A sequência destes slides foi alterada

15-213

"The course that gives CMU its Zip!"

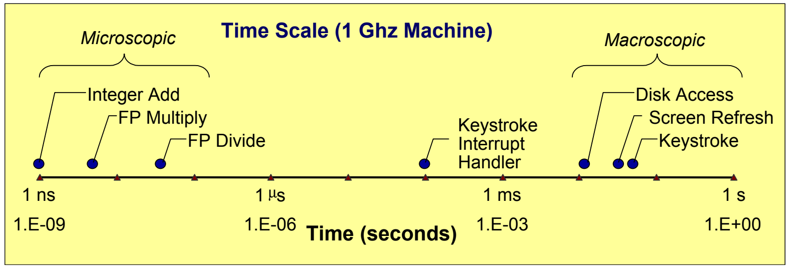
Time Measurement
Oct. 24, 2002

Topics

- Time scales
- Processes
- Interval counting
- Cycle counters
- K-best measurement scheme

class18.ppt

Computer Time Scales



Two Fundamental Time Scales

- Processor: ~10^{-9} sec.
- External events: ~10^{-2} sec.
  - Keyboard input
  - Disk seek
  - Screen refresh

Implication

- Can execute many instructions while waiting for external event to occur
- Can alternate among processes without anyone noticing

Measurement Challenge

How Much Time Does Program X Require?

- CPU time
  - How many total seconds are used when executing X?
  - Measure used for most applications
  - Small dependence on other system activities
- Actual ("Wall") Time
  - How many seconds elapse between the start and the completion of X?
  - Depends on system load, I/O times, etc.

Confounding Factors

- How does time get measured?
- Many <u>processes</u> share computing resources
  - Transient effects when switching from one process to another
  - Suddenly, the effects of alternating among processes become noticeable

Processes

Def: A *process* is an instance of a running program.

- One of the most profound ideas in computer science.
- Not the same as "program" or "processor"

Process provides each program with two key abstractions:

- <u>Private address space</u>
  - Each program seems to have exclusive use of main memory.
- <u>Logical control flow</u>
  - Each program seems to have exclusive use of the CPU.

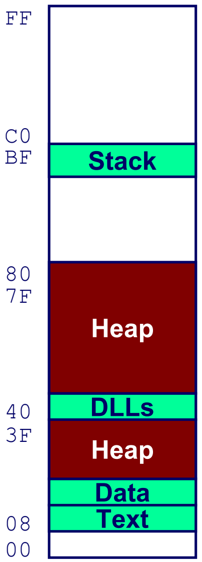
How are these Illusions maintained?

- Process executions interleaved (multitasking)
- Address spaces managed by virtual memory system

Private Address Spaces (1)

Upper 2 hex digits of address

Red Hat v. 6.2 ~1920MB memory limit



Linux Memory Layout

Stack

- Runtime stack (8MB limit)

Heap

- Dynamically allocated storage
- When call malloc, calloc, new

DLLs

- Dynamically Linked Libraries
- Library routines (e.g., printf, malloc)
- Linked into object code when first executed

Data

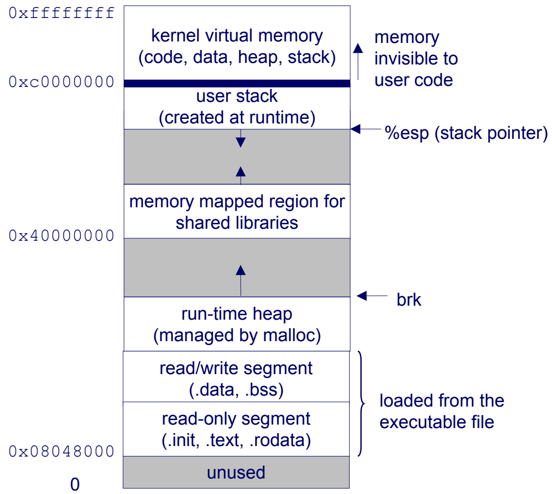
- Statically allocated data
- E.g., arrays & strings declared in code

Text

- Executable machine instructions
- Read-only

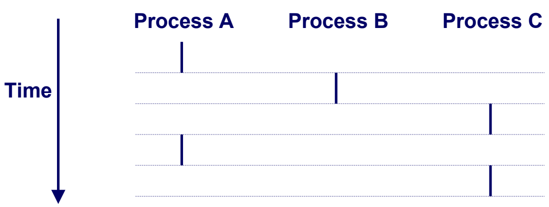
Private Address Spaces (2)

Each process has its own private address space.



Logical Control Flows

Each process has its own logical control flow



Altering the Control Flow

Up to Now: two mechanisms for changing control flow:

- Jumps and branches
- Call and return using the stack discipline.

Both react to changes in program state.

Insufficient for a useful system

- Difficult for the CPU to react to changes in system state.
  - Data arrives from a disk or a network adapter.
  - Instruction divides by zero
  - User hits ctl-c at the keyboard
  - System timer expires

System needs mechanisms for "exceptional control flow"

Exceptional Control Flow

- Mechanisms for exceptional control flow exists at all levels of a computer system.

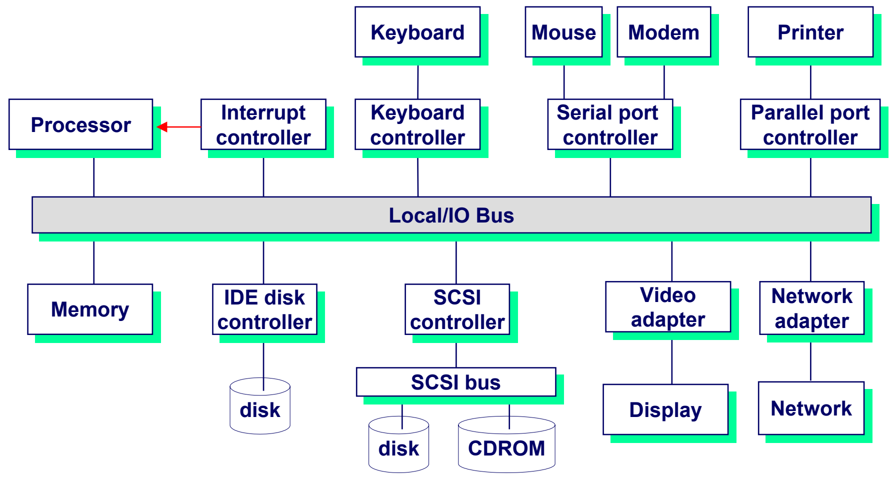
Low level Mechanism

- Exceptions
  - change in control flow in response to a system event (i.e., change in system state)
- Combination of hardware and OS software

Higher Level Mechanisms

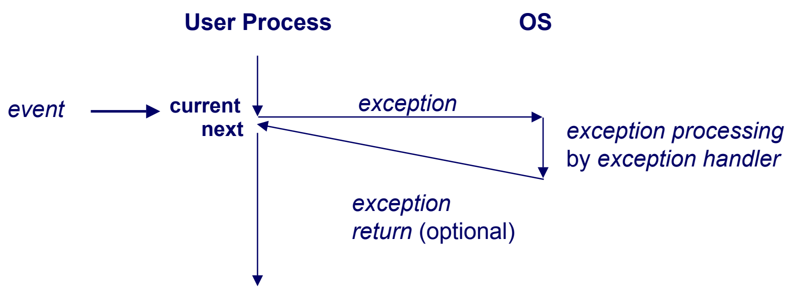
- Process context switch
- Signals
- Nonlocal jumps (setjmp/longjmp)
- Implemented by either:
  - OS software (context switch and signals).
  - C language runtime library: nonlocal jumps.

System context for exceptions



Exceptions

An *exception* is a transfer of control to the OS in response to some *event* (i.e., change in processor state)



Asynchronous Exceptions (Interrupts)

Caused by events external to the processor

- Indicated by setting the processor's interrupt pin
- *handler returns to "next" instruction.*

Examples:

- I/O interrupts
  - hitting ctl-c at the keyboard
  - arrival of a packet from a network
  - arrival of a data sector from a disk
- Hard reset interrupt
  - hitting the reset button
- Soft reset interrupt
  - hitting ctl-alt-delete on a PC

Synchronous Exceptions

Caused by events that occur as a result of executing an instruction:

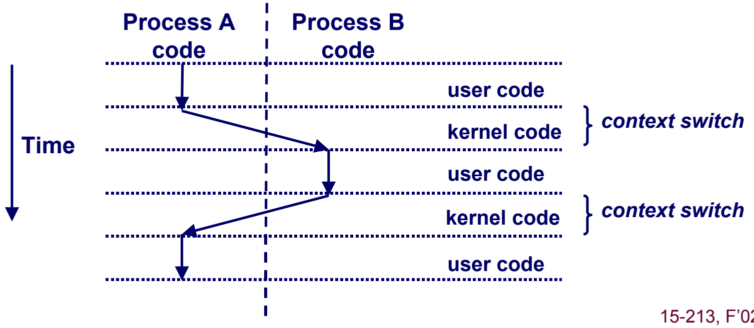
- Traps
  - Intentional
  - Examples: system calls, breakpoint traps, special instructions
  - Returns control to "next" instruction
- Faults
  - Unintentional but possibly recoverable
  - Examples: page faults (recoverable), protection faults (unrecoverable).
  - Either re-executes faulting ("current") instruction or aborts.
- Aborts
  - Unintentional and unrecoverable
  - Examples: parity error, machine check.
  - Aborts current program

Context Switching

Processes are managed by a shared chunk of OS code called the *kernel*

- Important: the kernel is not a separate process, but rather runs as part of some user process

Control flow passes from one process to another via a *context switch.*



"Time" on a Computer System

real (wall clock) time

= **user time** *(time executing instructions in the user process)*

= **system time** *(time executing instructions in kernel on behalf of user process)*

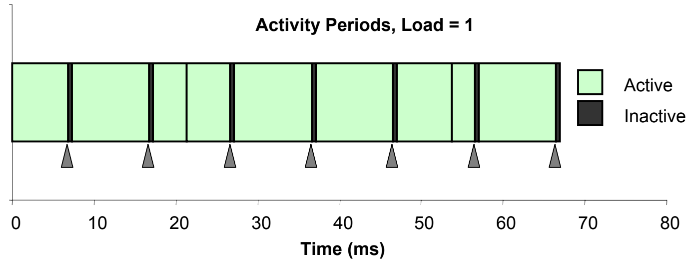
= **some other user's time** *(time executing instructions in different user's process)*

\+ + = real (wall clock) time

We will use the word "time" to refer to user time.

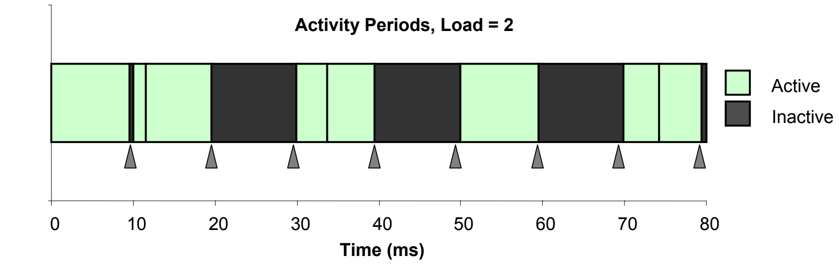
cumulative user time

Activity Periods: Light Load



- Most of the time spent executing one process
- Periodic interrupts every 10ms
  - Interval timer
  - Keep system from executing one process to exclusion of others
- Other interrupts
  - Due to I/O activity
- Inactivity periods
  - System time spent processing interrupts
  - ~250,000 clock cycles

Activity Periods: Heavy Load



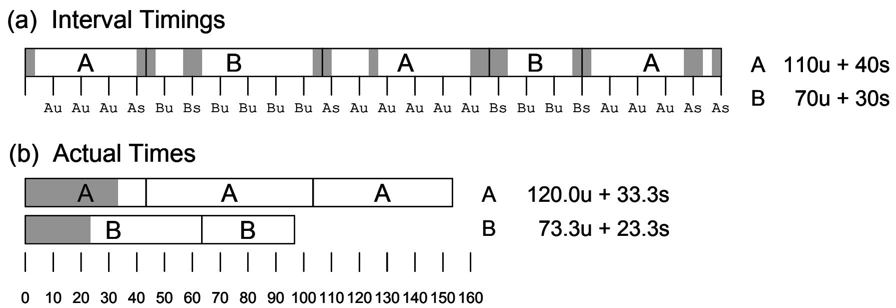
- Sharing processor with one other active process
- From perspective of this process, system appears to be "inactive" for ~50% of the time
  - Other process is executing

Interval Counting

OS Measures Runtimes Using Interval Timer

- Maintain 2 counts per process
  - User time
  - System time
- Each time get timer interrupt, increment counter for executing process
  - User time if running in user mode
  - System time if running in kernel mode

Interval Counting Example

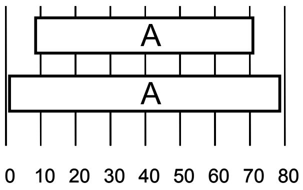


Unix `time` Command

```
time make osevent
gcc -O2 -Wall -g  -march=i486 -c clock.c
gcc -O2 -Wall -g  -march=i486 -c options.c
gcc -O2 -Wall -g  -march=i486 -c load.c
gcc -O2 -Wall -g  -march=i486 -o osevent osevent.c . . .
0.820u 0.300s 0:01.32 84.8%     0+0k 0+0io 4049pf+0w
```

- 0.82 seconds user time
  - 82 timer intervals
- 0.30 seconds system time
  - 30 timer intervals
- 1.32 seconds wall time
- 84.8% of total was used running these processes
  - (.82+0.3)/1.32 = .848

Accuracy of Interval Counting



Minimum

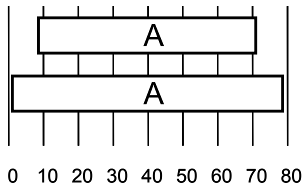
Maximum

- Computed time = 70ms
- Min Actual = 60 + ε
- Max Actual = 80 – ε

Worst Case Analysis

- Timer Interval = δ
- Single process segment measurement can be off by ±δ
- No bound on error for multiple segments
  - Could consistently underestimate, or consistently overestimate

Accuracy of Int. Cntg. (cont.)



Minimum

Maximum

- Computed time = 70ms
- Min Actual = 60 + ε
- Max Actual = 80 – ε

Average Case Analysis

- Over/underestimates tend to balance out
- As long as total run time is sufficiently large
  - Min run time ~1 second
  - 100 timer intervals
- Consistently miss 4% overhead due to timer interrupts

Time of Day Clock

- Unix `gettimeofday()` function
- Return elapsed time since reference time (Jan 1, 1970)
- Implementation
  - Uses interval counting on some machines
    - » Coarse grained
  - Uses cycle counter on others
    - » Fine grained, but significant overhead and only 1 microsecond resolution

```
#include <sys/time.h>
#include <unistd.h>

  struct timeval tstart, tfinish;
  double tsecs;
  gettimeofday(&tstart, NULL);
  P();
  gettimeofday(&tfinish, NULL);
  tsecs = (tfinish.tv_sec - tstart.tv_sec) +
      1e6 * (tfinish.tv_usec - tstart.tv_usec);
```


Measurement Summary

Timing is highly case and system dependent

- What is overall duration being measured?
  - > 1 second: interval counting is OK
  - << 1 second: must use cycle counters
- On what hardware / OS / OS version?
  - Accessing counters
    - » How `gettimeofday` is implemented
  - Timer interrupt overhead
  - Scheduling policy

Devising a Measurement Method

- Long durations: use Unix timing functions
- Short durations
  - If possible, use `gettimeofday`
  - Otherwise must work with cycle counters
  - K-best scheme most successful