WEEK 2

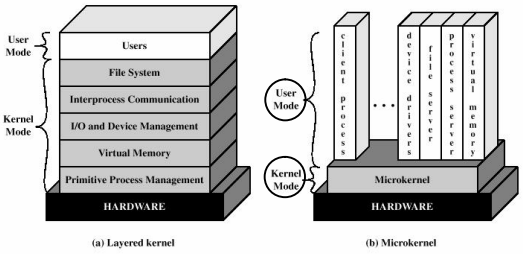
Operating Systems Examples and Process Management

Microkernels (4.3), SMP (4.2)
Examples (2.5 – 2.8)
Process management (Chapter 3)

1

Kernel

(Fig 4.10 [Stal 05])



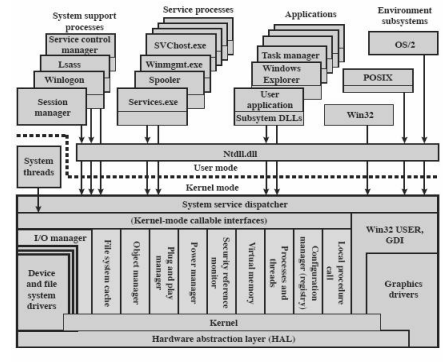
monolithic

2

WINDOWS

Ch 2.5 [Stal05]
(see also Tan01, Ch 11)

3



(kj:n peruspalvelut)

Figure 2.13 Windows 2000 Architecture [SOLO00]

4

WIN32 API palvelupyyntöjä

Fig 11-31 [Tane 01]

Win32 API function	UNIX	Description
CreateFile	open	Create a file or open an existing file; return a handle
DeleteFile	unlink	Destroy an existing file
CloseHandle	close	Close a file
ReadFile	read	Read data from a file
WriteFile	write	Write data to a file
SetFilePointer	lseek	Set the file pointer to a specific place in the file
GetFileAttributes	stat	Return the file properties
Lock		
Unlo		

API group	Description
Window management	Create, destroy, and manage windows,
Menus	Create, destroy, and append to menus and menu bars
Dialog boxes	Pop up a dialog box and collect information
Painting and drawing	Display points, lines, and geometric figures
Text	Display text in some font, size, and color
Bitmaps and icons	Placement of bitmaps and icons on the screen
Colors and palettes	Manage the set of colors available
The clipboard	Pass information from one application to another
Input	Get information from the mouse and keyboard

Fig 11-29 [Tane 01]

5

W2K Oliot

- W2K implementation is mostly based on objects
- Objects: files, processes, threads, semaphores, timers, windows, …
- Resource is handled as an object only when it could be shared or is accessed in user mode
- Object Manager

Fig. 2.13

- Objects in the kernel
  - microkernel control objects
  - dispatcher and synchronization objects

Tbl. 2.5
Tbl. 6.7

6

W2K Executive olioita

Type	Description
Process	User process
Thread	Thread within a process
Semaphore	Counting semaphore used for interprocess synchronization
Mutex	Binary semaphore used to enter a critical region
Event	Synchronization object with persistent state (signaled/not)
Port	Mechanism for interprocess message passing
Timer	Object allowing a thread to sleep for a fixed time interval
Queue	Object used for completion notification on asynchronous I/O
Open file	Object associated with an open file
Access token	Security descriptor for some object
Profile	Data structure used for profiling CPU usage
Section	Structure used for mapping files onto virtual address space
Key	Registry key
Object directory	Directory for grouping objects within the object manager
Symbolic link	Pointer to another object by name
Device	I/O device object
Device driver	Each loaded device driver has its own object

Fig 11-10 [Tan01]

7

LINUX

Example of a modern UNIX

Ch 2.8 [Stal05]
(see also Ch 10 [Tan01])

8

Linux

- Joint Internet-based development
  - Experts around the world
  - 1991 ->
- We are the home of LINUX
  - Linus Torvalds studied and worked at our department. Actually the earlier version of this course was the initial starting point in his development process
- Free Software Foundation & GNU Public License
  - Free distribution of the kernel code (C & assembler)
  - Free distribution of some (most) system software
    - GNU C, Gnome, KDE, Apache, Samba, ...
- Several commercial distributions
  - RedHat, SuSe, Debian, Mandrake, TurboLinux, jne...

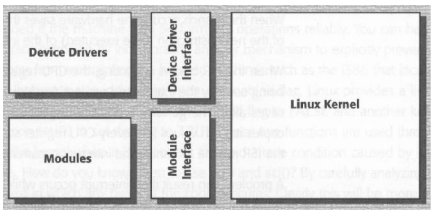
9

Linux

- Originally one monolithic kernel - configurable
  - Kernel is one big binary file
  - Fast and easy way of distributing information within the kernel
- Separate loadable modules
  - Everything need not be in the binary file
- Since it is open source, in addition to existing configurability and optimisations, you can modify the kernel based on your needs
- Runs on several different architectures
- LSB: Linux Standard Base

10

Linux: loadable modules



- Must be registered with the kernel
  - init_module(), delete_module(), ...
  - register_blkdev(), unregister_blkdev(), ...
  - register_filesystem(), unregister_filesystem(), ...

11

Linux: loadable modules

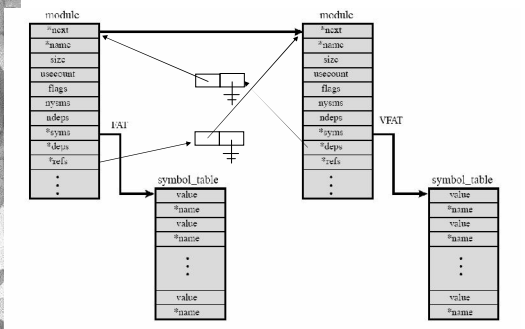


Fig 2.17 [Stal05]

12

Linux kernel components

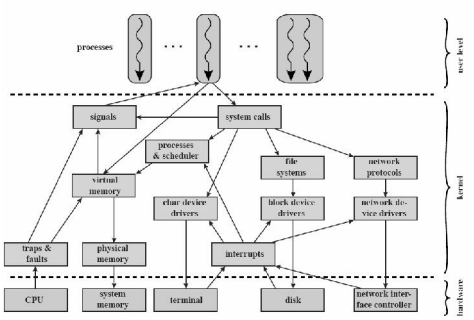


Fig 2.18 Stal 05]

Compare with W2000 in Fig. 2.13 [Stal05]

13

Linux: Information

Some books

- Bovet D.P., Cesati M.:
  Understanding the Linux Kernel. O'Reilly, 2nd ed., 2003.
- Beck M., Böhme H. & al. :
  Linux Kernel Programming. Addison-Wesley, 3rd ed., 2002
- Rubini A., Corbet J.:
  Linux Device Drivers. O'Reilly. 2001. 2nd ed.

Code

- http:// lxr.linux.no

14

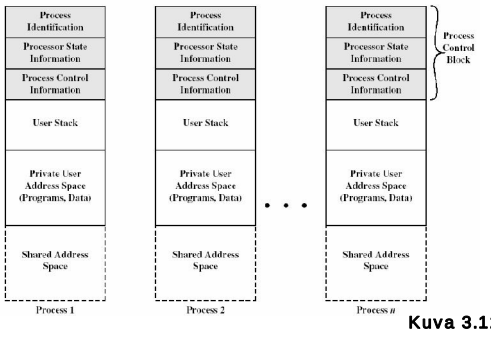
WEEK 2

Process management

Stallings, Chapter 3

15

Process



Kuva 3.12

16

Process

- Code = executable instructions
- Data = variables
- Stack = workarea
  - Parameter passing to subroutines/system calls
- Process Control Block, PCB
  = management information and structures

17

Process creation

- Create PCB
  - OS 'generates' a unique ID
- Allocate memory for the process
- Initiate PCB
- Link PCB to other structures
  - Place to Ready-queue, link to parent process, etc

18

Mode switch (vs. process switch)

- Interrupt
  - Change mode to kernel mode
  - Start executing OS as an interrupt handler
- Give CPU back to the interrupted process (no process switch)
- No need to copy all register contents to PCB
  - PC and PSW copied to stack during the call
  - Interrupt handlers stores to stack only the contents of registers it is going to use
- PCB unused and untouched
- Return:
  - **Copy registers from stack**
- In process switch more work to do
- Call dispatcher only if there is a reason to change process

19

Process switch

- Register values from CPU and stack to PCB
- Update counter fields etc.
- Update process state (Ready/Blocked/Exit...)
- Place to a queue according the state
- Choose next process to run
  - Ready ð Running
- Initiate MMU
- Register values from PCB to CPU

20

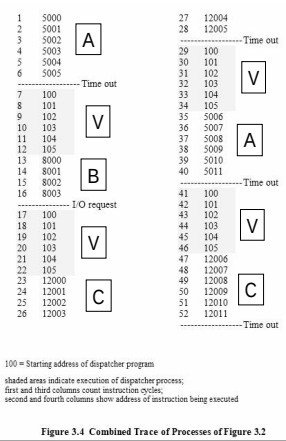


Figure 3.4 Combined Trace of Processes of Figure 3.2

A,B ja C processes
V Dispatcher

21

Process states

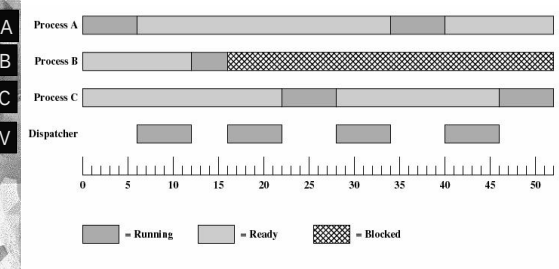
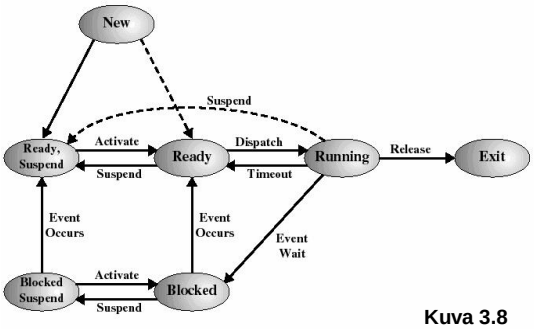


Figure 3.7 Process States for Trace of Figure 3.4

22

Process states

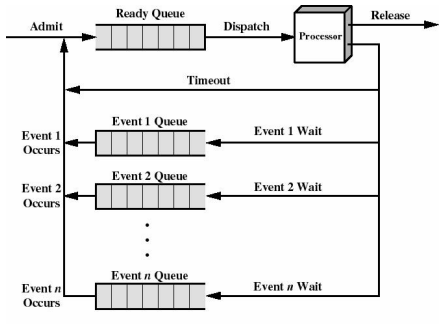


Kuva 3.8

23

Process queues

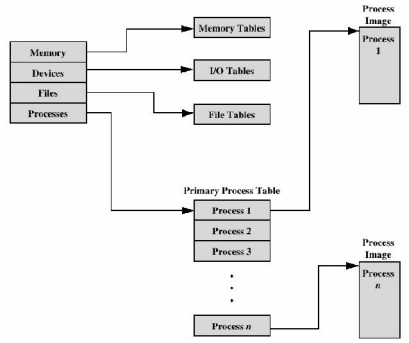
Kuva 3.7



24

OS structures

Kuva 3.10

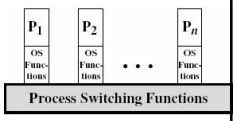


25

UNIX SVR4 Processes

(Fig. 3.15 (b) [Stal05])

- Most of the OS work is done in user mode within user process
- Only part of the OS work is done by specific OS processes in user mode partiallys



ks. Fig. 3.17 [Stal05]

- Priviledged process (in kernel mode) is not pre-emptable
  - It can loose processor only by requesting a resource that is cannot get just now
- Process 0 – boot
- Process 1 – init (mother of all)

26

UNIX: Process states and state transitions

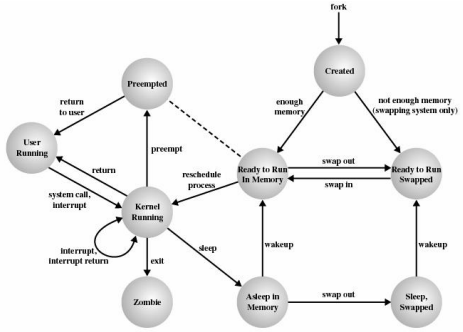


Figure 3.17 UNIX Process State Transition Diagram

27