Agenda

- Review and Introduction
- Principles of System Architectures
- System Types
- OS Examples
- Middleware (not in this course)
- Application Systems
Review and Introduction
Principles of
System Architectures

Monolithic Systems
Layered Systems
Kernel Systems
Micro-Kernel Systems
Principles of Structuring Systems

- Monolithic system (little structure)
- Layered system (e.g. THE, MULTICS)
- Kernel based system
  - Traditional (monolithic) kernel
  - Object oriented kernel
  - Extensible kernel
- Component- or server-oriented systems
  - Microkernel based
Monolithic Systems

- Mozilla
- Emacs
- Scheduler
- Virtual Memory
- Device Driver
- Device Driver

Hardware
Analysis: Monolithic Systems

- **Advantages**
  - Easy access to all system data (they are all shared)
  - Cost of module interaction is low (procedure call)
  - In principle extensible, but in practice **NOT**, e.g. ∃ an “extension industry” for Mac & PALM OS

- **Disadvantages**
  - No protection between system and applications
  - Not particularly stable nor robust
  - Adding extensions → **unpredictable** results
...can we imply some ordering within this structure?
Layering Principle

- System is divided into many layers (levels)
  - Each layer is built on top of lower layers
  - Bottom layer (layer 0) is hardware
  - Highest layer (layer N) is the user interface

- Each layer only uses functions (operations) and services of lower-level layers
  - Advantage: modularity ⇒ simpler debugging/maintenance
  - Not always possible: Does process management lie above or below memory management?
    - Need to reschedule processor while waiting for paging
    - May need to page in information about tasks

- Important: Machine-dependent versus independent layers
  - Easier migration between platforms
  - Easier evolution of hardware platform
Example: Dijkstra’s THE OS

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The operator</td>
</tr>
<tr>
<td>4</td>
<td>User programs</td>
</tr>
<tr>
<td>3</td>
<td>Input/output management</td>
</tr>
<tr>
<td>2</td>
<td>Operator-process communication</td>
</tr>
<tr>
<td>1</td>
<td>Memory and drum management</td>
</tr>
<tr>
<td>0</td>
<td>Processor allocation and multiprogramming</td>
</tr>
</tbody>
</table>

- Structure of the THE operating system
- Main advantage: Each layer can be tested and verified independently
System of Nested Layers

Emulation of an OS (e.g. Unix) on top of another kernel

- Unix application
- Unix application
- Unix system call interface
- Unix Emulation
- Kernel interface
- Kernel
- Terminal driver
- Process manager
- Memory manager
- Communication Software
- Network Driver
Problems with Layered Systems

- Completely hierarchical layering can be too inflexible
- Real systems include call-cycles, but not too many
  - File system requires virtual memory services for its buffers
  - Virtual memory would like to use files for its backing store
- Reduced performance:
  - Each layer crossing has some overhead associated with it
Monolithic Kernels

System Structure

Application

... System Call

Application

Scheduler

Privileged Kernel

File System

Device Driver

Protocol Stack

Device Driver

Hardware
Analysis Monolithic Kernels

- **Advantages**
  - Well understood
  - “Good” performance
  - Sufficient protection between applications

- **Disadvantages**
  - No protection between kernel components
  - Not very extensible
  - Overall structure soon **too complicated**
    - Every kernel entity is intermixed
    - **∃** no clear boundaries between modules
Approaches tackling Complexity

- **Safe kernel extensions**
  - SPIN – safe programming language (Modula 3) at Univ. of Washington (UoW)
  - VINO – sandboxing (hardware protection) at Harvard
  - NOW (Berkeley)
  - Spring (SUN)
  - Scout (Uni. of Arizona)
  - Synthetix (OGI*)

- **Exokernel (MIT)**
  - Kernel offers multiplexing of raw HW
  - All other control is done at application level

- **Microkernels

*Oregon Graduate Institute
Extensible Kernel Systems

Rating:
Pro: Saves memory by only loading modules (e.g. drivers) that are needed
Pro: Makes it easier to develop new kernel code outside of the main tree
Con: Once loaded, module can destroy kernel if not running in a sandbox
Micro-Kernel Systems

- Application
- File System
- Task Server
- Spooler
- Protocol Stack
- Disc Driver
- External Pager
- Printer Driver
- Network Driver

Threads, Address Spaces & Basic IPC
Micro Kernel

Hardware

Only very few microkernel functions (e.g. L4 offers 7 different ones)! Why?
Pros ↔ Cons of Micro-Kernels

Pros:
- Easier to test/prove/modify
- Improved robustness & security (each system component in user level is protected from itself)
- Improved maintainability
- Coexistence of $n$ APIs
- Natural extensibility (add a new server, delete a no longer needed old server)

Cons:
- Additional decomposing
- Expensive to re-implement everything using a new model
- Low performance due to communication overhead
- Bad experiences with IBM's Workflow (91-95):
  - 1 kernel for OS/2, OS/400, AIX
  - Based on Mach 3.0
  - ~ $2 billion loss
Arguments against Micro-Kernels

1. Low Performance

2. Still Large and Inflexible Vehicles

Comment: True for micro-kernels of 1st generation, e.g.

- Mach (Bershad, Rashid, CMU, OSF)
- Chorus (Rozier, Gien, INRIA, Chorus)
- Amoeba (Tanenbaum, Vrije Universiteit)
- L3 (Liedtke, GMD)
MACH “Micro-Kernel” and OS

- User program
- File Manager
- Memory Manager
- Terminal I/O

Scheduler, Message Transfer, Basic I/O, Memory

Hardware

User mode
Kernel mode
2nd Generation of Micro-Kernels

- L4 (GMD, IBM, U of Karlsruhe)
- Exokernel (MIT)
- EROS (U of Pennsylvania, Johns Hopkins)
- Flux (U of Utah)
- PARAS (C-DAC, India)
- Pebbles (Bell Labs)
- QNX (Quantum Software Systems)
- GNU Hurd (Free Software Foundation)

have shown **far better performance**

If you are really interested in OS affairs read the related overview papers of these micro-kernels and take the course [micro kernel construction](#) next summer term
Arch. Costs I (Traditional Kernel)

Application

User Mode

write_file(file a, record xyz)

Kernel Mode

Monolithic Kernel

File Server

get_disk_block()

Disk Driver

Both modules can share common data
Architectural Costs II

- 1 System Call
  - (including kernel entrance + leaving)

- 1 Procedure Call + 1 Return
  - (both within kernel address space, potential sharing of data, buffers, ...)

© 2008 Universität Karlsruhe(TH), System Architecture Group
Architectural Costs: Microkernel

Application

write_file(file a, record xyz)

File Server

Disk Driver

µ-Kernel

User Mode

Kernel Mode

SVC-Interface
Architectural Costs (Microkernel)

1. µK ipc call to file server (“write_file …”), including
   - µK Entrance/Exit
   - message transfer
   - address-space switch to server

2. µK ipc call to disk driver (“write_block …”), costs see (1.)

3. µK ipc reply to file server (“done …”), costs see (1.)

4. µK ipc reply to application (“done …”), costs see (1.)

Result:
Exchanging messages implies additional overhead compared to a monolithic kernel, which can use shared memory for that purpose.
Challenge

Regular OS operations (system calls) can imply many communications
Conclusion

Micro-kernel operations have to be:

fast, faster, fastest !!!

... and additionally we need a fast micro-kernel entrance and micro-kernel exit

⇒ You have to know your HW very well
Microkernel Based System Performance

AIM VII Benchmark (TU Dresden '97)

- Blue line: Linux
- Green line: MkLinux
- Red line: L4Linux
Summary

- OS can be quite complicated
- The structure of an OS dominates, at least influences the result of whole system
  - *What is the ideal OS structure?*
- Well, it depends where you visit the OS course
  - **CMU et al.: Extensible kernel based systems**
  - **KIT et al.: Micro kernel based systems**
System Types

Database Systems
Operating Systems
Real Time Systems
Middleware
Application Systems

Our Focus
Goal of this Section

- Try to find out common features, e.g.
  - problems
  - requirements
    - concurrency
    - performance
    - robustness
  - ...

of the different types of systems
Database Systems

Example: Database with flight information

Clerks at remote terminals

Booking system

Network
Requirements for DB Systems

- Support separate concurrent activities
- Support concurrent accesses and updates to the database without interference ⇒
  - We need synchronization features
  - Confidence in consistent state of the database
  - Solution: **TRANSACTIONS**
- Results of a transaction are recorded permanently and securely before a client is told that her/his request has completed

See next summer term
Operating Systems

System Types

- Single-User System
- Multi-User System
- Virtual Machine
- Distributed System (e.g. Client-Server)
Single User OS

- **Devices**
  - disk
  - printer

- **Memory**
  - user’s program(s)
  - operating system
  - processor (cpu)

- **User’s I/O devices**
  - screen
  - keyboard
Multi User OS

Devices
- disk
- disk
- disk
- disk
- tape
- tape
- printer
- printer
- printer

Memory
- user’s programs
- operating system
- processor (cpu)
- processor (cpu)

Users’ I/O devices
- screen
- keyboard
- screen
- keyboard
- screen
- keyboard
- screen
- keyboard

System Types

© 2008 Universität Karlsruhe(TH), System Architecture Group
Multiprogramming Systems

- I/O routines supplied by the system
- Memory management – the system must allocate the memory to several jobs
- CPU scheduling – the system must choose among several jobs ready to run
- Allocation of devices
Time-Sharing Systems

- CPU is multiplexed among several jobs that are kept in memory and on disk
  - CPU is allocated to a job only if the job is in RAM
- A job swaps in and out of RAM to the disk
- On-line interaction between user & system
  - When OS finishes execution of one command, it waits for the next “control statement” from the user’s keyboard
- On-line system must be available for users to access data and code
Desktop Systems

- PCs – computer system dedicated to a single user
- I/O devices – keyboards, mice, display screens, small printers
- Provides user with convenience and responsiveness
- Can adopt technology developed for larger OSs
  - Often individuals have sole use of computer and do not need advanced CPU utilization or protection features
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux, …)
Requirements for OSes

- Separate activities on application level
- Even separate activities within OS
  - Separate system activities might work together (e.g. file system, spooler)
- Resource management
- Synchronization and communication features, e.g. reading/writing shared data
- Protection and security
Virtual Machines

*Notion also used in context of Java

Virtual machines are one of our current research topics.
Advantages of Virtual Machines

- A VM provides complete protection of resources since each VM is isolated from all other VMs. This isolation permits no direct sharing of resources.

- VM = perfect for OS research + development
  - development is done on the VM ⇒
  - does not disrupt normal system operation

- Implementing an VM is a bit tricky, because you have to provide an exact duplicate of the underlying physical machine ⇒
  - Tools to support, e.g. "afterburner" from Joshua LeVasseur
Example 1: VMware

<table>
<thead>
<tr>
<th>application</th>
<th>application</th>
<th>application</th>
<th>application</th>
</tr>
</thead>
<tbody>
<tr>
<td>guest operating system (free BSD) virtual CPU virtual memory virtual devices</td>
<td>guest operating system (Windows NT) virtual CPU virtual memory virtual devices</td>
<td>guest operating system (Windows XP) virtual CPU virtual memory virtual devices</td>
<td></td>
</tr>
</tbody>
</table>

virtualization layer

host operating system (Linux)

hardware

CPU memory I/O devices
Example 2: Java VM

- Java program .class files
- class loader
- Java interpreter
- host system (Windows, Linux, etc.)
- Java API .class files
Local Client Server Model

Note: Microkernels are an appropriate base for an “LCSM”
Real-Time Systems (RTS)

- Controlling dedicated applications, e.g.
  - measuring scientific experiments
  - representing medical images to a surgeon
  - controlling industrial robots

- Timing constraints

- Two types of RTS
  - hard real time system
  - soft real-time system
Real-Time Systems (2)

- **Hard real-time**
  - Persistent (secondary) storage limited or absent, data stored in short term memory or read-only memory (ROM) or MEMS
  - Must fulfill deadlines, otherwise disaster

- **Soft real-time**
  - Limited use in industrial control of robotics
  - Combinable with time-sharing systems
  - Useful in applications (multimedia, virtual reality) requiring tight response times
Real-Time Systems (3)

- **Process Control (hard real time requirements)**
  - Responds to an “alarm signal” in predictable time, otherwise an expensive disaster or even a catastrophe might happen,
    - Fuel injection control, ABS or Air Bag in a car
    - Flight control, nuclear power station, military equipment, …
  - Signals may arrive in some unpredictable way

- **Multimedia (soft real time requirements)**
  - Responds to a “signal” in a more or less specified time, otherwise something unpleasant will happen (not a catastrophe)
    - Unsynchronized audio and video signals
  - Signals tend to occur more or less periodically
Process Control Systems

- Actor
- Sensor
- Network

External “Process” to be controlled

e.g. a patient in an emergency room
Component Computer

Sensor Interface

Actuator Interface

Monitoring:
gathers data
analyses dataeports results

Controlling:
responds to alarms
responds to management

Network Interface

Network
Multi Media Systems

- Videophone, -mail or -conferencing
- Multimedia docs (museum catalogue, video archive, etc.)
Req. for Real-Time Systems

- Support separate concurrent activities (some are periodic, some are sporadic, i.e. unpredictable)

- Requirements for the scheduling of activities (meet deadlines, ensure quality of service)

- Support for teamwork in some activities (achieving a common goal)
Handheld Systems

- Personal Digital Assistants (PDAs)
- Cellular telephones

Issues

- Limited memory
- Slow processors
- Small displays
Computing Environments

- Traditional computing
  - PCs, servers, limited remote access

- Web-Based Computing
  - Client/Server and web services, convenient remote access, location-less servers

- Embedded Computing
  - Most computers (auto engine controllers, microwaves)
  - Up to now limited OS features
  - Little or no user interface, remote access
Example OS

Unix
http://www.cl.cam.ac.uk/~smh22/os-net.html#unix

Windows
http://www.sysinternals.com/

Linux
http://www.linuxhq.com/guides/TLK/tlk.html
An OS interface independent of implementation is **POSIX**: Portable Operating System Interface based on Unix.
Windows NT Architecture

Figure 2.13 Windows 2000 Architecture
Application Systems
Why Application Systems?

- Without an application-specific controlling interface, you can only abort and restart a simulation (via usual system commands such as kill).

- With an application-specific controlling interface, you can stop, protocol an endlessly running simulation, and resume it later with corrected input data.

⇒ Paradigms, principles, policies, and mechanisms of OSes can also be used within other systems.

⇒ Try to parallelize your applications whenever you can.
Challenge of Parallel Computing

- *How do we get a speedup of f(N) on an N-way multi-processor*?  
  - Software must be parallelizable

- Speedup can refer to
  - **Turnaround time**: length of time to complete a single task
  - **Throughput**: rate at which tasks are completed

---

1. In practice speedup is quite limited
Parallelization Theory

- Amdahl's Law predicts speedup on a parallel machine:

\[
\text{speedup} = \frac{1}{F + \frac{(1-F)}{N}}
\]

- \( N \): number of processors
- \( F \): fraction of computation that is sequential
Parallelization in Practice

- Predicting performance is difficult because there are many flavors of parallelism
  - Multiple processors (in a SMP)
  - Multi-core processors
  - Multi-threaded processors (Hyper threading)
  - Clusters of machines

- Running and measuring your application software is the only way to know for sure
Parallel Application

Manager of parallel computations
(initiating algorithm, synchronizing results)

Copy of code to execute data

Data flow

Control flow

Copy of code to execute data

Copy of code to execute data

Copy of code to execute data

Copy of code to execute data

Part of data

Part of data

Part of data

Part of data

Hint: Design a simple model to show the potential performance improvements.
Parallel Application (Pipeline)

First input

First phase of code

Data 5

Second phase of code

Data 4

Third phase of code

Data 3

Fourth phase of code

Data 2

Intermediate outputs/inputs

Data 1

Last output
Req. for Parallel Applications

- Support for separate activities including controlling them (i.e. create, start, stop activities)
- Support for synchronizing activities and/or allowing cooperation on shared data
- Support for communication mechanisms
- ...

Middleware and “Upperware” Systems
Ultimate Design Goal

- Only few dedicated systems can be optimized to achieve high performance
- In general, we must live with compromises that are suitable (due to conflicting requirements)
- However, any system we are modeling or implementing has to be correct

Correctness is mandatory

- Although correctness is hardly achievable, we have to work hard to make a system “as correct as possible”