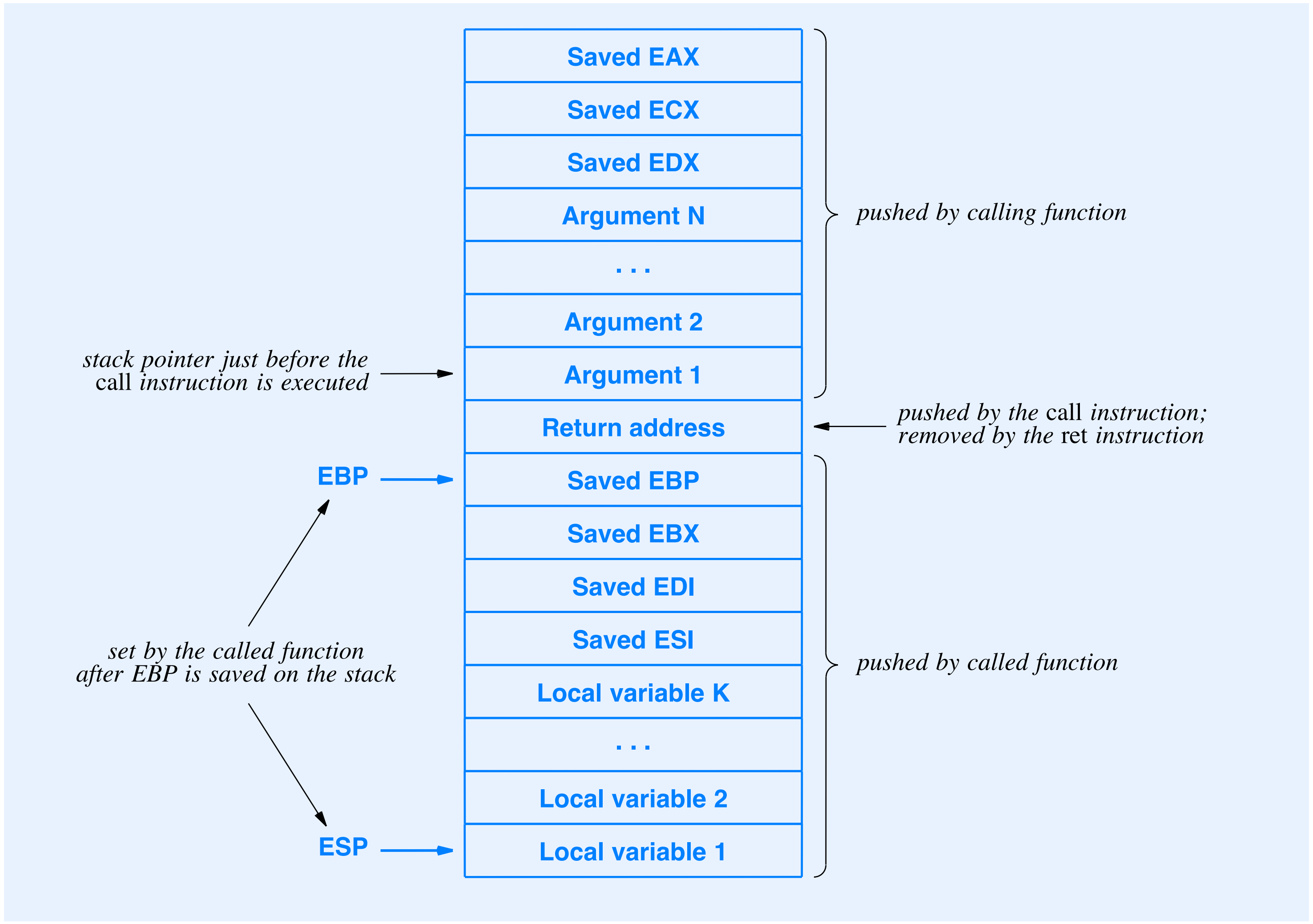


- Memory is often discontinuous
- Traditional PCs have a 'hole' from 640KB to 1 MB
- Does not affect applications
- But affects OSs

Calling conventions

- Calling conventions determine the set of steps taken during a call (e.g., function call)
- The conventions specify
  - How arguments and return values are passed?
  - Which registers must be saved by the caller?
  - Which registers must be saved by the callee?
  - Where registers are saved (e.g., on the stack)?
  - How to prepare the stack?
- Specific details may depend on:
  - The hardware
  - The compiler being used

Calling conventions for Intel


Saved EAX
Saved ECX
Saved EDX
Argument N
. . .
Argument 2
Argument 1
Return address
Saved EBP
Saved EBX
Saved EDI
Saved ESI
Local variable K
. . .
Local variable 2
Local variable 1
pushed by calling function
stack pointer just before the call instruction is executed
pushed by the call instruction; removed by the ret instruction
EBP
ESP
set by the called function after EBP is saved on the stack
pushed by called function

Interrupt mechanism

- Fundamental role in modern systems
- Permits I/O concurrent with execution
- Allows device priority
- Informs processor when I/O finished
- There are also software interrupts

Interrupt-driven I/O

- Processor start a device
- Device operates concurrently
- Device interrupts the processor when operation finishes

Interrupt mask

- Bit mask kept in a processor status register
- Determines whether interrupts are enabled
- HW sets mask during interrupt to prevent further interrupts
- Interrupt priority scheme:
    - Offered by some hardware (e.g., PIC or APIC)
    - Each device assigned a priority
    - When servicing level K interrupt, hardware sets mask to disable interrupts at level K and lower
    - Example: See how Xinu initializes the PIC (8259A)
        - `initevec()` in `system/evec.c`
- Inter-processor interrupt?

Interrupt processing

- Operating system must:
  - Store address of interrupt code in vector for each device
  - Arrange for interrupt code to save registers used during the interrupt and restore registers before returning

Returning from an interrupt

- Special HW instruction is used
- Atomically restores:
  - The old program state
  - Interrupt mask
  - Instruction pointer (program counter)
- HW returns to location when interrupt occurred
  - Processing resumes as if no interrupt had occured

Transfer size and interrupts

- Interrupt occrs after I/O operation completes
- Transfer size depends on device
  - Serial device transfers one character
  - Disk controller transfers one block (e.g., 512 bytes)
  - Network interface transfers one packet
- Large transfers use Direct Memory Access (DMA)

DMA

- Hardware mechanism
- I/O device transfers large block of data to/from memory without using the processor
- Example: network interface places incoming network packets in memory buffer
- Advantage: allows processor to execute during I/O transfer
- Disadvantage:
  - More expensive
  - More complex to design and program

Summary

- Memory system
  - Endianness, caches
- Bus operations
- Calling conventions
- I/O communication
  - Interrupts
  - DMA