Introduzione a Linux

Linux teoria
Parte 1

Ruggero Donida Labati

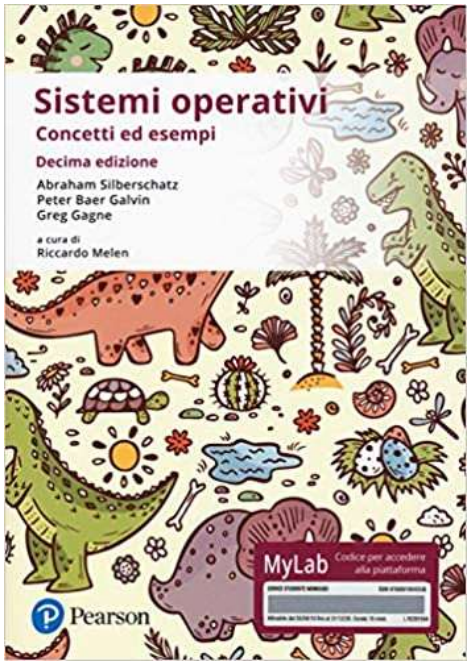
Laboratorio di Sistemi Operativi

Università degli Studi di Milano
Dipartimento di Informatica

A.A. 2022/2023

Panoramica della lezione

- Verranno presentate alcune caratteristiche di progettazione del sistema operativo Linux
- Saranno descritti alcuni componenti del kernel di Linux
- Saranno illustrati i meccanismi di gestione dei processi in Linux

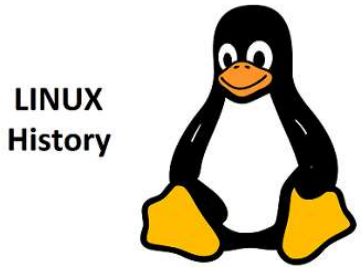


Sommario

1. Storia di Linux
2. Principi di progettazione
   - Sviluppo del software
   - Caratteristiche di progettazione
3. Componenti del sistema operativo
   - Elenco dei componenti
   - Kernel
4. Gestione dei processi
   - Descrizione
   - Scheduling
   - Processi e thread
   - Scheduling
   - Sincronizzazione

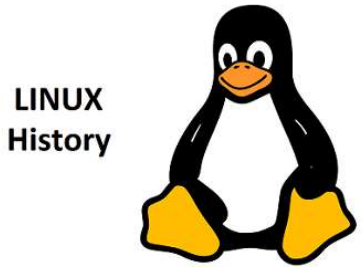
1. Storia di Linux (1/3)

- Basato su Unix
- Kernel sviluppato nel 1991 da Linus Torvalds
  - Compatibilità Unix
  - Open source
- Sviluppato in modo collaborativo dalla comunità
  - Attraverso Internet
- Compatibilità e affidabilità su numerose piattaforme hardware
- Numerose distribuzioni
  - Kernel, applicazioni, software di gestione



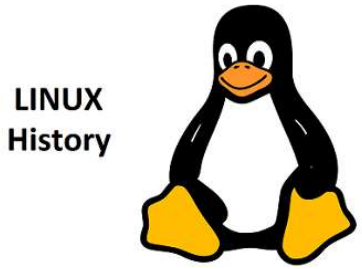
1. Storia di Linux (2/3)

- Versione 0.01 (1991)
  - No rete
  - Solo 80386
  - Solo Minix file system
  - Pochissimi driver
- Versione 1.0 (1994)
  - TCP/IP, socket
  - Supporto file system
  - Supporto periferiche memorizzazione
- Versione 1.2 (1995)
  - Kernel Linux solo PC



1. Storia di Linux (3/3)

- Versione 2.0 (1996)
  - Architetture multiple
    - 64 bit
    - Motorola, Sun Sparc, PC, Mac
  - Multiprocessore
  - Gestione memoria migliorata
  - Miglioramenti kernel
    - Thread
    - Caricamento moduli
  - Interfaccia di configurazione standard
- Versione 3.0 (2011)
  - Virtualizzazione
  - Completely Fair Scheduler

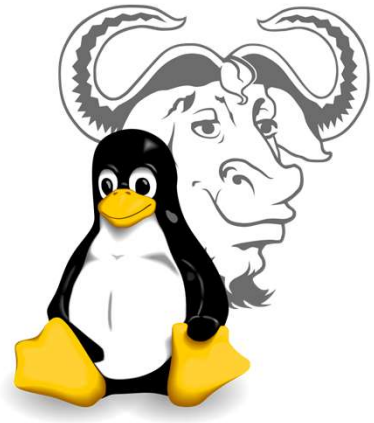


2. Principi di progettazione

1. Sviluppo del software
2. Caratteristiche di progettazione

Sviluppo del software (1/3)

- Uso di diversi tool esistenti
  - BSD
  - MIT X Window
  - GNU project
- Collaborazione distribuita
  - Sviluppatori collaborano attraverso internet
  - FTP funzionano come repository «standard» de facto
  - File System Hierarchy Standard
    - Assicura compatibilità tra i vari componenti
    - Layout del file system
    - Posizione dei file di configurazione e binari



Sviluppo del software (2/3)

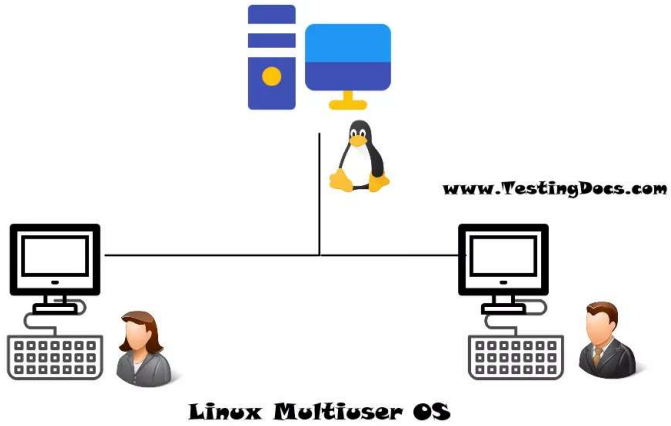
- Distribuzioni
  - Kernel Linux
  - Insieme di pacchetti precompilati
    - RPM: compatibilità tra distribuzioni diverse
  - Software di installazione e gestione
  - Alcune distribuzioni più «popolari»
    - Debian
    - SuSe
    - Red Hat
    - …

Sviluppo del software (3/3)

- Software «free»
  - GNU General Public License (GPL)
  - Le modifiche non possono essere rese proprietarie
  - Software modificato non può essere distribuito solo come binario
- Programmazione
  - Interfaccia di programmazione rispetta la semantica SVR4 UNIX

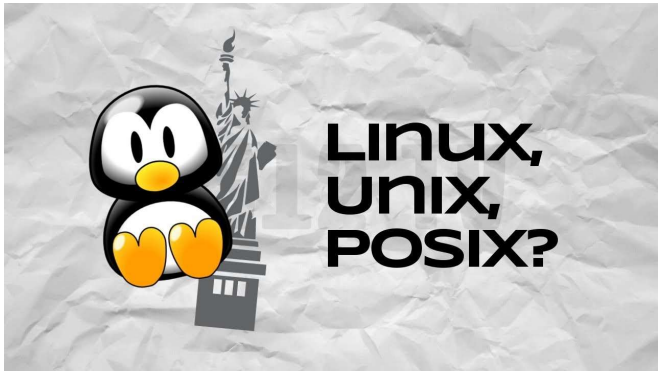
Caratteristiche di progettazione (1/2)

- Multiprogrammazione
  - Multi-utente
  - Multi-tasking
- File system
  - Semantica UNIX
- Rete
  - Modello di rete UNIX



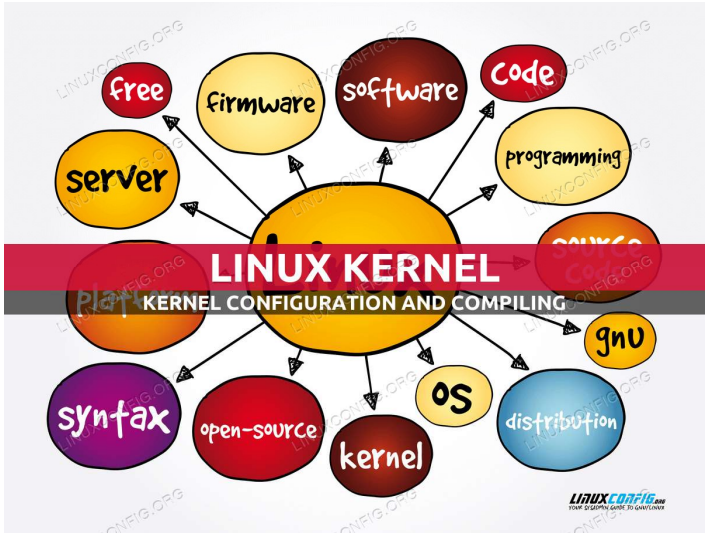
Caratteristiche di progettazione (2/2)

- POSIX
  - Portable Operating System Interface
  - Standard per la comunicazione tra sistemi operativi
  - Alcune distribuzioni sono certificate POSIX
    - Pthreads
    - Controllo processi POSIX real-time

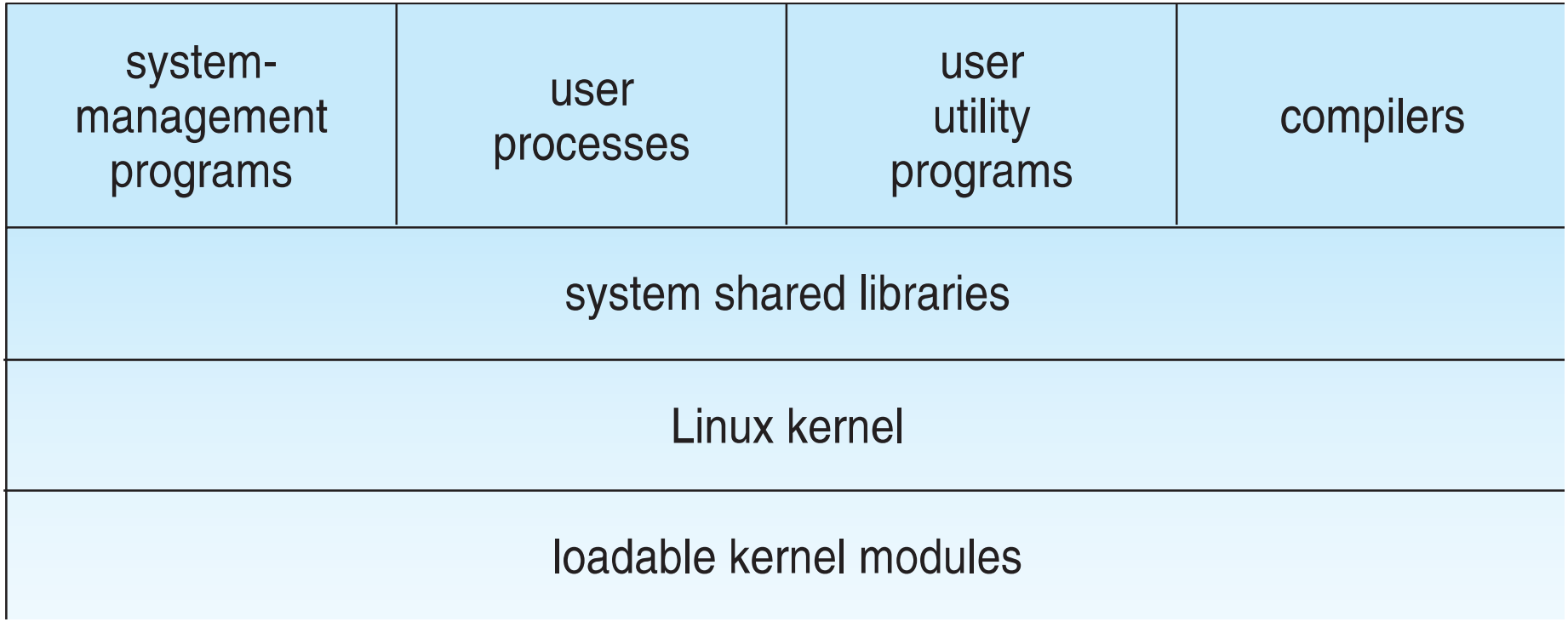


3. Componenti del sistema operativo

1. Elenco dei componenti
2. Kernel

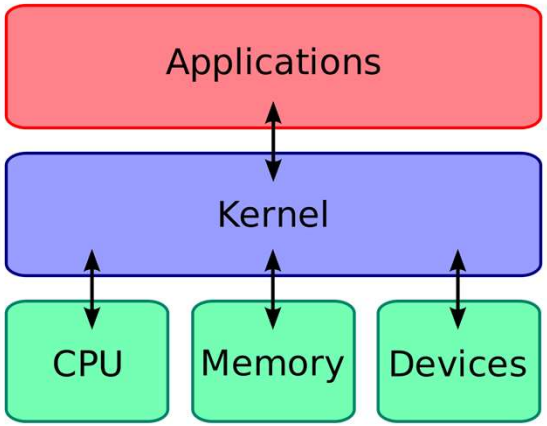


Elenco dei componenti (1/3)



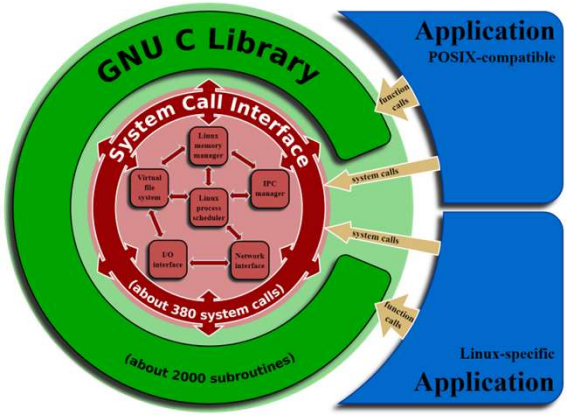
Elenco dei componenti (2/3)

- Due componenti fondamentali
  - Kernel
  - «Tutto il resto»
- Kernel
  - «Cuore» del sistema operativo
  - Contiene le primitive per la gestione dell'hardware
- «Tutto il resto»
  - Librerie di sistema
  - Utility di sistema
  - Applicazioni



Elenco dei componenti (3/3)

- Librerie di sistema
  - Definiscono un insieme standard di funzioni
  - Interazione tra applicazioni e kernel
  - Funzionalità del kernel che non richiede privilegi di esecuzione in modalità kernel
- Utility di sistema
  - Funzionalità di gestione specifiche
- Applicazioni utente
  - E.g. shells (bash)

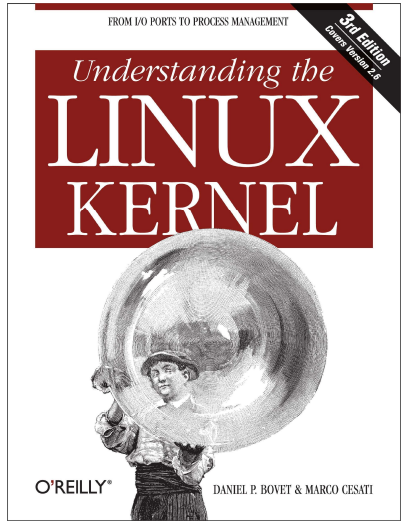


Kernel (1/8)

- Kernel
  - Realizza l'astrazione dell'hardware
  - Esegue codice in modalità kernel
    - Accesso completo a tutte le risorse fisiche della macchina
  - Codice e strutture dati del kernel sono mantenute in un unico spazio di indirizzamento

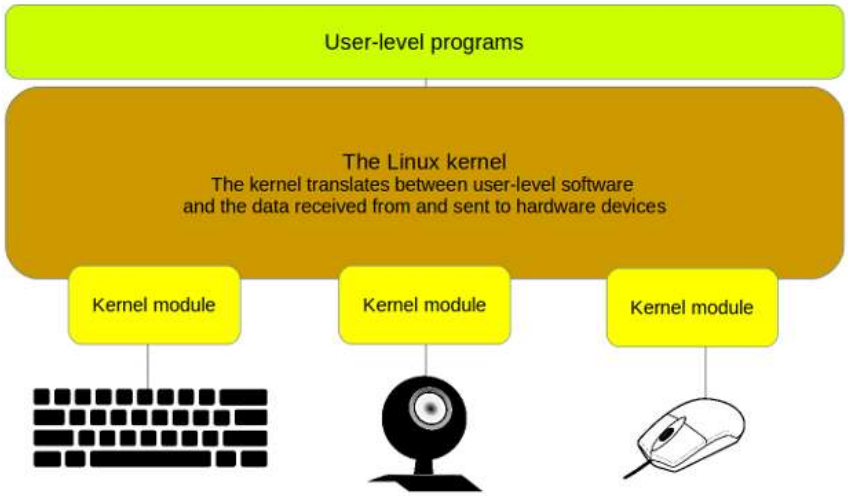
Kernel (2/8)

- Scritto sulla base di diversi kernel UNIX
- Monolitico
  - Contiene tutte le funzioni di base del sistema operativo e i driver per gestire l'hardware
  - Organizzato in moduli logici
    - Moduli aggiuntivi possono essere caricati a runtime
- Multi-threaded



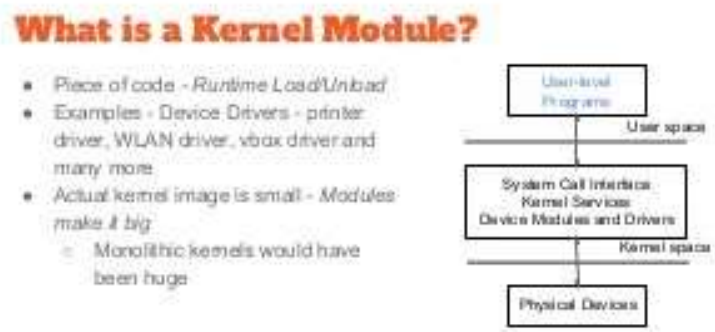
Kernel (3/8)

- Moduli del kernel
  - Sezioni del kernel che possono essere compilate, caricate e scaricate in modo indipendente dal resto del kernel
  - Componenti non strettamente legati alle primitive di gestione dell'hardware
    - Driver di dispositivo
    - File system
    - Protocollo di rete



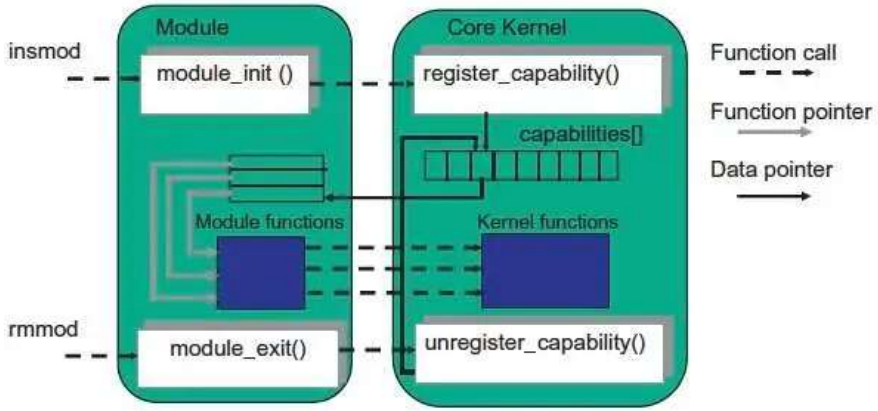
Kernel (4/8)

- Interfaccia standard per moduli
  - Terze parti possono progettare e distribuire moduli kernel che non possono essere distribuiti come GPL
  - Possibilità di distribuire un kernel «minimale», senza moduli aggiuntivi



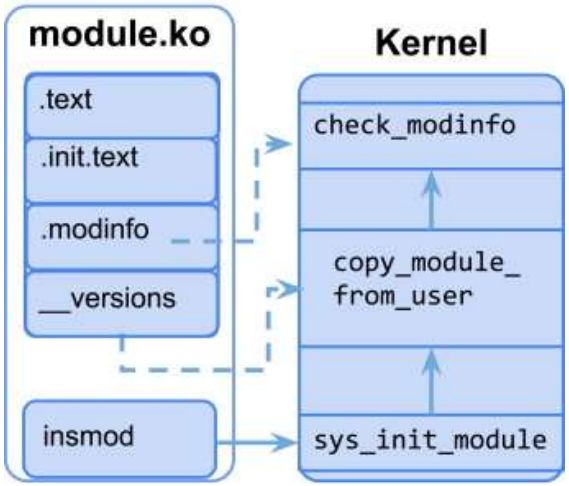
Kernel (5/8)

- Componenti di gestione dei moduli Kernel
  - *module-management system*
  - *module loader and unloader*
  - *driver-registration system*
  - *conflict-resolution mechanism*



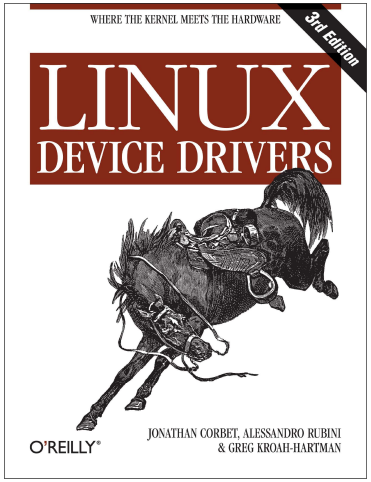
Kernel (6/8)

- Module management
  - Caricamento dei moduli in memoria
  - Comunicazione con il resto del kernel
  - «Pulizia»
    - Chiede regolarmente al kernel se un modulo è ancora in uso
    - Scarica dalla memoria in caso negativo



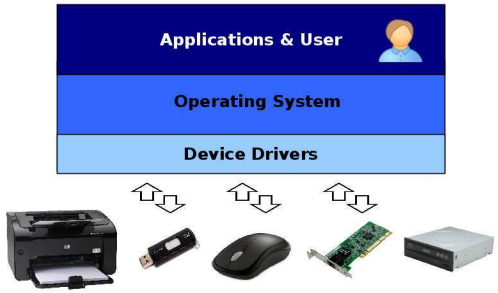
Kernel (7/8)

- Driver registration
  - Permette ai moduli di dire al kernel che un nuovo driver è disponibile
  - Il kernel mantiene una tabella di tutti i driver a disposizione
    - Aggiornata dinamicamente
    - Routine per aggiungere o rimuovere driver dalla tabella
  - Tabelle del kernel
    - Driver di dispositivo
    - File system
    - Protocolli di rete
    - Formati binari



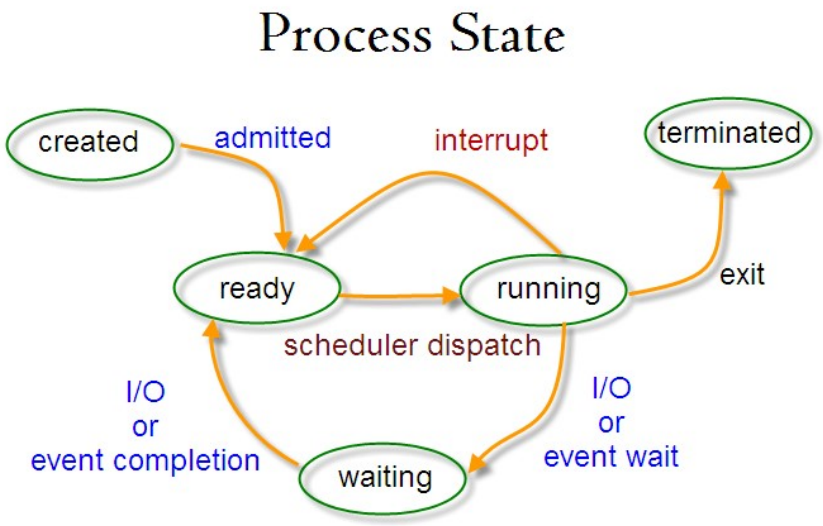
Kernel (8/8)

- Conflict resolution
  - Meccanismo che permette ai driver di «prenotare» risorse hardware
  - Proteggere le risorse dall'uso accidentale da parte di altri driver
  - Risolvere i conflitti causati da più driver che vogliono accedere allo stesso hardware
    - Lista aggiornata delle risorse hardware allocata
    - Il driver chiede la risorsa controllando la lista
    - La richiesta viene permessa o negata in base alla lista di allocazione



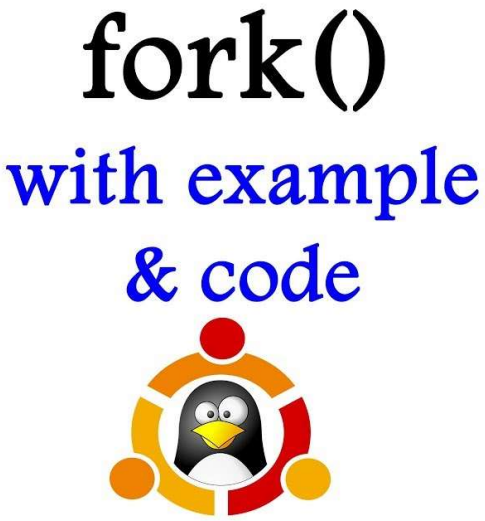
4. Gestione dei processi

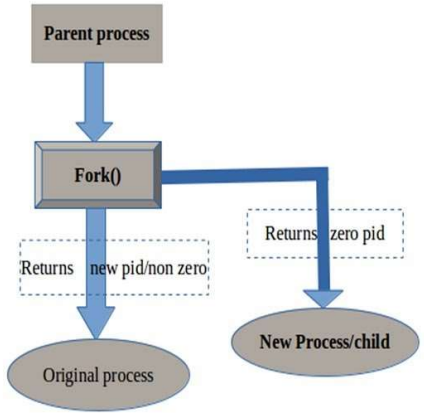
1. Creazione
2. Descrizione
3. Processi e thread
4. Scheduling
5. Sincronizzazione



Creazione dei processi

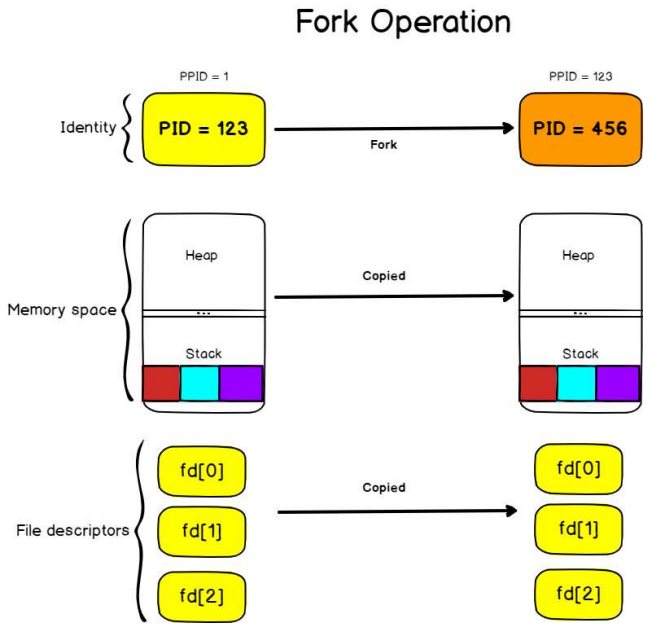
- I sistemi UNIX creano i processi attraverso due fasi distinte
  - *Fork():* crea un nuovo processo
  - *Exec():* esegue il codice del processo





Descrizione dei processi (1/5)

- I sistemi UNIX mantengono tutte le informazioni necessarie al sistema operativo per gestire l'esecuzione di un processo
- Le proprietà dei processi sono di tre tipi
  - *Identità* del processo
  - *Ambiente* del processo
  - *Contesto* del processo

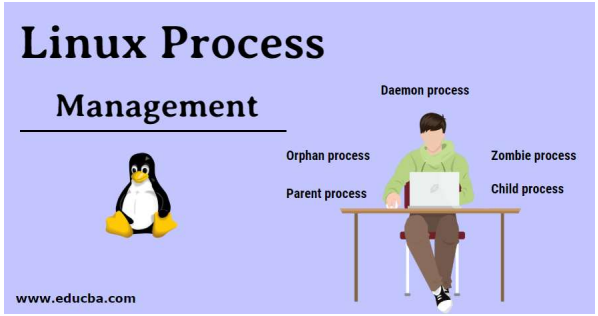


Descrizione dei processi (2/5)

- Identità del processo
  - *Process ID (PID):* identificatore unico del processo, usato dal sistema operativo per schedulazione e comunicazione
  - *Credenziali:* contiene lo user ID associato al processo, e uno o più group ID, che determinano i permessi del processo di accedere alle risorse del sistema
  - *Personality:* indicatore che può modificare la semantica delle chiamate di sistema
  - *Namespace:* vista specifica della gerarchia del file system

Descrizione dei processi (3/5)

- Ambiente del processo
  - Ereditato dal processo padre
  - Due vettori
    - *Vettore degli argomenti:* elenca gli argomenti passati a linea di comando. Tipicamente inizia con il nome del programma
    - *Vettore delle variabili d'ambiente:* lista «NOME=VALORE» che associa a delle variabili d'ambiente valori testuali
  - Il passaggio delle variabili d'ambiente è una modalità flessibile per dare informazioni al processo
    - Possibilità di configurare informazioni per ogni processo

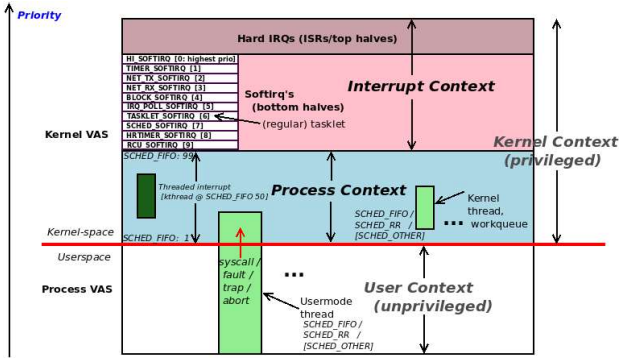


Descrizione dei processi (4/5)

- Contesto del processo
  - Lo stato (in continuo cambiamento) di un processo in esecuzione, nel tempo
  - *Contesto di schedulazione:* informazione necessaria allo scheduler per sospendere e riprendere il processo
  - *Accounting:* informazioni riguardo le risorse occupate da ciascun processo, sia allo stato attuale che nel tempo
  - *Tabella dei file:* vettore di puntatori alle strutture dati del kernel
    - In caso di chiamate di sistema relative a I/O, i processi si riferiscono ai file attraverso l'indicizzazione di questa tabella (*file descriptor – fd*)

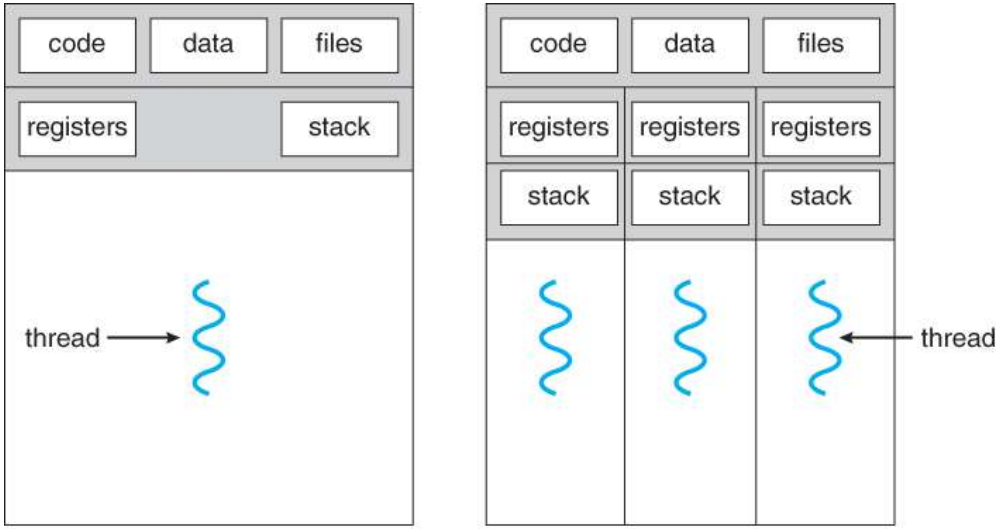
Descrizione dei processi (5/5)

- Contesto del processo
  - *File-system context:* struttura dati a cui il processo accede quando deve aprire nuovi file.
    - Elenca le directory di root e la directory di default usate per cercare i file da aprire
  - *Signal-handler table:* definisce le routine (nello spazio di memoria del processo) da chiamare in caso di specifici segnali
  - *Virtual-memory context:* descrive il contenuto completo dello spazio di memoria del processo



Processi e thread (1/2)

- Linux usa la stessa rappresentazione interna sia per i processi che per i thread
  - Processi e thread sono chiamati **Task**
  - *Thread:* nuovo processo che condivide lo stesso spazio di memoria del processo padre



single-threaded process

multithreaded process

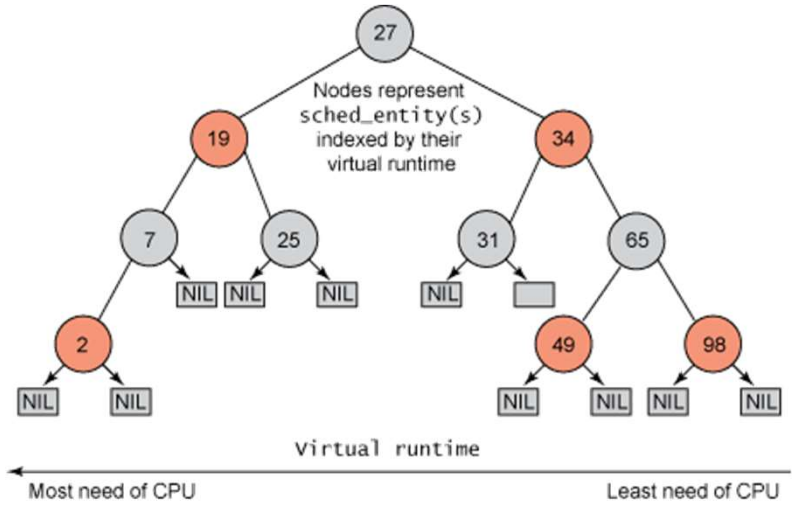
Processi e thread (2/2)

- Distinzione solo quando il thread è creato tramite la chiamata di sistema *clone()*
  - *fork():* nuovo task con il suo proprio contesto
  - *clone():* nuovo task con la propria identità, ma può usare le strutture dati del processo padre
    - Permette un controllo fine di cosa può essere condiviso tra i thread

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

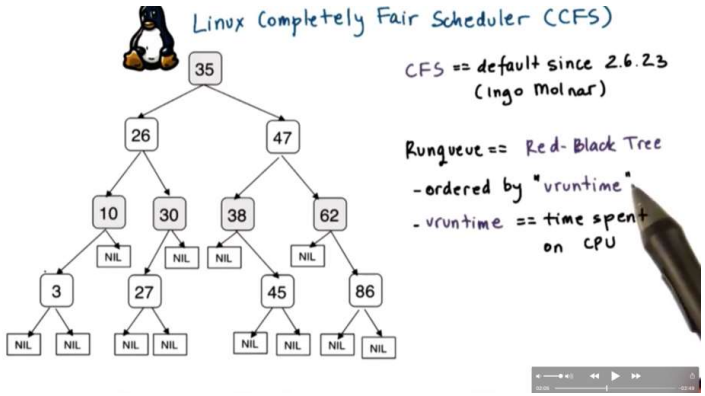
Scheduling (1/3)

- Compito del sistema operativo di allocare tempo di CPU ai vari processi
  - Interruzione e ripresa dei processi
  - Esecuzione di task a livello kernel per la gestione dello scheduling
- Linux 2.6: **Completely Fair Scheduler (CFS)**



Scheduling (2/3)

- Completely Fair Scheduler (CFS)
  - Elimina l'idea tradizionale del quanto di tempo
  - Alloca a tutti i task una porzione del tempo del processore
  - CFS calcola il tempo di esecuzione di un processo in base al numero totale dei task
  - N task totali: ognuno vede $1/N$ del tempo di CPU
  - Pesa ogni task con il numero *nice*
    - *Nice* basso: peso alto (priorità maggiore)

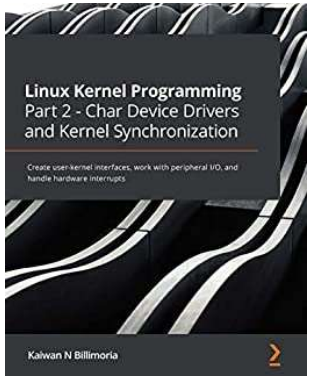


Scheduling (3/3)

- Completely Fair Scheduler (CFS)
  - Ogni task è eseguito dalla CPU per un tempo proporzionale al peso, diviso per il peso totale di tutti i task
  - *Target latency:* intervallo in cui si desidera che ogni task venga eseguito almeno una volta
  - 2 task, con peso uguale, target latency di 10 ms
    - Ognuno viene eseguito per 5 ms
  - 10 task: ognuno viene eseguito per 1 ms
    - Target latency sarebbe violata
    - La target latency assicura la *granularità minima*: ogni task è eseguito almeno per un tempo minimo

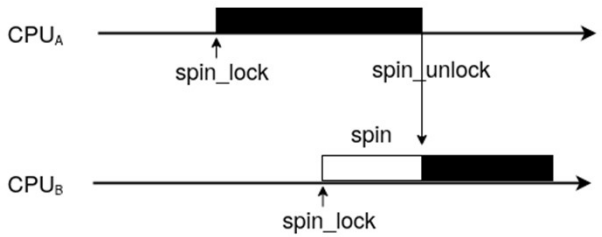
Sincronizzazione dei processi (1/5)

- Sincronizzazione Kernel
  - Richiesta per l'esecuzione in modalità kernel può avvenire in due modi
    - Un programma in esecuzione richiede un servizio del sistema operativo
      - ✓ *Esplicitamente:* chiamata di sistema
      - ✓ *Implicitamente:* es. page fault
    - Un driver di dispositivo causa un interrupt hardware
      - ✓ La CPU inizia ad eseguire un gestore a livello kernel
  - La sincronizzazione Kernel richiede che le sezioni critiche del Kernel siano eseguite senza interruzioni



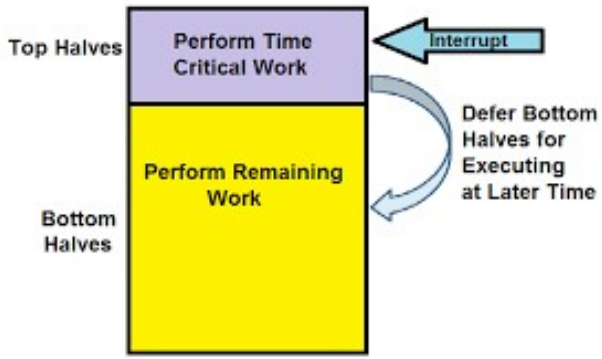
Sincronizzazione dei processi (2/5)

- Linux usa due modalità per «proteggere» le sezioni critiche
  - Il codice kernel è non pre-emptible: quando il sistema operativo sta eseguendo una routine kernel, setta *need_resched* in modo che lo scheduler sia fatto girare dopo che la sezione critica è finita
  - Il sistema operativo disabilita gli interrupt durante l'esecuzione di una sezione critica
    - Elimina il rischio di accesso concorrente a strutture dati condivise



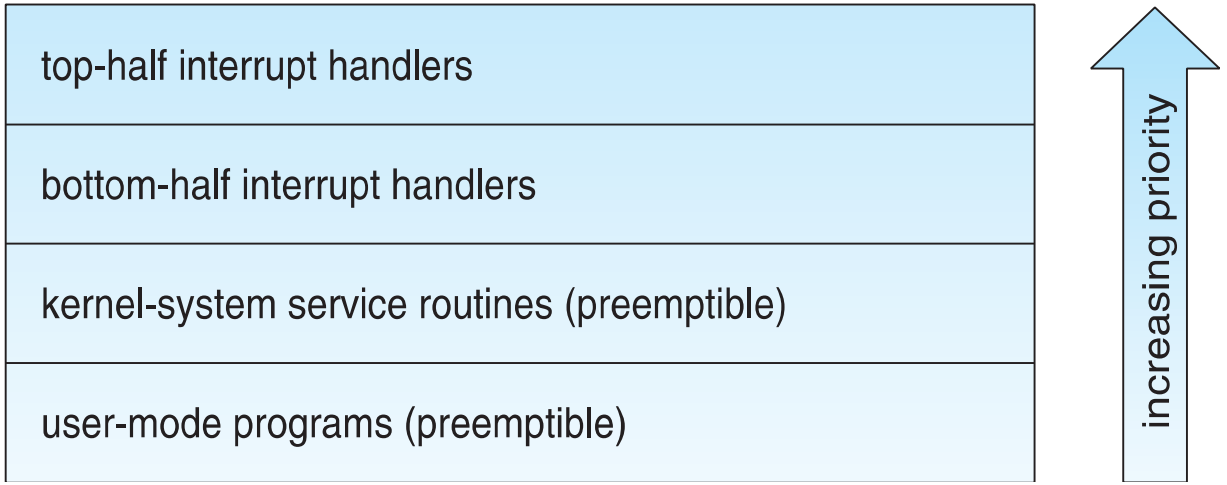
Sincronizzazione dei processi (3/5)

- Sezioni critiche lunghe
  - Per evitare decadimenti di prestazioni, Linux permette di eseguire sezioni critiche lunghe senza necessariamente disabilitare gli interrupt
  - Separazione in top half e bottom half
    - *Top half:* eseguita disabilitando gli interrupt
    - *Bottom half:* eseguita con interrupt abilitati, da uno scheduler che assicura che la bottom half non interrompa se stessa
    - Meccanismo che permette di disabilitare bottom half durante l'esecuzione di codice kernel



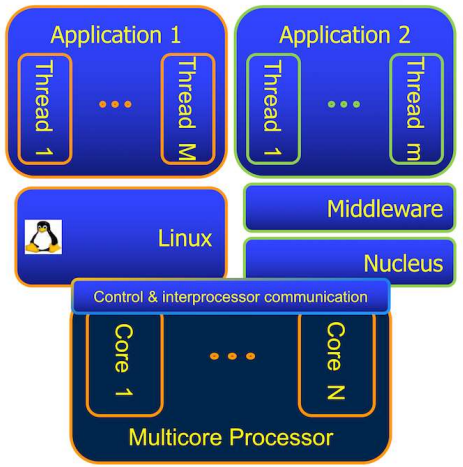
Sincronizzazione dei processi (4/5)

- Gerarchia degli interrupt
  - Ogni livello può essere interrotto da codice che gira ad un livello superiore
  - NON può essere interrotto da codice che gira allo stesso livello o inferiore
  - Processi utente possono essere sempre pre-empted nel caso di un interrupt di scheduling per time sharing



Sincronizzazione dei processi (5/5)

- Multiprocessing simmetrico
  - Processi e thread possono essere eseguiti in parallelo su processori separati
  - Fino alla versione 2.2: restrizione che solo un processore alla volta può eseguire codice in modalità Kernel
  - Versioni successive: aumento della scalabilità usando strutture dati per la sincronizzazione in grado di gestire più processori



In sintesi

1. Storia di Linux
2. Principi di progettazione
3. Componenti del sistema operativo
4. Gestione dei processi