

Da UNIX a GNU/Linux
Parte 1 - Indice

1. Genesi
2. Visione generale
3. Gestione dei processi
4. Ordinamento dei processi
5. Inizializzazione

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 127

Da UNIX a GNU/Linux
Genesi - 1

- **DTSS** (Dartmouth College Time Sharing System, 1964)
 - Il primo elaboratore multiutente a divisione di tempo
 - Programmato in BASIC ed 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 ad utente singolo
- **UNICS** (UNiplexed "ICS", 1969) UNIX

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 128

Da UNIX a GNU/Linux
Genesi - 2

- **1974**
 - Nuova versione di UNIX per PDP-11, completamente riscritta in C con Dennis Ritchie
 - Linguaggio definito appositamente come evoluzione del rudimentale BCPL (*Basic Combined Programming Language*)
 - Enorme successo grazie alla diffusione di PDP-11 (*Programmed Data Processor*) nelle università
 - 2 kB *cache*, 2 MB RAM
- **1979**
 - Rilascio di **UNIX v7**, "la" versione di riferimento
 - Perfino Microsoft lo ha inizialmente commercializzato!
 - Sotto il nome di **Xenix**, ma solo a costruttori di *computer* (p.es.: Intel)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 129

Da UNIX a GNU/Linux
Genesi - 3

- **Portabilità di programmi**
 - Programma scritto in un linguaggio ad alto livello, dotato di compilatore per più elaboratori
 - È desiderabile che il compilatore stesso sia portabile
 - Dipendenze limitate ad aspetti specifici della architettura 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 ...
 - **4.x BSD** (Berkeley Software Distribution)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 130

Da UNIX a GNU/Linux
Genesi - 4

- **Standardizzazione** (1986 -)
 - **POSIX** (Portable Operating System Interface for UNIX) racchiude e completa il meglio di **System V** e **BSD** secondo esperti "neutrali"
 - IEEE → ISO/IEC
 - Definisce l'insieme standard di procedure di libreria
 - La maggior parte contiene chiamate di sistema
 - Servizi utilizzabili da linguaggi ad alto livello

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 131

Da UNIX a GNU/Linux
Genesi - 5

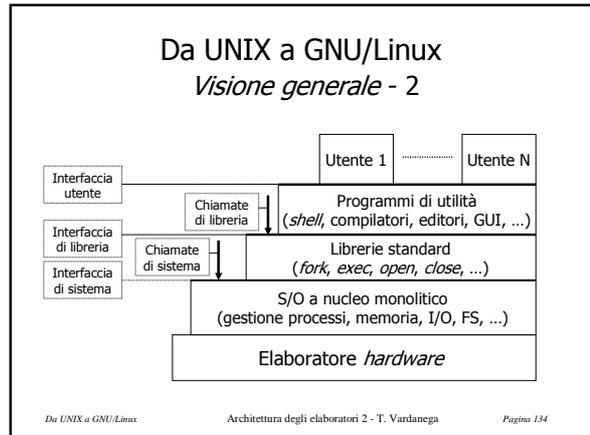
- **Scelte architetturali per cloni UNIX**
 - **Microkernel**: MINIX (Tanenbaum, 1987)
 - Nel nucleo solo processi e comunicazione; il resto dei servizi (p.es. : FS) realizzato in processi utente; non copre tutti i servizi UNIX
 - **Nucleo monolitico** : GNU/Linux (Linus Torvalds, 1991)
 - Clone completo, 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**)
 - Parallelo al progetto **GNU** (GNU is NOT UNIX!) <http://www.gnu.org>

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 132

Da UNIX a GNU/Linux Visione generale - 1

- Sistema **multiprogrammato** multiutente
 - Orientato allo **sviluppo** di applicazioni
 - Progettato per semplicità, eleganza e consistenza
 - **NO** a complessità da compromessi di compatibilità verso il passato
 - **SI** a specializzazione e flessibilità
 - Ogni servizio fa una sola cosa (bene)
 - Facile combinare liberamente più servizi in attività più sofisticate
- Architettura a livelli gerarchici

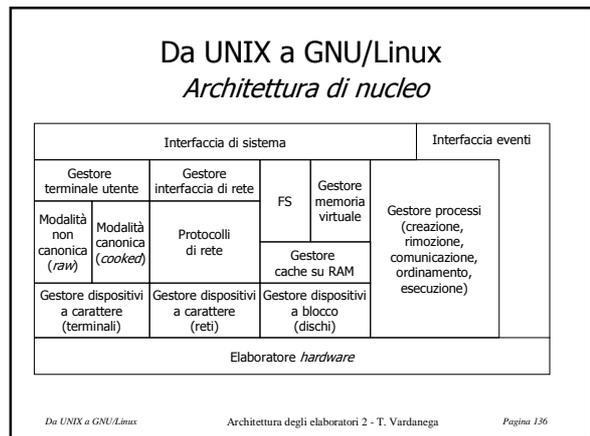
Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 133



Da UNIX a GNU/Linux Interfaccia utente

- UNIX nasce con interfaccia per linea di comando (*shell*)
 - Più potente e flessibile di GUI, ma intesa per utenti più esperti
 - Una *shell* per ogni terminale (**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 all'utente dove editare il comando
 - Comandi composti 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*)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 135



Da UNIX a GNU/Linux Gestione dei processi (UNIX) - 1

- **Processo**: la sola entità attiva nel sistema
 - Inizialmente definito come sequenziale (a singolo flusso di controllo)
 - Concorrenza a livello di processi
 - Molti attivati direttamente dal sistema (*daemon*)
 - Creazione mediante `fork()`
 - La discendenza di un processo costituisce un "gruppo"
 - Comunicazione mediante scambio messaggi (*pipe*) ed invio di segnali (*signal*) entro un gruppo

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 137

Da UNIX a GNU/Linux Gestione dei processi (UNIX) - 2

- Processi con più flussi di controllo (*thread*)
 - 2 scelte realizzative possibili
 - Nello spazio di nucleo
 - Nello spazio di utente
 - Più facile da aggiungere
 - Il nucleo però continua ad ordinare per processi
 - POSIX non impone una particolare scelta
 - Una *thread* può svolgere compiti specializzati
 - La creazione di una *thread* le assegna identità, attributi, compito ed argomenti

```

res = pthread_create( (&tid), attr, fun, arg )
    
```

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 138

Da UNIX a GNU/Linux Gestione dei processi (UNIX) - 3

- Completato il lavoro, la *thread* termina se stessa volontariamente (`pthread_exit`)
- Una *thread* può sincronizzarsi con la terminazione di un'altra (`pthread_join`)
- L'accesso a risorse condivise viene sincronizzato mediante semafori a mutua esclusione
`pthread_mutex{ _init, _destroy }`
- L'attesa su condizioni logiche (p.es. : risorsa libera) avviene mediante variabili speciali simili a *condition variables* (ma senza monitor)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 139

Da UNIX a GNU/Linux Gestione dei processi (UNIX) - 4

- Una *thread* può avere 2 "modi" operativi
 - **Normale** : esecuzione nello spazio di utente (propri diritti, memoria virtuale, risorse)
 - **Nucleo** : esecuzione con i privilegi, la memoria virtuale e le risorse di nucleo
 - Il cambio di modo consegue ad una chiamata di sistema
 - Chiamata incapsulata in una *procedura di libreria*, per consentirne l'invocazione da C

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 140

Da UNIX a GNU/Linux Gestione dei processi (UNIX) - 5

- 2 strutture di nucleo per gestione processi
 - **Tabella dei processi**
 - Permanentemente in RAM, descrive 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
 - **Descrittore degli utenti**
 - In RAM solo per i processi attivi
 - Parte del contesto (immagine dei registri dell'elaboratore)
 - Stato dall'ultima chiamata di sistema (parametri, risultato)
 - Tabella dei descrittori dei *file* aperti
 - Crediti del processo
 - *Stack* da usare in modo nucleo

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 141

Esempio Esecuzione di comando di shell - 1

Codice semplificato di un processo *shell*

```

while (TRUE) {
    type_prompt(); // mostra prompt sullo schermo
    read_command(cmd, par); // legge linea comando
    1 pid = fork();
    if (pid < 0) {
        printf("Errore di attivazione processo.\n");
        continue; // ripeti ciclo
    };
    if (pid != 0) {
        waitpid(-1, &status, 0); // attende la terminazione
        // di qualunque figlio
    }
    2 else {
        execve(cmd, par, 0);
    }
}
    
```

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 142

Esempio Esecuzione di comando di shell - 2

1 Il processo chiamante passa in **modo nucleo** e prova ad inserire i dati sul figlio nella Tabella dei Processi (incluso il suo PID). Se riesce, alloca memoria per *stack* e dati del figlio. 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" per il processo

In realtà, le aree dati e codice sono copiate per il figlio solo se questi le modifica (*copy-on-write*) e caricate solo quando riferite (*page-on-demand*)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 143

Esempio Esecuzione di comando di shell - 3

Modo normale cmd = ["cp", "uno", "due"]

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 144

Da UNIX a GNU/Linux

Gestione dei processi (UNIX) - 6

- **fork()** clona il processo chiamante (padre)
 - Che accade se questi include più *thread*?
- 2 possibilità
 - **Tutte le *thread* del padre vengono clonate**
 - Difficile gestire l'accesso concorrente ai dati ed alle risorse necessarie alle *thread* del padre
 - **Solo una *thread* del padre viene clonata**
 - Possibile inconsistenza nella sua aspettativa di cooperazione con le *thread* non clonate
- Il *multi-threading* aggiunge complessità
 - Al FS: consistenza nell'uso concorrente di *file*
 - Alla comunicazione: quale *thread* è destinataria di un segnale inviato ad un processo?

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 145

Da UNIX a GNU/Linux

Gestione dei processi (GNU/Linux) - 7

- Maggior granularità nel trattamento delle strutture di controllo dei processi e della creazione di *thread*
- Chiamata di sistema per creazione di *thread*

```
pid = clone(fun, stack, ctrl, par);
```

 - *fun* : programma da eseguire nella *thread*, con argomento *par*
 - *stack* : indirizzo dello *stack* assegnato alla nuova *thread*
 - *ctrl* : livello di condivisione tra la nuova *thread* e l'ambiente del chiamante
 - Spazio di memoria, FS, *file*, segnali, identità

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 146

Da UNIX a GNU/Linux

Ordinamento dei processi (UNIX) - 1

- **Sistema destinato ad uso interattivo**
 - Politica di ordinamento con prerilascio
 - Deve garantire buoni tempi di risposta e soddisfare le aspettative degli utenti
- **Algoritmo di ordinamento a 2 livelli**
 - Livello basso (*dispatcher*)
 - Seleziona un processo tra quelli pronti (in RAM)
 - Livello alto (*scheduler*)
 - Assicura che tutti i processi abbiano opportunità di esecuzione spostando processi tra RAM e disco

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 147

Da UNIX a GNU/Linux

Ordinamento dei processi (UNIX) - 2

- **Politica di ordinamento**
 - Selezione per priorità decrescente
 - Priorità alta (< 0) per esecuzione in modo nucleò
 - Priorità bassa (> 0) per esecuzione in modo normale
 - Più processi possono avere la stessa priorità
 - 1 coda di processi \forall gruppo di priorità distinte
 - Selezione *round robin* da testa della coda
 - Esecuzione a quanti con ritorno in fondo alla coda
 - Aggiornamento periodico della priorità dei processi

$$\text{Priorità} = (\text{consumo}) + \text{cortesía} + (\text{urgenza})$$

Esecuzione media per 1 s (>0)

Importanza evento atteso (<0)

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 148

Da UNIX a GNU/Linux

Ordinamento dei processi (UNIX) - 3

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 149

Da UNIX a GNU/Linux

Ordinamento dei processi (GNU/Linux) - 1

- Le *thread* sono gestite direttamente dal nucleò
 - Ordinamento per *thread*, non per processi
 - Selezione per migliore fattore di *goodness*
 - Prerilascio per priorità, fine quanto ed attesa evento
- 3 classi di priorità di *thread*
 - **Tempo reale con FIFO**
 - Prerilascio a priorità solo tra *thread* di stesso livello (= quanto infinito)
 - *Goodness* alta (= 1000 + priorità)
 - **Tempo reale con quanti**
 - Prerilascio per quanti con ritorno in fondo alla coda
 - *Goodness* media (= ultimo quanto non utilizzato + priorità)
 - **Divisione di tempo**
 - *Goodness* bassa (= 0)
- Quando $Q_i = 0$ per *thread* $i \rightarrow Q_j = Q_j/2 + \text{priorità}, \forall j$

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 150

Da UNIX a GNU/Linux Inizializzazione (GNU/Linux) - 1

- BIOS carica l'MBR 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 in *assembler* (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 ed attivazione del processo 0

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 151

Da UNIX a GNU/Linux Inizializzazione (GNU/Linux) - 2

- **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/tty*s
 - **fork()** √ terminale abilitato ed **exec("getty")**
- **Processo getty**
 - Configurazione del terminale ed 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")** e poi si procede come mostrato alle pagine 142-144

Da UNIX a GNU/Linux Architettura degli elaboratori 2 - T. Vardanega Pagina 152

