Jan 20, 2005
Lecture 2: Multiprogramming OS

February 17, 2005

1 Review

"OS mediates between hardware and user software"

QUIZ: Q: What is the most important function in an OS? A: To support multiprogramming

REMARK: A somewhat more abstract answer, given in sections 1.1.1, 1.1.2 of [Tanenbaum] gives two functions: (a) to provided an Extended (Virtual) Machine such as through an API, and (b) to serve as a Resource Manager.

We might say that our "multiprogramming" answer involves both these aspects.

Q: What is a process? A: "process" = program executing in time... associated with address space registers

Q: An OS provide services. How does a user access these services? A: Make system calls.

MECHANICS of a System Call, e.g., READ. [Tanenbaum p.45]

User Program loads the arguments and calls a system library READ routine This routine loads the code for reading and TRAPS CPU switch to kernel mode, and executes at fixed address in the kernel. CPU returns to the library READ routine Library READ returns to user program.

KEY FUNCTION: provides multiprogramming environment

Usefulness of multiprogramming: I/O bound vs CPU bound processes User processes vs Kernel processes When useful?

What is needed for multiprogramming? Process management (process table) Memory management Interrupts (and ability to turn on/off interrupts) ...

2 Memory Management for Multiprogramming

Requirements: to isolate processes from each other, to isolate processes from kernel, to relocate programs.

Relocation problem: when a program is compiled, it must be assumed that each instruction is placed in sum location, and the locations conventionally start

at 0. When this program is LOADED, it will begin at some address determined at runtime. Also, a certain amount of space is allocated for data.

THE CDC MODEL:

- Each process occupies a contiguous chunk of memory.
- THIS CALLS for a BASE REGISTER and a LIMIT REGISTER: Limit register tells the size of the program+data. Chunk = [base, base + limit] These registers are protected from users
- How does this modify our execution cycle? Each instruction or data fetch is checked to be within the limit, and then "offset" by base. But actually, it is done by the MMU.
- THIS illustrates –virtual address vs physical address. –context switching: the values of the register pair is a context!
- Drawback: we need to predict how much data is needed. So everyone overestimates.
- MODIFIED CDC MODEL: Have two pairs of Base+Limit registers. Pair 1 for program, Pair 2 for data. ADVANTAGE: processes can share program!

3 Processes

Our first programming effort will involve multi-processes. So...

We will examine processes in Unix and Windows!

How are processes created?

- At system initialization (foreground process, e.g., login shells), (background, e.g., email, http daemons)
- Users can create them (e.g., multiple window processes)

  E.g., ¿cat chap1.txt chap2.txt — grep process

  Current process creates two new process, one to run "cat" (which contenates 2 files), the other to run "grep" (to look for word "process").
- UNIX: "fork" (creates a clone of caller process! but the the child can next do some manipulation and ultimately execute a "execve" to do its job)

  BUT both have different address spaces.

  THEY CAN share resources like open files.
- Windoz: "CreateProcess" with 10 parameters! and lots of related functions.

Process Hierarchies

- Windoz: no hierarchy! The parent process has only one special advantage: it has the "token" or "handle" to control the child, but can give the handle away!
- Unix: from fork, get parent-child relation. ALL PROCESSES are descendents of the "init" process! The hierarchy cannot change except through termination!
- Unix: Process group – a process and all its descendents. If a signal is sent to one(or root of group?), it is delivered to all?

How are Processes terminated?

- Normal exit (voluntary),

  Error exit (voluntary),

  Fatal error exit (involuntary),

  Killed by another process exit (involuntary),

Process States and scheduler

- States: Running, Ready, Blocked.

  (Ready-suspended, blocked-suspended).
- Possible state transitions:

  running → blocked → ready.
- Names of transitions:

  create, terminate

  preempt, run

  block, unblock,
- (running → ready) and (ready → running)

  are done by process scheduler!
- Scheduler [p.142-3]. Round Robin is simplest preemptive scheduler:

  (1) Ready processes are in FIFO queue (ready queue)

  (2) Each process runs in a fixed quantum (e.g. 50ms)

  (3) Current process P runs till it BLOCKS, in which case it is moved to "blocked queue", or it finished its quantum, and put in end of "ready queue".

  (4) Next process at top of FIFO queue is next running.

Process Management

- Process Table (one entry/process)

- Each entry: PC, SP, memory allocation, status of open files, accounting, scheduling info, info for context switching.
- Each I/O Class (floppy, hard disk, timers, terminals,...) has a location called an **interrupt vector**. It has the address of the interrupt routine.

  If process 3 is running when a disk interrupt occurs: then HARDWARE causes process 3's PC, registers, etc to be pushed onto the current stack, and make CPU jumps to the interrupt vector.

  That is all the hardware does. From then on, the interrupt service procedure takes over.

  When done, returns to the scheduler.

What is a THREAD?

- A process can have SEVERAL threads of execution.

  The threads all SHARE the same address space.
- The reason for threads are similar to the ones for processes BUT:

  easy to create and destroy

  the need to share data

  useful when threads are I/O bound
- E.g. Word processor

  backup of file in background

  reformatting of file while we continue to edit the file
- Processes share resources, but threads are executional units.
- Thread can be implemented in kernel or user space [p.93ff.]

4 Busses

We did not manage to talk about these:
PC's have 8 BUSSES

- original IBM:

  ISA (ISA bridge ↔ various devices)
- successor to ISA:

  PCI (PCI bridge ↔ various slots, inc.ISA Bridge)
- New:

  cache (CPU↔Level 2 cache)

  local (CPU↔PCI bridge)

memory (PCI bridge↔ Main Memory)

SCSI

USB

IDE

- New: FireWire (IEEE 1394) (50 MB/sec) like USB but much faster (useful for camcorders, etc)

  Viewed as replacement for USB: (1.5MB/sec) for slow devices like keyboard, mouse.