

## 6 – Caso di Studio Linux

**Sommario**

- [Introduzione](#)
- [Storia](#)
- [Panoramica Linux](#)
  - [Obiettivi](#)
  - [Interfaccia e distribuzione](#)
  - [Struttura](#)
- [Architettura del Nucleo](#)
  - [Piattaforma Hardware e moduli del nucleo caricabili](#)
- [Processi in Linux](#)
  - [Organizzazione e dei Processi e Thread](#)
  - [Chiamate e implementazione](#)
  - [Scheduling dei Processi](#)
- [Gestione della memoria in Linux](#)
  - [implementazione: allocazione e deallocazione della memoria fisica](#)
  - [Sostituzione delle pagine e Swapping](#)
- [Dispositivi di I/O](#)
- [File System](#)
  - [File System Virtuale](#)
  - [Cache del File System Virtuale](#)
  - [NFS](#)

S. Balsamo – Università Ca' Foscari Venezia – SO.6.0

## Obiettivi

- [Architettura](#) del nucleo di Linux
- [Implementazione](#) delle componenti del S.O. Linux: processi, memoria e gestione dei file
- [Livelli software](#) che compongono il nucleo di Linux
- Come Linux organizza e gestisce i [dispositivi](#) del sistema
- Come Linux gestisce [operazioni di I/O](#)
- Cenni ai meccanismi di [comunicazione e sincronizzazione](#) tra processi in Linux
- Cenni alle funzioni di [sicurezza](#) di Linux

S. Balsamo – Università Ca' Foscari Venezia – SO.6.1

## Introduzione

- Linux kernel
  - Nucleo del S.O. [open-source](#) più diffuso, distribuito gratuitamente e completo
  - codice sorgente di Linux è disponibile a tutti per studio, installazione e possibile modifica
- Diffuso anche per server di fascia alta, sistemi *desktop* e sistemi dedicati (*embedded*)
- Supporta anche molte [caratteristiche avanzate](#)
  - SMP (Symmetric Multiprocessing)
  - accesso alla memoria non uniforme (NUMA)
  - accesso ai file di diversi sistemi hardware

S. Balsamo – Università Ca' Foscari Venezia – SO.6.2

## Storia di Unix e Linux

- [CTSS](#) poi [Multics](#) ([MULTIplexed information and Computing Service](#)) sviluppato all'M.I.T. poi da Bell Labs e GE, timesharing e multiprogrammazione
- Unics ([UNIplexed information and Computing Service](#)) versione di Ken Thompson su minicomputer PDP poi [UNIX](#)
- Compilatore C portabile
- [Porting](#)
- [BSD](#) Berkeley software distribution networking, editor (*vi*), shell (*csh*), compilatori
- Due versioni maggiormente usate 4.3BSD e System V
- [POSIX portable operating system](#) standard per procedure di libreria

S. Balsamo – Università Ca' Foscari Venezia – SO.6.3

Storia di Unix e Linux

- Creato nel 1991 come **evoluzione di Unix**, da uno studente dell'Università di Helsinki, Finlandia, Linus Torvalds dal quale deriva il nome Linux combinazione di Linus e **UNIX**
  - Utilizza come punto di partenza il codice sorgente libero del S.O. **Minix** (sviluppato da A. Tanenbaum, docente dell'Università di Amsterdam, 1987) e tende a migliorarlo
  - Si basa sul contributo di opinioni e supporto da parte della comunità
  - Gli sviluppatori hanno poi continuato a sostenere il concetto di un nuovo S.O. **disponibile liberamente**



S. Balsamo – Università Ca' Foscari Venezia – SO 6.4

Storia

- Successivamente Linux è stato esteso con possibilità di interoperatività
  - Obiettivo: conformità allo **standard** delle **interfacce** delle applicazioni e servizi del S.O., POSIX (*Portable Operating System Interface*) per Unix
  - Dal 1994 include caratteristiche più avanzate
    - Multiprogrammazione
    - Memoria virtuale
    - Supporto TCP/IP
    - Caricamento a richiesta
  - Obiettivi comuni a Unix
    - Multiprogrammazione e multiutente
    - Facilitare la condivisione e cooperazione
  - Per altre funzioni (es. gestione account, GUI, editor):
    - **distribuzioni** Linux

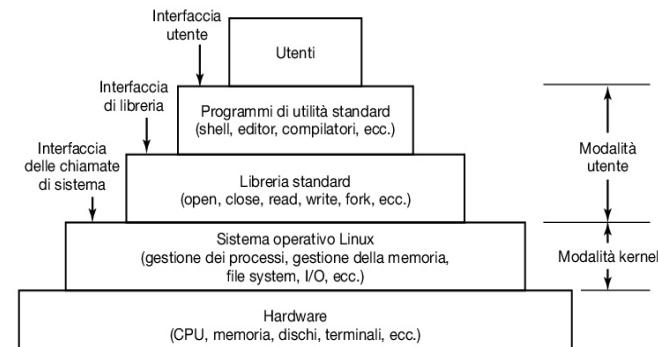
S. Balsamo – Università Ca' Foscari Venezia – SO 6.5

Storia

- Distribuzioni Linux
    - Consente agli utenti non hanno familiarità con i dettagli Linux di installarlo e usarlo
    - Include il software come
      - Il nucleo di Linux
      - applicazioni di sistema
        - Es.: gestione degli account utente, la gestione della rete e strumenti di sicurezza
      - applicazioni utente
        - Es: GUI, browser web, editor di testo, applicazioni e-mail, database e giochi
      - Strumenti per semplificare il processo di installazione
    - Molte distribuzioni modificano il nucleo per aggiungere altri driver o caratteristiche specifiche

S. Balsamo – Università Ca' Foscari Venezia – SO.6.6

Storia Unix – Linux - Livelli



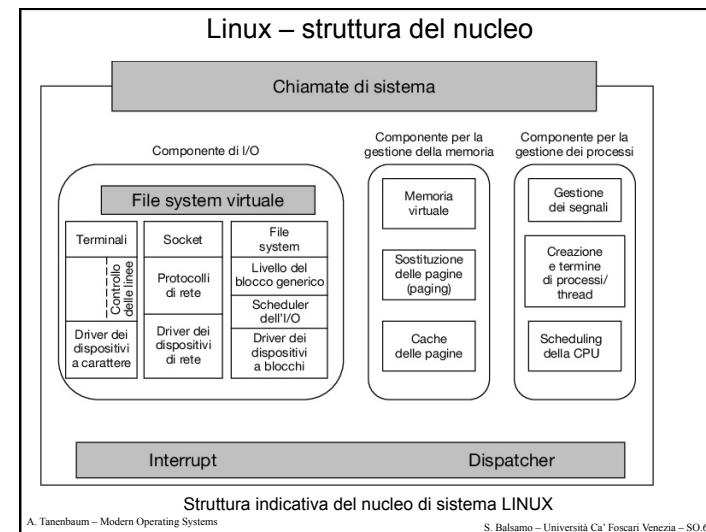
A. Tanenbaum – Modern Operating Systems

S. Balsamo – Università Ca' Foscari Venezia – SO.6.7

Storia Unix – programmi utilities	
Programma	Uso comune
cat	Concatena molteplici file sullo standard output
chmod	Modifica la modalità di protezione del file
cp	Copia uno o più file
cut	Taglia colonne di testo da un file
grep	Esamina un file per trovare un pattern
head	Estrae le prime righe di un file
ls	Elenco il contenuto della directory
make	Compila i file per costruire un binario
mkdir	Crea una directory
od	Trasforma un file in ottale
paste	Incolla colonne di testo in un file
pr	Formatta un file per la stampa
ps	Elenco i processi in esecuzione
rm	Rimuove uno o più file
rmdir	Rimuove una directory
sort	Ordina un file di righe in ordine alfabetico
tail	Estrae le ultime righe di un file
tr	Trasforma tra insiemi di caratteri

Alcuni dei programmi utility UNIX richiesti da POSIX

A. Tanenbaum – Modern Operating Systems      S. Balsamo – Università Ca' Foscari Venezia – SO.6.8



### Storia

- Numeri di versione
  - Incrementato a discrezione da Torvalds per ogni versione del nucleo che contiene un set di funzionalità significativamente diverso da quello della precedente
  - numero di versione minore (cifra dopo il primo punto)
    - Fino alla versione 2.6 il numero **pari** indica una versione **stabile**
    - Numero **dispari** indica una versione **minore**, es. 2.6.1, indica una versione in fase di sviluppo
  - Le cifre che seguono il secondo punto decimale sono incrementate per ogni aggiornamento minore del nucleo
- Esempi
  - Linux 2.0 (1996), 2.2 (1999 con SMP), 2.4 (2001, supporto di diversi hw, migliori prestazioni e scalabilità), 2.6 (dal 2003), 3 (2011), 4 (2015)...

S. Balsamo – Università Ca' Foscari Venezia – SO.6.10

### Overview di Linux

- I sistemi Linux includono **interfacce utente** e altre applicazioni oltre al nucleo
- Eredita da UNIX il modello a **livelli**
- Accesso tramite **interfaccia utente**
- Per le **chiamate** di sistema
- Il S.O. contiene **thread** del nucleo per eseguire i **servizi**
  - Implementati come **demoni**, che possono essere dormienti fino a quando non sono risvegliati da un componente del nucleo
- Sistema multitentativo**
  - Diritti di accesso
  - Sincronizzazione
  - Limita l'accesso alle operazioni importanti per gli utenti con privilegi da **superutente** (detto **root** o **superuser**)

S. Balsamo – Università Ca' Foscari Venezia – SO.6.11

## Sviluppo e comunità

- Torvalds controlla tutte le modifiche del nucleo
- Si appoggia su un gruppo una ventina di **sviluppatori** fidati per la gestione dei miglioramenti del nucleo
- Quando un nucleo di sviluppo si avvicina al completamento
  - Congelamento delle caratteristiche (**feature-freeze**): non si aggiungono nuove funzionalità al nucleo, correzioni per migliorare le prestazioni
  - Congelamento del codice (**code-freeze**): accettate solo correzioni di bug importanti al codice
- Molte aziende sostengono e migliorano lo sviluppo di Linux
- Linux è distribuito sotto la [GNU Public License \(GPL\)](#)
  - General Public Licence
  - GNU progetto della Free Software Foundation (1984) mira a fornire software libero e S.O. simili a Unix
- Linux è gratuito, ma software protetto da copyright

S. Balsamo – Università Ca' Foscari Venezia – SO 6.12

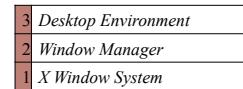
## Distribuzioni

- Oltre 300 distribuzioni disponibili
- Solitamente organizzati in **packages**, ciascuno con [un solo servizio o applicazione](#)
- Alcune fra le più diffuse distribuzioni (commerciali o senza scopo di lucro)
  - Debian
  - Ubuntu
  - Arch Linux
  - CentOS
  - Fedora
  - Linux Mint
  - Mandrake (c)
  - Red Hat (c)
  - SuSE (c)
  - Slackware
  - ...

S. Balsamo – Università Ca' Foscari Venezia – SO 6.13

## Interfaccia

- Si può accedere tramite terminale emulato, tramite la riga di comando di una shell come *bash*, *csh* e *esh*
- La maggior parte delle **interfacce grafiche** (GUI) di Linux sono [a livelli](#)
  - **X Window System** (MIT 1984, Interfaccia grafica di basso livello)
    - Livello più basso
    - Fornisce ai livelli GUI più alti meccanismi per creare e manipolare componenti grafiche
  - **Window manager**
    - Costruito sui meccanismi nell'interfaccia X Window per controllare il posizionamento, l'aspetto, le dimensioni e altri attributi della finestra
  - **Ambiente desktop** (ad esempio, KDE, GNOME) – (opzionale)
    - Fornisce agli utenti interfacce per applicazioni e servizi



S. Balsamo – Università Ca' Foscari Venezia – SO 6.14

## Standard

- Linux è conforme agli standard diffusi come [POSIX](#)
- [Single UNIX Specification \(SUS\)](#)
  - Suite di **standard** che definiscono le **interfacce utente** per la programmazione delle applicazioni e degli utenti per S.O. UNIX, le *shell* e le *utilities*
  - Versione 3 del SUS combina diversi standard (inclusi POSIX, standard ISO e le versioni precedenti di SUS)
- [Linux Standard Base \(LSB\)](#)
  - Progetto che mira a **standardizzare Linux** in modo che le applicazioni scritte per una distribuzione conforme a LSB compilino e si comportino come su qualsiasi altra distribuzione conforme

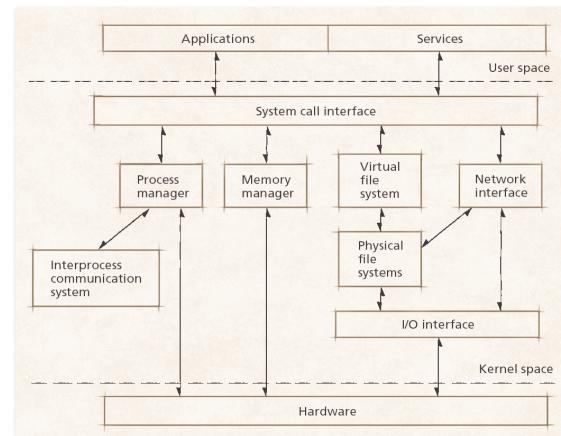
S. Balsamo – Università Ca' Foscari Venezia – SO 6.15

## Architettura del Nucleo

- Nucleo **monolitico** ma
  - Contiene **componenti** modulari
- S.O. UNIX-like o UNIX-based  
(es. servizi simili a Unix System V, BSD Unix)
- Sei principali sottosistemi
  1. Gestione dei **processi**
  2. Interprocess communication
  3. Gestione della **memoria**
  4. Gestione del **File system**  
VFS (virtual File System): fornisce una unica interfaccia a più file systems
  5. Gestione dei dispositivi di **I/O**
  6. Gestione della **rete**

S. Balsamo – Università Ca' Foscari Venezia – SO 6.16

## Architettura del Nucleo - Linux



S. Balsamo – Università Ca' Foscari Venezia – SO 6.17

## Piattaforme Hardware

- Supporta molte piattaforme fra le quali
  - Ese: x86 (incluso Intel IA-32), HP/Compaq Alpha AXP, Sun SPARC, Sun UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390 e zSeries, MIPS, HP PA-RISC, Intel IA-64, AMD x86-64, H8/300, V850 e CRIS
- Codice specifico per l'architettura
  - Esegue operazioni realizzate in modo diverso dalle piattaforme con le istruzioni di basso livello
- Porting
  - Processo di **modifica del nucleo** per supportare una **nuova** piattaforma
  - Il codice specifico per il **porting** è separato dal nucleo e si trova in /arch
- Albero del codice sorgente (**Source tree**)
  - Organizza il nucleo in componenti separati in directory
  - Nelle directory in /arch vi sono i **codici** per ogni **architettura**
- User-Mode Linux (UML)
  - strumento importante per lo sviluppo del nucleo (verifica, messa a punto e correzione). Eseguito in modalità utente su dispositivi virtuali

S. Balsamo – Università Ca' Foscari Venezia – SO 6.18

## Moduli caricabili del nucleo

- Alternativa alla modifica del nucleo monolitico per estenderlo:  
uso di moduli caricabili per **integrale** le funzionalità del nucleo
- Modulo kernel: contiene il **codice oggetto** che, una volta **caricato**, è **collegato** dinamicamente al nucleo in esecuzione
- Eseguiti in **modalità nucleo**
  - sicurezza
- Consente il **caricamento a richiesta** del codice
  - Riduce l'occupazione di memoria del nucleo
  - Es.: file system, alcune periferiche hw
- I moduli scritti per versioni del nucleo diversi da quello corrente potrebbero non funzionare correttamente
- **Kmod**: sottosistema del nucleo che gestisce i moduli senza l'intervento dell'utente (abilitabile)
  - Determina le dipendenze dei moduli e li carica su richiesta

S. Balsamo – Università Ca' Foscari Venezia – SO 6.19

## Gestione dei Processi

- Gestore di processi: **scheduler dei processi**
  - Responsabile innanzitutto di **assegnare i processori** ai processi
  - Spedisce anche i **segnali**
  - **Carica** i moduli del nucleo
  - Riceve gli **interrupt**

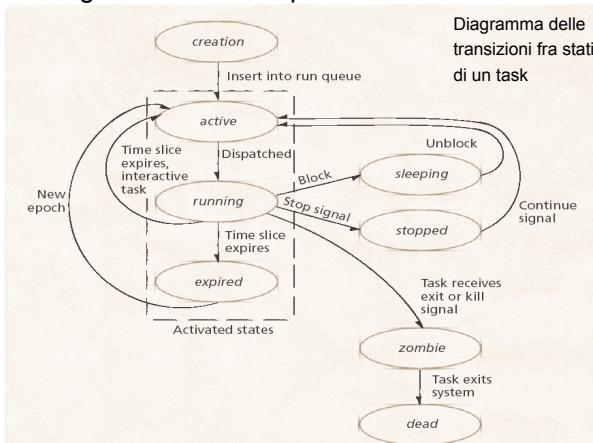
S. Balsamo – Università Ca' Foscari Venezia – SO 6.20

## Organizzazione dei processi e dei Thread

- Processi e *thread* sono chiamati **Tasks**
  - Processi e thread sono rappresentati internamente da una struttura (descrittore di processo) **task\_struct**
- Il gestore di processi mantiene i riferimenti a tutti i task tramite
  - una **lista circolare** doppia di task
  - una **tavella hash**
- **Creazione** del task: assegnamento di un **PID** (*Process IDentifier*) usato per determinare con una funzione hash la **posizione** nella tabella dei processi
- Stati dei Task (variabile **state**)
  - *Running*
  - *Sleeping* (bloccato)
  - *Zombie* (terminato, ma non rimosso)
  - *Dead* (terminato e rimuovibile)
  - *Stopped* (sospeso)
  - *Attivo e terminato (expired)* (non memorizzato dalla variabile state)

S. Balsamo – Università Ca' Foscari Venezia – SO 6.21

## Organizzazione dei processi e dei Thread



© Deitel &amp; Ass. Inc.

S. Balsamo – Università Ca' Foscari Venezia – SO 6.22

## Organizzazione dei processi e dei Thread

- descrittore di processo - **task\_struct**
- Informazioni su parametri
  - Scheduling immagine, puntatori ai segmenti testo, dati e stack
  - Memoria (maschere)
  - Segnali da usare per le *trap*
  - Registri Stato della chiamata di sistema, parametri, risultati
  - Tabella dei descrittori di file
  - Accounting
  - Stack del nucleo stack fisso che il kernel può usare
  - Altro stato del processo, segnale atteso, tempo del prossimo allarme, PID,...

S. Balsamo – Università Ca' Foscari Venezia – SO 6.23

## Organizzazione dei processi e dei Thread

- Init**

- Processo iniziale che usa il nucleo per creare tutti gli altri task
  - La chiamata di sistema **clone** crea nuovi task
  - La chiamata di sistema **fork** crea task che inizialmente condividono lo spazio di indirizzi con i genitori utilizzando **copy-on-write** (analogamente ai processi), la scrittura provoca la copia
  - Quando un processo fa una chiamata di sistema **clone** può specificare **quali strutture dati condividere** con il padre
    - Se lo spazio indirizzo è **condiviso**, clone crea un thread tradizionale
    - Se clone viene chiamato da un processo del **nucleo**, crea un **thread del nucleo** che condivide lo **spazio di indirizzamento del nucleo**
  - Anche se meno portabile di Pthread, i thread Linux possono facilitare la programmazione e migliorare l'efficienza delle applicazioni sfruttando la condivisione di risorse fra task
    - Library of Native POSIX Thread (NPTL) progetto di libreria di thread conformi a POSIX

S. Balsamo – Università Ca' Foscari Venezia – SO 6.24

## Organizzazione dei processi e dei Thread

Flag	Significato quando impostato	Significato quando non impostato
CLONE_VM	Crea un nuovo thread	Crea un nuovo processo
CLONE_FS	Condivide umask, directory principale e di lavoro	Non condividerli
CLONE_FILES	Condivide i descrittori dei file	Copia i descrittori dei file
CLONE_SIGHAND	Condivide la tabella dei gestori dei segnali	Copia la tabella
CLONE_PID	Il nuovo thread riceve il vecchio PID	Il nuovo thread riceve il suo PID
CLONE_PARENT	Il nuovo thread ha lo stesso padre del chiamante	Il padre del nuovo thread è il chiamante

Bit nella `sharing_flags` bitmap - flags per la chiamata **clone** in Linux  
Condivisione selettiva

A. Tanenbaum – Modern Operating Systems

S. Balsamo – Università Ca' Foscari Venezia – SO 6.26

## Creazione di processi in Linux - fork

```
pid = fork( );
if (pid < 0) {
    handle_error( );
} else if (pid > 0) {
    /* il codice del padre va qui */
} else {
    /* il codice del figlio va qui */
}
```

**fork** restituisce 0 al figlio e PID del figlio al padre

S. Balsamo – Università Ca' Foscari Venezia – SO 6.25

## Scheduling dei Processi

- Obiettivi** dello Scheduler
  - Eseguire tutte le attività entro un ragionevole lasso di **tempo**
  - Rispettare le **priorità** dei task
  - Mantenere di un elevato **utilizzo** delle risorse
  - Alto **throughput**
  - **Ridurre l'overhead** di operazioni di scheduling
  - Scalabilità
- Tre classi di thread: Real-time FIFO, Real-time Round Robin, time-sharing
  - Real-time FIFO massima priorità senza prelazione da parte di altri tipi
  - Real-time RR con quanto di tempo
  - time-sharing priorità in [100,139]

Nota: i task 'real-time' non hanno scadenze associate, hanno priorità in [0,99]
- Obiettivo: tutte le operazioni di scheduling devono essere eseguite in **tempo costante** scheduler O(1)
  - Migliora la **scalabilità** se il tempo di esecuzione è indipendente dal numero di task nel sistema

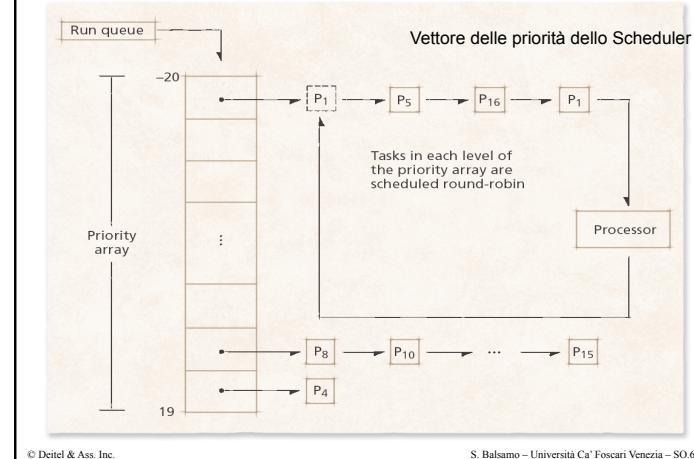
S. Balsamo – Università Ca' Foscari Venezia – SO 6.27

## Scheduling dei Processi

- Scheduler a **prelazione**
  - Ogni task è eseguito fino
    - allo **scadere del suo quanto** o intervallo di tempo
    - oppure diventa eseguibile un processo di **priorità maggiore**
    - oppure il processo si **blocca**
  - I task sono nella coda RUN (simili alle **code multilivello con feedback**)
  - Il **vettore di priorità** mantiene un puntatore a ogni livello della **coda run**
    - Un task con priorità  $i$  è posto nella coda della  $i$ -sima posizione del vettore di priorità della coda run
  - Lo Scheduler avvia il task in testa alla lista nel livello più alto del vettore di priorità
    - La coda per un livello priorità con più task è gestita dallo scheduler con **round-robin**
    - Quando un task entra in stato **bloccato** o **sleeping** (i.e., *waiting*) , o comunque non può essere eseguito, viene rimosso dalla sua coda run

S. Balsamo – Università Ca' Foscari Venezia – SO 6.28

## Scheduling dei Processi



S. Balsamo – Università Ca' Foscari Venezia – SO 6.29

## Scheduling dei Processi

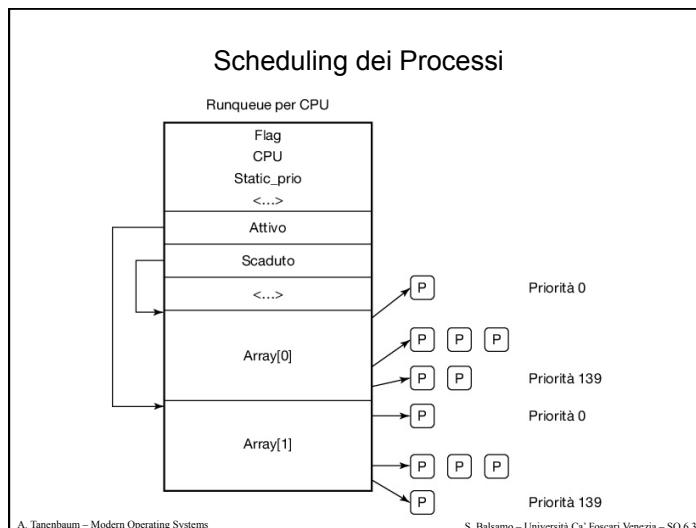
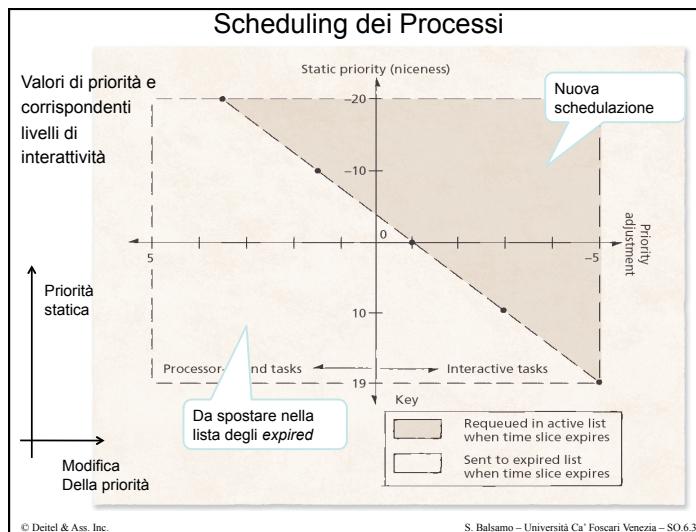
- Per evitare l'**attesa infinita**, ogni task nella coda run è eseguito almeno una volta all'interno di un periodo detto **epoch** (epoch)
  - EPOCH è definita da un limite di massima attesa infinita, derivato empiricamente
    - Es.: 10  $n$  sec. quando ci sono  $n$  task nella coda run
- Lo Scheduler organizza i task in **due liste** con stati **active** e **expired**
  - Si possono **eseguire solo** i task nella lista **active**
  - Quando viene raggiunto il limite di attesa infinita, ogni task- quando scade il suo quanto- è spostato dallo stato **active** e inserito nella lista **expired**
  - Sospende temporaneamente i task ad alta priorità (eccetto quelli in tempo reale)
    - Garantisce che i task a bassa priorità alla fine siano eseguiti
  - Quando tutti sono nello stato **expired** (fine di un'epoch) sposta tutti i task da inattivi ad attivi e inizia una nuova epoch

S. Balsamo – Università Ca' Foscari Venezia – SO 6.30

## Scheduling dei Processi

- Priorità
  - Ad **ogni task creato** è assegnata un **valore** interpretabile come priorità statica (detto anche valore di affinità - **nice**) 0 di default
    - Modificabile con la chiamata `nice(value)`
    - 40 livelli in [-20, 19]
    - Valori bassi: priorità alta
- Obiettivo: **alta interattività**
- I task sono schedulati in base alla loro **effettiva priorità**
  - I task **I/O-bound** ricevono la priorità **alta**
  - I task processor-bound sono penalizzati con priorità più bassa
- Una task ad alta priorità può essere riprogrammato, ovvero variata dinamicamente la priorità, alla scadenza dell'intervallo di tempo
  - (entro una data soglia, e.g. differenza di 5 dalla prima priorità assegnata)

S. Balsamo – Università Ca' Foscari Venezia – SO 6.31



## Scheduling dei Processi

- Scheduler O(1)
    - Esecuzione a tempo costante
    - Runqueue divisa in due code (array): attivo (active) e scaduto (expired)
    - Ogni campo punta a un vettore di 140 liste di priorità (da 0 a 139)
    - La testa della lista punta ad una lista doppia di processi di quella priorità
    - Lo scheduler seleziona un task fra quelli a priorità massima
    - Se scade il quanto il task viene spostato nella lista dei processi scaduti (eventualmente ad una diversa priorità)
    - Quando un processo è bloccato al ritorno viene rimesso nell'array di task attivi originale, tenendo conto del tempo
    - Quando terminano i task attivi i puntatori delle due liste sono scambiati così gli scaduti diventano attivi e viceversa
    - Tempi di quanto diverso a diversi livelli di priorità

S. Balcamo - Università Ca' Foscari Venezia - SO 6 33

## Scheduling dei Processi

- Scheduling per sistemi multiprocessore
    - Una coda di task indipendente per ogni processore
    - Località della *cache*
    - Scheduler esegue il **bilanciamento dinamico del carico**
    - Cerca di ridurre lo sbilanciamento del carico, non bilanciare perfettamente le code run
    - Cerca di migrare solo i task *cache-cold*
      - la *cache* non contiene molti dei suoi dati
  - Scheduling real-time
    - Scheduler soft real-time
    - I task RT possono usare una politica di scheduling round-robin, FIFO o di default
    - I task RT sono sempre rischedulati alla fine del quanto di tempo
    - I task RT possono essere creati solo da utenti con privilegi di root

S. Balsamo – Università Ca' Foscari Venezia – SO.6.35