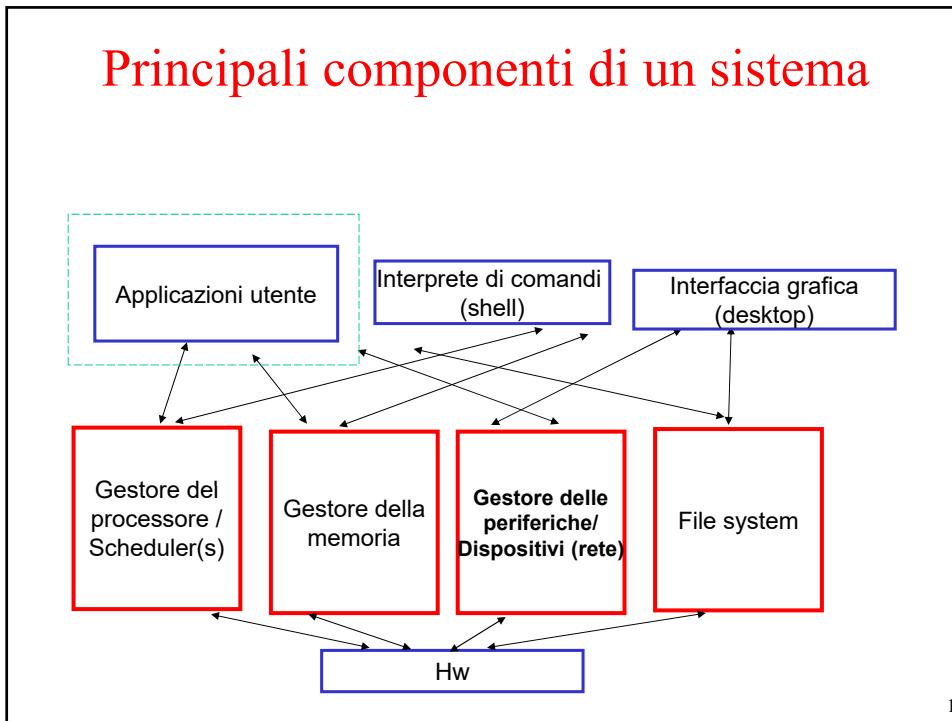


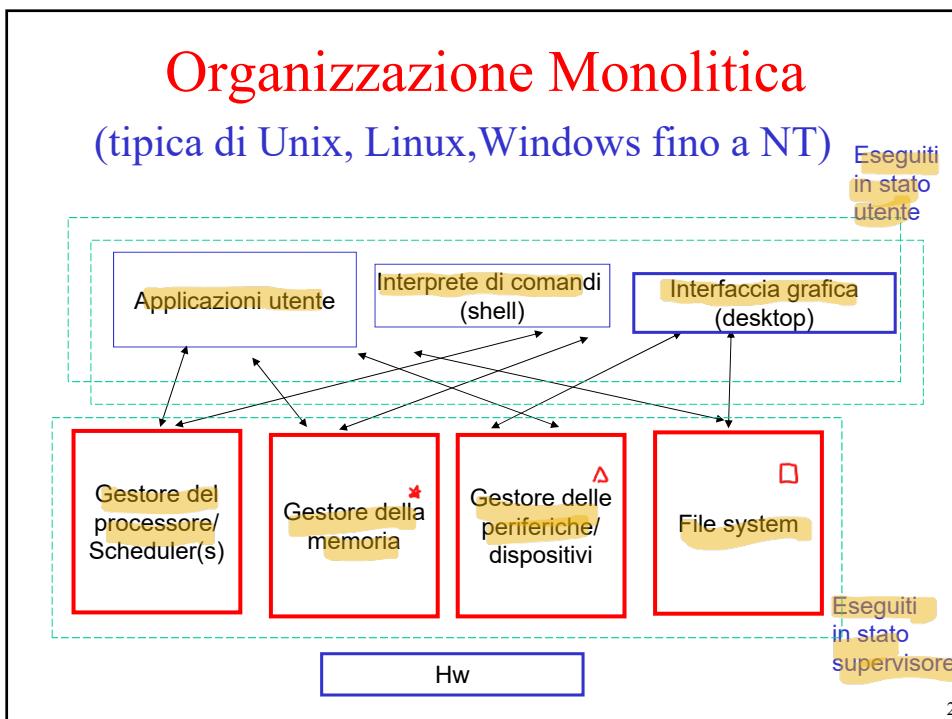
Principali componenti di un sistema



1

Prima distinzione: stato utente/supervisore Seconda distinzione: funzionalità

Organizzazione Monolitica (tipica di Unix, Linux, Windows fino a NT)



2

Una organizzazione a livelli (es. rete), ma questo non massimizza performance

* Altraverso il concetto di memoria virtuale, posso salvare su disco pezzi del processo invece che su memoria centrale. Però danneggia la performance. Gestione memoria è fondamentale.
MEMORIA VIRTUALE: Spazio di memoria che un processo ha a disposizione

Stato utente e stato supervisore

- Stato utente :
 - modalità di funzionamento dell'hw che permette l'accesso solo a un sottinsieme delle risorse disponibili
 - es : un sottoinsieme delle istruzioni assembler (non si può accedere alle istruzioni che comunicano con le interfacce di I/O), una sola parte della RAM etc.
- Stato supervisore o kernel :
 - modalità che permette l'accesso a tutte le risorse

3

Come può interagire con l'OS un programer? Attraverso delle system call

Organizzazione Monolitica (system call)

- Organizzazione del sw del SO :
 - insieme di procedure compilate in un unico oggetto
 - ogni procedura può chiamare tutte le altre/ha visibilità delle strutture dati globali
- I programmi che girano in stato utente richiedono servizi al SO tramite invocazione di funzioni 'speciali' → procedure eseguite dal sistema : es. leggere e scrivere da file
 - system call o chiamate di sistema
- Le SC portano il sistema in stato kernel e mandano in esecuzione il SO

4

Le SC dipendono dal sistema operativo. Con POSIX si standardizza un sottoinsieme delle chiamate

Organizzazione Monolitica (system call bloccanti e non)

- Il programma utente può essere riattivato con due politiche :
 - alla fine del servizio :
 - si parla di *system call bloccanti* perché l'esecuzione del processo viene bloccata in attesa della fine della gestione della richiesta
 - alla fine della richiesta :
 - quando la richiesta è stata accettata dal SO il processo può continuare a fare altre cose
 - è necessario un meccanismo aggiuntivo per decidere quando la richiesta è stata servita

5

Una SC può richiedere tempi molto più lunghi rispetto a quelli di esecuzione dei processi

Organizzazione Monolitica (meccanismi basati sulle interruzioni)

- Il sistema operativo può *interrompere* l'esecuzione di un programma utente per effettuare operazioni di gestione
 - questo avviene attraverso il meccanismo delle interruzioni hw
- L'attivazione delle system call si basa sul meccanismo delle trap (interruzioni software). Le Interrupt sono **ASINCRONE** rispetto al programma in esecuzione mentre le Trap sono **SINCRONE**: causate dal programma in esecuzione

6

Un interrupt causa trigger di moduli di supervisione, che esegue ISR



Come passano da lato supervisor a utente? Quando copio nel registro di stato lo faccio in automatico.
 Al contrario però? Le interrupt sono asincrone: mi servirebbe un modo per mettere il sistema in eccezione: 21/02/2017
 Le system call presentano una trap per portare sistema in stato supervisore. Come se fosse una chiamata da una ISR volontaria. Trap sta normalmente all'interno delle SC.

Le trap: interrupt software

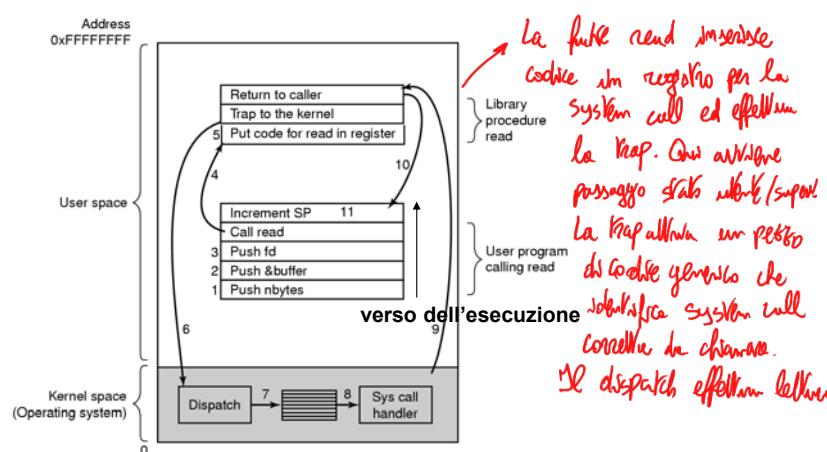
Le trap sono un tipo di interrupt, provocato dall'esecuzione di una determinata istruzione del programma in esecuzione:

Le trap vengono trattate in modo del tutto analogo agli interrupt: ciascun tipo di trap ha un identificatore che viene usato per cercare l'indirizzo della prima istruzione del corrispondente interrupt handler (ISR).

Le chiamate alle system call sono realizzate tramite trap. Alla ricezione di un interrupt/trap la CPU passa da user mode a kernel mode: in questo modo l'interrupt handler (che è un componente del software di sistema operativo) esegue in kernel mode. Alla istruzione di ritorno da interrupt, viene ripristinata la vecchia status word (PSW) del programma interrotto, e così si ritorna a user mode.

7

Passi nel trattamento di una System Call



Esempio: gli 11 passi necessari per eseguire la read (fd, buffer, nbytes)

8

1. Salvo punto di ritorno e salto. Al termine ritorno alla subroutine.

Con la read ho una semplice subroutine. La read è una funzione utente.

⇒ 1. Normali chiamate di procedura. 2. Identif. del codice. 3. Chiamata del dispatch che attiva sys call.

4. Ritorno al chiamante.

Alcuni esempi di system call: Process Management

Process management	
Call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Sostituire cuore del processo

↳ Sostituire codice chiamato con codice dei dati presi da un file eseguibile

→ Creazione di un nuovo processo
Modo in cui OS Unix
crea nuovo processo

9

Alcuni esempi di system call: File Management

File management	
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Operazioni
sui file

10

Alcuni esempi di system call: Directory Management

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

11

Alcuni esempi di system call: Varie ed eventuali ...

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

12

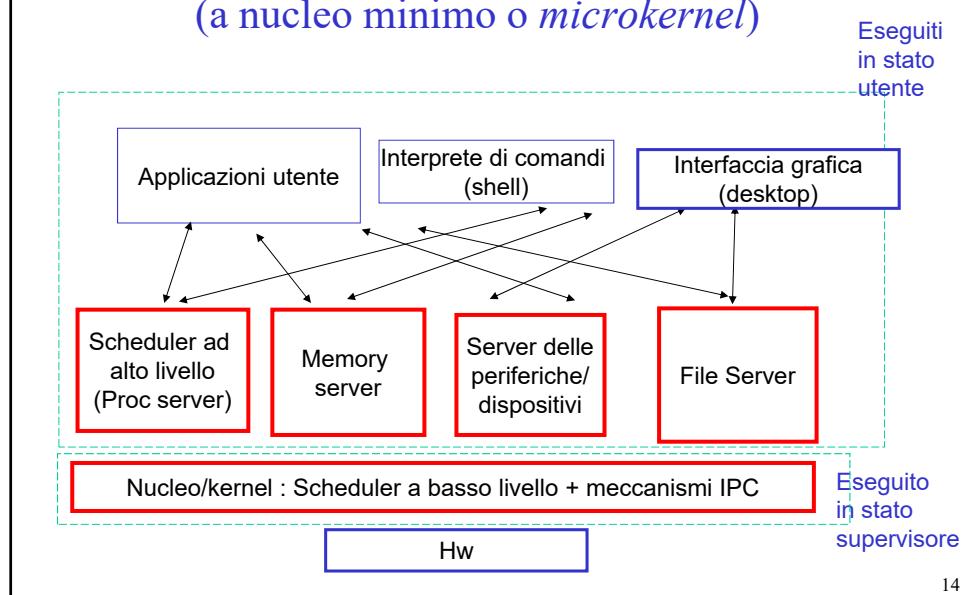
Alcuni esempi di system call: Win32 Application Programming Interface

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve (none)	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link (none)	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount (none)	(none)	Win32 does not support mount
umount (none)	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod (none)	(none)	Win32 does not support security (although NT does)
kill (none)	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

13

Mondatica: gran parte del SO che lavora n/n moduli supervisori.
 Problema: errori verificati n/n supervisori sono difficili da gestire

Organizzazione Client-Server (a nucleo minimo o *microkernel*)



14

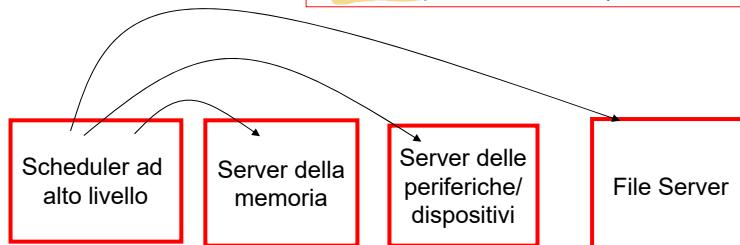
Struttura architettonica a micro kernel: riduce al minimo le parti che lavorano n/n moduli supervisori. Problema: meno efficiente

Ogni spazio di memoria corrisponde a un ammesso. Bisogna usare meccanismi di comunicazione tra processi legati a scambi di messaggi.

Organizzazione Client-Server (2)

(a nucleo minimo o *microkernel*)

I vari processi server non hanno SD e comunicano fra di loro con i normali meccanismi di comunicazione tra processi (IPC) usati dai processi utenti (es. send-receive)



Hw

15

Organizzazione Client-Server (3)

- Quando un processo richiede un servizio *comunica* con uno dei processi server
 - es: effettua una *send* al file server per richiedere la lettura da un file
- L'attesa della terminazione di un servizio avviene come *attesa di una comunicazione*
 - es: effettua una *receive* al file server per ottenere le informazioni lette

16

Organizzazione Client-Server (4)

- Minimizza le funzioni del SO che girano in modo kernel
- Molte funzioni sono realizzate da processi server che girano in modo utente
- Nucleo minimo (Microkernel) :
 - funzioni base per la gestione dei processi e comunicazione fra processi (IPC)
 - comunicazione con i dispositivi vista come messaggi “speciali”

17

Client-server vs modello monolitico

- Più sicuro
- Meno efficiente
- Si adatta bene ai sistemi operativi di rete
- Windows NT 3.0 adottava un modello ispirato al client/server (ibrido)
 - scartato perché troppo lento
- Studiato in ambito accademico
 - es: MACH, Minix, sono versioni di Unix a microkernel

18