Lezione 13

Linux

- Introduzione
- Architettura del kernel
- Gestione dei processi
- Gestione della memoria
- File system
- I/O

Introduzione

- Il kernel di Linux e' il cuore del piu' popolare e completo s.o. open source Unix-like
- Il codice sorgente e' liberamente disponibile al pubblico Linux per modifiche ed installazione
- Oggi molto diffuso su servers, desktop computers e sistemi embedded

⇩

Numerosi vantaggi per gli utenti

- Economici
- Di personalizzazione del s.o.
- Disponibilita' di molti user groups

⇩

RAPIDISSIMA DIFFUSIONE

Storia

- Versione 0.1 creata nel 1991 da Linus Torvalds a partire dal s.o. "didattico" Minix e dagli strumenti sw sviluppati da Stallman nel progetto GNU
- Guidati da Torvalds, una comunita' di sviluppatori rilascia la versione 1.0 nel 1994
  - Conformita' a POSIX
  - Caratteristiche dei s.o. avanzati (mem. virt., networking, multiprogrammazione,…)
- Versione 2.0 nel 1994
  - Supporto SMP
  - Supporto a vari file system
  - Tools di amministrazione
- Versione 2.6 nel 2003
  - Architetture a 64 bit
  - Riscrittura di numerosi moduli del kernel

Le distribuzioni (qualche centinaio)

- Nate per facilitare il processo di installazione e configurazione del sistema, nonche' il suo uso
- Di solito includono
  - Il kernel di Linux
  - Applicazioni di sistema (tools per la sicurezza e la gestione del sistema)
  - Applicazioni utente (compilatori, browser, librerie di office automation,…)
  - Interfacce grafiche
- Distrubuzioni famose
  - Red hat
  - Ubuntu
  - Suse
  - Debian
  - Fedora

Interfacce utente

- A differenza di Windows e MacOS X, il kernel di Linux non specifica una interfaccia grafica utente (il desktop)
- L'accesso standard al sistema e' attraverso interfacce "a linea di comando" chiamate shell (Bourne shell, C shell, Bourne again shell)
- Le comuni interfacce grafiche fanno parte delle distribuzioni e sono costituite da 3 livelli:

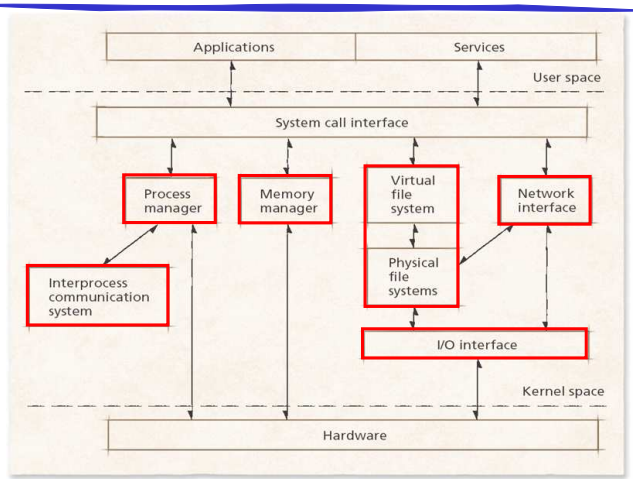
Gestione file e accesso a programmi	desktop
Posizionamento, aspetto e dimensione finestre	Window manager
Creazione e manipolazione finestre	Sistema X window

Sistema operativo

Architettura del kernel

- Linux nasce con un kernel monolitico
  - La complessita' delle ultime versioni ha reso necessaria una organizzazione modulare
- Sei sottosistemi primari:
  - Gestione dei processi
  - Comunicazione tra processi (…approccio microkernel !!)
  - Gestione della memoria
  - File system
  - Gestione I/O
  - Networking

Architettura del kernel



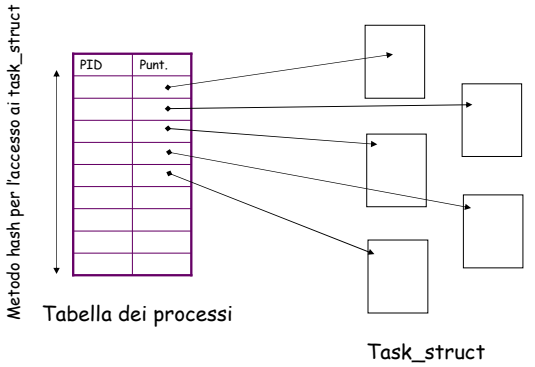
Piattaforme hardware

- Originariamente sviluppato per processori Intel x86
- Il successo ha reso necessario lo sviluppo per un gran numero di piattaforme:
  - x86 (incluso Intel IA-32), HP/Compaq Alpha AXP, Sun SPARC, Sun UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390 and zSeries, MIPS, HP PA-RISC, Intel IA-64,AMD x86-64,H8/300,V850 e CRIS.
- La parte del sistema dipendente dall'architettura e' detta Architecture-specific code, e' separata dal kernel e risiede nella directory /arch
- Lo sviluppo di versioni per nuove piattaforme avviene mediante una versione speciale del kernel chiamata User Mode Linux (UML) che viene eseguito in modalita' utente e non danneggia il sistema dove e' eseguito

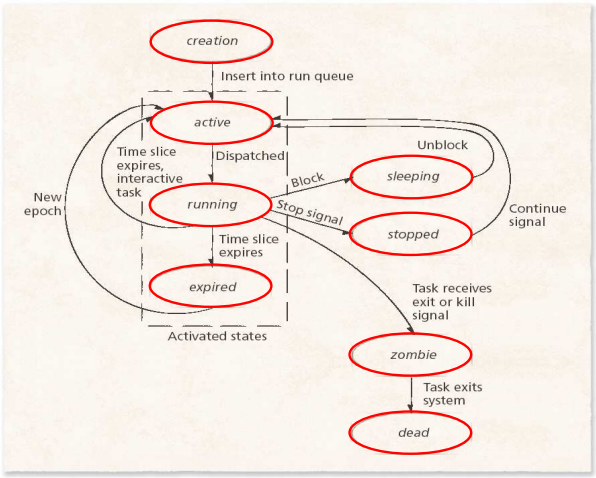
Sottosistema per la gestione dei processi

- Responsabile per
  - allocazione del processore ai processi
  - gestione dei segnali
- Linux non fa differenza tra processi e thread
  - Processi e thread hanno un'unica rappresentazione e vengono chiamati task
  - Il PCB prende il nome di **task_struct**
- Oltre ai task_struct (una per processo) il kernel mantiene una tabella dei processi
- Ogni processo e' identificato univocamente da un Process ID
- Un metodo hashing consente un accesso veloce alla tabella dei processi

Strutture dati per la gestione dei task



Stati di transizione



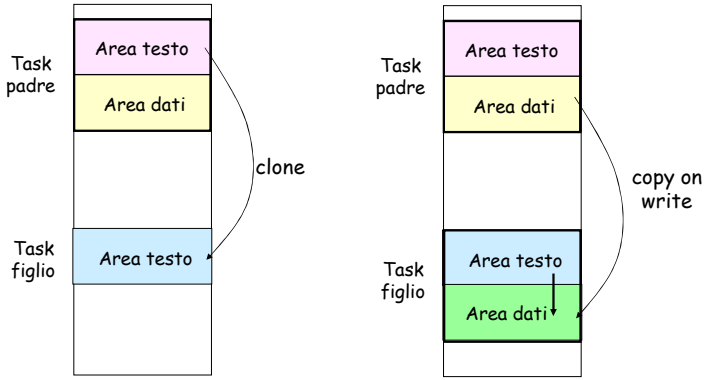
Stati di transizione

- **Creazione**
  - Viene creata la task_struct e creato un riferimento con il PID nella tabella dei processi
- **Active**
  - Il task e' in attesa di essere schedulato per l'esecuzione (stato ready)
- **Running**
  - Il task utilizza la CPU
- **Sleeping** o **stopped**
  - Il task e' bloccato o sospeso per I/O, attesa di messaggi,…(stato waiting)
- **Expired**
  - Il task sta dando priorita' ad alti task
- **Zombie**
  - Il task e' terminato ma le strutture non sono state ancora rimosse dal sistema (ad es. se un altro task deve ricevere dati dal task terminato ma non e' ancora pronto)
- **Dead**
  - Il task puo' essere rimosso dal sistema

Creazione dei task

- Un task puo' generare altri tasks mediante la chiamata di sistema **clone**
- Mediante tale funzione e' possibile specificare quali risorse (memoria, registri, stack, descrittori di file) far condividere tra task padre e task figlio
  - Poche risorse ➔ creazione di processi
  - Molte risorse (es. sp. indirizzamento)➔ creazione di threads
- I task sono le unita' di schedulazione da parte del kernel, quindi i threads generati con clone hanno una naturale implementazione uno-uno
- La tradizionale funzione **fork** di Unix viene convertita in **clone**

Implementazione della fork mediante clone



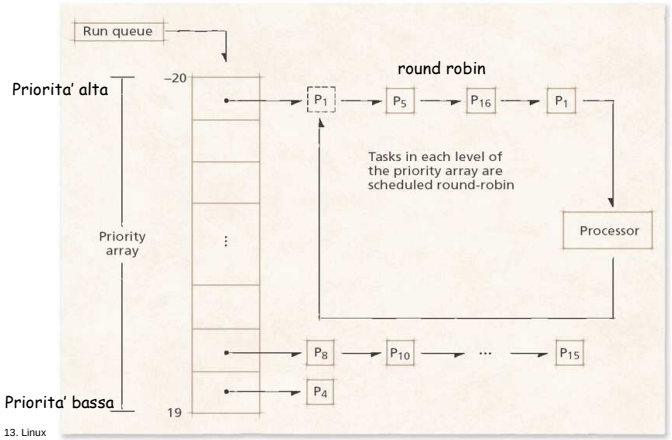
La duplicazione dello spazio di indirizzamento avviene con il meccanismo di copy-on-write: le pagine vengono copiate solo quando il figlio tenta una scrittura

Scheduling dei processi (kernel 2.4)

- 40 livelli di priorita': da -19 (la massima) a 20 (la minima)
- I task vengono eseguiti secondo l'ordine di priorita'
  - Un array delle priorita' conserva i puntatori alle liste dei task con la stessa priorita'
- Ogni livello gestito in modalita' round robin
- Quanto di tempo q variabile a secondo della priorita'
  - Time slice piccolo per task a bassa priorita'
  - Time slice grande per task a alta priorita'
- Ogni task e' in esecuzione fino a:
  - Esaurimento del quanto di tempo
  - Operazioni di I/O
  - E' disponibile un task con priorita' maggiore

Scheduling dei processi

Scheduler con prelazione basato su code multiple e priorita'



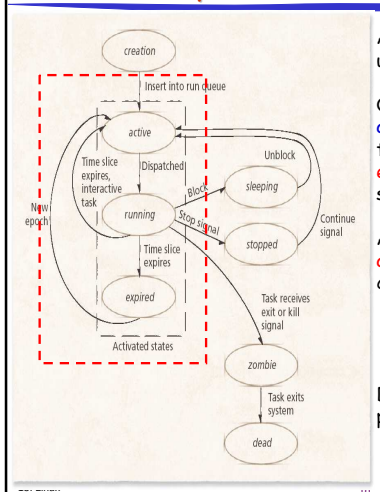
Epoche

- Per evitare la starvation dei processi il kernel divide il tempo di CPU in Epoche

EPOCA =
Periodo di tempo predefinito (starvation limit)
entro cui tutti i processi sono eseguiti almeno una volta

- All'inizio di ogni epoca viene assegnato un quanto di tempo q ad ogni processo, in funzione della sua priorita' e del suo quanto di tempo base Q
- Durante un'epoca la CPU puo' essere assegnata round robin piu' volte allo stesso processo, fino a quando q non e' esaurito
- L'epoca scade quando:
  - la coda dei processi attivi e' vuota
  - Scade la starvation limit

Lo stato expired



Alla creazione di un processo si assegna un valore di default q= Q = 200ms

Quando i processi esauriscono il loro quanto oppure scade la starvation limit tutti i processi passano allo stato expired e non possono essere piu' schedulati

All'inizio di una nuova epoca il nuovo quanto di tempo q viene ricalcolato come

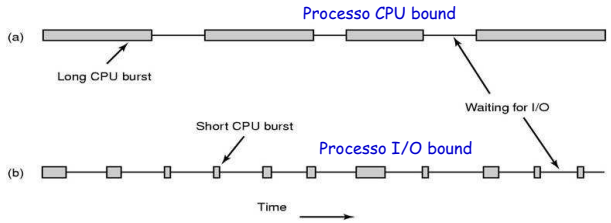
q = Q + T/2
(priorita' maggiore)

Dove T e' il tempo non usato nell'epoca precedente (es. erano bloccati per I/O)

Idea alla base

I task possono essere divisi in due classi

- I/O bound: uso prevalente dei dispositivi di I/O. Tendono a non utilizzare per intero il Q a loro assegnato ➔ Sono interattivi
- CPU bound: uso prevalente della CPU. Tendono ad usare tutto il Q a disposizione ➔ In generale sono poco interattivi



Ai task interattivi (I/O bound) si cerca di dare una priorita' maggiore

Problemi dello scheduler Linux 2.4

- All'inizio di ogni epoca bisogna calcolare il quanto di tutti i processi

Scarsa scalabilita'

Lo scheduler Linux 2.6 ha parzialmente risolto il problema con un diverso modo di calcolare le priorita' dinamiche ad ogni inizio epoca

Scheduling real time

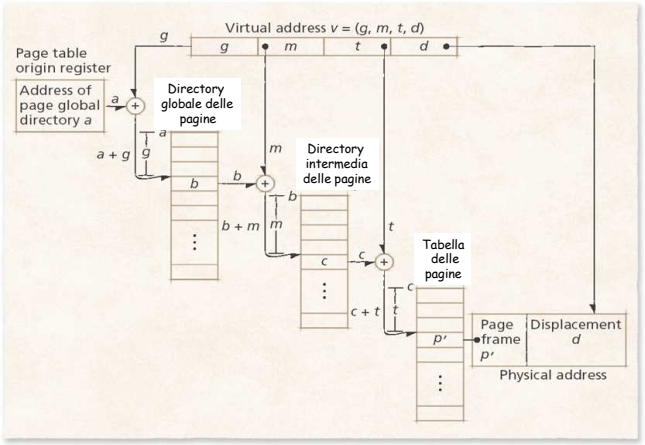
- Oltre allo scheduling a partizione di tempo, Linux gestisce anche la classe di scheduling real time
- I processi sono gestiti o con una coda FIFO senza prelazione o in modalita' round-robin
- Per prevenire un utilizzo improprio possono essere creati solo dall'amministratore di sistema.

Gestione della memoria

- La memoria centrale e' divisa in frame di 4 KB o 8 KB
- Linux puo' gestire indirizzamenti a
  - 32 bit ➔ spazio di ind. 4 GB
  - 64 bit ➔ spazio di ind. 2 PB (es. Intel Itanium, ma con 51 bit)
- Tale spazio di indirizzamento richiede l'uso della memoria virtuale
  - 3 livelli di tabelle delle pagine
  - 1 livello viene disabilitato per architetture a 32 bit
- Lo spazio di indirizzamento virtuale e' organizzato in aree di memoria virtuale che raggruppano pagine con caratteristiche omogenee (heap, stack, area testo, ...).

  Segmentazione paginata

Memoria virtuale



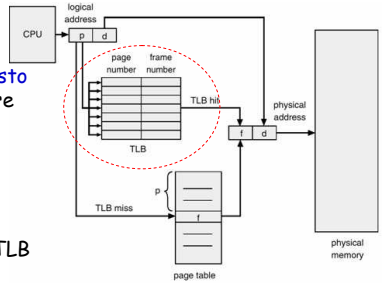
Un problema: flushing dei TLB

Nella gestione della memoria virtuale i registri associativi (TLB) contengono i riferimenti alle pagine usate piu' di recente

In occasione dei cambio di contesto il contenuto del TLB deve essere salvato in memoria (flushing)

Accesso alla memoria ad ogni cambio di contesto, facendo perdere il vantaggio nell'uso dei TLB

Riduzione delle prestazioni



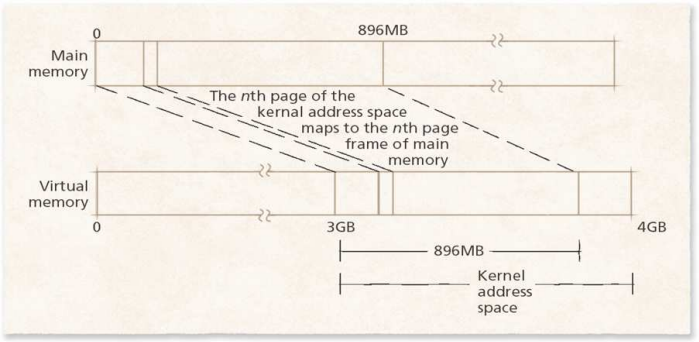
Soluzione

- I 4 GB dello spazio di indirizzamento sono divisi in
  - 3 GB per i processi (indirizzi da 0 a 3 GB)
  - 1 GB per il kernel (indirizzi da 3 GB a 4 GB), sempre lo stesso per tutti i processi

Non e' mai necessario svuotare i TLB relativi alle pagine dello spazio di indirizzamento del kernel

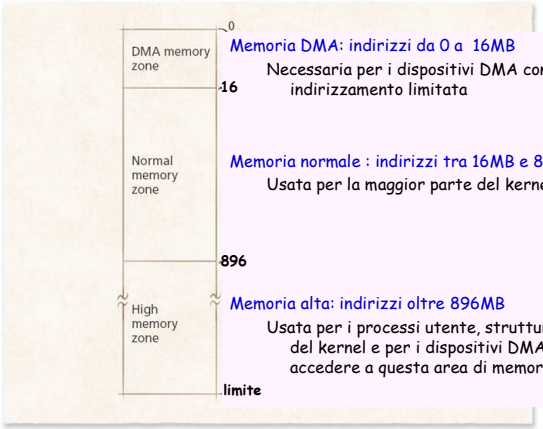
- Per semplicita' e per motivi di compatibilita' lo spazio di indirizzamento del kernel e' mappato nel primo GB della memoria centrale

Esempio



Il kernel associa permanentemente i primi 896 MB delle proprie pagine virtuali ai primi 896 MB della memoria fisica
I rimanenti 128 MB sono usati come cache o per strutture dati temporanee

Organizzazione della memoria fisica (IA-32)



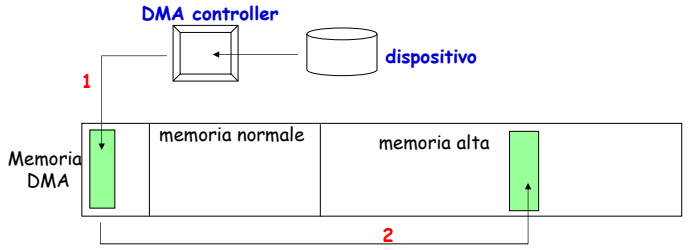
Memoria DMA: indirizzi da 0 a 16MB
Necessaria per i dispositivi DMA con capacita' di indirizzamento limitata

Memoria normale : indirizzi tra 16MB e 896MB
Usata per la maggior parte del kernel

Memoria alta: indirizzi oltre 896MB
Usata per i processi utente, strutture temporanee del kernel e per i dispositivi DMA capaci di accedere a questa area di memoria

Buffer bounce (memoria tampone di rimbalzo)

- I dispositivi DMA sono in grado di indirizzare limitate quantita' di memoria
  1. viene creata una area di memoria transitoria nella zona di memoria DMA (memoria bassa)
  2. le pagine della zona di memoria DMA vengono copiate dal kernel nella zona di memoria alta

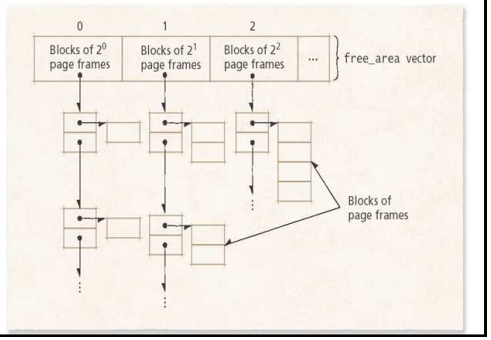


Allocatore di zona

- Modulo che ha il compito di trovare frame liberi per un processo. Nell'ordine cerca frame liberi ne:
  - Zona di memoria alta
  - Zona di memoria normale se quella alta e' indisponibile
  - Zona di memoria DMA se quella normale e' indisponibile

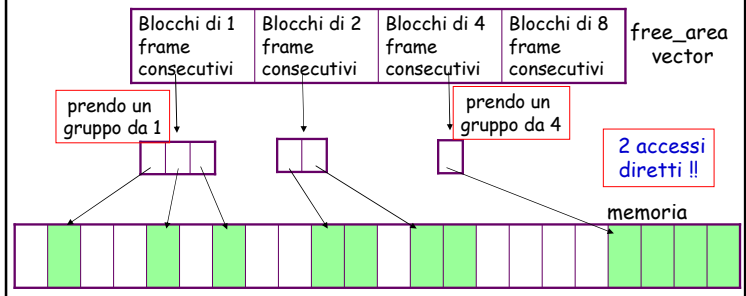
I frame liberi sono organizzati in liste di frame adiacenti in numero pari a potenze di 2

Alle liste si accede attraverso un vettore



esempio

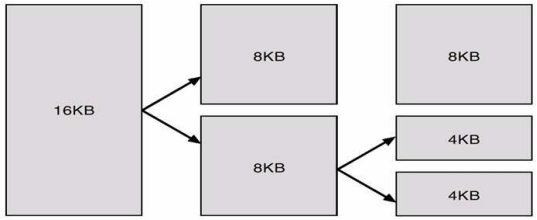
- Processo di 5 pagine ➔ serve un gruppo di 4 e un gruppo di 1



Alternativa: ricerca di 5 frame liberi

Il "buddy algorithm" (buddy = compagno)

- per ottenere gruppi di frame in numero uguali a potenza di due, l'allocatore di zona
  - in fase di allocazione divide frame grandi in 2 frame piu' piccoli (i compagni) fino alla dimensione richiesta (min=4KB)
  - in fase di deallocazione raggruppa 2 frame in frame piu' grandi



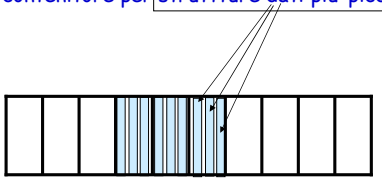
Passi del buddy algorithm per l'allocazione di un processo di 3 KB

Allocatore di slab

- Tipicamente la dimensione delle pagine in un sistema Linux e' di 4 KB
- Molte strutture del kernel richiedono molto meno di una pagina

Rischio di frammentazione

- il kernel conserva alcune pagine (slab) usate come contenitore per struttture dati piu' piccole di una pagina



Slab 4KB

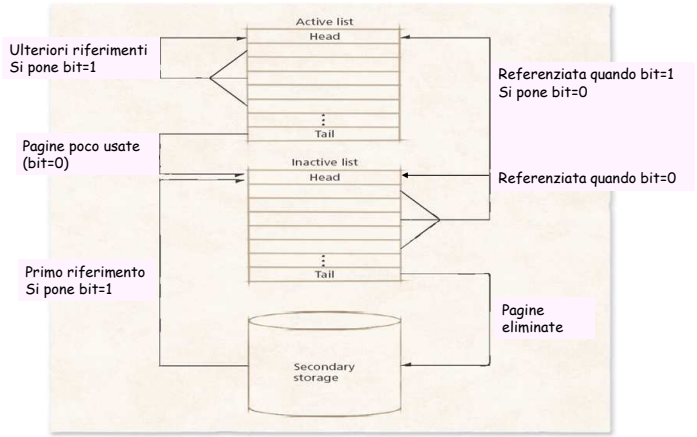
Algoritmo di paginazione

- Caratteristiche generali:
  - Quando le pagine sono lette in memoria sono inserite in una cache
  - Per migliorare le prestazioni, le pagine sporche sono copiate periodicamente (5-10sec) sul disco (write back caching)
- Due tipi di pagine:
  - Attive (referenziate di recente)
  - Inattive (candidate all'eleminazione)
- L'algoritmo di paginazione e' una variante del FIFO seconda chance
  - FIFO seconda chance a 2 livelli

FIFO seconda chance a 2 livelli

- 2 liste di dimensione variabile:
  - Pagine attive e pagine inattive
- FIFO seconda chance in ogni lista
  - Le pagine entrano nella lista delle pagine inattive con bit=1
- Nella lista delle pagine inattive un secondo riferimento "promuove" la pagina nella lista delle pagine inattive con bit=0
- Periodicamente si riorganizzano le liste:
  - Si spostano alcune pagine attive poco usate nella lista delle pagine inattive
  - Circa 2/3 nella lista delle pagine attive e 1/3 tra quelle inattive

Algoritmo di paginazione



swapping

- Quando i frame iniziano a scarseggiare si eliminano le pagine meno utilizzate
  - Obiettivo: avere sempre frame liberi
- *kswapd* e' il demone del kernel che
  - Periodicamente copia le pagine sporche sul disco
  - Elimina le pagine dalla coda della inactive list
- Una pagina non puo' essere eliminata immediatamente se
  - La pagina e' condivisa tra processi
  - La pagina e' modificata
  - La pagina e' bloccata (in attesa di I/O)

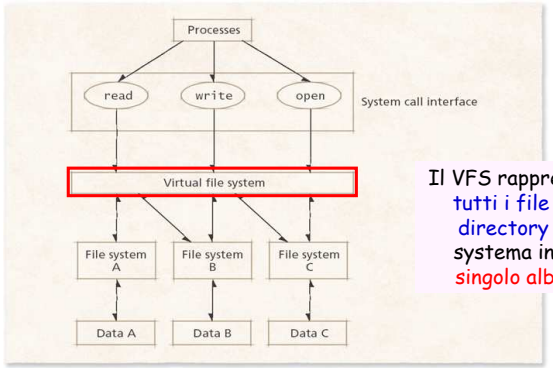
File system

- Ogni particolare file system determina come sono memorizzati i file
- Il kernel di Linux supporta oltre 40 file systems
  - File system generici (ext2, FAT, …)
  - File system di rete (NFS, …)
  - File system residenti in memoria centrale (procfs, sysfs, …)

Problema

Come e' possibile gestire piu' file system contemporaneamente?

File syste virtuale (VFS)

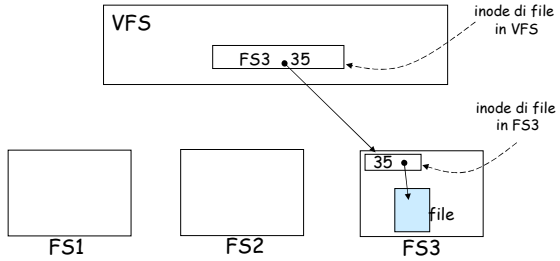


Il VFS rappresenta tutti i file e le directory del systema in un singolo albero

Gli utenti possono accedere a ogni file senza sapere in quale file system e' memorizzato

File system virtuale

- Il VFS usa degli oggetti chiamati inode per accedere ai file e le directory dei vari file system
- Gli inode contengono informazioni sul
  - File system che contiene il file
  - Inode che identifica univocamente il file nel suo file system



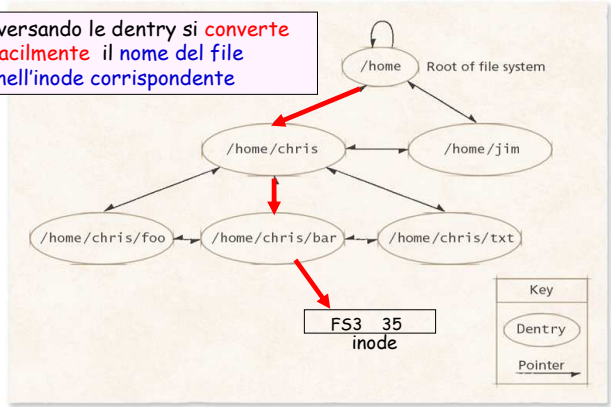
File system virtuale

Oltre agli inode, per descrivere un file, il VFS utilizza

- Descrittore di file
  - Informazioni sull'inode da referenziare
  - Informazione sulla posizione del file
  - Flag sulla modalita' di accesso
- Dentry (directory entry)
  - Contiene un puntatore all'inode corrispondente
  - Contiene il nome del file che l'inode rappresenta
  - Contiene puntatori alle dentry del padre del figlio e dei fratelli

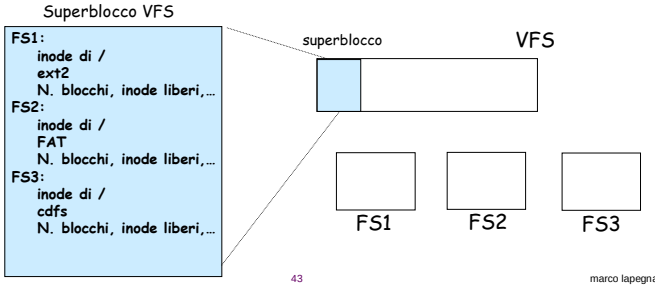
Esempio: accesso a /home/chris/bar

Attraversando le dentry si converte facilmente il nome del file nell'inode corrispondente



Superblocco del VFS

- Contiene informazioni sui file system montati:
  - Tipo del file system
  - Inode della directory principale
  - Informazioni per proteggere l'integrita' del F.S.
- Creato quando il VFS e' montato ed e' residente in memoria centrale

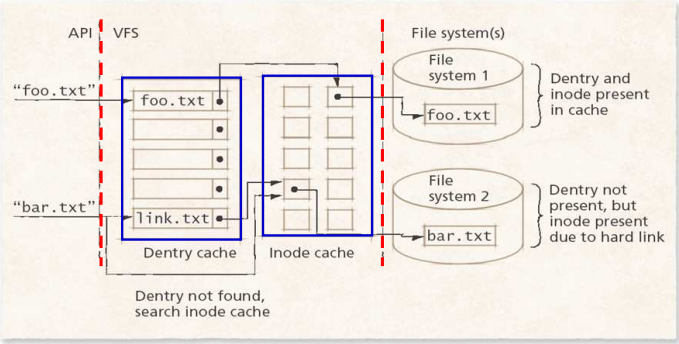


Operazioni definite sul VFS

VFS operation	*Intended use*
read	Copy data from a file to a location in memory.
write	Write data from a location in memory to a file.
open	Locate the inode corresponding to a file.
release	Release the inode associated with a file. This can be performed only when all open file descriptors for that inode are closed.
ioctl	Perform a device-specific operation on a device (represented by an inode and file).
lookup	Resolve a pathname to a file system inode and return a dentry corresponding to it.

- Poiche' ogni file system montato ha la sua organizzazione e implementazione, il VFS non implementa le chiamate di sistema
- Richiede che ogni file system montato supporti le stesse operazione

Caches del VFS

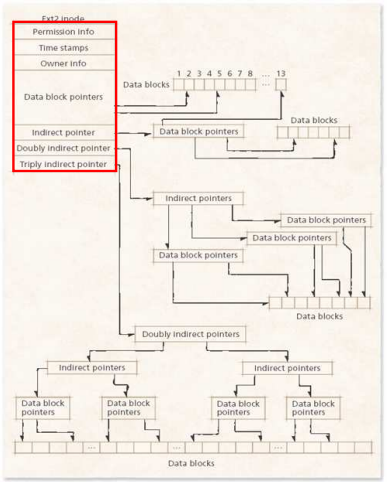


La presenza di due caches (dentry cache e inode cache) permette di migliorare le prestazioni per l'accesso ai file e alle directory

Second Extended File System (ext2fs)

- Principale FS dei sistemi Linux
- Obiettivi: performance e robustezza
- Basato su inode e directory (UNIX-like)
- Nuove funzionalita' rispetto a BSD file system (es. recupero file)
- Dimensione dei blocchi: 1KB (tipica), 2KB, 4KB o 8 KB
- 5% dei blocchi riservati all'amministratore

Inode di ext2fs



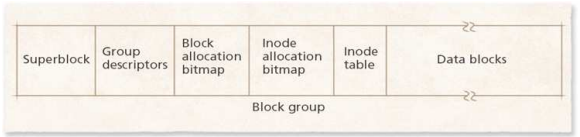
- Accesso veloce ai file piccoli
- Supporto a file grandi

Dimensione massima dei file = da 16 a 4096 GB
(a secondo della dimensione dei blocchi)

Gruppi di blocchi

Il file system cerca di memorizzare i dati di un file in blocchi fisicamente vicini (gruppi di blocchi)

Riduzione del seek time relativo ai blocchi del file

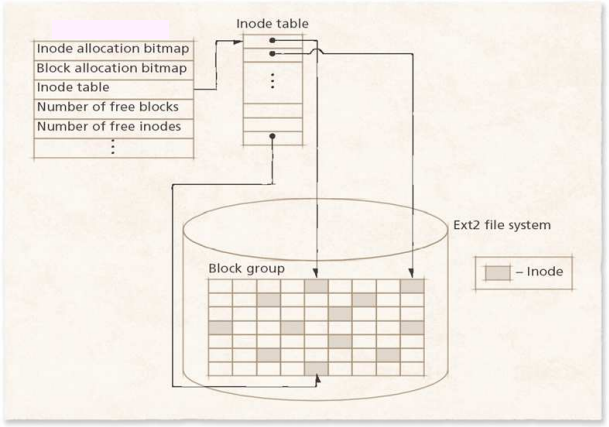


Struttura di un gruppo di blocchi

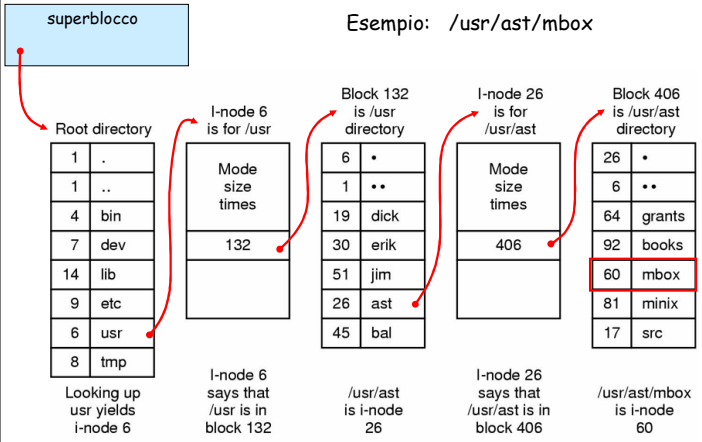
Struttura di un gruppo di blocchi

- Superblocco
  - Contiene informazione critiche su tutto il file system (dimensione, numero di file, …)
  - Replicato in differenti gruppi per motivi di affidabilita'
- Dscrittore del gruppo
  - Contiene informazioni sui blocchi del gruppo (numero, blocchi liberi,…)
- Bitmap di allocazione dei blocchi
  - Mantiene traccia dei blocchi in uso
- Tabella degli inode
  - Contiene gli inode dei file memorizzati nel gruppo
- Bitmap di allocazione degli inode
  - Mantiene traccia degli inode in uso
- Blocchi di dati
  - Dati dei file e directory

Gruppo di blocchi



Localizzazione di un file



File System Proc

- Ext2fs e' un file system per la memorizzazione permanente di dati sul disco
- File system Proc
  - Creato per fornire, in tempo reale, informazioni sullo stato del kernel e dei processi
  - Permette ai processi utenti di accedere alle informazioni del sistema attraverso il contenuto della directory /proc
  - Non e' associato ad un dispositivo di memorizzazione secondario
    - Esiste solo in memoria centrale
    - I dati sono creati "on demand"
    - Permette agli utenti di inviare dati al kernel

Esempio

```
root> ls /proc
1      20535  20656  751  978          interrupts  pci
10     20538  20657  792  acpi         iomem       self
137    20539  20658  8    asound       ioports     slabinfo
19902  20540  20696  811  buddyinfo    irq         stat
2      20572  20697  829  bus          kcore       swaps
20473  20576  20750  883  cmdline      kmsg        sys
20484  20577  3      9    cpuinfo      ksyms       sysvipc
20485  20578  4      919  crypto       loadavg     tty
20489  20579  469    940  devices      locks       uptime
20505  20581  5      960  dma          meminfo     version
20507  20583  536    961  dri          misc        vmstat
20522  20586  541    962  driver       modules
20526  20587  561    963  execdomains  mounts
20527  20591  589    964  filesystems  mtrr
20529  20621  6      965  fs           net
20534  20624  7      966  ide          partitions
root>
```

Dir. con le info del proc. 20507

Dir. con le info dei dispositivi

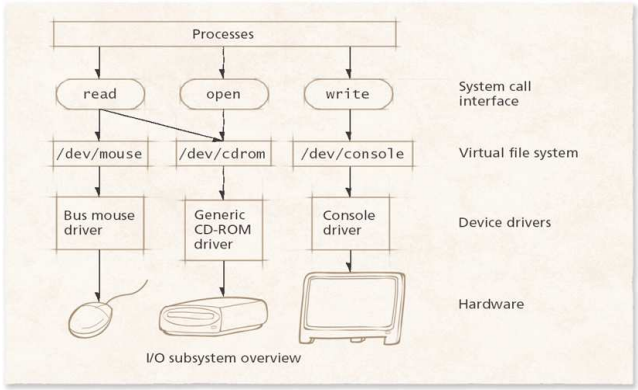
Dir. con le info sulla cpu

Dir. con le info sulla memoria

Sottosistema di I/O

- I dispositivi di I/O sono rappresentati da file speciali del dispositivo che forniscono accesso al dispositivo
  - Presenti nella directory /dev
- Questa rappresentazione permette ai processi di accedere ai dispositivi attraverso chiamate standard di libreria (fopen, fclose, fprintf, …) dirette al VFS
- Il sistema operativo si interfaccia con i dispositivi attraverso i driver dei dispositivi
  - Scritti dai produttori e programmatori indipendenti

Interfacciamento tra S.O. e dispositivi



I/O subsystem overview

Identificazione dei dispositivi

- Il kernel assegna ai dispositivi un
  - Numero primario di identificazione
  - Numero secondario di identificazione
  - I dispositivi con lo stesso numero primario eseguono funzioni simili e sono controllati dallo stesso driver
  - il numero secondario permette al driver di distinguere i singoli dispositivi
- Esempio:
  - Ad un disco viene assegnato un determinato numero primario di identificazione
  - Alle singole partizioni del disco un differente numero secondario

Driver dei dispositivi

Driver disponibili nella directory /proc/devices

```
root> cat /proc/devices
Character devices:
  1 mem                Physical memory access
  2 pty                BSD-style terminal (TTY) devices
  3 ttyp               BSD-style terminal (TTY) devices
  4 vc/%d              Virtual console
  5 ptmx               Multiplexor for AT&T-style terminal (TTY) devices
  6 lp                 Parallel printer
  7 vcs                Virtual console capture devices
 10 misc               Non-serial mice, other devices
 13 input              Input core (typically contains a mouse)
 14 sound              Audio device
116 alsa               Advanced Linux Sound Driver
128 ptm                AT&T-style terminal (TTY) devices
136 pts                AT&T-style terminal (TTY) devices
180 usb                USB device
226 drm                Direct Rendering Manager (video card)
Block devices:
  2 fd                 Floppy disk drive
  3 ide0               Primary IDE channel
 22 ide1               Secondary IDE channel
root>
```

Numeri primari di identificazione

Classificazione dei dispositivi

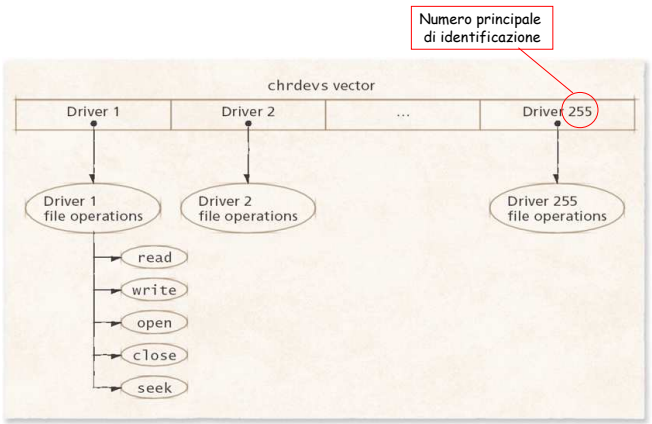
- Linux supporta dispositivi appartenenti a tre classi principali
  - Caratteri
    - Stampanti, console, mouse, tastiera,…
  - Blocchi
    - dischi (floppy, CDROM, disco fisso,…)
  - Rete
    - Schede di rete (LAN, FDDI, …)

Dispositivi a carattere

- Trasmettono dati come un flusso sequenziale di bytes
- Ogni dispositivo e' rappresentato da una struttura dati che contiene
  - Nome del driver da usare
  - Puntatore ad una struttura con le operazioni consentite sul dispositivo
- Per inizializzare un dispositivo a carattere e' necessario informare il VFS del tipo di operazioni supportate dal dispositivo

aggiunta della relativa struttura dati ad un vettore dei driver registrati

Vettore dei driver registrati



Dispositivi a blocchi

- Accedono a blocchi di byte di dimensione fissa
- Prevalentemente supporti di memorizzazione secondaria
- Accesso non sequenziale

Algoritmi per l'ottimizzazione dell'uso dei dischi

- Per minimizzare il tempo di accesso ai blocchi del dispositivo vengono impiegate strategie di caching
- Se il blocco richiesto non e' presente nella cache, la richiesta viene accodata ad una lista delle richieste

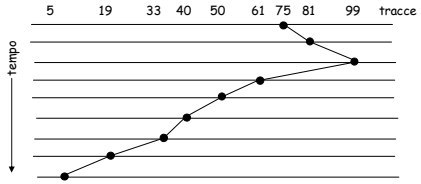
Sottosistema per i dispositivi a blocchi

Diversi strati sw per esaudire una richiesta

Virtual file system	Richiesta di accesso al dispositivo
File systems	Accesso a /dev
Page cache	Gestore della cache
Block layer, bios	Scheduler del disco
Drivers	Accesso al singolo blocco
Hardware	Dispositivo fisico

Scheduler del disco

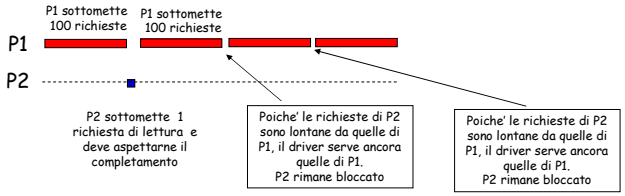
- L'algoritmo e' una variante del LOOK algorithm
- Esempio : 75, 81, 99, 61, 50, 40, 33, 19, 5



- Poiche' il file system ext2 cerca di memorizzare i dati di un file in blocchi contigui, il kernel prova a fondere richieste a blocchi adiacenti in un'unica richiesta

Un problema

- P1 deve scrivere 200 MB di dati (circa 200000 blocchi da 1KB)
- P2 deve visualizzare sul terminale il contenuto di un file (lettura dal disco e scrittura sul terminale)



Il processo P2 rimane bloccato a causa della posticipazione indefinita per le richieste di lettura (read starvation)

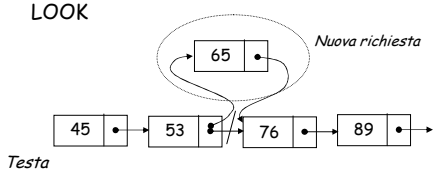
Scheduler per deadline

- Oltre all'algoritmo LOOK, il kernel utilizza un algoritmo per deadline per prevenire la read starvation
  - Assegna una deadline ad ogni richiesta (default 500ms) e assicura che tutte le operazioni di lettura siano eseguite entro la deadline
- Esempio .
  - Deadline di 450 ms
  - Ogni richiesta di 15 ms

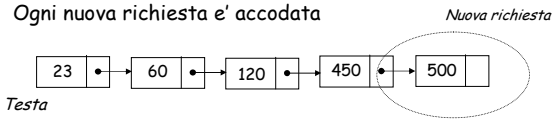
Dopo 30 richieste per il processo P1 si passa alle richieste di P2

Implementazione: 2 strutture dati

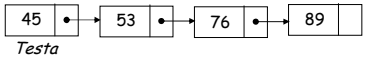
- Una lista ordinata secondo la politica LOOK in base alle tracce
  - Ogni nuova richiesta e' inserita nella lista secondo la politica LOOK



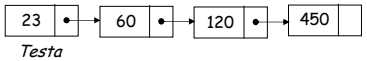
- Una coda FIFO secondo la deadline
  - Ogni nuova richiesta e' accodata



Esempio 1ms per traccia



Lista LOOK



Coda FIFO

Se traccia corrente = 30 ➔
15 ms per servire la richiesta LOOK < 23 ➔ OK

Se traccia corrente =10 ➔
35 ms per servire la richiesta LOOK > 23 ➔
serve secondo la coda FIFO

Dispositivi di rete

- I/O di rete
  - I processi utente accedono alle schede di rete solo attraverso il sottosistema di IPC
  - Il processo definisce l'indirizzo di rete, ma il sottosistema di rete determina il dispositivo da usare
- Il traffico di rete e' asincrono
  - Le istruzioni di read e write non sono sufficienti
  - Necessarie anche quelle per sincronizzazione con il kernel, avvio e arresto del dispositivo e spedizione dei pacchetti

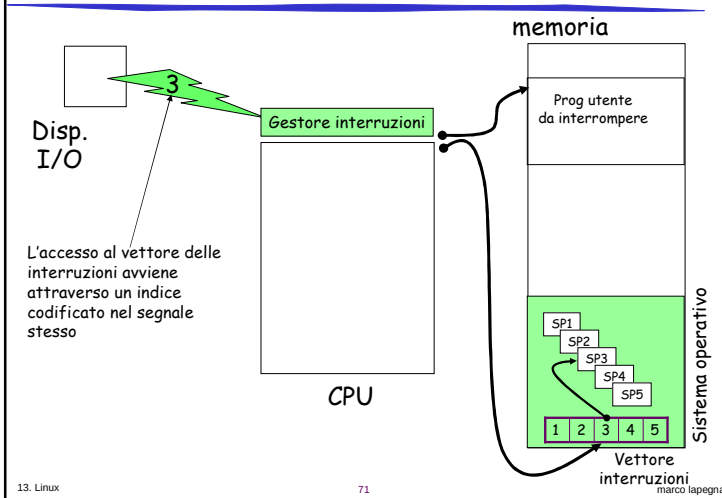
Dispositivi di rete

- Trasmissione a pacchetti
  - Il kernel esamina una tabella interna per accoppiare l'indirizzo con l'appropriato dispositivo
  - Una volta che il kernel ha preparato i pacchetti, li passa al driver della scheda di rete che li accoda secondo una propria politica
  - Il kernel risveglia il dispositivo per spedire i pacchetti

Interruzioni

- Dispositivi e kernel interagiscono attraverso le interruzioni
  - Quando il kernel riceve una interruzione da un particolare dispositivo passa il controllo a un gestore delle interruzioni
  - Per motivi di efficienza i processi relativi al gestore delle interruzioni non possono essere schedulati, prelazionati o produrre a loro volta altre interruzioni
  - Necesita' di routine di servizio che elaborino le interruzioni il piu' velocemente possibile

Gestione delle interruzioni



Interruzioni

- Per migliorare l'efficienza il kernel divide il codice delle routine di servizio in due parti:
  - Meta' superiore : gestisce solo l'accettazione dell'interruzione e viene eseguita come descritto prima. Passa subito il controllo alla meta' inferiore.
  - Meta' inferiore : esegue la maggior parte delcodice (es. gestisce la manipolazione delle strutture dati) e puo' essere schedulata successivamente.
  - Durante l'esecuzione della meta' superiore sono disabilitate le interruzioni

La meta' superiore puo' prelazionare quella inferiore ma non viceversa

Sincronizzazione del kernel

- I processi utente non possono accedere direttamente ai dati del kernel o alle risorse hw del sistema (es. I dispositivi)
- Attraverso le chiamate di sistema il kernel accede alle risorse in nome dei processi
- Se due processi fanno richieste agli stessi dati contemporaneamente c'e' il rischio di race condition

Il principale meccanismo che il kernel di Linux fornisce per la mutua esclusione alle risorse critiche sono i semafori

Semafori del kernel

- Semafori di tipo contatore, inizializzati al numero di processi abilitati ad entrare nelle sezioni critiche
- Funzione wait
  - Se contatore > 0 , wait lo decrementa e continua l'esecuzione
  - Se contatore ≤ 0, wait lo decrementa e il processo si pone nello stato di sleeping
- Funzione signal
  - Se contatore > 0, down lo incrementa
  - Se contatore ≤ 0, down lo incrementa e un processo in stato sleeping passa allo stato ready

Inter Process Comunication

- Linux supporta tutti i principali tipi di comunicazione tra processi di UNIX
  - Segnali
  - PIPE / FIFO
  - Socket
  - Memoria condivisa
  - Semafori

Segnali

- Uno dei primi meccanismi di comunicazione tra processi di UNIX
- Conformi allo standard POSIX
- Creati dal kernel per avvisare un processo di un evento inaspettato
  - Risultato dell'esecuzione di una istruzione illegale
  - Completamento di I/O
- Non permettono la comunicazione di dati
- Alla ricezione del segnale:
  - Il processo interrompe la sua esecuzione
  - Invoca la routine di servizio del segnale corrispondente. Se arriva un segnale dello steso tipo viene ignorato
  - Al termine dell'esecuzione della routine di servizio riprende l'esecuzione

Segnali

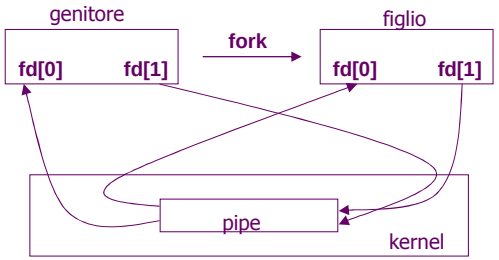
- Un processo o un thread possono gestire un segnale nei seguenti modi
  - Ignorare il segnale (tutti tranne SIGKILL e SIGSTOP)
  - Catturare il segnale : viene invocata una routine di servizio
  - Eseguire una azione di default specifica per quel segnale
- Azioni di default definite da POSIX
  - Abort (fine immediata del processo)
  - Dump (fine con salvataggio dell'area di memoria utile per il debug)
  - Ignore
  - Stop (il processo viene fermato ma non rimosso dalla memoria)
  - Continue (il processo passa dallo stato suspended allo stato ready)

Alcuni segnali POSIX

Signal	Type	Default Action	Description
1	SIGHUP	Abort	Hang-up detected on terminal or death of controlling process
2	SIGINT	Abort	Interrupt from keyboard
3	SIGQUIT	Dump	Quit from keyboard
4	SIGILL	Dump	Illegal instruction
5	SIGTRAP	Dump	Trace/breakpoint trap
6	SIGABRT	Dump	Abort signal from `abort` function
7	SIGBUS	Dump	Bus error
8	SIGFPE	Dump	Floating point exception
9	SIGKILL	Abort	Kill signal
10	SIGUSR1	Abort	User-defined signal 1
11	SIGSEGV	Dump	Invalid memory reference
12	SIGUSR2	Abort	User-defined signal 2
13	SIGPIPE	Abort	Broken pipe: write to pipe with no readers
14	SIGALRM	Abort	Timer signal from `alarm` function
15	SIGTERM	Abort	Termination signal
16	SIGSTKFLT	Abort	Stack fault on coprocessor
17	SIGCHLD	Ignore	Child stopped or terminated
18	SIGCONT	Continue	Continue if stopped
19	SIGSTOP	Stop	Stop process
20	SIGTSTP	Stop	Stop typed at terminal device

PIPE e FIFO

- Permettono lo scambio di dati tra due processi
- Quando una PIPE e' creata il kernel crea un file in un file system residente in memoria centrale ed accessibile attraverso il VFS
  - L'accesso alla PIPE avviene mediante un file descriptor
  - I due processi devono condividere il file descriptor ➔ il modo piu' naturale e' far comunicare genitore e figlio



PIPE e FIFO

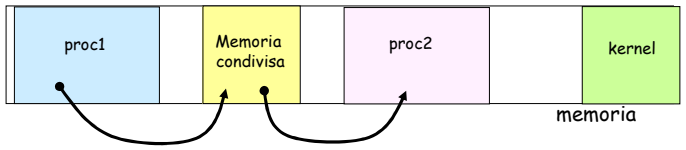
- La comunicazione e' secondo il protocollo produttore/consumatore a buffer limitato (pipe buffer)
- Principali limitazioni: dimensione limitata del buffer e comunicazione unidirezionale
- Uno strumento piu' flessibile, che non richiede la condivisione del file descriptor, ma che si basa sullo stesso meccanismo e' la pipe con nome (detta anche FIFO)

Socket

- Permettono a coppie di processi di scambiarsi dati attraverso un canale di comunicazione bidirezionale
- Principalmente utilizzato per la comunicazione tra processi residenti su differenti sistemi
- Utilizzano il protocollo client/server
  - Il server rimane in attesa delle richieste del client
- 3 tipi disponibili
  - Socket di flusso. Il piu' usato, usa il TCP, dati trasferiti come flussi di byte
  - Socket a datagramma. Piu' veloce ma meno affidabile.
  - Socket accoppiati. Utilizzabili solo sullo stesso sistema

Memoria condivisa

- Coppie di processi possono scambiarsi dati attraverso un'area di memoria facente parte di entrambi gli spazi di indirizzamento
- Vantaggi
  - Non richiede intervento del kernel per l'accesso ai dati
  - Buone prestazioni per processi che accedono spesso ai dati
  - Interfacce standard (POSIX, System V)



Memoria condivisa

- Intervento del kernel
  - Limitato alla sola creazione
    - determina se un processo ha i permessi necessari
    - Alloca la memoria condivisa
    - Collega l'area di memoria allo spazio di indirizzamento del processo
- Implementazione
  - Mediante un file nel file system /tmpfs (temporary file system)
  - Il file system e' temporaneo e le pagine di memoria non sono persistenti
  - Le pagine possono essere portate su memoria di massa se i frame in memoria scarseggiano
  - E' possibile definire permessi

Semafori

- Oltre ai semafori del kernel i processi possono utilizzare i semafori System V per sincronizzare le loro attivita'
- Accessibili mediante chiamate di sistema
- Possibilita' di definire "array di semafori" per proteggere gruppi di risorse

Varianti di Linux

- Versioni per architetture a 64 bit
- Versione per sistemi paralleli
- Versione per sistemi embedded
- Versione per sistemi real time