CS323 Operating Systems
Input/Output III

Yuanyuan Zhou
Lecture 21
3/12/2003

Content of this lecture

- Administrative announcements
- Clock and timer
- Power management
- Summary

Administrative

- MP1 grade
  - Available by Friday
- MP3 tutorial
- Midterm 1
  - Solution will made available on Thursday
- Drop-class

Review

- I/O software
- Three ways to do I/O
  - Programmed I/O
  - Interrupt-based I/O
  - I/O using DMA
- Device driver
- User-space I/O system calls

Blocking and Non-blocking I/O

- When an application calls a **blocking** system call, the execution of the application is suspended.
  - The application waits for the I/O result.
  - The physical action performed by I/O devices is generally asynchronous (they take unpredictable/varying amount of time)
- **Non-blocking** I/O system call means that the application calls the I/O system call, and returns quickly with the returned value
  - Examples: video application which reads from the disk and simultaneously displays video frames

Asynchronous System Call

- Alternative to non-blocking system call is asynchronous system call.
- An asynchronous system call returns immediately, without waiting for the I/O to complete.
  - The application continues to execute its code and the completion of the I/O call is communicated to the application in the future
- Multi-threaded applications benefit from non-blocking system calls

Performance

- Heavy demands on CPU to execute device driver code and to schedule processes fairly and efficiently as they block and unblock.
- Several principles to improve the efficiency of I/O:
  - reduce the number of context switches
  - reduce the number of times that data must be copied in memory while passing between device and application
  - reduce frequency of interrupts (user large transfers, smart controllers, polling, etc)
  - increase concurrency by using DMA controllers
  - move processing primitives into hardware
  - balance CPU, memory subsystem, bus and I/O performance
  - Caching
- Place to implement I/O functionality
  - device hardware, device driver, kernel, application?
  - Device-functionality progression (the lower level implementation, the better performance, efficiency, however also increased development cost, and decreased flexibility.

Discussion

- Tradeoff between
  - Blocking I/O
  - Non-blocking I/O
  - Asynchronous I/O

Clocks and Timers

- Hardware clocks (also called timers)
  - give the current time
  - give the elapsed time
  - set a timer to trigger operation X at time T
- Programmable interval timer - hardware to measure elapsed time and trigger operations
- This mechanism is used by the scheduler, disk I/O subsystem, network subsystem

Clock Hardware

- Older/simpler clocks
  - tied to the 110 or 220 volt power line and cause interrupt on every voltage cycle at 50-60 Hz.
- Current clocks consist of three components:
  - a crystal oscillator
  - a counter
  - a holding register.
  - Feature:
    - The quartz crystal generates a periodic signal of very high accuracy (several hundred MHz). This signal is fed into the counter which counts down to zero. When the counter is zero, it causes CPU interrupt. The holding register is used to load the counter.

Programmable Clock

- Modes of programmable clocks:
  - **one-shot mode**: when the clock starts (started explicitly by the software), it copies the register value into the counter, decrements the counter, causes an interrupts when counter is zero, and stops ;
  - **square-wave mode**: when the clock starts, it copies the register value into the counter, decrements the counter, causes an interrupts when counter is zero, and then automatically the holding register value is copied into the counter, and the whole process **repeats** (i.e., there is no explicit start of the clock by the software)
- The periodic interrupts are called **clock ticks**.
- Programmable clock chips usually contain two or three independently programmable clocks.

Clock Issues

- battery-powered backup clocks
- Some clocks drift and need to be resynchronized.
  - manually resynchronize the clocks.
  - get the current time from a remote host which carries the UTC (Universal Coordinated Time), formally known as Greenwich Mean Time).
- NTP (Network Time Protocol) is a standard protocol to re-synchronize clocks between two networked systems (regularly the system administrators re-synchronize all clocks in our machines using NTP).

Clock Software

- Clock hardware generates interrupts at known intervals.
- The clock hardware is served by the clock driver software.
- Functions of the clock driver:
  - Maintaining the time of day
  - Preventing processes from running longer than they are allowed to
  - Accounting for CPU usage
  - Handling alarm system call made by the user process
  - Providing watchdog timers for parts of the system itself
  - Doing profiling, monitoring and statistics gathering

Discussion

- How can clock help debugging?

Clock Overflow

- *Time of the day*(real clock): requires incrementing a counter at each clock tick.
- One difficulty is the overflow of the counter (e.g., a 32-bit counter overflows in two years).
- Solutions:
  - Use 64 bit counter, but maintaining the counter is more expensive
  - Maintain the time of day in seconds, rather than ticks, using a subsidiary counter to counter ticks until a whole second has been accumulated. Because seconds is more than 136 years, this method works until the 22 century.
  - Count ticks, but only relative to the time the system was booted, rather than relative to a fixed external moment.

Other Functions

- *Preventing processes from running too long*
  - whenever a process is started, the scheduler initializes a counter to the value of that process' quantum in clock ticks. As the clock ticks, the counter is decremented, and when counter is zero, the clock driver calls the scheduler to set up another process.
- *CPU accounting*
  - The system should start a second timer (different from the main timer). Whenever the process starts, the second timer starts counting. When a process stops, the timer can be read out to tell how long the process has run. When a process is interrupted, the timer value should be stored and when the process restarts, the timer value should be restored.
- Simpler way to do CPU accounting
  - currently running process keeps a field in the process table entry. At every clock tick, this field in the current process is incremented, ie. every clock tick is ``charged'' to the process running at the time (problems might arise if many interrupts occur).

Timer for Individual Process

- Providing timers (alarm) to requests from processes in order to give a warning after a certain time interval
  - If the clock driver had enough clocks, it could set a separate clock for each request. This is not the case.
- Simulate virtual clocks with a single physical clock.
  - maintain (a) table in which the signal time for all pending times is kept, and (b) a variable giving the time of the next one. When time of day is updated, the driver checks for the closest signal to occur. Once it has, the table is searched for the next one to occur.

Watchdog Timer and Profiling

- A timer is set, and when the timer goes off, instead of causing signal as it is the case for user signals, the clock driver calls a procedure supplied by the caller.
- Use it for profiling
  - Build histogram of the program counter and see where it is spending its time.
  - Debugging

Soft Timers

- Software-based timer, called soft timer, is a timer which
  - sets to the requested frequency as needed
  - is checked by the kernel when entries are made for other reasons such as system calls, TLB misses, page faults, I/O interrupts, CPU going idle.
  - If the soft timer has expired, the scheduled event is performed (e.g., packet transmission) with no need to switch to kernel mode.
  - The kernel resets the soft timer again to go off. The kernel copies the current clock value to the timer and adds the timeout interval to it.
- Soft timer can avoid interrupts
  - the frequency of kernel entries is so frequent that the kernel can check on the soft timer and schedule the events without sending an interrupt.

Power Management

- ENIAC - first general purpose electronic computer had 18,000 vacuum tubes and consumed 140,000 watts of power.
- Invention of transistors dropped the power usage dramatically, and computer industry stopped considering power requirements for some time.
- Currently big change: mobile environments, mobile devices which run only on batteries (e.g., battery-powered computers laptops - IBM Think-pads, palmtops - Palm VII, HP Journadas, Webpads)
- Battery/Energy performance results do not follow the Moore's law (Moore's law = double performance of hardware such as processors, memory every 18 months)

Approaches to reduce energy consumption

- OS turns off parts of the computer when are not in use (mostly IO devices such as display)
- Application program uses less energy, possibly degrading quality of the user experience

Reminder