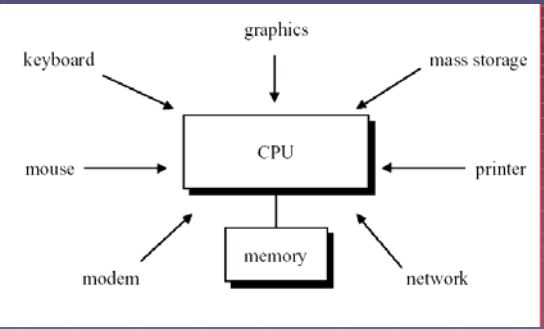
Input/Output Management

GU Jianhua
School of Computer Science
Northwestern Polytechnical University
谷建华
西北工业大学计算机学院

I/O System Management

- I/O hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Buffering
- Disk Schedule

I/O Devices



I/O 系统: 设备及其接口线路，控制部件，管理软件

需要解决的主要问题

- 主机与I/O设备速度不匹配
  - 中断、DMA、通道、缓冲区技术
- 用户直接使用I/O不方便
  - 虚拟设备、设备管理程序、设备驱动程序

要　　求

- 向用户提供使用外设的手段，满足用户的各种要求
- 充分发挥主机和外设的效率，使CPU与外设高度并行工作
- 均衡分配设备，防止设备忙闲不均
- 实现程序与设备无关性。

设备分类

- 按使用方式:
  - 独占、共享和虚拟设备
- 按隶属关系
  - 系统设备和用户设备
- 按传输数据量:
  - 字符设备和块设备
- 按传输速率
  - 低速设备、中速设备和高速设备。

I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - Port
  - Bus (daisy chain or shared direct access)
  - Controller (host adapter): interface CPU & I/O
- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O

Where is the Control of I/O Functions?

- Processor directly controls device.
- Controller or I/O module is added. Processor uses programmed I/O without interrupts.
- Same, but with interrupts.
- The I/O module is given direct control of memory via DMA.
- I/O module is separate processor, with special instruction set and access to memory to execute I/O program. No CPU intervention.
- I/O module has local memory of its own. Can control several I/O devices.
- Intelligent I/O (I_2O)
- Data-near computation: portions of applications are migrated to I/O controllers.

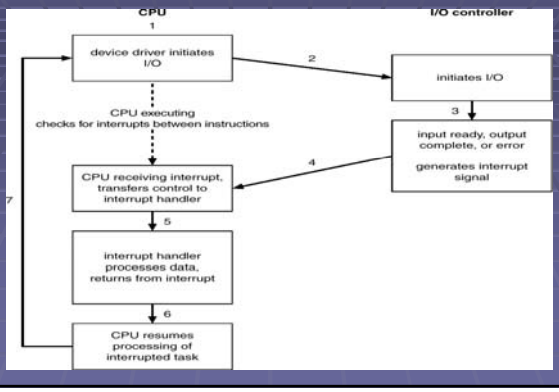
Polling

- Determines state of device
  - `command-ready`
  - `busy`
  - `error`
- Busy-wait cycle to wait for I/O from device

Interrupts

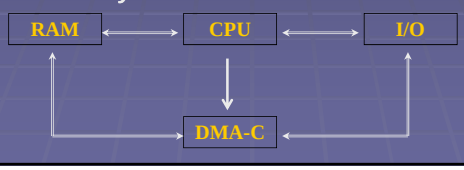
- CPU Interrupt request line triggered by I/O device
- Interrupt handler receives interrupts
- Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some unmaskable
- Interrupt mechanism also used for exceptions (software interrupt)

Interrupt-drive I/O Cycle



Direct Memory Access(DMA)

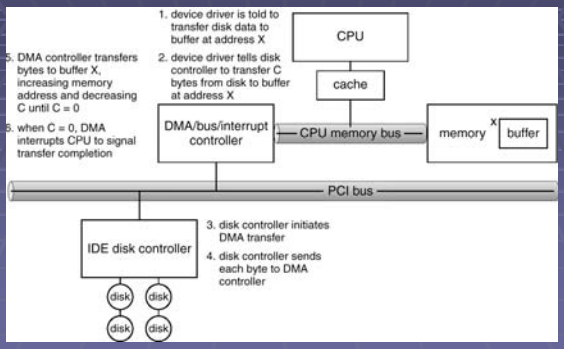
- Used to avoid programmed I/O for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory



Typical I/O operation involving DMA

- CPU sends DMA request with:
  - direction of transfer (read or write) on control control line
  - address of I/O device, on data lines
  - starting location in memory, on data lines, in address register
  - number of words to transfer, on data lines, in data count register
- CPU continues with other work, while DMA transfers block of data.
- When transfer is complete, DMA interrupts the CPU.

Six steps process to perform DMA transfer



I/O 系统的软件组织(1)

- I/O软件设计的目标
  - 软件与设备无关性
  - 出错处理: 错误尽可能在接近硬件的地方处理
  - 支持同步/异步传输
  - 能够处理独占设备和共享设备的I/O操作
- 基本思想
  - 软件组织成层次结构,底层软件用来屏蔽硬件细节，高层软件主要为用户提供简洁、规范的接口。

I/O 系统的软件组织(2)

- 中断处理程序
- 设备驱动程序
- 与设备无关的I/O软件
- 用户空间的I/O软件

设备驱动程序

- 设备硬件与操作系统的接口
- 包含了所有与设备相关的代码。
- 每个设备驱动程序只处理一种设备获一类密切相关设备。
- 从上层软件（设备无关软件）中接收抽象的I/O请求。
- 如有错误，则向设备无关软件报告。

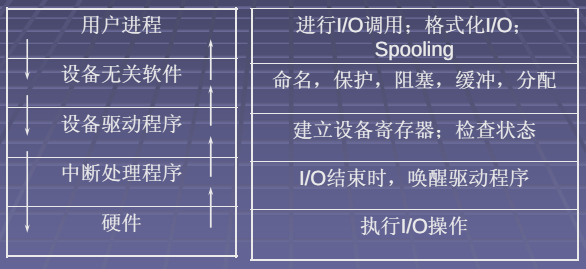
与设备无关的I/O软件

- 提供适用于所有设备的常用I/O功能，向用户提供一致接口。主要功能：
  - 设备命名
  - 设备保护
  - 数据缓冲
  - 设备的分配与释放
  - 错误处理

用户空间的I/O软件

- 以库例程形式提供的系统调用
  count=write(fd, buffer, nbyte)
- 以守护进程形势提供的Spooling系统。

I/O系统的层次结构



Application I/O Interface

- OS must abstract the sorted of devices so as to manage them
- I/O system calls encapsulate device behaviors in generic classes, left the interface to application
- Device-driver layer hides differences among I/O controllers from kernel
- Device-Independent
- Difference type of device has difference I/O interfaces

Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
  - Commands include `get, put`
  - Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows/NT include socket interface
  - Separates network protocol from network operation
  - Includes `select` functionality
- Other approaches to IPC and network communication (pipes, FIFOs, streams, queues, mailboxes)

Clocks and Timers

- Provide the function to
  - ***get current time***,
  - ***elapsed time***,
  - ***Timer***
- Timer: Timer is to trigger operation X at time T if programmable interval time used for timings, periodic interrupts

Task Delaying Management

- 任务延时系统调用的一般格式为：
  DELAY（s, t）
  s: 指向调用者TCB的指针
  t: 延时时间。
- 设置延时队列系统
  - 把所有延时进程的TCB表按要求的延时时间从小到大排序，称为延迟队列 。
  - 每次时钟中断，把队列中的所有等待时间减1。为0则使之就绪。需要遍历整个队列。

改进方法

- $\triangle t_i$：任务i的延迟时间与前面任务延迟时间的差值
- t_i:任务i要等的绝对时间。

$$t_i = \Delta t_1 + \Delta t_2 + \cdots + \Delta t_i = \sum_{k=1}^{i} \Delta t_k = \Delta t_i + \sum_{k=1}^{i-1} \Delta t_k$$

$$\Delta t_i = t_i - \sum_{k=1}^{i-1} \Delta t_k$$

延时队列的插入

- 设任务Q调用延时命令，它的延迟时间为t_Q。
  - 寻找插入位置：若TCB_Q要插入TCB_{i-1}与TCB_i之间，则必须有满足如下条件

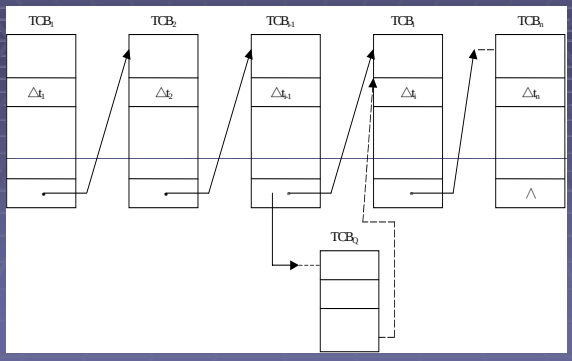
$$\sum_{k=1}^{i-1} \Delta t_k \le t_Q < \sum_{k=1}^{i} \Delta t_k$$

  - 计算Q的相对时间。然后从$\triangle t_i$中减去$\triangle t_Q$。

$$\Delta t_Q = t_Q - \sum_{k=1}^{i-1} \Delta t_k$$

  - 插入适当位置 ：特殊状况：1）队列为空，插到队首；2）延迟最长时，要挂在队尾 。

延时队列



延迟队列的操作

- 每次时钟中断，将队首TCB时间减1。若非零，则进行其它操作，若为零，把它从延时队列中取出并插入就绪队列（还要检查以后的TCB看是否也为零）。
- 延迟队列互斥保护
  - 延时命令处理和时钟中断处理，必须进行互斥访问延迟队列 。
  - 如何处理时钟停走？

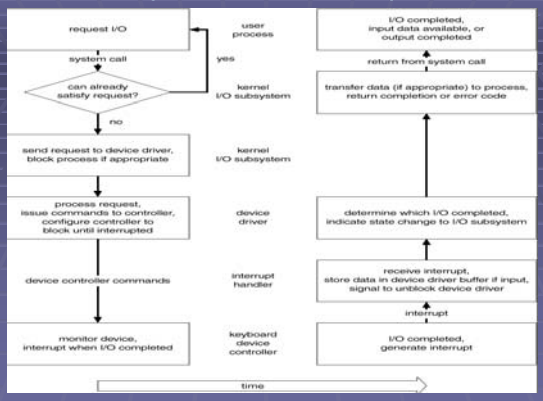
Blocking and Nonblocking I/O

- Blocking - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- Nonblocking - I/O call returns as much as available
  - Example: Be used to receive user input while display data on screen
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
- Asynchronous - process runs while I/O executes
  - Difficult to use(a callback function is needed)
  - I/O subsystem signals process when I/O completed
- Blocking until some time

I/O Requests to Hardware Operations

- Consider reading a file from disk for a process
  - Process issues a request to OS
  - OS queues the request
  - Driver get a request and determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

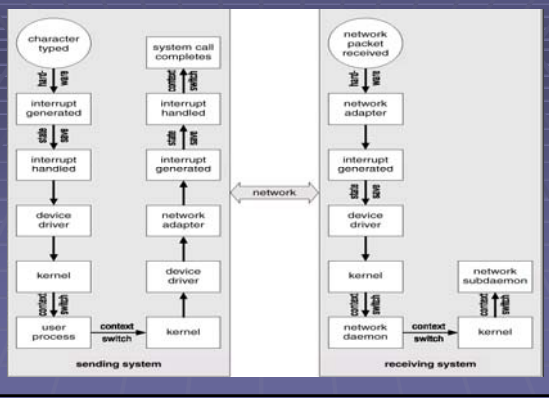
Life Cycle of an I/O Request



Performance

- I/O a major factor in system performance
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Intercomputer communications



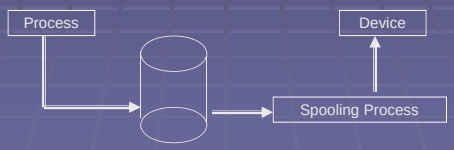
Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Spooling

- Offline

- Spooling

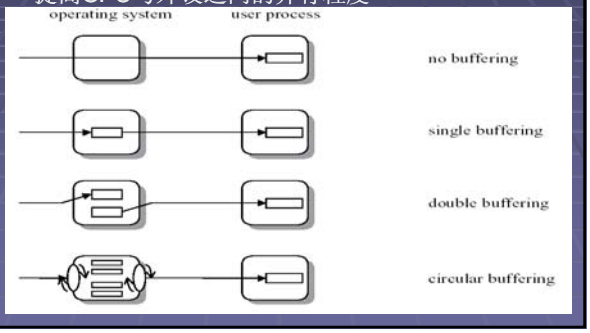


Features of Spooling

- Speed up the device
- Turn exclusive device to shared device
- Process use the virtual device

Buffering

- CPU与外设之间的速度不匹配
- 提高CPU与外设之间的并行程度



Buffer Pool Management

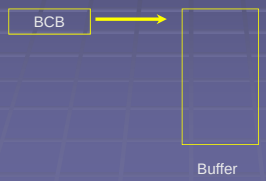
- Buffer pool consists of many buffers
- The buffer in buffer pool are shared by processes
- Available buffer; Input buffer; Output buffer
- Operations on buffer
  - *getbuf(type);* get buffer from queue *type*
  - *putbuf(type,buf);*free buffer to *queue type*

Buffer Management on UNIX

- Block Device Buffer Management
- Character Device Buffer Management

Block Device Buffer Management

- Buffer Control Block(BCB) and Buffer
- Organize the BCB to queue
  - Available queue
  - Device queues
  - I/O request queues



Buffer Control Block

```
struct buf
 {  struct buf *b_forw, *b_back;
    struct buf *av_forw, *av_back;
    char *b_blkno;
    ……  } buf[NBUF];
```

- Two queues
  - Free block queue
  - Device block queues(one device, one queue)
- Free block queue: FIFO algorithm
- BCB releases to free queue and remains in device after I/O operation

getblk() and brelse

- getblk(dev,blkno)
  - search in device with parameters
  - If(found) use it directly;
  - else get one BUF from head of free queue;
    remove from device queue;
    use it;
- brelse(bp) /* bp pointer to a BUF */
  - If there is process waiting this BUF, wake it up
  - put BUF in the end of free queue
- That is a LRU algorithm

Device Read/Write with buffer

- Read: user area ← input buffer device
- Write: user area → output buffer device
- Delay writing: buffer is not written fully.
- When write this buffer
  - Buffer is full
  - Buffer is used by other device

Character Device Buffer Management

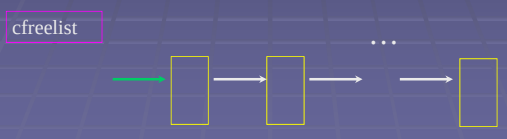
- Features of Character Device
  - 字符设备工作速度低
  - 一次I/O要求传输的字符少且不固定
  - 在传输过程中一般需要做若干即时处理
  - 字符一般不重用

Unix Char Buffer Management

- Buffer
  struct cblock
  { struct cblock *c_next;
  char info[6]; /* char buffer */
  }
- Buffer pool
  struct cblock cfree[NCLIST];
- Free character buffer queue
- I/O character buffer queues

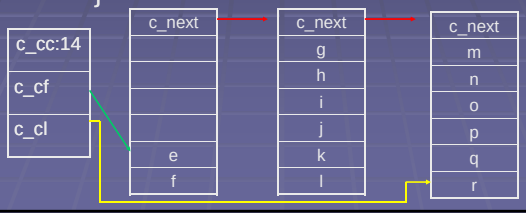
Free character buffer queue

- Put buffer into this queue when it is not be used.
- Head pointer :
  struct cblock *cfreelist;
- Algorithm of allocation and release: ***stack***



I/O character buffer queues

- Head pointer of queue
  struct clist
  { int c_cc; /* available char counter*/
  int c_cf; /* first character */
  int c_cl; /* last character */
  }



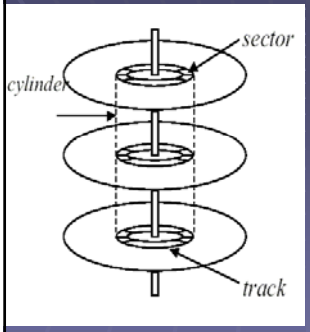
Read/Write Character

- ***User process gets a char from buffer***
  - Get a char from buffer one by one according to the ***c_cf*** , c_cf++;
  - After all of char in a buffer are read, release this buffer to free buffer queue
- ***Device puts char into a buffer***
  - Put a char into buffer according to the ***c_cl,*** c_cl++
  - After a buffer is full, allocate a buffer from free buffer queue

Disk scheduling

- Disk Structure
- Disk scheduling
  - FCFS Scheduling
  - Shortest-Seek-Time-First (SSTF)
  - SCAN (Elevator Algorithm)
  - C-SCAN (circular SCAN)

Disk Structure



- Disk speed:
  - ***seek time*** : head moves to correct track
  - ***rotational delay*** : wait until sector is under head
  - ***transfer time*** : transfer data between disk and memory
- Seek time is max

Disk Performance

- Seek Time : T_S
  - n = number of tracks traversed
  - m = "track traversal time"
  - s = startup time
  - $T_S = m*n + s$
- Rotational Delay (Latency Time): T_R
  - r = # revolutions per time unit (rotation speed)
  - $T_R = 1/(2*r)$
- Transfer Time: T_T
  - b = # bytes to be transferred
  - N = number of bytes on track
  - $T_T = b/(r*N)$
- Disk Access Time: $T_a = T_S + T_R + T_T$

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Objective: Minimize seek time.
- Seek time ≈ seek distance.
- We illustrate them with a request queue (0-199).

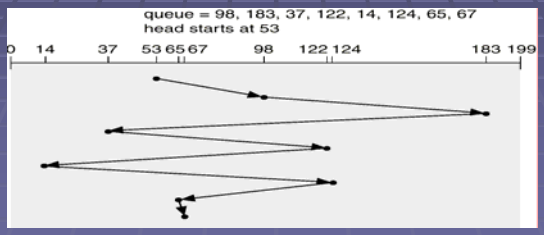
98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS Scheduling

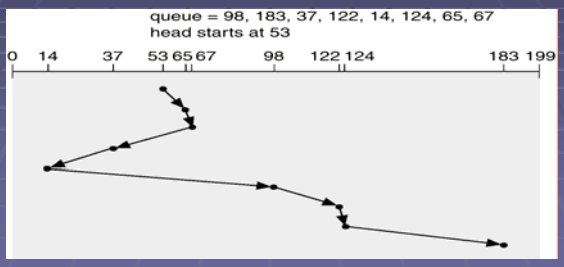
- Advantages:
  - simple
  - fair
- Disadvantages
  - poor average service time

Total head movement: 640



Shortest-Seek-Time-First (SSTF)

- Selects the request with the minimum seek time from the current head position.
- Problem: Starvation

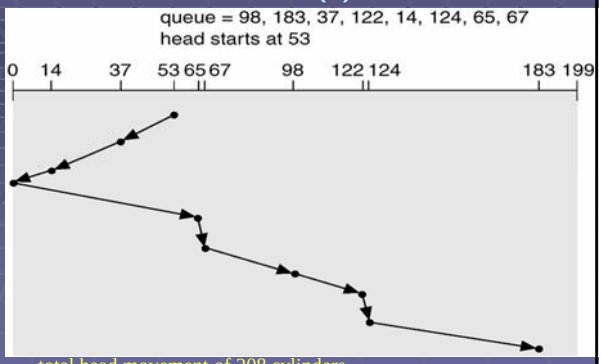


total head movement of 236 cylinders

SCAN(1)

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- 两个优先
  - 相同优先
  - 最短优先
- Problem:
  - When we change direction at end, requests there are very new, other end request has more delay

SCAN(2)

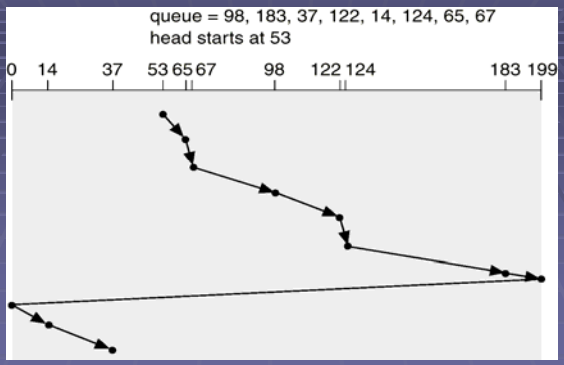


total head movement of 208 cylinders.

C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

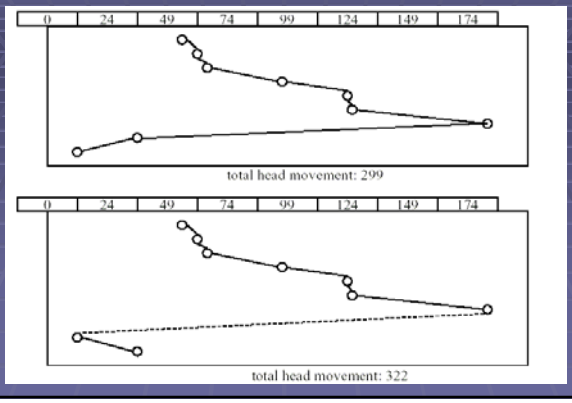
C-SCAN (Cont.)



LOOK and C-LOOK

- Version of LOOK and C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

LOOK and C-LOOK



Disk Management

- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - *Partition* the disk into one or more groups of cylinders.
  - *Logical formatting* or "making a file system".
- Boot block initializes system.
  - The bootstrap is stored in ROM.
  - *Bootstrap loader* program.
- Methods such as *sector sparing* used to handle bad blocks.