Input and Output

Chapter 15

Programmed And Interrupt-Driven I/O

Topics

- Introduction
- I/O paradigms
- Programmed I/O
- Synchronization
- Polling
- Code for polling
- Control and status register
- Processor use and polling
- First, second, and third generation computers

Topics

- Interrupt-driven I/O
- A hardware interrupt mechanism
- Interrupts and the fetch-execute cycle
- Handling an interrupt
- Interrupt vectors
- Enabling and disabling interrupts and initialization
- Preventing interrupt code from being interrupted
- Multiple levels of interrupts
- Assignment of interrupt vectors and priorities

Topics

- Dynamic bus connections and pluggable devices
- The advantage of interrupts
- Smart devices and further improvements in I/O performance
- Direct memory access (DMA)
- Buffer chaining
- Scatter read and gather write operations
- Operation chaining
- Summary

Introduction

- This chapter discusses
  - interrupt driven I/O
  - how device driver software in the OS interacts with an external device

I/O Paradigms

- We have seen that processor issues fetch and store operations to bus addresses
- Unanswered questions
  - what control operations should each device support?
  - How does application software running on the processor access a given device without understanding the hardware details?
  - can the interaction between a processor and I/O devices affect the overall system performance?

Programmed I/O

- The early computers took the approach
  - CPU handles all the details
  - external device has digital circuit that controls the hardware in response to fetch and store operations
  - device consists of digital circuits
  - device consists of no intelligence
  - CPU issues the commands and controls device operations using fetch and store instructions
- In the above case, the I/O is said to be programmable

Synchronization

- Nonintelligent devices can't remember commands, each command is executed by the device when the processor sends the command.
- Processor operates much faster than an I/O device
- Programmed I/O needs synchronization, the processor must interact with device to determine when the device is ready for another command

  Because a processor operates orders of magnitude faster than an I/O device, programmed I/O requires the processor to synchronize with the device that is being controlled

Polling

- One way to synchronize is polling
  - processor repeatedly asks the device whether an operation has been completed, before the processor starts the next operation.

- Software for polling
  - polling uses a fetch operation
  - one or more addresses assigned to the device correspond to the status information
  - when the processor fetches a value from the address, the device responds by giving its current status

Polling Example

Addresses	Operation	Meaning
0 through 3	store	Nonzero starts paper advance
4 through 7	store	Nonzero starts head moving to beginning of line
8 through 11	store	Character to print (low-order byte)
9 through 12	store	Nonzero starts hammer striking
13 through 16	fetch	Busy: nonzero when device is busy

- Example of a specification that shows how a device uses the fetch-store paradigm to allow the processor to control the device or determine the current status (The specification is for an imaginary printing device).

Control And Status Registers

- Control And Status Registers (CSRs) refer to the set of addresses that a device uses.
  - control register: contiguous set of addresses that respond to store operation
  - status register: contiguous set of addresses that respond to a fetch operation

Processor Use And Polling

- Advantage of programmed I/O
  - inexpensive, as they don’t contain sophisticated digital circuits
- Disadvantage of programmed I/O
  - computation overhead, each step requires the processor to interact with the I/O device

Why is Polling Undesirable?

Because a typical processor is much faster than an I/O device, the speed of a system that uses polling depends only on the speed of the I/O device; using a fast processor will not increase the rate at which I/O is performed.

- Faster processor merely wastes more cycles waiting for an I/O device

Different Generations of Computers

Generation	Description
1	Vacuum tubes used to build digital circuits
2	Transistors used to build digital circuits
3	Interrupt mechanism used to control I/O

- Three generations of computer systems and characteristics of each

Interrupt Driven I/O

- Allow a processor to continue to perform computation while an I/O device operates.
- Changes are needed in the following
  - I/O device hardware
  - bus architecture
  - processor Architecture
  - programming paradigm

Interrupt Driven I/O

- I/O device hardware
  - I/O device must operate independently once it has started.
  - must inform processor after completion
- Bus architecture
  - allow two-way communication between processor and device
- Processor Architecture
  - processor need s mechanism to temporarily stop normal processing and handle a device

Interrupt Driven I/O

- Programming paradigm
  - shift from synchronous to asynchronous mode
  - synchronous: programmer specifi es each step of the operation an I/O performs
  - asynchronous: programmer writes code to handle events

A Hardware Interrupt Mechanism

An interrupt mechanisms temporarily borrows the processor to handle an I/O device. When an interrupt occurs, the hardware saves the state of the computation, and restarts the computation when interrupt processing finishes.

Interrupts and the Fetch-Execute Cycle

- Interrupts are implemented by modifying the fetch-execute cycle as shown below.
- I/O hardware does not actually interrupt a processor!

Interrupts and the Fetch-Execute Cycle

- Cause the printer to advance the paper
  - Poll to determine when paper has advanced
    - Move the print head to the beginning of the line
      - Poll to determine when the print head reaches the beginning of the line
        - Specify a character to print
          - Poll to determine when the character is locked in position
            - Cause the hammer to strike the character

- Poll to determine when the hammer is finished striking

The fetch-execute cycle used with a processor that uses interrupts. The processor tests for interrupts before each instruction.

Handling and Interrupt

- Five steps that processor hardware performs to handle an interrupt. The steps are hidden from a programmer
  - save the current execution state
  - determine which device interrupted
  - call the procedure that handles the device
  - clear the interrupt signal on the bus
  - restore the current execution state

Interrupt Vectors

- How does the processor know which device is interrupting?
  - processor sends special command over bus to determine which device needs service
  - only one device can respond at a time. Device has a unique number which is sent in the message
- Interrupt vector
  - an array of pointers indexed by device number.
  - pointer points to the software which handles the device
  - software is called interrupt handler

Interrupt Vector

- Cause the printer to advance the paper
  - Poll to determine when paper has advanced
    - Move the print head to the beginning of the line
      - Poll to determine when the print head reaches the beginning of the line
        - Specify a character to print
          - Poll to determine when the character is locked in position
            - Cause the hammer to strike the character

- Poll to determine when the hammer is finished striking

Illustration of interrupt vectors. Each vector points to code that serves as an interrupt handler for the device.

Enabling and Disabling Interrupts and Initialization

- How are values installed in interrupt vector table?
  - neither processor nor device hardware enters or modifies the table.
  - both assume vector table has been initialized.
- How to ensure interrupts don't occur before table is initialized
  - processors start in interrupt disabled mode.
  - after operating system has initialized the vector, software must execute instruction to enable interrupts

Preventing Interrupt Code from being Interrupted

- How to prevent a second device from interrupting when processor is handling an interrupt ?
  - policy that disables interrupts once one interrupt occurs
  - interrupts enabled after processor returns from previous interrupt

Multiple Level of Interrupts

- How to deal with devices that need interrupts to be serviced in a short time, devices that don't need service immediately, devices that take long to service and so on ..
- Solution: Multiple level interrupts or multiple interrupt priorities

  When operating at priority level K, a processor can only be interrupted by a device that has been assigned to level K+1 or higher

  A processor that has interrupt priority levels zero through N and uses zero for application programs can have up to N interrupts in progress at a time. However, only one interrupt can be in progress at any given priority level

Assignment of Interrupt Vectors and Priorities

- Fixed, manual assignment
  - person confi gures, interrupt vector addresses is entered by setting switches
  - used on small embedded systems
- Automated assignment
  - processor uses bus during boot time to determine which devices are attached. Processor assigns interrupt vector and priority for each device.
  - used on general purpose systems

Dynamic Bus Connections and Pluggable Devices

- What about devices that can be dynamically attached and detached?
- Universal Serial Bus (USB): permits user to plug device anytime
- USB assigned a interrupt vector and handler placed in memory. Uses a secondary handler after a device is attached.

Advantage of Interrupts

A computer that uses interrupts is both easier to program and offers better I/O performance than a computer that uses polling.

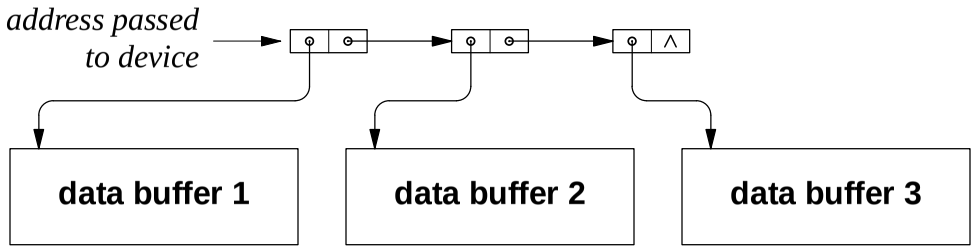
Smart Devices

- To improve performance beyond that of a basic interrupt
- More digital logic in I/O device, less dependency on processor - smart device.

- Processor requests a *read* operation by specifying the location on the disk and the location in memory
  - Disk performs all steps of the operation and interrupts when the operation completes

Example of the interaction between a dumb disk device and a processor. The processor controls each step of the operation.

Smart Devices



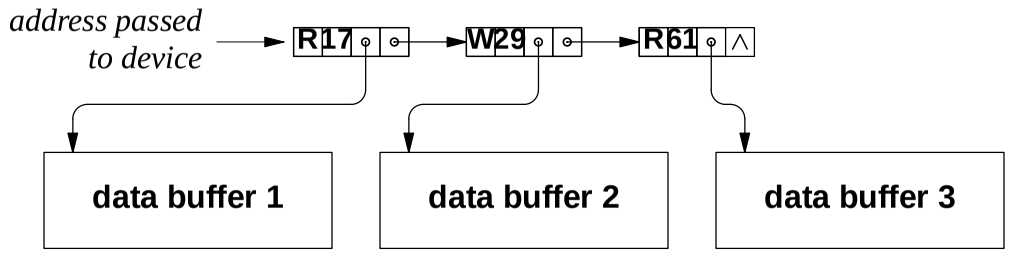
Example of the interaction between a smart disk device and a processor. The disk device performs individual steps of the operation without interrupting the processor.

Direct Memory Access

A technology known as direct memory access (DMA) allows a smart I/O device to access memory directly. A device that uses DMA can transfer data between the device and memory without using the processor.

Buffer Chaining

- Packet arrives in bursts: several packets arrive back-to-back with minimum time between packets
- Buffer chaining solves above problem
- Processor allocates multiple buffers and creates a linked list in memory.



A processor passes a list of buffers to a smart I/O device, and the device fi lls each buffer on the list without waiting for the processor.

Scatter Read and Gather Write Operations

- Scatter read: divide a large block of coming data into multiple small buffers
- Gather write: combining data from multiple small buffers into a single output block

Operation Chaining

- Buffer chaining handles situations in which given operation is repeated over many buffers.
- Further optimization is possible where device can perform multiple operations
- Start another operation as soon as current operation completes: this technology is called operation chaining

Operation Chaining

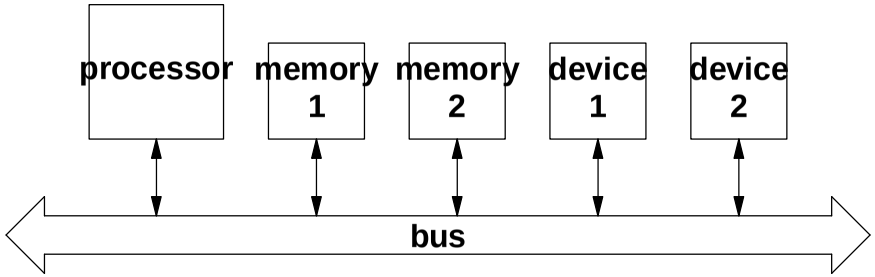


Illustration of operation chaining. Each node specifi es an operation (R or W), a disk block number, and a buffer in memory.

Summary

- Two paradigms are used to handle I/O devices: programmed I/O and interrupt driven I/O
- Programmed I/O needs processor to handle each step of operation; as the processor is faster, it must wait for the device.
- In interrupt driven I/O device can complete operation before informing the processor. Processor tests for interrupt during the fetch-execute cycle.
- Smart I/O devices contain additional logic that allows them to perform a series of steps without assistance from processor.
- Smart devices uses buffer and operation chaining to further optimize performance