Operating Systems

Eike Ritter
University of Birmingham

2-hour Lecture Course in Semester 1 2008/2009
Monday, 2pm
Wednesday, 12.00 noon

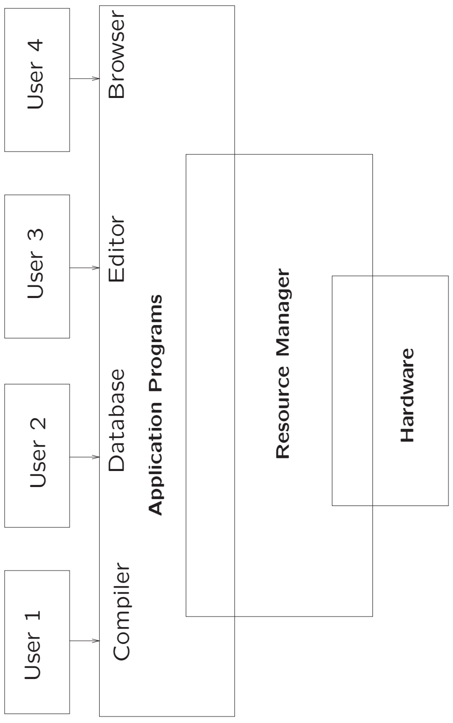
Examination:
For normal module (06 15258): 100% exam
For extended module (06 15257): 80% exam, 20% extra coursework

References

A. Silberschatz, P.B. Galvin, G. Gagne. *Operating Systems Concepts*, Seventh edition, Addison Wesley, 2004.

A.S. Tanenbaum. *Distributed Operating Systems: Principles and Paradigms.* Second Edition, Prentice-Hall 2007.

A.S. Tanenbaum, *Modern Operating Systems*. Third Edition, Prentice-Hall, 2007.



1 What is an Operating System?

Functions:

1.) Implement multi-user programming:

- several processes ready to be executed; OS chooses next one
- Have to simulate parallelism on a sequential machine
  ⇒ Must avoid Starvation, Deadlock and achieve fairness.
- Protection of processes from each other
  ⇒ Separation of logical and physical address spaces.

2.) Memory Management

Main memory is fast but expensive
Disc storage is slow but cheap

⇒ Only part of memory needed for process execution in main memory
⇒ OS manages memory allocation

View from process: One large address space (“virtual memory”).

3) Input/Output

OS manages highly complex interaction with I/O-devices:

real-time constraints have to be observed

4.) Distributed Computing

Data or programs can be on different computer
Aim: Transparency (same interface for local and remote access)
⇒ need to support protocols for file transfer, remote login as part of the OS

Examples of Operating Systems

- **Unix**: Started 1969 as Multi-user Time-sharing System
  Vital for development of the Internet in the 1980’s
- **Linux**: Unix derivative, started in 1991 when PC-HW was powerful enough to run Unix
- **MS-DOS/Windows 95/98/ME**: Started in 1980’s as OS for PC’s
  Restricted Power of 1980’s PC’s meant very limited functionality
- **Windows NT/2000/XP/Vista**:
  Re-implementation started in the 1980’s with extended functionality
- **Mac OS** OS with limited functionality for Apple Mac
- **Mac OS X** Unix-like re-implementation for Apple Mac

OS for embedded systems

Have variety of embedded systems
from special-purpose controllers to programmable chips
⇒ wide variety of OS’s to satisfy very different needs systems

Due to limited resources of embedded systems re-emergence of OS-issues of 1970’s and 1980’s

Hardware prerequisites

OS relies on hardware to ensure protection of processes from each other.

Need at least two different execution modi for hardware:

1. Kernel or Supervisor mode: Allows unrestricted access to all resources
2. User Mode: Only instructions not affecting other users are allowed; sanity checks are enforced.

Instructions allowed only in Supervisor mode are called privileged instructions.

Interrupts

OS requires feedback from hardware when operations are finished.

Standard mechanism: Interrupt:

- Hardware generates signal, which is transferred to processor
- Processor interrupts current activity
- Processor executes appropriate interrupt service routine
- Processor resumes previous activity

Short response time important, so interrupt service routines tend to be small

Interface to application programs

OS accessible only via specified procedures (system calls)

Execution of a system call:

- OS reads call parameters and checks appropriate privileges
- OS executes requested function in Supervisor mode
- OS returns result