Lecture 2
O/S Design: Trends and Motivations

Today’s Talk will cover:
1. Review
2. O/S design History/Survey
3. UNIX Design Overview
4. Trends in Parallel Systems
5. O/S support for Distributed Systems/Networks

---

Computer - System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from the local buffers.
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.
Common Function of Interrupts

- Interrupt transfers control to the interrupt service routine, generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.

Common Function of Interrupts

- A trap is a software-generated interrupt caused either by an error or a user request.
- An operating system is interrupt driven.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - polling
  - vectored interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt.
I/O Structure

• After I/O starts, control returns to user program only upon I/O completion.
  – Wait instruction idles the CPU until the next interrupt.
  – Wait loop (contention for memory access).
  – At most one I/O request is outstanding at a time; no simultaneous I/O processing.

• After I/O starts, control returns to user program without waiting for I/O completion.
  – System call - request to the operating system to allow user to wait for I/O completion.
  – Device-status table contains entry for each I/O device indicating its type, address, and state.
  – Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Storage Hierarchy

• Storage systems organized in hierarchy:
  – speed
  – cost
  – volatility
• Caching - copying information into faster storage system; main memory can be viewed as a fast cache for secondary memory.
Storage Hierarchy Continued

- Software Protection
- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

Direct Memory Access (DMA) Structure

- Schema

![Diagram of CPU, Memory, I/O Devices, and I/O Instructions]

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.
This section considers the availability of tools.

**1st Generation: 1945 - 1955**

1st Generation computers were made of Vacuum tubes and plugboards. Computers were expensive and had limited peripherals, programs were entered by toggling switches on the control panel. Minimal O/S were used.

**2nd Generation: 1955 - 1965**

2nd Generation computers are characterized by Transistors /Batch Programming. Non-interactive peripheral access improved (i.e. card readers) but computers were still relatively expensive relative to user/programmer time. Access to the machine was restricted to operators, users submitted jobs to the operators.

**3rd Generation: 1965 - 1980**

IC's reduced computing costs and made interactive peripherals available. Computers are still expensive, and have to be shared. Time-Sharing permits many users to share the machine, and encouraged interactive applications.
Technological History of O/S Design

Tanenbaum calls Personal Computing fourth generation.

Users had dedicated machines and initially multitasking was not used. Simple networking was prevalent.

Parallel Computing: 1990-present
Transparent tools for interconnecting machines are prevalent.

Tightly Coupled Architectures
Coupling in parallel systems refers to how processors are connected. Processors sharing common memory are said to be tightly coupled. (figure 3)

Tightly coupled systems have limited scalability.

Loosely coupled processors have distributed memory.
Inter-processor communication latency is larger in a loosely coupled system than in a tightly coupled system, however such systems scale well.

Types of Applications

- Resource Utilization:
  - Computation intensive jobs
  - I/O intensive jobs
- Application Driven Needs:
  - Real time jobs
  - Fault Tolerant/High Availability
  - Virtual Machine
  - Multi User
  - Multitasking
  - Interactive Jobs
  - Batch Jobs
  - On Line Transaction Processing
  - Transparent Interconnection

Some well Known Operating Systems

We can characterize well known current O/Ss:

1. Apple's Mac OS
2. IBM's OS/2
3. UNIX/Linux/SunOs/AIX/IRIX
4. Microsoft Windows
5. QNX
6. IBM's VM
7. IBM's MVS
8. Inferno
MVS History and Goals
IBM developed the System/360 and System/370 and their O/Ss:
1. PCP - Principle control program
2. MFT - Multiprogramming with a fixed number of tasks
3. MVT - Multiprogramming with a variable number of tasks
4. MVS - Multiple virtual storage.

MVS is designed to Support:
1. High performance (I/O throughput)
2. On Line Transaction Processing (OLTP)
3. Maintain backward compatibility
4. Have high availability/fault tolerance
5. Support tightly coupled multiprocessing

MVS Jargon and Components
MVS has internals and an outer layer of support tools:

MVS Outer Layer
MVS has the following outer layers shown in:
1. Compilers, Link Editors, Loader
2. Error Recovery Management
3. Job Management - The command environment (shell):
   a) Interprets operator commands
   b) Read and write Job input data to peripherals
   c) Allocate I/O devices and notify operator of mount requests
   d) Convert the Job into tasks for task management
MVS Internals
Under the hood MVS has:
• Dispatcher --- Schedules tasks on processors
• Task Management --- process control
• Interrupt Handling
• Program Management --- Runs Programs (run time loader)
• Storage Management --- Virtual and real memory
• Systems resource Management --- Partitions resource among tasks
• Access Methods --- User I/O interface
• I/O supervisor --- Low level device access

MVS System Structure
1. Job Management
2. Access Methods
3. Program Mgmt.
4. Dispatcher
5. Interrupt Handlers
6. Storage Mgmt.
7. I/O Supervisor
8. Task Mgmt.
11. Compilers, Linkage Editors, Loaders

Unix System Structure

Figure 6
Unix - An Evolving O/S

Figure 7: (Now Ancient) History of UNIX

Some History

• 1969 Multics Abandoned by Bell Labs GE 645 Removed
• Thompson begins building UNIX file system on GECOS
• Ritchie and Thompson ported/extended UNIX for PDP 7 "Space Travel" --- Assembler/Command interpreter added
• 1970-71 Version 1 on PDP 11/20 Added multi-user support, process management and most major command utilities.

• 1972 Version 1 on PDP 11/20 --- C created (no structs or global variables yet).
• 1973 --- UNIX Rewritten in C
• 1975 --- UNIX Version 6 made publicly available (inexpensive)
• 1977 --- UNIX ported to Interdata 8/32 (eliminating many machine dependencies)
• 1979 --- UNIX System 7 released (first widely used UNIX Version)
UNIX Programming Environment

UNIX programming environment features:
- Text Editors
- Text Processing
- C/C++ compilers
- make utility
- Debuggers (dbx/sdb/adb/gdb)
- Profilers
- Compiler Construction Aids
  - (Lex and YACC)
- Source code version control
  - (sccs, rcs, cvs)
- Memory Leak checkers

UNIX Programming Philosophy

Program design philosophy:
1. Arrange each program to perform a single function.
2. Avoid extraneous output, another program might use it as input.
3. If possible, use or modify existing tools rather write a new one.
4. Create the design first, then start with a small prototype and add features incrementally.

UNIX System Concepts

Some Central Concepts to UNIX include:
1. The File System
   a) Every file is a sequence of bytes (characters) representing either a program or data. No record structure is imposed.
   b) A directory holds the names of other files or directories, hence the file system is hierarchical.
   c) Input or output devices are treated as special files using the standard file interface. Information is provided from/to the device directly.
UNIX System Concepts

2. Processes do all user work in UNIX.
   a) Process creation is done by copying, the original is the parent, the copy is the child.
   b) Parent and Child are identical except the parent may wait for the child to finish.
   c) A process may replace itself by running another program.

3. The Shell is the Unix command language.
   a) The shell executes command from a terminal or a file.
   b) Users can create commands using script files.
   c) Many commands use standard input or standard output.
   d) Pipes send the standard output from one process to the standard input of another.

Pipes, Processes, and I/O Redirection

Recall that a process is a program in execution.
In UNIX a process has access to the following files:
1. Standard Input (stdin) - The keyboard by default, but could be a file or the output of another process (pipe)
2. Standard Output (stdout) — Where normally generated output goes.
3. Standard Error (stderr) — Reserved for error messages (so they don’t get piped as input to another process).

Additional files can be read or written.
Redirection: sort < junk > /tmp/junk.srt
Pipe: "du -a -s sort -r -n more "

UNIX System Concepts
Spell Checking Using Filters

Steve Johnson constructed the following prototype of spell using filters. The sequence of actions is:

- Break into Words
- Sort and Display Differences
- Compare to Dictionary
- Error

![Diagram of spell checking process]

Figure 8

The program looks like:

- `prepare filename` #Remove Formatting
- `tr '[A-Z]' '[a-z]'` #Convert to lower case
- `tr -c '[a-z]' '/n'` #Separate into words
- `sort -u` #Sort step
- `comm -2 dict` #In dictionary?

Contributions of UNIX

Some UNIX Contributions to O/S design and practice include:

1. Simplicity
2. Portable implementation using a high level language
3. Uniform treatment of peripherals/files
4. Ease of I/O redirection
5. Flexibility/ease of extension
6. (Initially) System size
7. Good software development tools
8. Initially easily obtained source code
9. Interactive multitasking
Problems With UNIX

1. (Historically) Easy to crash the system due to limited error detection and recovery
2. Command names are not obvious
3. Documentation assumes you know what you are doing
4. Administration tools assume UNIX expertise
5. Limited system security
6. Code Bloat (particularly in the Kernel)
7. Filter model initially unintuitive to users (sometimes no output is generated)

Microsoft Windows NT

David Cutler was the chief architect of NT.

NT is a portable single user multitasking system with highly modular components, designed to support multi processing.

Figure 9: Structure of Windows NT
NT Components (Layers)

NT has the following layers:
2. Kernel: Provides basic O/S services not belonging in user space: scheduling, context switching, interrupt handling, etc...
3. Subsystems: Modules designed to provide specific functions within the O/S outside the Kernel.
4. System Services: Modules designed to provide functions for application level support

Parallelism and OOP

Models of parallelism employed:
1. Client Server Model
2. Threads
3. Symmetric Multiprocessing (SMP)

NT has some object oriented features, with:
1. Encapsulation
2. Objects and Instantiations (named instances have security)

Memory Speeds Compared

Figure 10:
Speed of various memory devices
Memory Management

Programmers and Users demand:
1. More Memory
2. Fast Memory Access
The von Neumann bottleneck: Memory speed limits systems performance.
Ways of improving memory performance:
1. Parallelize Memory Access - Possible Asynchrony

Memory Management

2. Hierarchical Memory — Has two common approaches:
a) Large amounts of cheap slow memory extend available memory.
b) Small amounts of expensive fast memory improve systems performance.

Paralleling Memory Access

Memory tends to be written sequentially. The cost of sequential memory access can be described:
\[ T_A = T_L + T_C \]

Where:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_A )</td>
<td>Total Memory access time</td>
</tr>
<tr>
<td>( T_L )</td>
<td>Memory latency (startup cost)</td>
</tr>
<tr>
<td>( T_C )</td>
<td>Completion Time (service cost)</td>
</tr>
</tbody>
</table>
Paralleling Memory Access

A technique to improve memory access is to initiate several memory accesses in parallel when so that the processor can continue operating while waiting for a memory operation to finish. Typically this is done by distributing sequential memory addresses across different memory units.

Hierarchical Memory

Hierarchical memory was first used in the Atlas operating system in 1962. (figure 11)

References
