

# Sistemi Operativi Modulo I

Primo canale (A-L) e Teledidattica

A.A. 2021/2022

Corso di Laurea in Informatica

## Nozioni di Base

Igor Melatti

Sapienza Università di Roma  
Dipartimento di Informatica



# Informazioni Generali sul Corso

- Per il corso in presenza ci sono 2 canali
- Chi è del primo canale fa l'esame con me, chi è del secondo canale fa l'esame con il prof. Tolomei
- Le **uniche** eccezioni ammesse sono quelle relative ai cambi di canale *ufficiali*
  - ovvero, seguendo le istruzioni di <http://www.studiareinformatica.uniroma1.it/laurea/assegnazione-canale/cambio-canale>
- La precedente regola *non* vale per gli studenti in teledidattica, i quali faranno *tutti* l'esame con me, qualsiasi sia il loro cognome

# Come Seguire le Lezioni

- In presenza: solo se siete stati esplicitamente autorizzati dall'applicativo PRODIGIT  
<https://prodigit.uniroma1.it/>
  - ci sono controlli all'entrata della Sapienza, più controlli a random in aula da parte del docente
- A distanza: tramite l'applicativo zoom
  - si consiglia l'uso dell'applicazione (scaricabile da <https://zoom.us/download>)
  - è comunque possibile usare la versione web (<https://zoom.us/>), che però ha limitazioni

# Come Seguire le Lezioni a Distanza

- Gli ID delle conferenze zoom sono i seguenti:
  - **964 2360 7293** per tutti i lunedì
  - **944 7438 1096** per tutti i mercoledì
- Le password verranno comunicate tramite email (istituzionale) a tutti gli iscritti al link `http://151.100.17.205/iscrizione/corso.php?id_corso=12`
  - l'iscrizione va fatta una volta sola ed è valida fino all'ultima lezione del corso

# Come Seguire le Lezioni a Distanza

- C'è una waiting room: dovrete aspettare che il docente vi ammetta
  - l'attesa potrà protrarsi al massimo fino a 10 minuti dopo l'orario stabilito
  - waiting room comunque controllata ogni 20-30 minuti per ammettere eventuali ritardatari
  - selezionare "join with computer audio"
- **Importante 1:** settare il proprio numero di matricola (e nient'altro) come nickname zoom
  - vi viene chiesto sia dall'applicazione che dal sito web
  - non verrete ammessi se non vi presentate in questo modo
- **Importante 2:** spegnere immediatamente sia il vostro video che il vostro microfono
  - per ragioni di privacy, siete sconsigliati dal settare una vostra foto di profilo

# Come Seguire le Lezioni

- Per fare domande durante la lezione:
  - se siete in presenza, chiedete e basta; parlate a voce non troppo alta e mantenendo la mascherina
  - se siete a distanza, dovete usare la chat di zoom
    - o per scrivere direttamente la vostra domanda
    - o per chiedere il permesso di porla a voce: una volta ottenutolo, attivare il microfono, porre la domanda, e disattivarlo nuovamente
  - il docente risponderà sempre a voce, ripetendo la domanda
- Tutte le lezioni sono registrate
  - copia della registrazione verrà mandata a tutti coloro che si sono prenotati su [http://151.100.17.205/iscrizione/corso.php?id\\_corso=12](http://151.100.17.205/iscrizione/corso.php?id_corso=12) (già menzionato sopra)

# E Se le Regole Non Vengono Rispettate?

- Violare le regole in presenza, e rifiutarsi di rispettarle dopo i richiami, porterà all'immediata *cancellazione della lezione*
  - il docente se ne va, e gli argomenti ancora da trattare vengono considerati come se fossero stati spiegati
  - esempi non esaustivi di regole infrante:
    - non avere la mascherina, o non portarla nella maniera corretta
    - sedersi su un posto non occupabile
    - essere presenti senza autorizzazione
    - parlare a voce alta
- Violare le regole a distanza, e rifiutarsi di rispettarle dopo i richiami, porta all'immediata *espulsione dalla conferenza zoom* del colpevole
  - esempi non esaustivi di regole infrante:
    - attivare video o microfono senza permesso del docente
    - disturbare sulla chat

# Informazioni Generali sul Corso

- Sistemi Operativi è un corso a 2 facce
  - lo stesso vale anche per Basi di Dati
- Da un punto di vista delle lezioni frontali e delle modalità d'esame, è diviso in 2 corsi indipendenti: modulo 1 e modulo 2
  - modulo 1 adesso, modulo 2 nel secondo semestre
  - 6 CFU per ogni modulo
  - anche i docenti sono diversi
    - il sottoscritto fa eccezione: vi insegnerò anche il modulo 2
- Da un punto di vista della verbalizzazione, è un esame solo
  - su Infostud, troverete solo Sistemi Operativi, senza nessun riferimento ai moduli
  - l'esame verbalizzato vale 12 CFU



# Informazioni Generali sul Corso

- Un po' di conseguenze (valide per chi è in corso, ovvero attualmente all'inizio del secondo anno):
  - non potrete verbalizzare prima di giugno
  - a gennaio e febbraio, che voi superiate o no l'esame, la verbalizzazione su Infostud sarà "Rinuncia"
- Un po' di conseguenze (valide per tutti):
  - se superate lo scritto in un appello, ma ancora non avete fatto il modulo 2, la verbalizzazione sarà "Rinuncia"
  - se per un esame "normale" (non a moduli) ci sono  $n$  appelli su Infostud, per Sistemi Operativi ne vedrete  $2n$ 
    - per ogni appello, c'è un esame per il primo modulo e uno per il secondo
    - attenzione: gli appelli del modulo 2 di gennaio/febbraio *non* sono per chi è adesso all'inizio del secondo anno
    - come distinguerli? guardando la descrizione dell'appello e, soprattutto, consultando il sito del corso (ora ci arriviamo)

# Informazioni Generali sul Corso

- Validità dei voti dei singoli moduli: c'è una regola semplice
- I moduli possono essere superati in qualsiasi ordine
  - quindi, va bene anche superare prima il modulo 2 e poi il modulo 1
- Una volta superato un modulo, si ha tempo fino all'anno solare successivo per superare anche l'altro
  - ad esempio: se si supera il modulo 1 a gennaio 2021, occorre superare il modulo 2 entro settembre 2022
  - ad esempio: se si supera il modulo 2 a settembre 2021, occorre superare il modulo 1 entro settembre 2022
  - se si ha diritto all'appello straordinario, allora anche novembre 2022
  - ad esempio: se si supera il modulo 1 a settembre 2021, e il modulo 2 a gennaio 2023, occorre nuovamente superare il modulo 1 entro settembre (o novembre) 2024

# Informazioni Generali sul Corso

- Da qui in poi, si tratterà solo del primo modulo, primo canale (e teledidattica)
- Docente: Igor Melatti
- Dove potete trovare materiale: <http://twiki.di.uniroma1.it/twiki/view/S0/S01213AL/SistemiOperativi12CFUModulo1Canale120202021>
  - ci saranno anche informazioni sugli esami ed in generale sull'intero corso
  - invece, da <http://twiki.di.uniroma1.it/twiki/view/S0/S01213AL/SistemiOperativi12CFUModulo1> si possono trovare informazioni sugli anni passati
- Ogni settimana 2 lezioni per 5 ore complessive

# Regole per gli Esami

- Ci sono 5 appelli d'esame nel corso dell'anno, più due "straordinari"
  - uno a gennaio (subito dopo la fine del corso), uno a fine gennaio/inizio febbraio, due a giugno/luglio e uno a settembre 2021
  - i due appelli straordinari (verso novembre il primo e verso aprile il secondo) sono riservati a studenti fuoricorso, ripetenti, part-time e lavoratori
  - l'appello straordinario di novembre 2020 fa ancora concettualmente parte dell'A.A. 2019/2020, e sarà quindi gestito con le vecchie regole di esame
- Gli studenti con DSA o disabilità sono invitati a contattare la prof.ssa Tiziana Calamoneri ([calamo@di.uniroma1.it](mailto:calamo@di.uniroma1.it))

# Regole per gli Esami

- Per partecipare agli appelli straordinari, occorre seguire le istruzioni pubblicate qui: <http://www.studiareinformatica.uniroma1.it/appelli-d-esame>
- Per gli appelli “normali”, è necessaria e sufficiente la prenotazione su Infostud
  - le iscrizioni su Infostud si chiuderanno sempre  $k$  giorni prima del giorno dello scritto
    - sul sito del corso viene riportato il termine ultimo per l'iscrizione
    - finché gli esami saranno a distanza,  $k = 5$ , altrimenti  $k = 2$
  - attenzione a distinguere tra appelli per il modulo 1 e il modulo 2
  - i numeri dei verbali sono riportati sul sito del corso

# Regole per gli Esami

- Ciascun appello d'esame è composto da uno scritto e da un orale
- È possibile partecipare a qualsiasi numero di esami (scritti e/o orali) nel corso dell'anno accademico
- Per superare l'esame scritto occorre aver preso almeno 18
  - fa fede il voto dell'*ultimo* esame scritto sostenuto
  - nota bene: questo significa che non superare un esame scritto invalida eventuali esami scritti superati in precedenza
- Il voto dell'esame scritto sarà al massimo 22
  - sono possibili eccezioni a discrezione del docente
- Chiunque abbia superato lo scritto può accettare direttamente il voto dello scritto stesso, senza un esame orale
  - sono possibili eccezioni a discrezione del docente (in caso di dubbio di copiatura)

# Regole per gli Esami

- Ogni studente può chiedere di alzare il voto dello scritto tramite un orale
- Chi venga bocciato all'orale dovrà nuovamente superare un esame scritto
- L'orale può essere anche sostenuto (a discrezione dello studente) in un appello diverso da quello in cui ha superato lo scritto, ma *entro lo stesso anno accademico*
- Occorre sempre iscriversi su Infostud, sia per gli esami scritti che per quelli orali, con la seguente eccezione:
  - se si fa l'esame orale nello stesso appello in cui si fa lo scritto, basta iscriversi una volta sola (prima dello scritto)

# Regole per gli Esami: lo Scritto

- Le domande verteranno sempre sull'intero programma del corso
- Si tratta di un compito a quiz da fare direttamente al computer
- Ci saranno 25 domande da fare in 30 minuti
- Per ogni domanda, ci sono 4 opzioni, delle quali una sola è giusta
- Tutti gli studenti hanno lo stesso compito, ma con le domande (e le opzioni) mischiate
- È possibile consultare copie *cartacee* di libro, slide, appunti
- Non è consentito, ovviamente, comunicare con altri studenti, né consultare archivi elettronici

# Regole per gli Esami: lo Scritto

- Per ciascuno viene calcolato il punteggio come  $2E - S$ , con  $E$  numero di risposte esatte ed  $S$  numero di risposte sbagliate
  - quindi il punteggio va da -25 a 50
- Per assegnare un voto da insufficiente a 22 a ciascun punteggio, si valuteranno i punteggi di tutti i partecipanti all'esame, seguendo (*cum grano salis*) una distribuzione a campana di Gauss
  - conseguenza: se ci sono copiatore, molti compiti avranno punteggi vicini, e quindi il voto si abbassa. Morale: non conviene copiare
  - non è necessario rispondere a tutte le domande
  - chi fa il compito migliore rispetto agli altri ha il punteggio più alto

## Regole per gli Esami: l'Orale (Facoltativo)

- Per accedere all'orale è necessario aver superato uno scritto
  - anche in appelli precedenti, ma sempre dello stesso anno accademico
- Sarà necessario prenotarsi tramite un apposito form online, immediatamente dopo la fine dell'ultimo turno dello scritto
  - chi non vuole fare l'orale, e non vuole verbalizzare, non deve comunicare nulla
- È possibile prendere 30 e lode partendo da un 18 allo scritto, così come è possibile venire bocciati dopo aver preso 22 allo scritto
- È possibile ritirarsi entro 5 minuti dalla proposizione dei quesiti, mantenendo così il voto dello scritto

# Regole per gli Esami: l'Orale (Facoltativo)

- *Tutte* le possibili domande per l'orale sono pubblicate con all'incirca un mese di anticipo sulla sessione d'esame
  - all'indirizzo [http://151.100.17.236/~melatti/so1\\_domande\\_20202021/orali.html](http://151.100.17.236/~melatti/so1_domande_20202021/orali.html)
- Ci sono 3 fasce di domande: da 4, da 8 e da 12 punti
  - risolvere e discutere correttamente una domanda da  $x$  punti comporta un'aggiunta di  $x$  sul voto dello scritto
  - per esempio, risolvendo e discutendo correttamente la domanda da 8 punti a partire da uno scritto con 19, si ottiene 27 come voto finale (per il solo modulo 1, ovviamente)
  - lo studente deve scegliere *esattamente una domanda* alla quale rispondere; la difficoltà dell'orale è ovviamente crescente rispetto ad  $x$
  - per ogni compito, ci sarà 1 domanda per ciascuna fascia
- Le domande per l'orale potranno subire aggiunte per le sessioni estiva ed autunnale

# Regole per gli Esami: Organizzazione

- Ci saranno 4 opzioni per l'esame
  - ① in presenza: tutti gli studenti fanno l'esame in un laboratorio della Sapienza attrezzato con computer
  - ② a distanza: tutti gli studenti fanno l'esame a distanza, ciascuno a casa sua (possibile solo in caso di lockdown che coinvolga la Sapienza)
  - ③ in presenza con eccezioni: l'esame è in presenza, ma gli studenti che lo desiderano possono chiedere di farlo a distanza
  - ④ a distanza con eccezioni: l'esame è a distanza, ma gli studenti che lo desiderano possono chiedere di farlo in presenza
- Per ogni appello, verrà chiaramente indicata la modalità scelta
- Finché perdurerà l'emergenza CoViD-19, e potendo scegliere, il docente di questo corso effettuerà tutti gli esami in modalità 4
  - in caso di lockdown che coinvolga la Sapienza, si passerà alla modalità 2

# Esami a Distanza: Prerequisiti Hardware

- PC (portatile o fisso) con webcam, microfono e altoparlanti (o cuffie)
- Smartphone (o anche tablet)
- Connessione Internet stabile attiva su entrambi i dispositivi
- Tutti i dispositivi devono essere collegati all'alimentazione elettrica durante l'esame
- Un documento di riconoscimento in corso di validità

# Esami a Distanza: Prerequisiti Software

- Lo smartphone deve essere configurato, per tutta la durata dell'esame, in modo che non vada in sleep
  - si può fare o con app come wakey, o configurando opportunamente le impostazioni
- Un browser sul PC; accertarsi di essere in grado di eseguire GoogleMeet a partire dal proprio account di posta istituzionale (@studenti.uniroma1.it)
- L'applicazione Zoom sul PC (nota bene: per gli scopi dell'esame, non è possibile utilizzarla da browser)
- L'applicazione Zoom sullo smartphone
- Copie *cartacee* degli appunti del corso, o libri di testo
- Un'applicazione di posta elettronica (anche su browser) configurata sull'account istituzionale, sia sul PC che sullo smartphone.

# Esami a Distanza: Fallimento Prerequisiti

- Chi non soddisfa tali requisiti minimi, non può partecipare all'esame a distanza da casa
- Per gli appelli in presenza con eccezioni (modalità 3), deve pertanto optare forzatamente per la presenza
- Per gli appelli a distanza senza eccezioni (modalità 2), non potrà fare l'esame
- Per gli appelli a distanza con eccezioni, se ha la possibilità di recarsi personalmente nelle strutture della Sapienza, può chiedere che gli venga assegnata una postazione dalla quale connettersi e fare l'esame
  - nessun hardware/software richiesto, si usano le risorse Sapienza
- A tal proposito, è necessario fare una richiesta scritta e motivata alla segreteria didattica ([segr.didattica@di.uniroma1.it](mailto:segr.didattica@di.uniroma1.it)) almeno 2 settimane prima dello scritto

# Regole per gli Esami in Presenza

Per ogni appello *in presenza*:

- 1 si fa lo scritto la mattina
  - se ci sono troppi iscritti, vengono divisi su più turni, ciascuno della durata di 45 minuti circa (comprensivi di appello e setup)
  - i turni sono comunicati per email (quella istituzionale @studenti.uniroma1.it)
- 2 circa 15 minuti dopo la fine dell'ultimo turno, i risultati vengono pubblicati sul sito del corso, insieme al form di iscrizione all'orale
  - si potrà anche chiedere la verbalizzazione col voto dello scritto, ma solo se si è già superato il modulo 2 nei giusti tempi
  - per chi ha appena fatto lo scritto, basta inviare un'email dal proprio account istituzionale; altrimenti, è richiesta la presenza
- 3 si fanno gli orali
  - con eventuale verbalizzazione
- 4 si fa la correzione (individuale) dello scritto
  - solo su esplicita richiesta; possibile anche prendere appuntamento per email

# Cosa Verrà Insegnato in Questo Corso

- Come fa un “computer” a funzionare?
  - computer: qualsiasi dispositivo elettronico con capacità di calcolo e programmabile
  - i più comuni: workstation (“computer fisso”), notebook e laptop (“computer portatile”), tablet, smartphone, ...
  - meno comuni ma importanti: cluster (vettoriali e non), minicomputer (Raspberry Pi, controllori), ...
- Com'è possibile che...
  - premendo un tasto sulla tastiera, io “produca” sullo schermo una lettera (se sto usando, ad esempio, un editor)?
  - muovendo il mouse (o equivalente), io veda il cursore che si muove sullo schermo?
  - cliccando con il mouse (o equivalente), io possa aprire un nuovo programma?
  - mentre navigo con il browser, io possa velocemente passare a leggere le email o scrivere il codice di un programma, mentre *in contemporanea* scarico un file?

# Cosa Verrà Insegnato in Questo Corso

- Com'è possibile che io possa scrivere un programma in maniera “semplice” e poi eseguirlo in modo altrettanto “semplice”?
  - devo acquisire un dato da tastiera?
    - `myint = input('Enter your input:')` (Python 3)
    - `int myint = keyboard.nextInt();` (Java)
    - `scanf("%d", &myint);` (C)
  - devo usare la RAM?
    - `myint = 3` (Python 3)
    - `int myint = 3;` (Java e C)
    - `oggetto_complesso = ClasseComplessa()` (Python 3)
    - `ClasseComplessa oggettoComplesso = new ClasseComplessa();` (Java)
    - `StrutturaComplessa *oggettoComplesso = malloc(sizeof(StrutturaComplessa));` (C)

# Cosa Verrà Insegnato in Questo Corso

- Com'è possibile che io possa scrivere un programma ed eseguirne contemporaneamente 2 istanze?
  - programma Python o Java che cerca il carattere più comune in un file dato come argomento
  - scrivo il codice, apro 2 diversi terminali e su ciascuno di questi eseguo il mio programma (con input diversi)
  - il programma che ho scritto è lo stesso
  - potrei quindi pensare che entrambe provino ad accedere alle stesse zone di memoria
  - come mai le 2 esecuzioni, invece, non si danno fastidio, e calcolano correttamente il loro output?

# Cosa Verrà Insegnato in Questo Corso

- La risposta corta è “perché c'è un *sistema operativo*”
- La risposta un po' più lunga e precisa è “perché c'è un *sistema operativo* che interagisce con l'hardware”
  - negli esempi di sopra, giocano un ruolo importante anche le *librerie* ed i *compilatori*, che però alla fine si devono basare sul sistema operativo
- Con il corso di Progettazione dei Sistemi Digitali prima e Architettura degli Elaboratori poi, avete cominciato ad intuire cosa succede a livello di hardware e di software di basso livello
- Ma manca un pezzo fondamentale, che viene colmato da questo corso: il *sistema operativo*

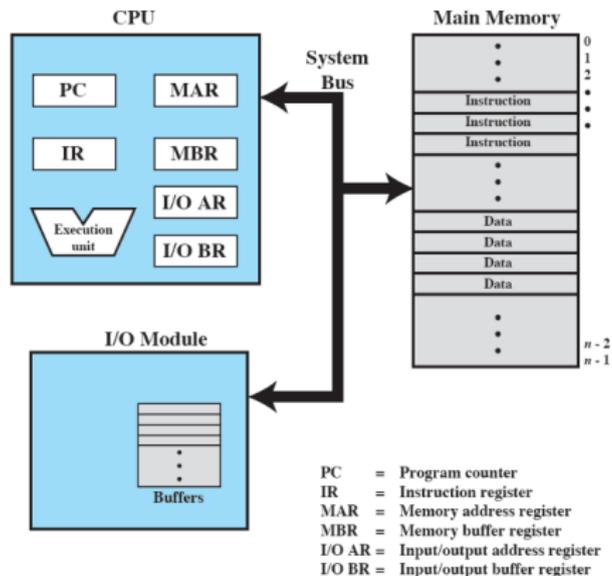
# Sistemi Operativi: Esempi

- Windows
  - ultima versione: Windows 10
- macOS
  - ultima versione: macOS 10.15 Catalina
- Linux Ubuntu
  - ultima versione: Ubuntu 20.04 (Focal Fossa)
- Windows 10 Mobile, iOS, Android
- Ad oggi, ne esistono un migliaio

# Sistema Operativo

- Gestisce le risorse hardware di un sistema computerizzato
  - uno o più processori
  - memoria primaria (RAM)
  - dispositivi di input/output
    - memoria secondaria (dischi)
    - rete
    - ...
- Il suo scopo è fornire un insieme di servizi agli utenti
  - servizi sia per gli sviluppatori che per i semplici utilizzatori
  - esempio: offre un ambiente di esecuzione “facilitato” alle applicazioni utente
  - esempio: offre un ambiente grafico per eseguire applicazioni

# Componenti di un Computer Monoprocessore



# Nozioni di Base: Parti Principali

- Processore
  - il cervello del computer: si occupa di tutte le computazioni
- Memoria Principale
  - volatile: se si spegne il computer, se ne perde il contenuto
  - talvolta chiamata memoria reale o primaria
- Moduli di input/output
  - dispositivi di memoria secondaria (dischi...), non volatile
  - dispositivi per la comunicazione (schede di rete...)
  - altri dispositivi: tastiera, monitor, stampante, mouse, ...
- “Bus” di sistema
  - mezzo per far comunicare tra loro le parti interne del computer: processori, memoria principale, e moduli di input/output

# Registri del Processore

- Registri visibili dall'utente
  - usati o da chi programma in assembler o dai compilatori di linguaggi *non* interpretati
  - obbligatori per alcune istruzioni su alcuni processori
  - facoltativi per ridurre accessi alla memoria principale
  - linguaggi compilati (“vecchi”): C, C++, Fortran; linguaggi interpretati (“nuovi”): Python, Java
- Registri di controllo e di stato
  - usati dal processore per controllare l'uso del processore stesso
  - usati da funzioni privilegiate del SO per controllare l'esecuzione dei programmi
- Registri “interni”
  - usati dal processore tramite microprogrammazione
  - comunicazione con memoria ed I/O

# Registri Visibili dall'Utente

- Gli unici che possono essere usati *direttamente* (con il loro nome) quando si programma in linguaggio macchina (o assembler), ad es.:
  - `mov $5, %eax` (Pentium con sintassi AT&T)
  - `li $t1, 5` (MIPS, ad es. R3000A nella PlayStation)
- Possono contenere dati o indirizzi
- Nel caso contengano indirizzi, possono essere
  - puntatori diretti
  - registri-indice: per ottenere l'indirizzo effettivo, occorre aggiungere il loro contenuto ad un indirizzo base
  - puntatori a segmento: se la memoria è divisa in segmenti, contengono l'indirizzo di inizio di un segmento
    - es.: `cs`, `ds`, `ss`, `es`, `fs`, `gs` nel Pentium
  - puntatori a stack: puntano alla cima di uno stack
    - es.: `esp` per Pentium, `$sp` per MIPS

# Registri Interni

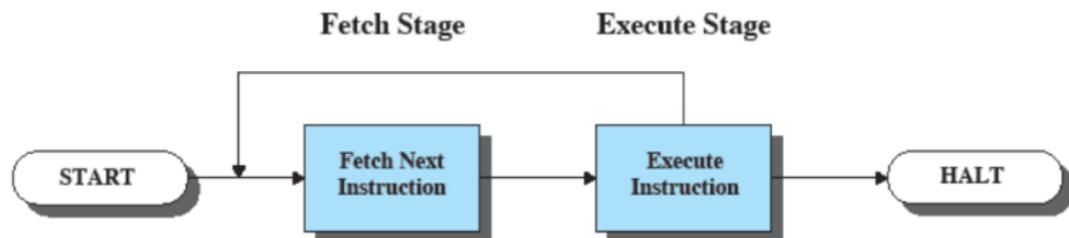
- Registro dell'indirizzo di memoria
  - Memory Address Register, o MAR
  - contiene l'indirizzo della prossima operazione di lettura/scrittura
- Registro di memoria temporanea
  - Memory Buffer Register, o MBR
  - contiene i dati da scrivere in memoria, o fornisce lo spazio dove scrivere i dati letti dalla memoria
  - `lw $s1, 0($s2)`: concettualmente, prima copia il contenuto di `$s2` in MAR, poi legge dalla memoria in MBR, e infine copia da MBR in `$s1` (microprogrammazione semplificata)
- Registro dell'indirizzo di input/output
  - I/O address register
- Registro di memoria temporanea per l'input/output
  - I/O buffer register

# Registri di Controllo e Stato

- Contatore di Programma (Program Counter, o PC)
  - contiene l'indirizzo di un'istruzione da prelevare dalla memoria
- Registro di Istruzione (Instruction Register, o IR)
  - contiene l'istruzione prelevata più di recente
- Stato di Programma (Program Status Word, o PSW)
  - contiene le informazioni di stato, ad es: interrupt disabilitati
- Codici di condizione (o flag)
  - singoli bit settati dal processore come risultato di operazioni
  - esempi: risultato positivo, negativo, zero, overflow, ...
- Vengono usualmente letti/modificati in modo *implicito* da opportune istruzioni assembler
  - esempio: una jump modifica il PC
- Nel Pentium sono considerati registri di controllo anche quelli per la gestione della memoria
  - ad esempio, i registri cr0 ... cr4 gestiscono le tabelle delle pagine
  - un bit di cr0 abilita la paginazione tout-court

# Esecuzione di Istruzioni

- Due passi
  - Il processore legge (preleva, fase di fetch) istruzioni dalla memoria (principale)
  - Il processore esegue ogni istruzione prelevata



# Prelievo ed Esecuzione di Istruzioni

- Il processore preleva l'istruzione dalla memoria principale
- Il PC mantiene l'indirizzo della prossima istruzione da prelevare
- Il PC è incrementato dopo ogni prelievo
- Se l'istruzione contiene una jump, il PC verrà ulteriormente modificato dall'istruzione stessa

# Registro dell'Istruzione

- L'istruzione prelevata viene caricata nell'IR
- Categorie di istruzioni
  - Scambio dati tra processore e memoria
  - Scambio dati tra processore e input/output
  - Manipolazione di dati
    - include operazioni aritmetiche
    - solitamente solo con registri (MIPS), ma in alcuni processori anche direttamente in RAM (Pentium)
  - Controllo
    - modifica del PC tramite salti condizionati o non
  - Operazioni riservate
    - (dis)abilitazione interrupt
    - (dis)abilitazione cache
    - (dis)abilitazione paginazione/segmentazione

# Caratteristiche di una Macchina Ipotetica



(a) Instruction format



(b) Integer format

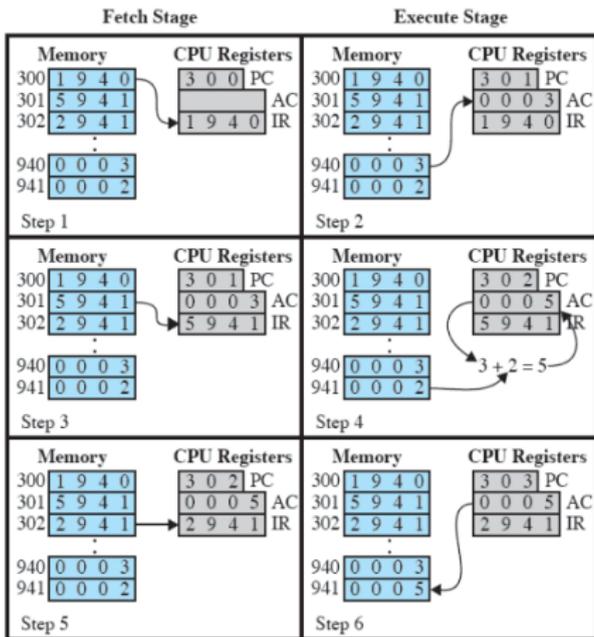
Program counter (PC) = Address of instruction  
Instruction register (IR) = Instruction being executed  
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory  
0010 = Store AC to memory  
0101 = Add to AC from memory

(d) Partial list of opcodes

# Esempio di Esecuzione di un Programma



# Interruzioni

- Paradigma dell'interazione hardware/software
- Interrompono la normale esecuzione sequenziale del processore
  - come conseguenza, viene eseguito del software “di sistema”, che tipicamente non è stato scritto dall'utente che sta eseguendo un certo programma
  - questo software di sistema, come si può intuire, è parte del sistema operativo
- Le cause sono molteplici, e danno luogo a diverse *classi* di interruzioni:
  - da programma (sincrone)
  - da I/O (asincrone)
  - da fallimento hardware (asincrone)
  - da timer (asincrone)

# Interruzioni Asincrone vs. Sincrone

- Le interruzioni da programma sono le uniche *sincrone*
  - ovvero, hanno luogo come *immediata* conseguenza di una certa istruzione
- Le altre sono asincrone perché vengono tipicamente sollevate (*molto*) *dopo* l'istruzione che le ha causate
  - addirittura, alcune non sono neanche causate dall'esecuzione di istruzioni

# Classi di Interruzioni *Asincrone*

- Interruzioni da input/output
  - generate dal controllore di un dispositivo di input/output
  - per la maggior parte, i dispositivi di input/output sono più lenti del processore
  - quindi, il processore manda un comando al dispositivo di I/O e poi aspetta che il dispositivo lo “interrompa” quando è riuscito a completare la richiesta
  - segnalano il completamento o l'errore di un'operazione di I/O
- Interruzioni da fallimento HW
  - improvvisa mancanza di potenza (power failure)
  - errore di parità nella memoria
- Interruzioni da comunicazione tra CPU
  - per sistemi dove ce n'è più d'una
- Interruzioni da timer
  - generate da un timer interno al processore
  - permettono al sistema operativo di eseguire alcune operazioni ad intervalli regolari
- Per processori Intel, gli *interrupt* sono solo questi

# Classi di Interruzioni *Sincrone*

- Interruzioni di programma, causate principalmente da:
  - overflow
  - divisione per 0
    - per Intel, anche quando si fa  $\frac{-2^{31}}{-1}$
  - debugging: single step o breakpoint
  - riferimento ad indirizzo di memoria fuori dallo spazio disponibile al programma
  - riferimento ad indirizzo di memoria momentaneamente non disponibile
    - memoria virtuale: ci ritorneremo
  - tentativo di esecuzione di un'istruzione macchina errata
    - opcode illegale, oppure operando non allineato
  - chiamata a *system call* (intenzionale!)
- Per processori Intel, queste vengono chiamate *exception*

# Interruzioni ed Istruzione di Ritorno

- Per le interruzioni asincrone, una volta che l'handler è terminato, si riprende dall'istruzione subito successiva a quella dove si è verificata l'interruzione
  - questo, ovviamente, solo se la computazione non è stata completamente abortita come conseguenza dell'interruzione
  - in realtà, potrebbe succedere che ci sia un *process switch*, ma ne parleremo nelle prossime lezioni
  - comunque, quando e se la computazione ritornerà al processo interrotto, si ricomincia dall'istruzione successiva
- Con le eccezioni sincrone, non è detto
  - *faults*: errore correggibile, viene *rieseguita la stessa istruzione*
    - es.: page fault
  - *aborts*: errore non correggibile, si esegue software collegato con l'errore
    - es.: segmentation fault
  - *traps* e *system calls*: si continua dall'istruzione successiva
    - es. per traps: debugging

# Fase di Interruzione

- Ad ogni ciclo fetch-execute, viene anche controllato se c'è stata un'interruzione (o una exception)
- Se così è, il programma viene *sospeso* e viene eseguita una funzione che gestisce l'interruzione (interrupt-handler routine)

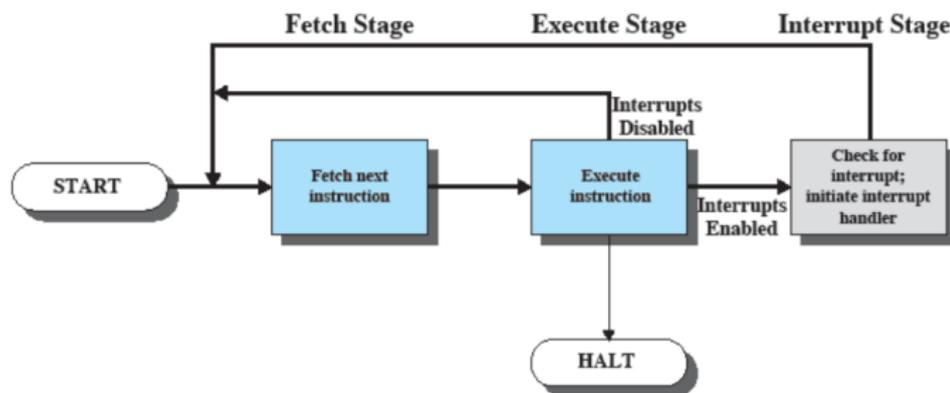
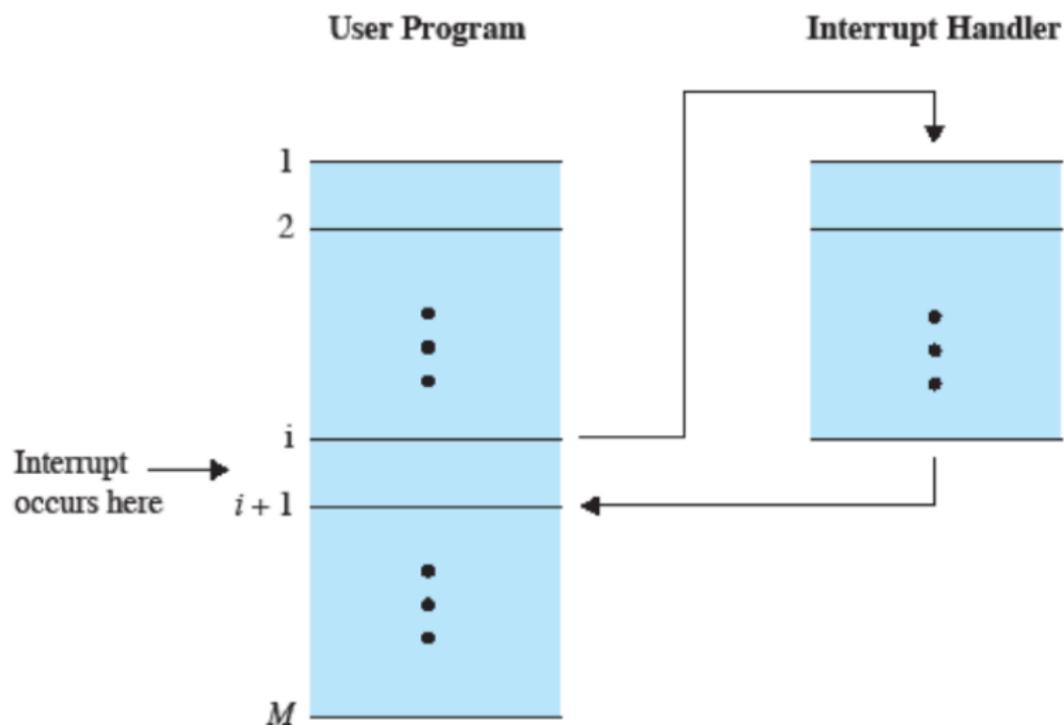


Figure 1.7 Instruction Cycle with Interrupts

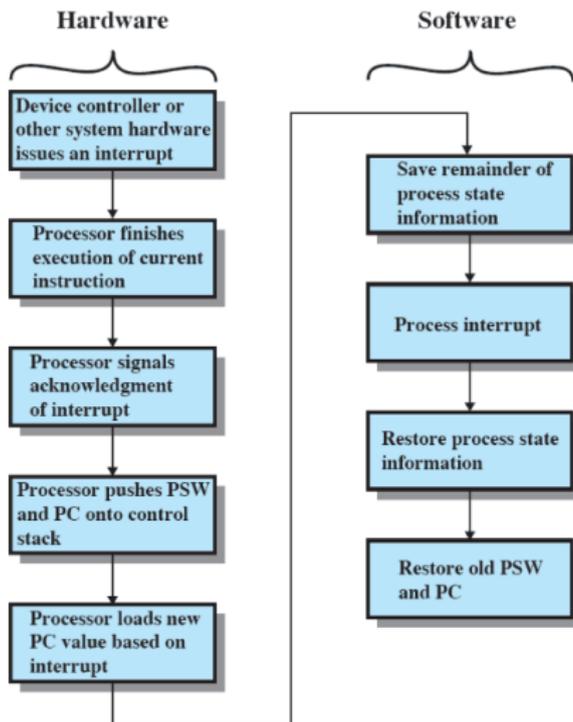
# Interruzioni: Trasferimento del Controllo

- L'interrupt handler è una funzione particolare: nel programma utente non era prevista
- Sistema operativo e hardware collaborano per salvare le informazioni (almeno registro di stato e program counter) e settare il program counter
  - normalmente, questo lo fa o il programmatore (se scrive direttamente in assembler) o il compilatore: è una chiamata a funzione
  - non è stata scritta dal programmatore, ma fin qui niente di così strano (esistono le librerie...)
  - il fatto è che, nel punto in cui avviene questa “chiamata”, il programma utente non prevedeva affatto di effettuare la chiamata all'interrupt handler!
  - soprattutto per le interruzioni “vere”; per le eccezioni, almeno si sa che è possibile che vengano sollevate
  - per funzioni “normali” salvare anche il registro di stato è solitamente superfluo, qui può essere importante

# Interruzioni: Trasferimento del Controllo

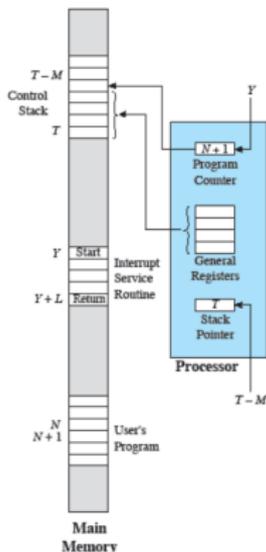


# Interruzioni: Trasferimento del Controllo



# Interruzioni: Modifiche a Memoria e Registri

Il processore sta eseguendo l'istruzione all'indirizzo  $N$  quando arriva un'interruzione, da gestire con l'handler all'indirizzo  $Y$

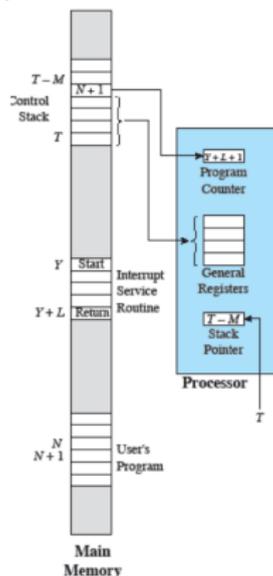


(a) Interrupt occurs after instruction at location  $N$

# Interruzioni: Modifiche a Memoria e Registri

L'handler è completato, si torna all'indirizzo  $N + 1$

Se l'interruzione era una *fault* correggibile, si torna all'indirizzo  $N$  (es.: dopo un page fault)



(b) Return from interrupt

# Interruzioni Disabilitate

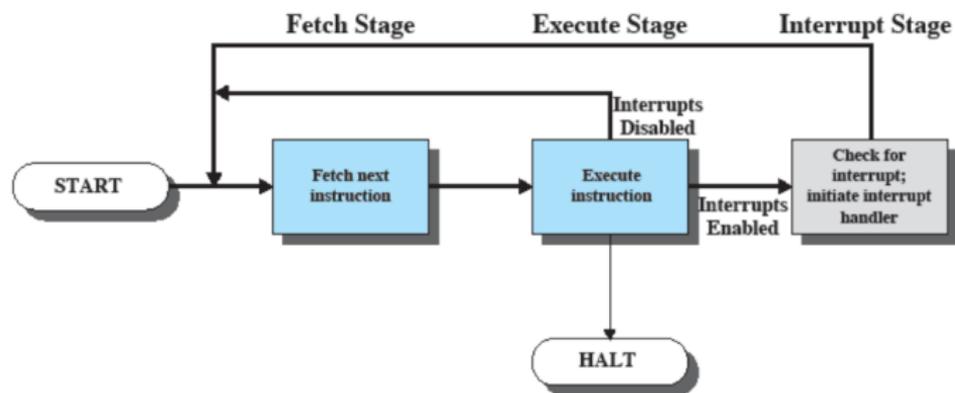
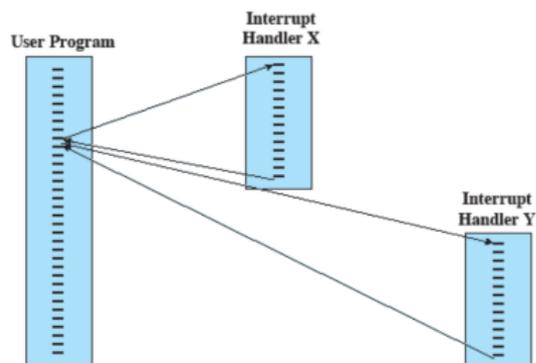
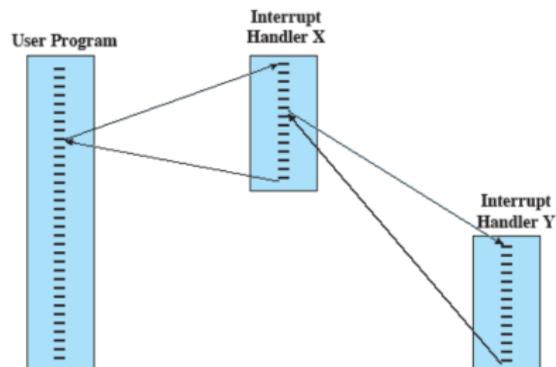


Figure 1.7 Instruction Cycle with Interrupts

# Interruzioni: Sequenziali ed Annidate

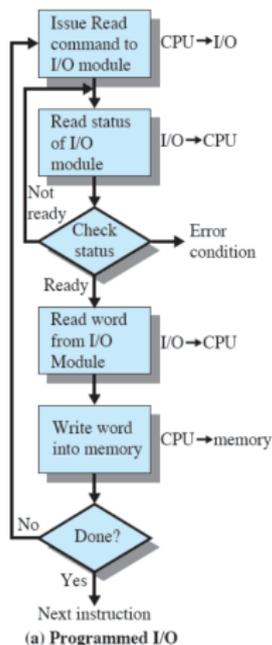


(a) Sequential interrupt processing



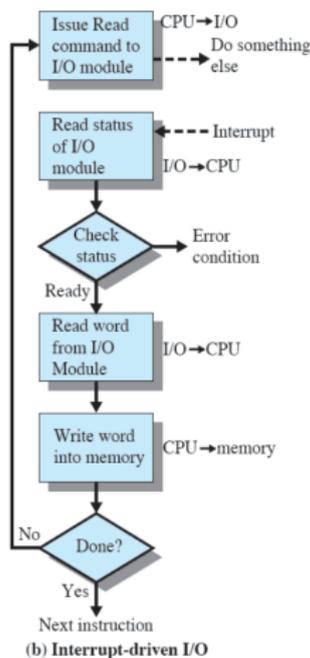
(b) Nested interrupt processing

# I/O Programmato



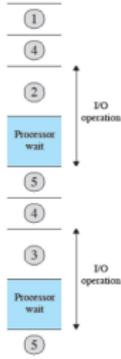
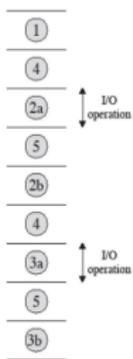
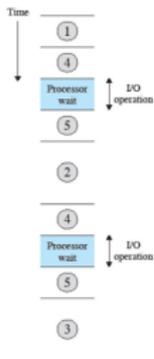
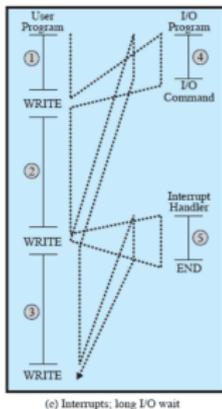
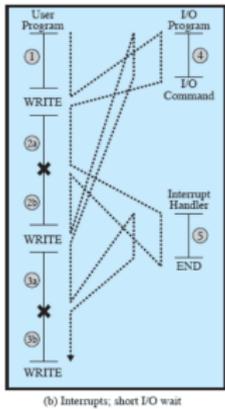
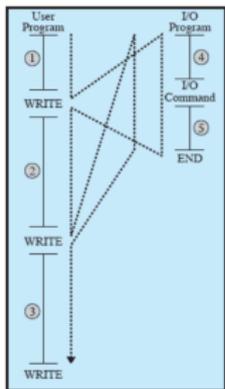
- Il più vecchio modo di fare I/O
- L'azione viene effettuata dal modulo di I/O, non dal processore
- Setta i bit appropriati sul registro di stato dell'I/O
- Niente interruzioni
- Il processore controlla lo status finché l'operazione non è completa

# I/O da Interruzioni



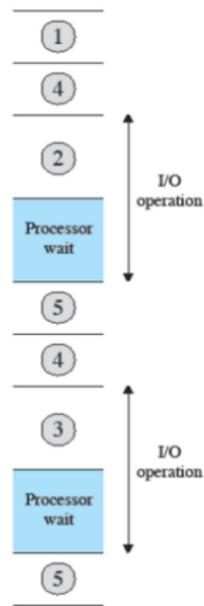
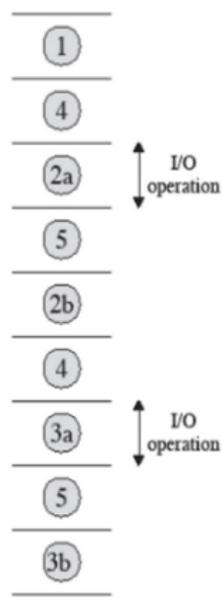
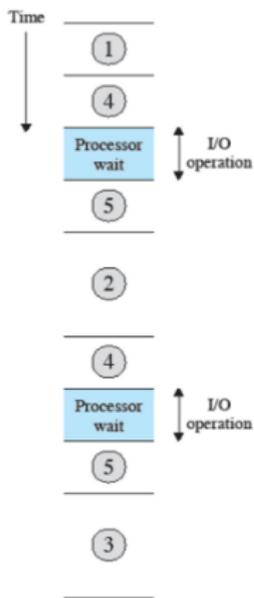
- È un modo un po' più moderno per fare I/O
- Il processore viene interrotto quando il modulo I/O è pronto a scambiare dati
- Il processore salva il contesto del programma che stava eseguendo e comincia ad eseguire il gestore dell'interruzione
- Non c'è inutile attesa
- Tuttavia, consuma molto tempo di processore, dal momento che ogni singolo dato letto o scritto deve passare per il processore

# Programma: Flusso di Controllo

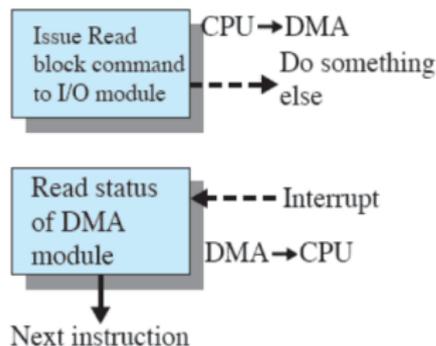




# Programma: Flusso di Controllo



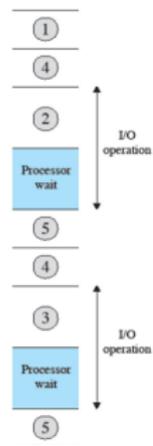
# Accesso Diretto in Memoria



(c) Direct memory access

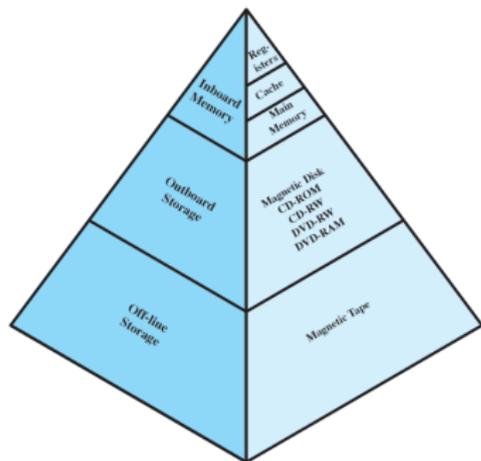
- È il metodo di I/O usato nei computer attuali
- Le istruzioni di I/O tipicamente richiedono di trasferire informazioni tra dispositivo di I/O e memoria
- Trasferisce un blocco di dati direttamente dalla/alla memoria
- Un'interruzione viene mandata quando il trasferimento è completato
- Più efficiente

# Multiprogrammazione



- Un processore deve eseguire più programmi contemporaneamente
- La sequenza con cui i programmi sono eseguiti dipende dalla loro priorità e dal fatto che siano o meno in attesa di input/output
- Alla fine della gestione di un'interruzione, il controllo potrebbe non tornare al programma che era in esecuzione al momento dell'interruzione

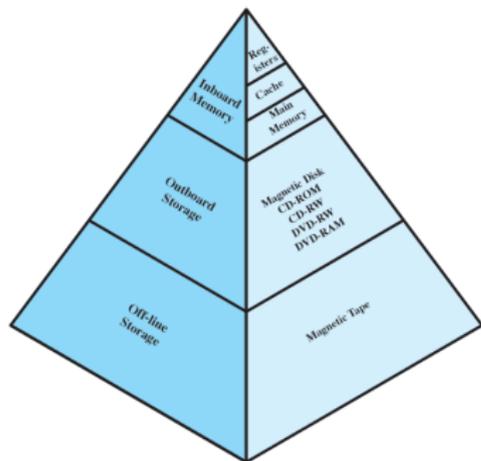
# Gerarchia della Memoria



## Dall'alto verso il basso

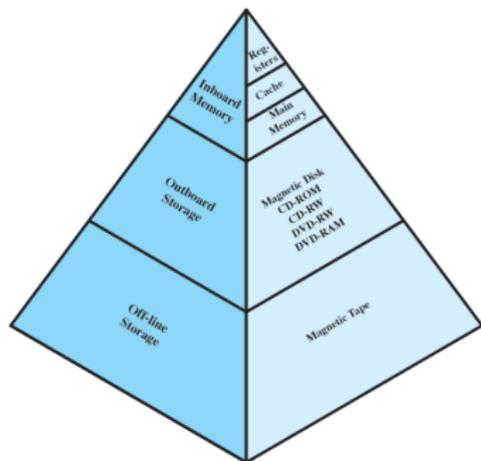
- Diminuisce la velocità di accesso
- Diminuisce il costo al bit
- Aumenta la capacità
- Diminuisce la frequenza di accesso alla memoria da parte del processore

# Gerarchia della Memoria: Memoria Secondaria



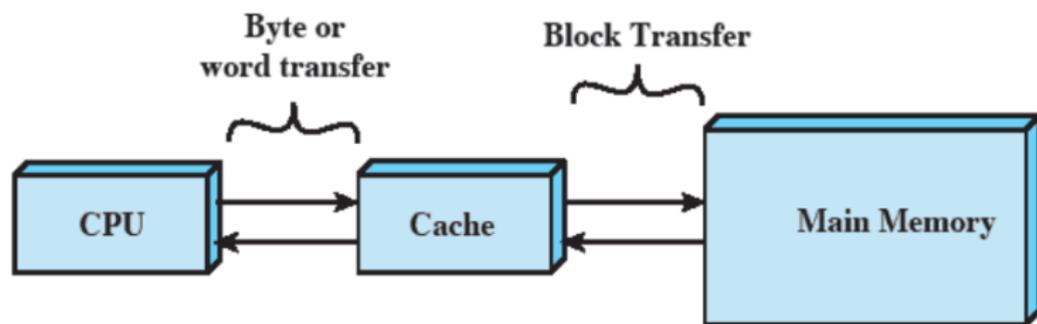
- Corrisponde all'outboard e all'offline storage
- Memoria "ausiliaria" ed "esterna"
- Non volatile: se si spegne il computer, il contenuto rimane
- Usata per memorizzare i files contenenti programmi o dati

# Gerarchia della Memoria: Memoria Cache



- Anche nell'inboard memory ci sono importanti differenze di velocità
- Infatti, la velocità del processore è maggiore della velocità di accesso alla memoria principale
- Per evitare eccessivi tempi di attesa, tutti i computer hanno una memoria cache
- Memoria piccola e veloce, che sfrutta il principio di località

# Cache e Memoria Principale



# Cache: Nozioni di Base

- Contiene copie di porzioni della memoria principale
- Il processore prima controlla se un dato è nella cache
- Se no, il corrispondente blocco di memoria principale viene caricato nella cache
- Siccome vale la *località dei riferimenti*, è probabile che il dato appena caricato serva ancora nell'immediato futuro
  - in generale, i futuri riferimenti in memoria ricadranno probabilmente nel blocco appena caricato
- Gestione totalmente demandata all'hardware
  - il programmatore non "vede" la cache, neanche se usa l'assembler
  - quindi, non la "vede" neanche un compilatore
  - e neanche il sistema operativo, che però la "imita" spesso

# Cache vs. Memoria Principale

$$\frac{2^n}{K} \gg C; |Tag| = \log_2 \frac{2^n}{K}; \text{tipicamente, } K = 4$$

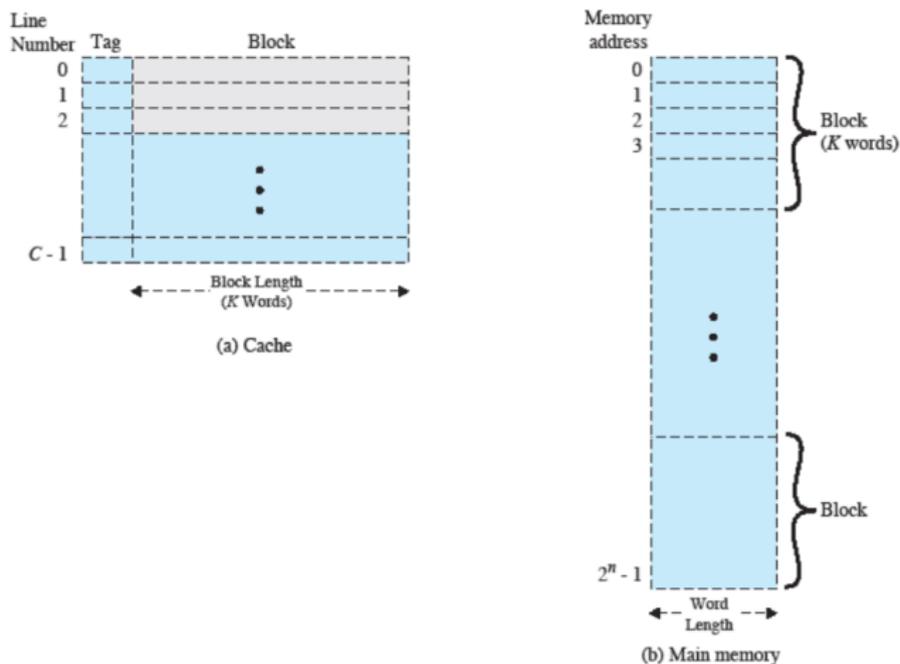
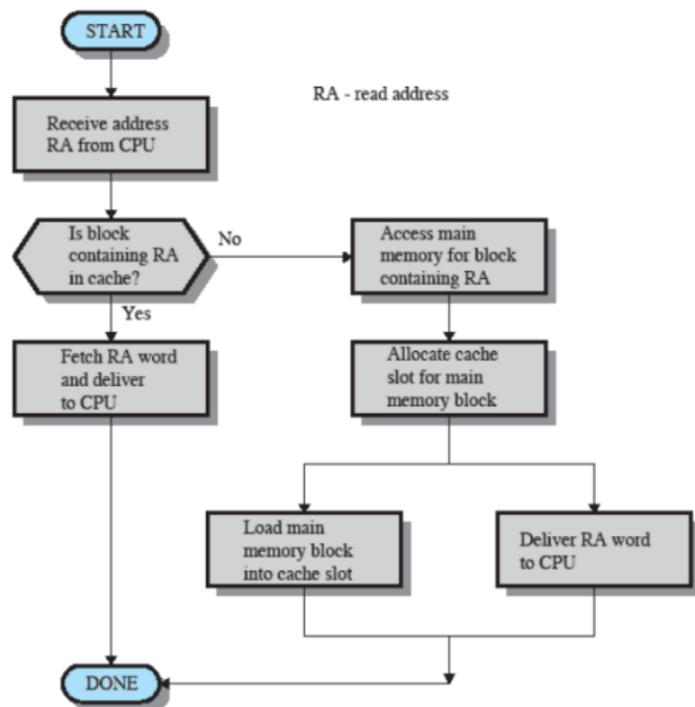


Figure 1.17 Cache/Main-Memory Structure

# Lettura dalla Cache



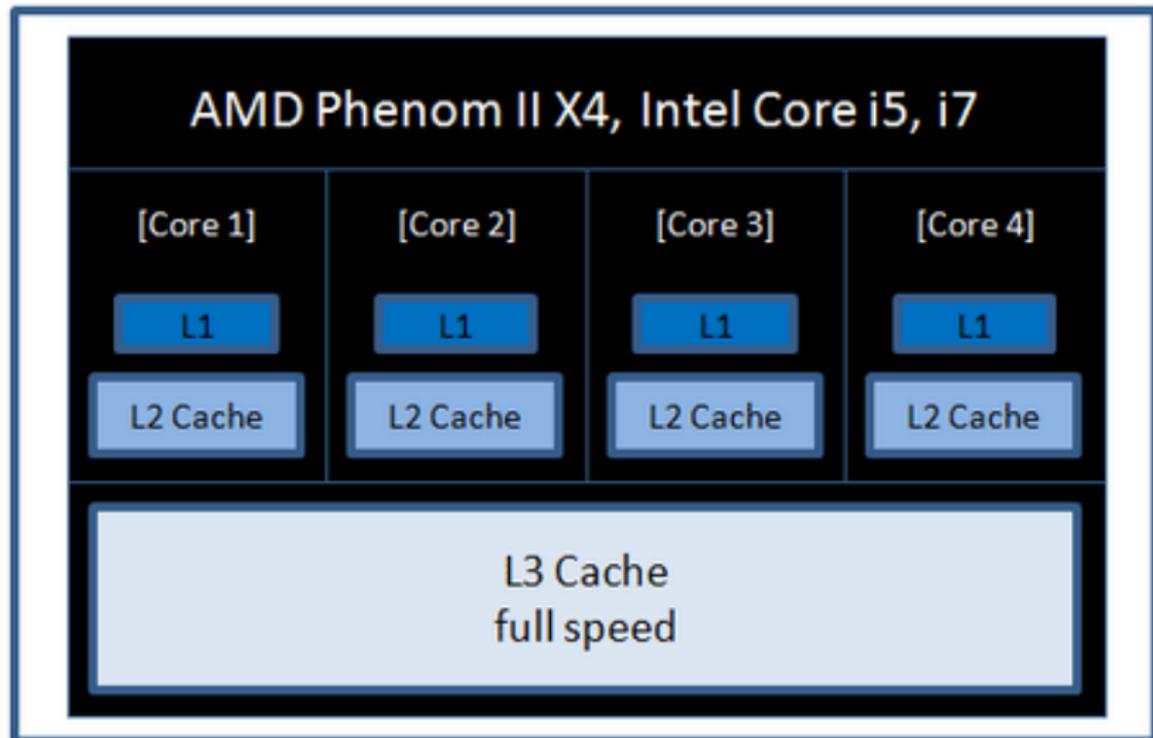
# Cache: Nozioni di Base

- Capacità della cache
  - cache anche piccole hanno un grande impatto sulle performance
- Misura dei blocchi
  - i dati scambiati tra memoria e cache sono in quantità multiple di un blocco
  - incrementare la misura del blocco aumenta il numero di accessi riusciti
  - ma incrementarla troppo è controproducente: saranno di più anche i dati che vengono rimossi
  - il che abbassa la probabilità di accesso riuscito

# Cache: Nozioni di Base

- Funzione di mappatura
  - determina la locazione della cache nella quale andrà messo il blocco proveniente dalla memoria
- Algoritmo di rimpiazzamento
  - sceglie il blocco da rimpiazzare
  - algoritmo Least-Recently-Used (LRU): si rimpiazza il blocco usato meno di recente (quindi, più vecchio)
- Politica di scrittura
  - determina quando occorre scrivere in memoria
  - può accadere ogni volta che un blocco viene modificato (*write-through*)
  - può accadere quando il blocco è rimpiazzato (*write-back*)
  - occorre minimizzare le operazioni di scrittura
  - questo vuol dire che la memoria può trovarsi in uno stato “obsoleto”, ovvero non in linea con il contenuto della cache

## La Cache nei Pentium i7



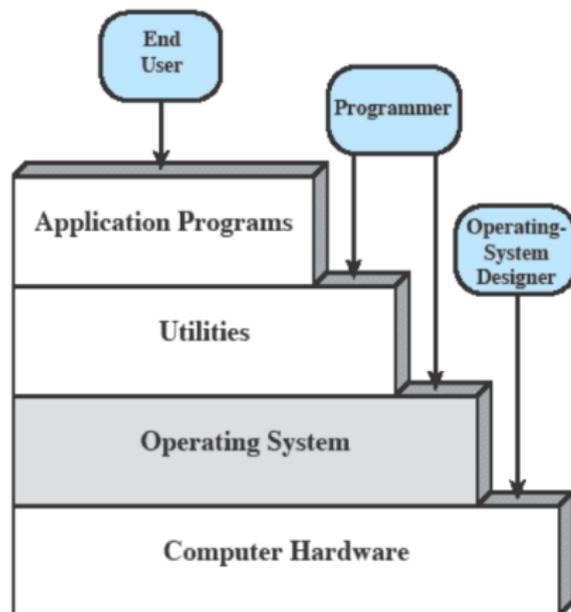
# La Cache nei Pentium i7

- Ogni cache line ha 64 byte e ogni word ha 64 bit, quindi  $K = 8$
- Ci sono 2 cache di primo livello (L1), più altre 2 cache di secondo e terzo livello (L2 e L3)
  - le 2 di primo livello sono l'una dedicata alle istruzioni, l'altra dedicata ai dati
  - si va a cascata: dato un indirizzo RAM, prima si cerca in L1, se fallisce in L2, se fallisce in L3, e se fallisce ancora in memoria
  - ovviamente, se si trova un dato in L2 e non in L1, L1 va aggiornata
- Le dimensioni della cache crescono da 32 KB per ciascuna delle L1, a 256 KB per L2, fino a 8MB per L3
- Il sistema operativo può disabilitare il caching
  - Linux non lo fa
- Politica di scrittura: la può decidere il sistema operativo
  - Linux sceglie sempre il write-back

# Sistema Operativo

- Gestisce le risorse hardware di un sistema computerizzato
  - uno o più processori
  - memoria primaria (RAM)
  - dispositivi di input/output
    - memoria secondaria (dischi)
    - rete
    - ...
- Il suo scopo è fornire un insieme di servizi agli utenti
  - servizi sia per gli sviluppatori che per i semplici utilizzatori
  - esempio: offre un ambiente di esecuzione “facilitato” alle applicazioni utente
  - esempio: offre un ambiente grafico per eseguire applicazioni

# Strati e Utenti



Quando si programma con un linguaggio interpretato, sostanzialmente ci si basa solo sulle “utilities”



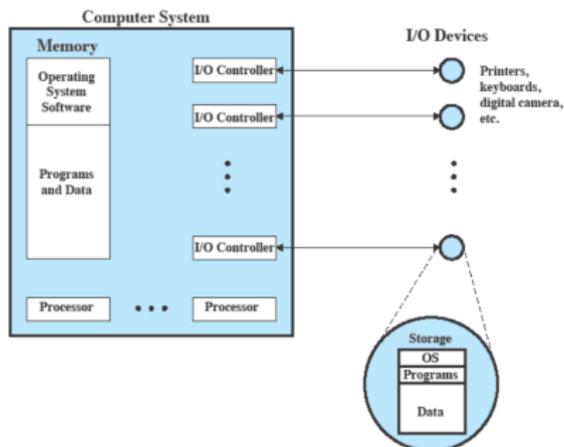
# Servizi Offerti da un Sistema Operativo

- Rilevamento di e reazione ad errori
  - errori di hardware interno ed esterno
  - errori software
  - richiesta di un applicativo non soddisfacibile
- Accounting (chi fa cosa)
  - collezione di statistiche dell'uso del sistema
  - monitoraggio delle performance
  - usato per capire cosa occorre migliorare
  - usato per far pagare in base all'uso del sistema

- Un programma che controlla l'esecuzione dei programmi applicativi
  - gli prepara l'ambiente, li manda in esecuzione, risponde a loro eventuali richieste, gestisce la loro terminazione
- Un'interfaccia tra le applicazioni e l'hardware
- **Obiettivi** di un sistema operativo
  - Convenienza
  - Efficienza
  - Capacità di evolvere

# Sistema Operativo

- Responsabile della gestione delle risorse
  - funziona allo stesso modo del software “normale”: è un programma in esecuzione
    - tuttavia, lo fa con privilegi più alti
  - concede il controllo del processore ad altri programmi
    - e controlla l'accesso alle altre risorse (RAM, I/O)



# Il Kernel

- La parte di sistema operativo che si trova sempre in memoria principale
- Contiene le funzioni più usate
- Letteralmente, sta per “nucleo”
- Restano fuori funzionalità che si usano per casi particolari
  - ad es.: file system poco usati

# Evoluzione dei Sistemi Operativi

- I sistemi operativi “primitivi” (anni Quaranta) erano molto diversi da quelli attuali
- Si è arrivati alla situazione attuale gradualmente, tramite un’*evoluzione*
- Evoluzione dovute principalmente a:
  - in seguito ad aggiornamento dell’hardware, o a nuovi tipi di hardware
  - nuovi servizi
  - correzione di errori
- Conoscere questa evoluzione storica è importante per capire i sistemi operativi moderni
  - inoltre, alcune metodologie abbandonate potrebbero tornare utili in altri ambiti
- <http://www.computerhistory.org/timeline/>

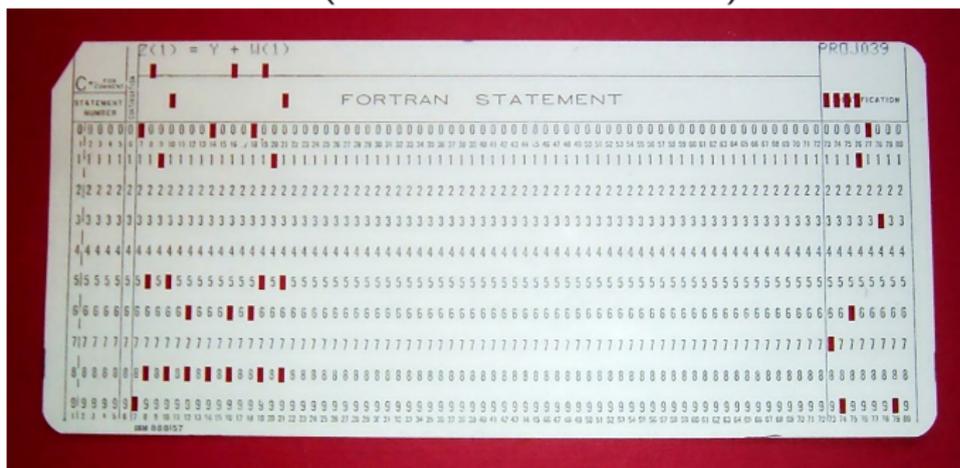
# Storia dei Sistemi Operativi

- Computazione seriale (anni Quaranta)
  - nessun sistema operativo
  - per fornire comandi ad un computer, si usavano speciali “console” con spie luminose ed interruttori, e una stampante per output



# Storia dei Sistemi Operativi

- Già all'inizio l'input viene parzialmente semplificato con dispositivi per leggere schede perforate (esistenti da 2 secoli)
- Per scrivere una scheda perforata, si poteva usare una speciale macchina da scrivere (ad es.: IBM 024 e 026)





# Caratteristiche Hardware

- Protezione della memoria
  - non permette che la zona di memoria contenente il monitor venga modificata
- Timer
  - impedisce che un job monopolizzi l'intero sistema
- Istruzioni privilegiate
  - alcune istruzioni macchina possono essere eseguite solo dal monitor
  - interruzioni
  - i primi modelli di computer non le avevano

# Protezione della Memoria

- I programmi utente vengono eseguiti in **modalità utente**
  - alcune istruzioni non possono essere eseguite
  
- Il monitor viene eseguito in **modalità sistema**
  - o modalità kernel
  - le istruzioni privilegiate possono essere eseguite
  - le aree protette della memoria possono essere accedute

## Sistemi Batch: Sottoutilizzazione

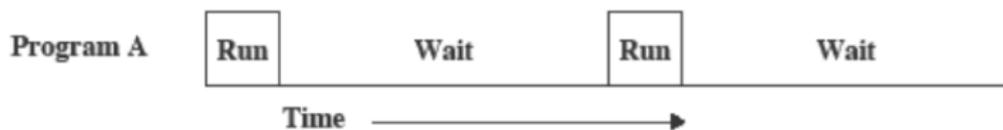
Più dell 96% del tempo è sprecato ad aspettare i dispositivi di I/O

Read one record from file	$15 \mu s$
Execute 100 instructions	$1 \mu s$
Write one record to file	<u><math>15 \mu s</math></u>
TOTAL	$31 \mu s$

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

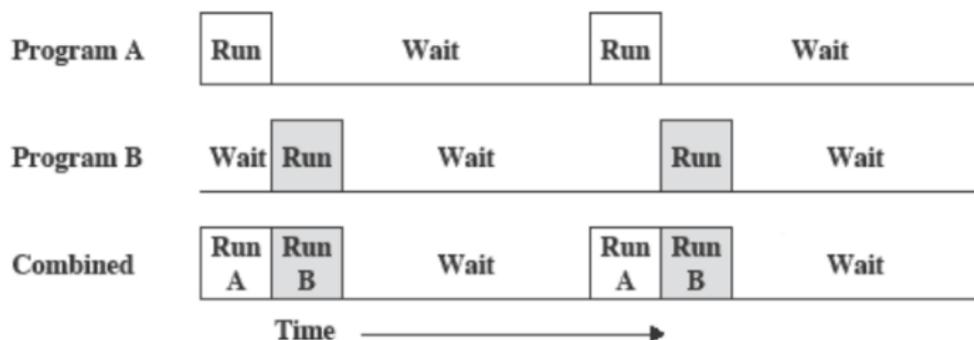
# Programmazione Singola

Il processore deve aspettare che le istruzioni di I/O siano completate prima di procedere

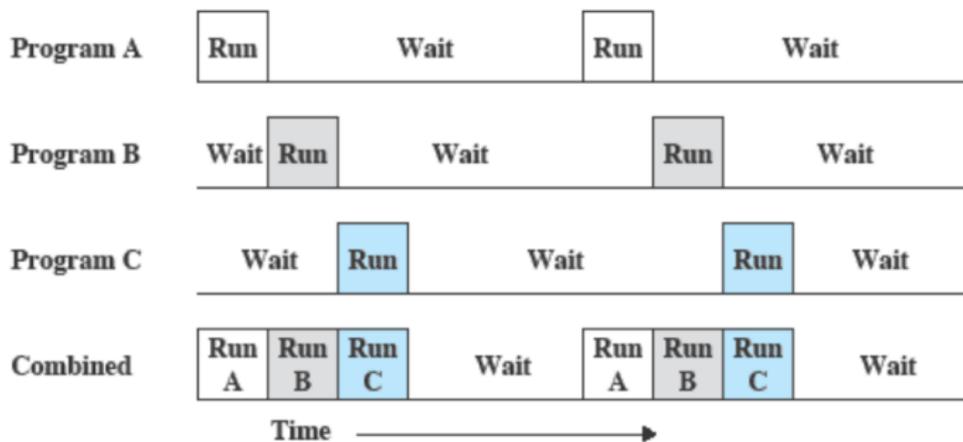


# Multiprogrammazione

Se un job deve aspettare che si completi dell'I/O, allora il processore può passare ad un altro job



# Multiprogrammazione

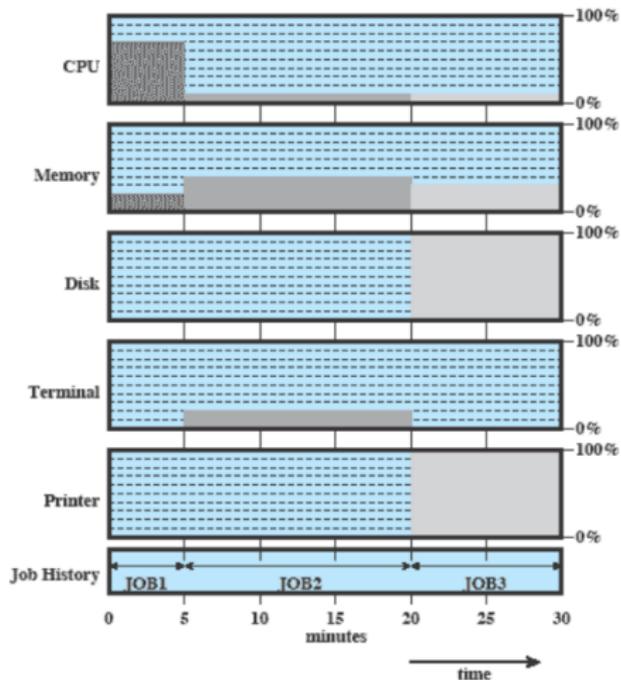


(c) Multiprogramming with three programs

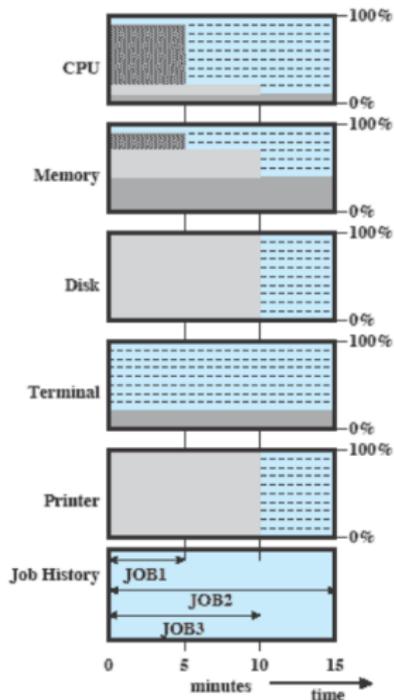
# Esempio

	JOB1	JOB2	JOB3
<b>Type of job</b>	Heavy compute	Heavy I/O	Heavy I/O
<b>Duration</b>	5 min	15 min	10 min
<b>Memory required</b>	50 M	100 M	75 M
<b>Need disk?</b>	No	No	Yes
<b>Need terminal?</b>	No	Yes	No
<b>Need printer?</b>	No	No	Yes

# Istogrammi di Utilizzo



(a) Uniprogramming



(b) Multiprogramming

# Uso del Processore

- Prime 4 righe: media tra le percentuali del uniprogramming e del multiprogramming
  - c'è un errore, il valore per la CPU uniprogramming è 26.7% (media tra 60%, 10%, 10%)
- Elapsed time: tempo per vedere completati tutti e 3 i job
- Throughput:  $\frac{\text{numero job completati}}{\text{ore}}$
- Mean response time: media dei tempi di completamento (nel caso di uniprogramming: 5, 20, 30  $\rightarrow$  18.3)

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

# Sistemi Time Sharing

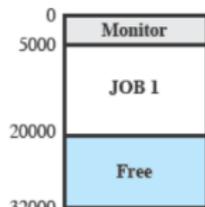
- Letteralmente, sistemi a condivisione di tempo (dagli anni Settanta)
- Uso della multiprogrammazione per gestire contemporaneamente più jobs *interattivi*
- Il tempo del processore è condiviso tra più utenti
- Più utenti contemporaneamente accedono al sistema tramite terminali

# Batch vs. Time Sharing

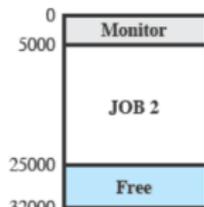
	<b>Batch</b>	<b>Time Sharing</b>
<b>Scopo principale</b>	Massimizzare l'uso del processore	Minimizzare il tempo di risposta
<b>Provenienza delle direttive al Sistemi Operativi</b>	Comandi del job control language, sottomessi con il job stesso	Comandi dati da terminale

# CTSS: Compatible Time-Sharing System

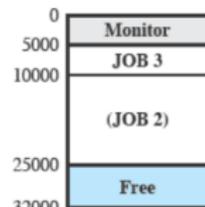
Sistema operativo anni '60 che supportava il time sharing



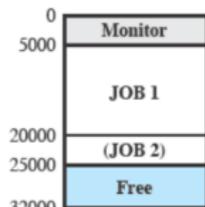
(a)



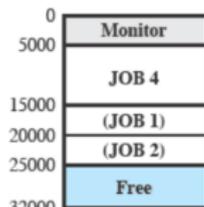
(b)



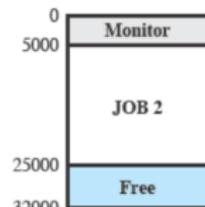
(c)



(d)



(e)



(f)

# Storia dei Sistemi Operativi

- Computazione seriale (anni Quaranta)
- Semplice sistema non interattivo o *batch* (anni Cinquanta/Sessanta)
  - multiprogrammazione
- Time-Sharing: sistemi a condivisione di tempo (dagli anni Settanta)
  - qui invece i job sono tipicamente interattivi

# Storia dei Sistemi Operativi: Risultati più Importanti

- Processi
- Gestione della memoria
- Sicurezza e protezione delle informazioni (privacy)
- Gestione dello scheduling e delle risorse
- Strutturazione del sistema

# Dal Job al Processo

- Il *processo* riunisce in un unico concetto il job non-interattivo e quello interattivo
- Incorpora anche un altro tipo di job che cominciò a manifestarsi dagli anni Settanta: quello transazionale real-time
  - ad es.: prenotazione biglietti aerei
- Un'unità di attività caratterizzata da:
  - un singolo flusso (thread) di esecuzione
  - uno stato corrente
  - un insieme di risorse di sistema ad esso associate

# Multiprogrammazione dei Processi: Difficoltà

- Errori di sincronizzazione
  - gli interrupt si perdono o vengono ricevuti 2 volte
- Violazione della mutua esclusione
  - se 2 processi vogliono accedere alla stessa risorsa, ci possono essere problemi
- Programmi con esecuzione non deterministica
  - un processo accede ad una porzione di memoria modificata da un altro processo
- Deadlock (stallo)
  - un processo A attende un processo B che attende A

# Gestione della Memoria

- Isolamento dei processi
- Protezione e controllo degli accessi
- Gestione (compresa allocazione/deallocazione) automatica
- Supporto per la programmazione modulare (stack)
- Memorizzazione a lungo termine
- Metodi attuali: paginazione + memoria virtuale

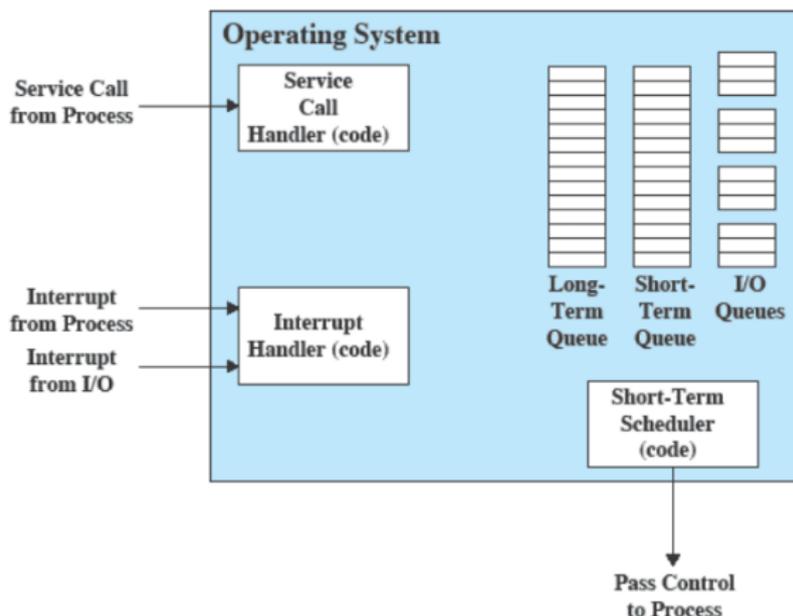
# Protezione dell'Informazione e Sicurezza

- Disponibilità (availability)
  - il dover proteggere il sistema contro l'interruzione di servizio (DoS attacks)
- Confidenzialità
  - garanzia che gli utenti non leggano informazioni per le quali non hanno l'autorizzazione
- Integrità dei dati
  - protezione dei dati da modifiche non autorizzate
- Autenticità
  - il dover verificare l'identità degli utenti, o la validità di messaggi e dati

# Pianificazione e Gestione delle Risorse

- Equità (fairness)
  - dare accesso alle risorse in modo egualitario ed equo
- Velocità di risposta differenziata
  - a seconda del tipo di processo
- Efficienza
  - massimizzare l'uso delle risorse per unità di tempo (throughput), minimizzare il tempo di risposta, e servire il maggior numero di utenti possibile

# Elementi Principali di un Sistemi Operativo



Nota: il "service call" o *system call* è in realtà anch'esso un interrupt (o eccezione, seguendo la terminologia Pentium)



# Livelli

- Livello 1
  - circuiti elettrici
  - gli oggetti sono registri, celle di memoria, porte logiche
  - le operazioni sono, ad esempio, resettare un registro o leggere una locazione di memoria
- Livello 2
  - insieme delle istruzioni macchina
  - operazioni come add, subtract, load, and store
- Livello 3
  - aggiunge il concetto di procedura (o subroutine), con operazioni di chiamata e ritorno
- Livello 4
  - interruzioni

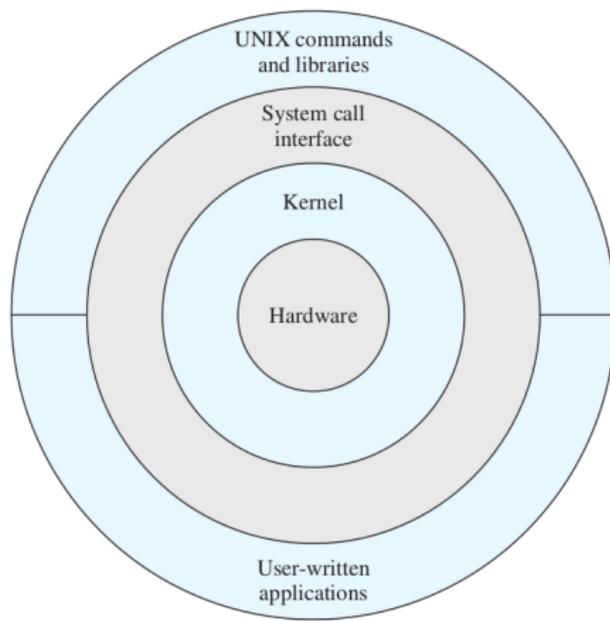
# Livelli: Multiprogrammazione

- Livello 5
  - processo come programma in esecuzione
  - sospensione e ripresa dell'esecuzione di un processo
- Livello 6
  - dispositivi di memorizzazione secondaria
  - trasferimento di blocchi di dati
- Livello 7
  - crea uno spazio logico degli indirizzi per i processi
  - organizza lo spazio degli indirizzi virtuali in blocchi

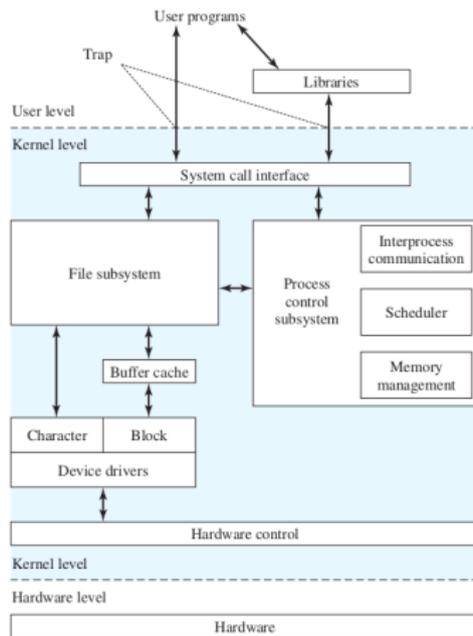
# Livelli: Dispositivi Esterni

- Livello 8
  - comunicazioni tra processi
- Livello 9
  - salvataggio di lungo termine di file con nome
- Livello 10
  - accesso a dispositivi esterni usando interfacce standardizzate
- Livello 11
  - associazione tra identificatori interni ed esterni
- Livello 12
  - supporto di alto livello per i processi
- Livello 13
  - interfaccia utente

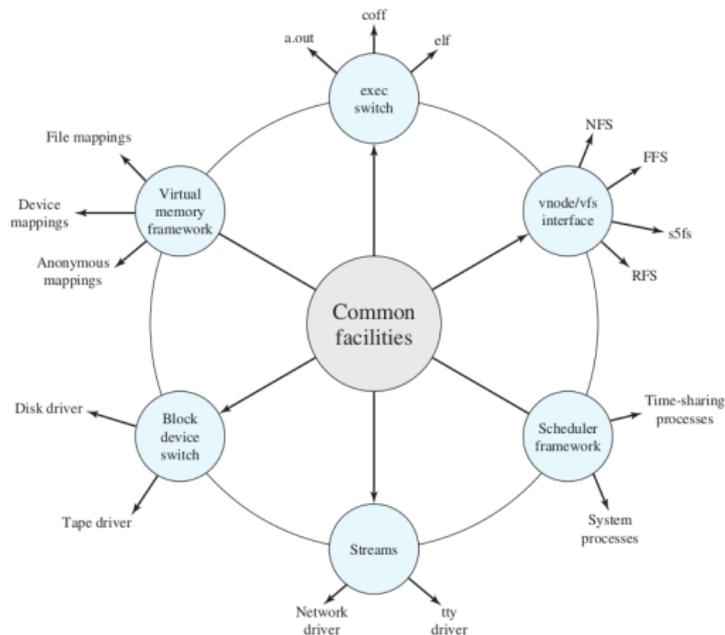
# Architettura di UNIX



# Kernel Tradizionale di UNIX



# Kernel Moderno di UNIX



# Kernel Moderno di Linux

- Compromesso tra kernel “monolitico” e “microkernel”
  - monolitico: kernel tutto in memoria dal boot allo spegnimento
  - microkernel: solo una minima parte del kernel in memoria, il resto caricato quando serve
    - sempre in memoria: scheduler, sincronizzazione
    - solo a richiesta: gestore memoria, filesystem, driver
  - monolitico più efficiente come velocità, ma ovviamente occupa più memoria e rende difficile la modularità
- Quasi tutti i sistemi operativi moderni sono a kernel monolitico
  - eccezione notevole: Mac OS X
- Linux è principalmente monolitico, ma ha i *moduli*
  - alcune parti particolari possono essere aggiunte e tolte a richiesta dall'immagine in memoria del kernel
  - essenzialmente, i diversi file system, i driver per determinati dispositivi di I/O, l'implementazione delle funzionalità di rete

# In questo corso...

- 1 Gestione dei processi
- 2 Scheduling dei processi
- 3 Gestione della memoria principale
- 4 Gestione della concorrenza tra processi
- 5 Gestione del deadlock
- 6 Gestione dell'input/output
- 7 Gestione dei file system
- 8 Nozioni basilari di sicurezza