CSE221 Lecture 2

Aronya Baksy

September 2024

1 TENEX - A time shared paging OS for the PDP-10

1.1 Design Goals

- Build a **state-of-the-art virtual machine**
  - Paged virtual address space ($\geq$ address space of processor)
  - Multiprocessing with appropriate IPC facilities
  - File system integrated into virtual address space
  - Make common operations available as single instructions (CISC)
- **Good human engineering**
  - Command language interpreter for common small programs as well as all subsystems/user programs
  - TUI with interrupt capability and full ASCII charset
  - Encourage cooperation among users and provide security against intrusion
- Be **implementable, maintainable, modifiable**
  - Modular software with clean interfaces
  - Debuggable and reliable software
  - Allow manual adjustment and be configurable without reassembly
  - Contain instrumentation for perf metrics

1.2 Hardware Changes

Only changes to PDP-10 h/w: adding address mapping (paging) device using current DEC modules and some changes to the PDP-10 processor.

1.2.1 The BBN Pager

- An interface between the memory bus and the PDP-10 processor
- Some notes on PDP-10:
  1. Word length: 36 bit
  2. Address length: 18 bit (hence 2 addresses fit in 1 word)
- Two levels of address translation:
  1. Associative registers on CPU: 9 high-order bits of the address and the request-type are compared in parallel with each register.
  2. Page table with 512 entries
  3. Locate in main memory
- Three other cases:
  - If mem addr referenced is not in core, protected, or nonexistent - page fault (trap)
  - Mem addr points to a shared page: the map contains a "shared" pointer to a system table which contains the page location

  - The page belongs to another process; in this case, the entry contains an "indirect" pointer to an entry in another page table.

- Shared pages with different virt addr but single physical location
- Processes can communicate using shared pages.
- Users share data using **copy-on-write mechanism** (each page has a dirty bit which produces a trap on a write reference and a system procedure that creates a pvt copy).
- **Core status table** in the memory maintains the activity of the pages in core memory table. The pager notes when and which processes have referenced a page, and whether the page has been written into.

1.3 The TENEX Virtual Machine

- User processes on TENEX run on a virtual machine with 256K addressable mem. Paging h/w traps all references and the *core manager* performs necessary I/O to load pages to core memory
- TENEX VM doesn't provide direct h/w access but more powerful instructions via *monitor routines*

1.3.1 Virtual Memory Structure

- Virtual memory map (aka page table) of 512 slots lists the contents of the virtual memory
- Content of map entry: physical address of virt mem page, type of access (r/w/x)
- Map slot can contain pointer to a private page, or an indirect ptr to a page in some other process

1.4 Job Structure

- Job - set of ≥ 1 hierarchical processes with - name of initiating user, a/c number, open files, hierarchical processes, attached devices
- IPC through shared memory (CoW pages), software interrupts or direct control (parent over child only)
- Pros: allows protection and sharing, allows "EXEC" to handle interrupts and terminals, allow programs to block on events, implement invisible debugging

1.5 Monitor Functions

- Processes can listen for interrupts from other processes or connected terminals for:
  - illegal memory references, processor overflow conditions, end-of-file, and data errors.
  - User hitting particular key(s) on terminal
- Retrieve system state (date/time, username)
- Save & restore program environment to restart stopped programs
- I/O conversions (int or float I/O, date to string etc.)
- Measuring functions are built into monitor routines. efficiency of scheduling, the core/CPU balance, and stats on running processes

1.6 Backward Compatibility

- Step 1: all TENEX monitor calls (except page traps) are handled by JSYS instruction (which is a custom instr. that allows context switching between user and monitor preserving the context of callee)
- Step 2: implement all DEC 10/50 system calls using TENEX instructions except for some system maintenance routines
- These routines are loaded in core memory, never seen by pure TENEX programs, and made available to the user space of 10/50 programs by the monitor
- The compat pkg is loaded into user space to:
  1. allow normal use of the interrupt system
  2. maintain it separately from the monitor
  3. protect the monitor from malfunction of the compat routines

1.7 TENEX File System

- Provide symbolic file-name management: translate filename to a file descriptor and then check access permissions
- File name consists of 5 fields: device name, directory name, file name, extension, and version number.
- Relationship between file and running program (which is attached to a job):
  - Directory attached to job is same as the one which owns the file
  - Directory attached to job is in the same group as the one that owns the file
  - NOTA
- Five access modes: directory listing, read, write, execute, append
- File I/O happens with processes in either thawed mode (any no. of readers and writers with no consustency guaranteed) or unthawed (either N readers or one writer).

1.8 Scheduling

- Share CPU equitably, prompt svc to interactive requests, maximise CPU usage by efficently using core memory, allow admin control of scheduling
- Each process is given a priority
- Balance set is the set of jobs with highest priority whose total working sets fit in core mem.
- Processes are grouped into queues: lower queues have lower priority but longer runtimes (this is shown to cause long-running processes to starve while short interactive processes consume all the CPU)
- Priority is assigned based on a long-running average ratio of CPU use to real time. While running, process priority decreases at rate C and while waiting, increases at rate C/N (N = no. of runnable processes)
- Escape clause: After a block wait of greater than minimum time, a process is given a short quantum at maximum priority.
- Admin control: admins can assign a fraction F to any process and the scheduler will attempt to give the CPU time C to that process such that $C/T > F$ where T is the real time since that process was last unblocked
- The scheduler maintains IDLE time, WAIT (time for which all procs wait for page fault svc), CORE (core mgmt overhead) and TRAP (time spent in dealing w pager traps)
- Also maintains no. of processes in balance set, core to disk transfers, terminal interactions, three running averages to indicate current load

$$A(T+t) = A(T)e^{-t/c} + N(1-e^{-t/c}) \tag{1}$$

for $c = 1, 5, 15$ mins and N = number of runnable processes

1.9 Core Memory Management

- The core system table (1.2.1) contains data used for mem management
- $PAV = 67ms$ is a system parameter.
- If process page faults more often than PAV then more pages need to be beought in to increase working set size (scheduler can schedule another process while pages are swapped in)
- If process page faults less often than PAV then core mgmt routines swap pages out of core mem using an LRU algorithm

2 The UNIX timesharing system

2.1 Goals

- Easy for programmers to write, test and debug programs, make it interactive
- Size and memory constraints lead to elegant and economical design
- Easy to maintain, re-write
- Programs available: assembler, linker/loader/compiler/symbolic debugger for C, `ed` text editor, maintenance/utility/recreation programs

2.2 The UNIX File System

- Text files are strings of text with new lines demarcated by `\n`.
- Binary programs are sequences of words as they appear in main memory

2.2.1 Directories

- Impose a structure on the whole file system
- Cannot be written to by unprivileged programs but can be read by anyone
- Root directory is the root of all directories in the file system
- System directories: one user directory per user, one directory for system *commands*
- Each directory is the child of exactly one other, hence all directories are arranged as a tree
- Files however can appear in multiple directories as *links* (symlinks). Hence file names are not tied to directories

2.2.2 Special Files

- Each I/O device supported by UNIX is associated with at least one device file present under the `/dev` directory
- Can read/write on these special files like any other, I/O requests simply activate the device
- Files for active disks and core memory are protected from indiscriminate access
- Advantages of the device as a file abstraction:
  1. Makes file I/O and device I/O similar at a high level
  2. Makes programming easy because device paths can be passed to programs that need file paths
  3. Protection mechanisms for regular files can be extended to devices

2.2.3 Removable File Systems

- The *mount* system request has 2 args: an existing standard file, and a direct access special file with an associated storage volume structured as an independent file system
- `mount` replaces a leaf of the hierarchy tree (the ordinary file) by a whole new subtree (the hierarchy stored on the removable volume).
- no links may exist between one file system hierarchy and another. This avoids complicated book-keeping when file systems are unmounted

2.2.4 Protection

- When a file is created, it is marked with the user ID of its owner
- Six bits associated with a file (r/w/x permissions for owner and other users).
- Seventh bit is the `setuid` bit, allows the caller of the setuid to temporarily take on the identity of the file owner when that file is run as a program
- The superuser (UID 0) is exempt from the usual constraints on file access

2.2.5 I/O Calls

- no distinction between "random" and sequential I/O, no logical record size imposed by the system.
- No user-visible locks for synchronization, no restriction on number of simultaneous readers/writers
- Not necessary because use cases do not call for single files manipulated by independent processes, not sufficient as they do not prevent confusion in all cases (e.g. two users simultaneously open a file in a text editor which makes its own copy of the file)
- Internal locks do exist to maintain the consistency of the FS

2.2.6 File System Implementation

- The directory entry for a file contains its name and a pointer called the i-number
- The i-number is an index into the i-table maintained in core memory, and the entry for a particular file is the i-node (i here means *index*)
- Inode for a file contains name, protection bits, disk address, size, time of last modification, number of links, and 2 bits for large files and special files
- Mount maintains a system table whose argument is the i-number and device name of the mount point file, and whose corresponding value is the device name of the indicated special file.
- Reads and writes appear synchronous and unbuffered to the user, while in reality there is a complex system of buffering (if lots of I/O is occuring then it is advantageous to do I/O in 512-byte chunks, which is the block size of the disk)
- Pros of i-node organization:
  - Simple relation between file name and its metadata
  - Simple verification of FS consistency by just scanning the i-table linearly

2.3 Processes and Images

- An image is a computer execution environment. It includes a core image, general register values, status of open files, current directory, etc.
- Process is an execution of an image. Processes remain in core memory until swapped out for a higher priority process
- Processes in core-memory consist of a text segment (shared between all processes that run the program), data segment and a stack
- New processes are created by `pid = fork()` which creates a child process containing an identical copy of the parent's core image and sharing all open files
- Processes communicate with each other using **pipes** which can be read and write the same way as files
- The `execute(file, args...)` replaces the caller's code and data with that from the called program, but retains open files, process relationships and current dir.
- The `pid = wait()` system call suspends the caller until one of its children has exited, and returns the PID of the exited child process
- `exit(status)` terminates a process, destroys its image, closes its open files. The *status* is available to `wait()` calls from parents

2.4 The Shell

- The command line interpreter for UNIX: reads lines typed by the user and interprets them as requests to execute other programs.
- Uses `read()` to read args, `fork()` to create a child process, `execute()` to execute the user's commands in that child, and `wait()` to wait for the child to finish unless the command contains an '&' character (bg process)
- The *init* program creates processes for each typewriter channel, and these processes wait for user to enter their login info before being ready to accept commands

2.5 Traps

- PDP-11 contains software traps to catch illegal memory accesses or unimplemented instructions
- Upon encountering some illegal process behaviour, the system stops the process and writes its core image to a file called *core*
- **Interrupt** signal generated by Delete key is used to halt infinitely-running or malfunctioning processes without producing a core-image dump
- The **quit** signal does the same thing but produces a core image dump