**UNIX-Linux**

**Genesi**

**DTSS** (Dartmouth College Time Sharing System, 1964)

– Il primo elaboratore multi-utente a divisione di tempo

– Programmato in BASIC e ALGOL

– Presto soppiantato da

**CTSS** (MIT Compatible Time Sharing System, in versione sperimentale dal 1961)

– Enorme successo nella comunità scientifica

– Induce MIT, Bell Labs e GE alla collaborazione nel progetto di

**MULTICS** (Multiplexed Information and Computing Service, 1965)

– Quando Bell Labs abbandona il progetto, Ken Thompson, uno degli autori di MULTICS, ne produce in assembler una versione a utente singolo

**UNICS** (UNiplexed “ICS”, 1969)

1974 – Nuova versione di **UNIX** per PDP-11 completamente riscritta inC con Dennis Ritchie

• PDP-11 (Programmed Data Processor)

– 2KB cache, 2 MB RAM

• Linguaggio C definito appositamente come evoluzione del rudimentale BCPL (Basic Combined Programming Language)

– Enorme successo grazie alla diffusione di PDP-11 nelle università

1979 – Rilascio di **UNIX v7**, “la” versione di riferimento

• Perfino Microsoft lo ha inizialmente commercializzato!

– Sotto il nome di Xenix, ma solo a costruttori dell’hardware degli elaboratori (p.es.: Intel)

• **Portabilità di programmi**

– Programma scritto in un linguaggio ad alto livello dotato di compilatore per più elaboratori

• È desiderabile che anche il compilatore sia portabile

– Dipendenze limitate ad aspetti specifici della architettura di destinazione

• Dispositivi di I/O, gestione interruzioni, gestione di basso livello della memoria

• **Diversificazione degli idiomi** (1979 – 1986)

– Avvento di v7 e divaricazione in due filoni distinti

• System V (AT&T . Novell . Santa Cruz Operation) – Incluso Xenix (Microsoft)

• 4.x BSD (Berkeley Software Distribution) – Incluso Virtual Memory e TCP/IP

• **Standardizzazione** (1986 – )

– POSIX (Portable Operating System Interface for UNIX) racchiude elementi selezionati di System V e BSD

• I più maturi e utili secondo l’opinione di esperti “neutrali” incaricati da IEEE e ISO/IEC

• Definisce l’insieme standard di procedure di libreria utili per operare su S/O compatibili

– La maggior parte contiene chiamate di sistema

– Servizi utilizzabili da linguaggi ad alto livello

Scelte architetturali per cloni UNIX:

– **Micro-kernel** : MINIX (Tanenbaum, 1987)

• Nel nucleo solo processi e comunicazione (affidabile e efficiente)

• Il resto dei servizi (p.es. : FS) realizzato in processi utente

– MINIX non copre tutti i servizi UNIX

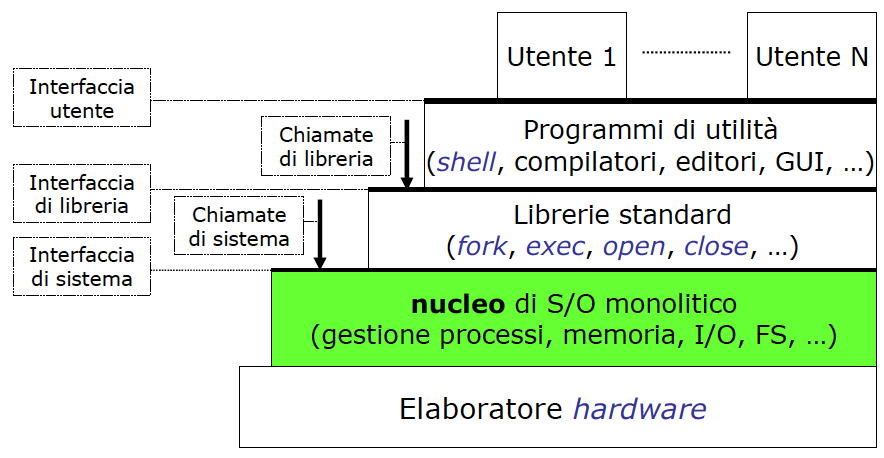
– **Nucleo monolitico** : GNU/Linux (Linus Torvalds, dav0.01 nel maggio 1991 a v3.8 di oggi)

• Clone completo aderente a POSIX con qualche libertà

– Il “meglio” di BSD e System V

• Modello open-source (scritto nel C compilato da gcc – GNU C compiler)

– Da non perdere: “Linux is obsolete”

**Livelli gerarchici**

Sistema multi-programmato multi-utente

Architettura a livelli gerarchici

**Interfaccia utente**

UNIX nasce con I/F per linea di comando (shell)

– Più potente e flessibile di GUI ma destinato a utenti esperti

– Una shell per ogni terminale utente (xterm)

• Lettura dal file “standard input”

• Scrittura sul file “standard output ”o“standard error ”

– Inizialmente associati al terminale stesso (visto come file)

• Possono essere re-diretti

– < per stdin, > per stdout

– Caratteri di prompt (%, $) indicano dove editare il comando

– Comandi compositi possono associare uscita di comandi ad ingresso di altri mediante | (pipe) e combinati in sequenze interpretate (script)

– In modalità normale la shell esegue un comando alla volta

– Comandi possono essere inviati all’esecuzione liberando la shell (&, background)

**Gestione dei processi (UNIX)**

**Processo**

– La principale entità attiva nel sistema

– Inizialmente definito come sequenziale

• Ossia dotato di un singolo flusso di controllo interno

– Concorrenza a livello di processi

• Molti processi attivati direttamente dal sistema (daemon)

– appuntamenti

• Creazione mediante fork()

– Clone con stessa memoria all’inizio e accesso a file aperti

• La discendenza di un processo costituisce un “gruppo”

• Comunicazione mediante scambio messaggi (pipe) e invio di segnali (signal) entro un gruppo

– Processi figli hanno memoria identica quella del processo genitore solo all’inizio

• Poi indipendente alla prima modifica

Processi con più flussi di controllo interni

– Detti thread

– La creazione di un thread gli assegna identità,attributi, compito e argomenti

– Thread condividono tutte le risorse logiche e fisichedel processo genitore (inclusi valori di variabili in uso)

Completato il proprio lavoro il thread termina se stesso volontariamente – Invocando la procedura pthread\_exit

Un thread può sincronizzarsi con la terminazione di un suo simile – Invocando la procedura pthread\_join

L’accesso a risorse condivise viene sincronizzato mediante semafori a mutua esclusione – Tramite le procedure pthread\_mutex{\_init, \_destroy}

L’attesa su condizioni logiche (p.es. : risorsa libera) avviene mediante variabili speciali simili a condition variables (ma senza monitor)

**Tabella dei processi**

– Permanentemente in RAM e per tutti i processi

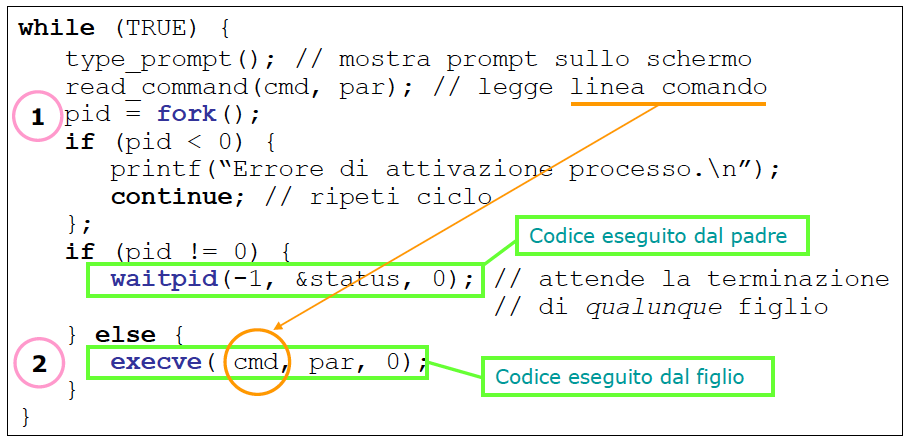
• Parametri di ordinamento (p.es. : priorità, tempo di esecuzione cumulato, tempo di sospensione in corso, …)

• Descrittore della memoria virtuale del processo

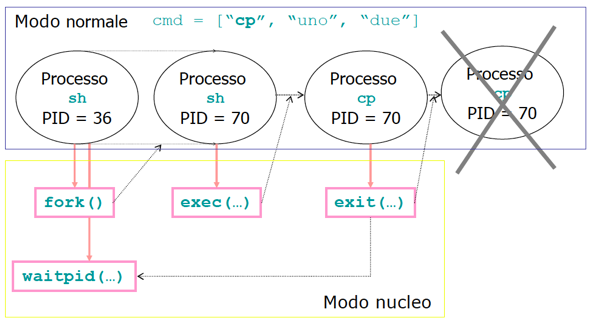
• Lista dei segnali significativi e del loro stato

• Stato, identità, relazioni di parentela, gruppo di appartenenza

**Esecuzione di comando di shell**

Codice semplificato di un processo shell

1) Il processo chiamante passa in modo nucleo e prova a inserire i propri dati per il figlio nella Tabella dei Processi (incluso il PID). Se riesce, alloca memoria per stack e dati del figlio. A questo punto il codice del figlio è ancora lo stesso del padre.



2) La linea di comando emessa dall’utente viene passata al processo figlio come array di stringhe. La exec, che opera in modo nucleo, localizza il programma da eseguire e lo sostituisce al codice del chiamante, passandogli la linea di comando e le “definizioni di ambiente” specifiche per il nuovo processo

**Gestione dei processi (UNIX)** – 5

• fork() duplica il processo chiamante creando un processo figlio uguale ma distinto

– Che accade se questi include più thread?

• Vi sono 2 possibilità

– Tutti i thread del padre vengono clonati

• Difficile gestire il loro accesso concorrente ai dati e alle risorse condivise conthread del padre

– Solo un thread del padre viene clonato

• Possibile sorgente di inconsistenza rispetto alle esigenze di cooperazionecon le thread non clonate

• Dunque il multi-threading aggiunge gradi di complessità

– Al FS

• Più difficile assicurare consistenza nell’uso concorrente di file

– Alla comunicazione tra entità attive

• Come decidere il thread destinazione di un segnale inviato a un processo?

• Un thread era bloccato da lettura da tastiera, i nuovi thread anche? E chiriceverà i caratteri digitati?

Maggior granularità nel trattamento della condivisione di strutture di controllo nella creazione

di processi e thread figli

Chiamata di sistema alternativa a fork()

pid = clone(function, stack\_ptr, ctrl, arg);

– function . programma da eseguire nel nuovo “task” (processo o thread) con argomento par

– Stack\_ptr . indirizzo dello stack assegnato al nuovo task

– ctrl . grado di condivisione desiderato tra il nuovo task e l’ambiente del chiamante

• Spazio di memoria, FS, file, segnali, identità

• Es. Solo copia o stesso address space?

**Ordinamento dei processi (GNU/Linux)**

I thread sono gestiti direttamente dal nucleo

– Ordinamento per task (thread o processo indistintamente)

– Selezione distinta tra classi distinte

– Prerilascio per fine quanto o per attesa di evento

• 3 classi di priorità di task

– Tempo reale con politica FCFS a priorità senza prerilascio

• A priorità uguale viene scelto il task in attesa da più tempo

– Tempo reale con politica RR a priorità

• Prerilascio per quanti con ritorno in fondo alla coda

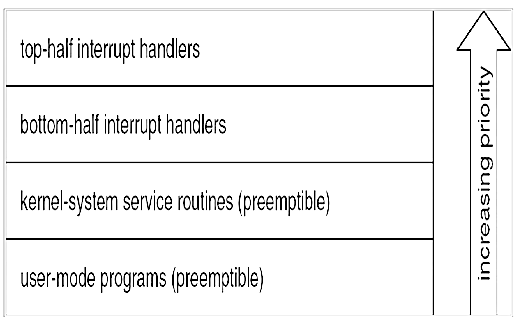
– Divisione di tempo RR a priorità (Timesharing)

• Priorità dinamica con premio o penalità rispetto al grado di interattività con I/O (alta . premio, bassa . penalità)

• Nuovo valore assegnato al task all’esaurimento del suo quanto corrente

Per versione < 2.6 l’attività dei processi in modo nucleo non ammetteva prerilascio

– Ma questo naturalmente causava pesanti problemi di inversione di priorità

Con versione ≥ 2.6 si usa granularità più fine

– Inibizione selettiva di prerilascio

• Per sezioni critiche corte

– Uso di semafori convenzionali

• Per sezioni critiche lunghe

– Uso minimale di disabilitazione delle interruzioni

• La parte immediata (top half) dei gestori disabilita

• La parte differita (bottom half) non disabilita ma il completamento della sua esecuzione viene preferito a ogni altra

– Tranne che di altre parti immediate

**Inizializzazione (GNU/Linux)**

BIOS carica l’MBR (Master Boot Record) da primo settore su disco di boot in RAM e lo “esegue”

– MBR = 1 settore di disco = 512 B

L’MBR carica il programma di boot dal corrispondente blocco della partizione attiva

– Lettura della struttura di FS, localizzazione e caricamento del nucleo di S/O

Il programma di inizializzazione del nucleo è scritto inassembler (specifico per l’elaboratore!)

– Poche azioni di configurazione di CPU e RAM

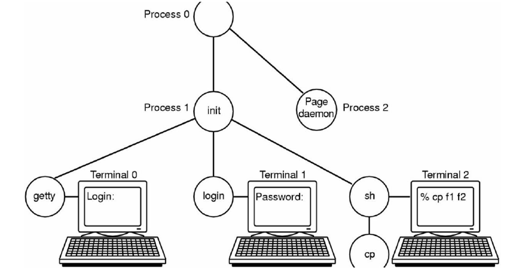
– Il controllo passa poi al main di nucleo

• Configurazione del sistema con caricamento dinamico dei gestori dei dispositivi rilevati

• Inizializzazione e attivazione del processo 0

• Processo 0 – Configurazione degli orologi, installazione del FS di sistema, creazione dei processi 1 (init) e 2 (daemon delle pagine)

• Processo 1 – Configurazione modo utente (singolo, multi)

• Esecuzione script di inizializzazione shell (/etc/rc etc.)

• Lettura numero e tipo terminali da /etc/ttys

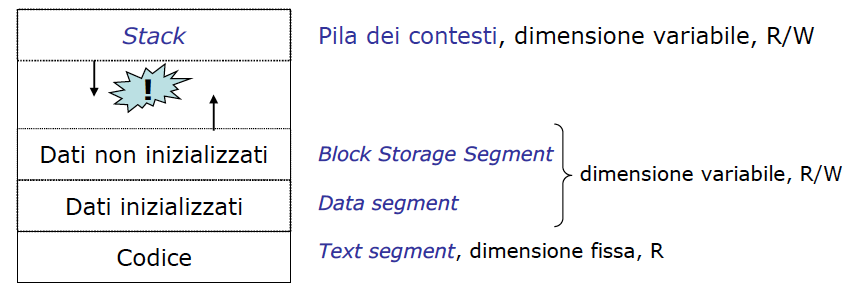
• fork() . terminale abilitato ed exec(“getty”)

• Processo getty

– Configurazione del terminale e attivazione del prompt di login

– Al login di utente, exec(“[/usr]/bin/login”) con verifica della password d’accesso (criptate in /etc/passwd)

– Infine exec(“shell”)

**Gestione della memoria**

Massima semplicità per massima portabilità su architetture fisiche diverse

Ogni processo possiede un proprio spazio di indirizzamento privato (memoria virtuale) – Suddiviso in 4 sezioni

Il segmento dati varia in dimensione a seconda delle attività del programma (cf. malloc() di C)

– POSIX non definisce queste chiamate di sistema

– Una parte del segmento dati può ospitare file mappati in memoria

Il segmento stack contiene l’ambiente d’esecuzione corrente (record di attivazione) e cresce in direzione opposta al segmento dati.

Il segmento codice può essere condiviso tra più processi

– Ma non gli altri segmenti tranne che per processi duplicati da fork()fintanto che non vengono modificati (Copy-on-write )

– contiene linguaggio macchina (da compilazione)

In origine l’allocazione di memoria principale avveniva mediante swap di processi

– Rimpiazzo di interi processi quando una particolare esecuzione rilevava mancanza di memoria

• A seguito di fork()

• A causa di allocazione esplicita richiesta dal programma

• Per allocazione implicita conseguente a chiamata di procedura

– Il gestore (swapper) creava lo spazio necessario salvando su disco i processi sospesi con più tempo d’esecuzione recente e minor priorità

In seguito, fu introdotta paginazione con modalità a richiesta (paging on demand)

– Un processo è eseguibile se il suo descrittore e la sua tabella delle pagine si trovano in RAM

• Il suo spazio di indirizzamento è caricato da disco per ogni riferimento che richiede dati non presenti in RAM

• Nessun caricamento anticipato di pagine

– No working set

– Il processo “2” gestisce lo stato dei page frame in RAM

• Page daemon

• Tenendo nucleo di S/O e “mappa delle pagine” (core map) sempre in RAM

• Il resto è paginato e ciascuna page frame indica il proprio uso

– Codice, dati, stack, tabella delle pagine (altrimenti in lista pagine libere)

**Mappa delle pagine (core map)**

Page daemon verifica con periodico 1/4 s che in RAM vi siano > lotsfree pagine liber

– Se ne mancano ne libera quante ne servono salvandone il contenuto corrente su un’area di disco specifica per pagina

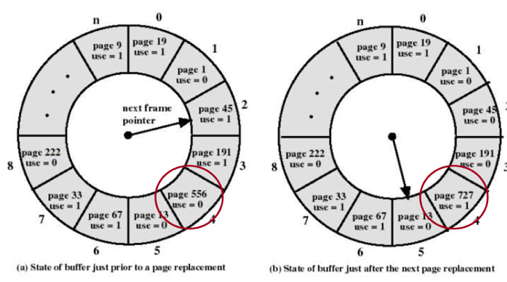
– La selezione delle pagine in uscita usa un algoritmo “a doppia passata”

• Two-handed clock algorithm

• Lista circolare delle pagine

• La 1a passata pone a 0 il bit di riferimento

• La 2a passata, a distanza programmabile, rimuove le pagine nel frattempo non riferite (bit vale 1 altrimenti)

**Orologio a una passata**

Se vi è spazio libero il page daemon riporta in RAM processi pronti selezionati con una euristica di “valore”

– Caricando solo il descrittore di processo e la sua tabella delle pagine

• Lasciando che il resto sia caricato via paging on demand

Per architetture a 32 bit la memoria virtuale di processo è ampia 4 GB

– 1 GB riservato e invisibile al modo operativo normale (user mode) per la tabella delle pagine del processo e per altri dati di controllo a uso del nucleo

Spazio suddiviso in regioni = sequenze contigue di pagine

Ogni regione ha un descrittore noto al nucleo

– Info su contenuto, pagine attive/inattive, aree libere

La fork() di GNU/Linux replica per il figlio l’intera lista di descrittori del padre

Le pagine del figlio sono fisicamente duplicate solo in caso di modifica (copy on write)

– La regione è marcata R/W

– Le sue pagine dati sono inizialmente marcate R

– Ogni richiesta di scrittura causa eccezione così il nucleo duplica la pagina richiesta e marca la copia come R/W

Il nucleo rimane sempre in RAM

Ad ogni istante nella RAM rimanente possono trovarsi:

– Le pagine attive dei processi utente

– Un insieme di pagine utente inattive ma presenti

– Una cache di blocchi di file usata dal file system

• Dimensione variabile organizzata per pagine

kswapd è il page daemon (periodo 1 s) checerca pagine da spostare su disco

Il daemon bdflush gestisce la riscrittura delle pagine modificate

La RAM è allocata in maniera dinamica e variabile

Algoritmo di allocazione primario (buddy)

– Ogni richiesta di ampiezza N è arrotondata a 2m . N

– La memoria disponibile viene frazionata in metà successive fino a frazioni di ampiezza 2m

• Una singola frazione viene assegnata al richiedente

• Una struttura ausiliaria contiene la testa di liste predefinite di frazioni di ampiezza 2i (i=0, …, m) per velocizzare la ricerca

– La memoria disponibile usata per l’allocazione è sempre la frazione libera di minore dimensione

– Al rilascio ogni frazione tornata libera si unisce con la frazione vicina se libera (il suo buddy)

– algoritmi sussidiari cercano di ridurre la frammentazione causata dall’algoritmo primario

• Strutture slab per evitare di usare 128 pagine per allocarne 65

**Gestione dell’I/O**

UNIX tratta i dispositivi di I/O come file di tipo speciale, ciascun con posizione specifica nel FS – Per esempio /dev/…

Un gestore (device driver) è associato in modo esclusivo a ciascun dispositivo o a famiglia di dispositivi dello stesso tipo

– Una coppia di indici <maggiore, minore> identifica precisamente ciascun dispositivo di I/O

• Maggiore: tipologia

• Minore: specifico di quel dispositivo

GNU/Linux consente invece caricamento dinamico dei moduli di gestione dei dispositivi

– Soluzione molto preferibile alla configurazione statica che richiede ogni volta una nuova compilazione dell’intero nucleo

• Inevitabile a fronte della grande varietà di hardware attuale

Il caricamento dinamico richiede al nucleo di effettuare diverse azioni di configurazione

– [1] Rilocazione dello spazio di indirizzamento del modulo

– [1-2] Allocazione delle risorse necessarie

• P. es.: interruzione assegnata al dispositivo

– [2] Configurazione del vettore delle interruzioni

– [2] Attivazione e inizializzazione del gestore

Un file speciale (detto socket) viene utilizzato per la connessione di rete e i relativi protocolli

– Può essere creato e distrutto dinamicamente

– Un socket è associato a uno specifico indirizzo di rete

Tre tipi di connessione con scelta alla creazione

– Connessione affidabile a flusso di caratteri (~ TCP)

• Il gestore garantisce la correttezza della trasmissione

• Invio e ricezione per blocchi di dimensione variabile

– Connessione affidabile a flusso di pacchetti (TCP)

• Come sopra, ma con invio e ricezione solo per pacchetti

– Trasmissione inaffidabile di pacchetti (UDP)

• L’utente deve occuparsi di trattare gli eventuali errori

**Caratteristiche del File System**

File visto da FS come sequenza di byte di significato arbitrario

– Fissato dal programma applicativo

File regolari, file repertorio (directory) e file speciali che mappano dispositivi di I/O

Nome inizialmente limitato a 14 caratteri (UNIX v7)

Poi esteso fino a 255 (UNIX BSD . GNU/Linux)

– Estensione non obbligatoria

– Convenzione di estensione a scelta del programma applicativo e/o dell’utente

File designato mediante cammino (path) assoluto o relativo

– Il cammino relativo richiede la nozione di directory di lavoro corrente

* pwd per visualizzarne la posizione assoluta
* cd per cambiare posizione

– Un intero FS B posto su una partizione visibile può essere ritenuto come parte di un FS A mediante mount

• La radice di B viene designata con un nome (cammino) specifico in A detto mount point

Controllo di accessi concorrenti (locking) -POSIX

– A grana grossa (per directory o per file)

• Mediante uso esplicito di semafori convenzionali

– A grana fine (per gruppi di byte in un file)

• Mediante meccanismi dedicati

Due distinte modalità d’uso

– Accesso simultaneo condiviso (shared lock)

• Più accessi R alla stessa zona ma anche a zone solo parzialmente

sovrapposte

– Accesso esclusivo (exclusive lock)

• Consente un solo accesso per zona selezionata

**Esempi di chiamate di File System**

Disponibili all’utente solo indirettamente tramite incapsulazione in procedure di libreria

– **lseek**

• Fissa l’indice di posizione all’interno di un file

– Come offset espresso in byte rispetto ad un riferimento dato

• Accesso diretto

– **stat**

• Fornisce informazioni su file prelevandole dall’i-node corrispondente

• Chiamata incapsulata dal comando stat di shell

– Provare per esercizio dopo aver letto “man stat“ .

**Realizzazione del FS in UNIX**

Struttura di partizione secondo UNIX v7

Il super-blocco (1) indica tra l’altro il # di i-node e di blocchi nel FS e fornisce il puntatore alla lista dei blocchi liberi (2)

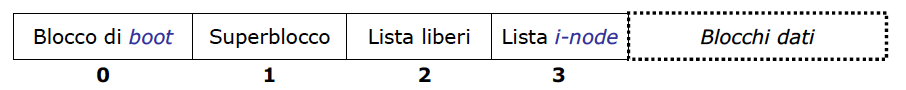
Gli i-node (3) sono numerati 1..N

Directory come insieme variabile e non ordinato di unità informative (entry)

– Ampie 16 B

• 14 B (codifica ASCII) per nome di file

• 2 B per numero di i-node

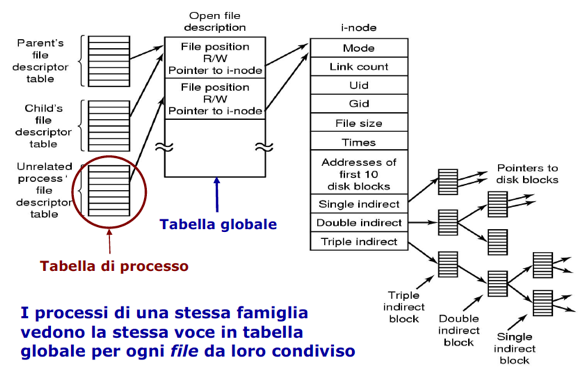


In nucleo usa due strutture di controllo

– Un insieme di tabelle di processo contiene “descrittori utente” dei file attualmente in uso a ciascun processo

• A ogni descrittore utente deve corrispondere l’attuale posizione di R/W

• Però ogni processo deve avere il suo proprio indice di posizione sui propri file aperti

 – Possono esistere più posizioni di R/W su uno

stesso file condiviso

– L’indice non può essere ritenuto nell’i-node

che è unico per file !

Una tabella globale mantiene la corrispondenza tra tutti i file aperti e i loro i-node

• Ciascuna voce nella tabella di processo punta a una voce nella tabella globale che specifica diritti e posizione di R/W corrente nel file

– La stessa voce . file condiviso da processi di una stessa famiglia

– Voce diversa per stesso file per processi non apparentati

L’i-node principale del file contiene (tra l’altro) l’indirizzo dei suoi primi 12 blocchi dati

– 1 i-node ha la dimensione di 1 frazione di blocco (64 B)

Per file più grandi 1 campo dell’i-node principale punta a 1 i-node secondario che contiene puntatori ad altri blocchi dati

– I-node principale con campo single-indirect

Per file ancora più grandi l’i-node secondario contiene puntatori a nodi single-indirect

– I-node principale con campo double-indirect

È previsto anche un campo triple-indirect

**Esempio d’uso di i-node** (UNIX v7)

• Ipotesi

– Blocco dati di capienza 4 KB

– i-node ampio 64 B

– Indici di blocco espressi su 4 B

• Esempio 1 (con uso di campo single-indirect)

– Max dimensione di file rappresentabile

• (12 . 64 B / 4 B) . 4 KB . (12 . 16) . 4 KB . 112 KB

• Esempio 2 (con uso di campo double-indirect)

– Max dimensione di file rappresentabile

• 112 KB . 162 . 4 KB . 1 MB . 112 KB

• Esempio 3 (con uso di campo triple-indirect)

– Max dimensione di file rappresentabile

• 1 MB . 112 KB . 163 . 4 KB . 17 MB . 112 KB

La versione BSD introduce alcune migliorie importanti

– Estensione del nome di file fino a 255 caratteri

– Directory di dimensione multipla di blocco

• Facilita e velocizza la scrittura su disco

• Comporta frammentazione interna

**Realizzazione del FS GNU/Linux**

Inizialmente basato sul FS di MINIX però subito abbandonato per le eccessive limitazioni

– Limitazioni imposte da MINIX

• Nomi . 14 caratteri

• Indirizzi di blocco su 2 B per blocchi ampi 1 KB

• Ampiezza massima di file ≤ 64 MB (perché?)

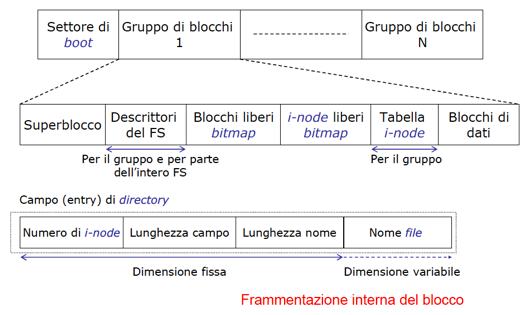
ext2 diviene presto la versione di riferimento

– Basata sulle scelte BSD con diversa struttura fisica

– La maggiore innovazione è stata la suddivisione della partizione in gruppi di blocchi

• Distribuzione uniforme delle directory su disco

• i-node e relativi blocchi dati sono tenuti vicini tramite preallocazione di alcuni blocchi al momento della creazione di un file – Riduce frammentazione



Dimensione di i-node estesa a 128 B

– Indirizzi di blocco ampi 4 B

• Per denotare fino a 232 = 4 G blocchi

– Blocchi di dimensione 1, 2, 4 KB scelta in fase di configurazione del FS

• Partizione di dimensione ≥ 4 TB

• 12 indirizzi diretti + 3 indiretti (single, double, triple)

• Informazioni di controllo

• Una parte riservata per uso futuro

Ogni aggiunta a file viene realizzata quanto più localmente possibile entro lo stesso gruppo

– Località tra file correlati tramite gruppi

– Località entro file mediante preallocazione di N < 8 blocchi contigui