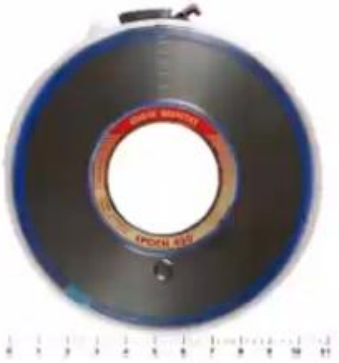
Storia dei SO

- **1ª generazione** (1945-55) valvole termoioniche:
  - La programmazione era in linguaggio macchina oppure attraverso il cablaggio di complessi circuiti elettrici.
  - Nel 1950 nasce il primo supporto dati cartaceo per la memorizzazione ed il caricamento automatico dei programmi.

Storia dei SO

- **2ª generazione** (1955-65) Transistor e sistemi batch:
  - Intorno alla metà degli anni '50 nasce il transistor.
  - I primi mainframe IBM occupano intere stanze, costano molti milioni e sono utilizzati soltanto da università e grandi gruppi industriali.
  - I programmi (**job**) sono scritti in FORTRAN (o assembler), prima memorizzato sulle schede perforate e successivamente su nastro utilizzando **sistemi batch** di elaborazione.



Storia dei SO

- **3ª generazione** (1965-1980) circuiti integrati e multiprogram:
  - Nasce l'IBM System/360 che utilizza i circuiti integrati ed il sistema operativo OS/360 che permette la multiprogrammazione (la **zSeries** è un suo discendente!).
  - Questi sistemi possono leggere i programmi dal disco attraverso una tecnica denominata di **spooling** (poi utilizzata per l'output).
  - Nasce il primo sistema **timesharing**, CTSS (Compatible Time Sharing System) sviluppato al M.I.T. e, successivamente il **MULTICS**. Da una versione derivata e ridotta, utilizzata su un minicomputer il PDP-7, si gettano le basi per la progettazione del sistema **UNIX**.

Storia dei SO

- **3ª generazione** (1965-1980) circuiti integrati e multiprogram:
  - Nasce l'IBM System/360 che utilizza i circuiti integrati ed il sistema operativo OS/360 che permette la multiprogrammazione (la **zSeries** è un suo discendente!).
  - Questi sistemi possono leggere i programmi dal disco attraverso una tecnica denominata di **spooling** (poi utilizzata per l'output).
  - Nasce il primo sistema **timesharing**, CTSS (Compatible Time Sharing System) sviluppato al M.I.T. e, successivamente il **MULTICS**. Da una versione derivata e ridotta, utilizzata su un minicomputer il PDP-7, si gettano le basi per la progettazione del sistema **UNIX**.

Storia dei SO

- **4ª generazione** (1980-oggi) Personal Computer
  - Digital Research riscrive il CP/M (Control Program for Microcomputers) per utilizzare il processore Zilog Z80.

  - IBM progetta il primo PC con il DOS (Disk Operating System) e il linguaggio Basic.
  - Apple progetta Lisa (troppo costoso) e Macintosh il primo ambiente **user-friendly**.

  - Microsoft, influenzata dal successo di Macintosh, realizza Windows e Apple MacOS.
  - Iniziano ad essere utilizzate alcune versioni di UNIX sui PC: Linux , FreeBSD,…

Astrazioni dei Sistemi Operativi

- I sistemi operativi fornisscono dei **concetti di base** e delle **astrazioni**:

Processi

Spazio indirizzi

File

I/O

Protezione

Shell

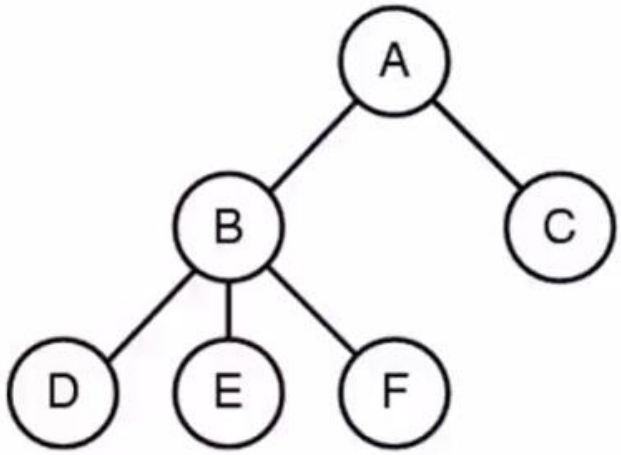
Processi

- Il processo è l'esecuzione di un programma, ogni processo ha associato:
  - uno **spazio degli indirizzi** dove sono memorizzati il programma eseguibile, i dati e lo stack;
  - un insieme di risorse: registri (inclusi PC e SP), file aperti, elenco di processi correlati, allarmi in sospeso,…;
  - una riga all'interno della **tabella dei processi**.

- I processi possono cooperare per raggiungere un obiettivo comune ed hanno bisogno di sincronizzarsi attraverso comunicazioni (**interprocess comunication** or **IPC**).

Processi

- Ai processi possono essere inviati dei **segnali** in analogia agli interrupt hardware.
- L'utente che ha avviato un processo ne è il proprietario (User IDentifier or **UID**).
- Un processo può creare processi figli (con stesso UID) in modo ricorsivo creando un struttura gerarchica.



Lo spazio degli indirizzi

- Ogni computer ha una memoria principale per gestire l'esecuzione dei programmi.
- Nei primi SO poteva stare in memoria un solo programma alla volta.
- Nei moderni SO più programmi possono risiedere in memoria nello stesso tempo, ma sono necessari dei meccanismi di protezione per evitare interferenze.
- La virtualizzazione della memoria consente al processo di vedere uno spazio indirizzi unico anche se parte dei dati sono davvero in memoria principale ed altri sul disco (e richiamati solo all'occorrenza).

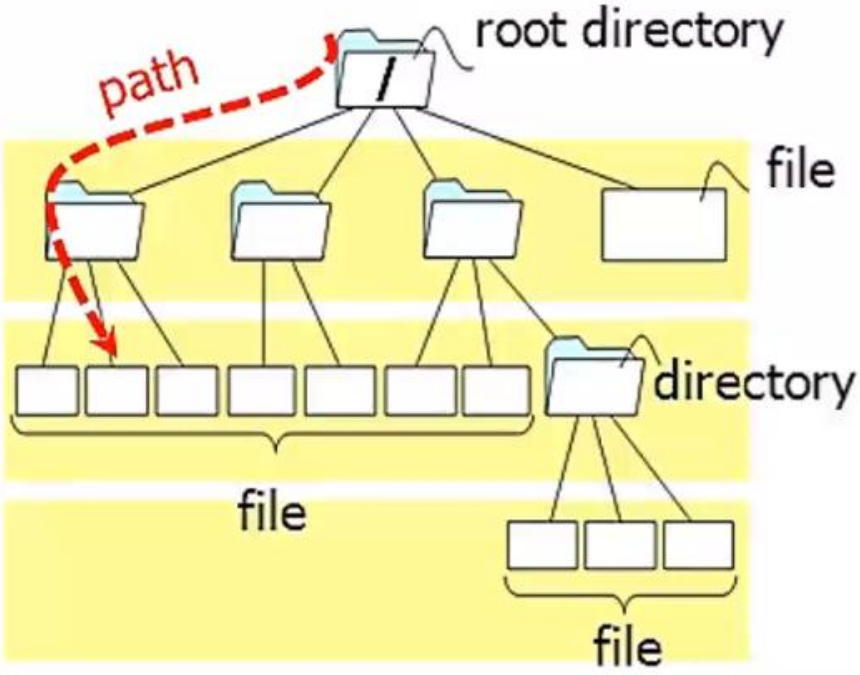
File

- I file sono il modello astratto per raggruppare insieme di dati nascondendo le peculiarità dell'hardware che li memorizza o, in generale, utilizza (nastri, dischi, dispositivi di I/O,...).

- Il file system è l'ambiente che gestisce i file ed è organizzato in modo gerarchico: ciascun nodo intermedio è un contenitore di altri file (directory) o altri contenitori, in modo ricorsivo.

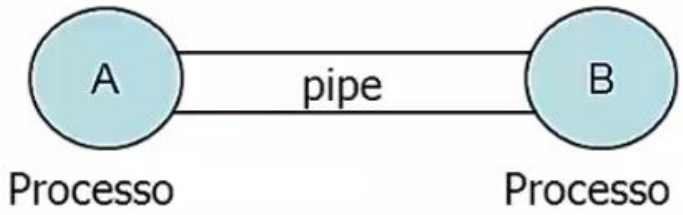
File

- L'elenco delle directory che occorre attraversare dalla radice (root) per raggingere il file è detto percorso o **path** del file (il simbolo slash o controslash è utilizzato come separatore dei nomi).



File

- Prima che un file possa essere letto/scritto è necessario aprirlo e in questo momento sono controllati i permessi, se l'accesso è consentito è restituito un numero intero detto **descrittore del file**.
- In UNIX i **file speciali** permettono di considerare i dispositivi di I/O come file e sono di due tipi:
  - **a blocchi**, utilizzati per modellare dispositivi in grado di indirizzare casualmente blocchi di dati (es. dischi);
  - **a caratteri**, utilizzati per modellare dispositivi che utilizzano le sequenze (o **stream**) di caratteri (es. stampanti, tastiere,…).
- Per connettere serialmente l'uscita di un processo con l'ingresso di un altro si può utilizzare uno pseudofile chiamato **pipe**:



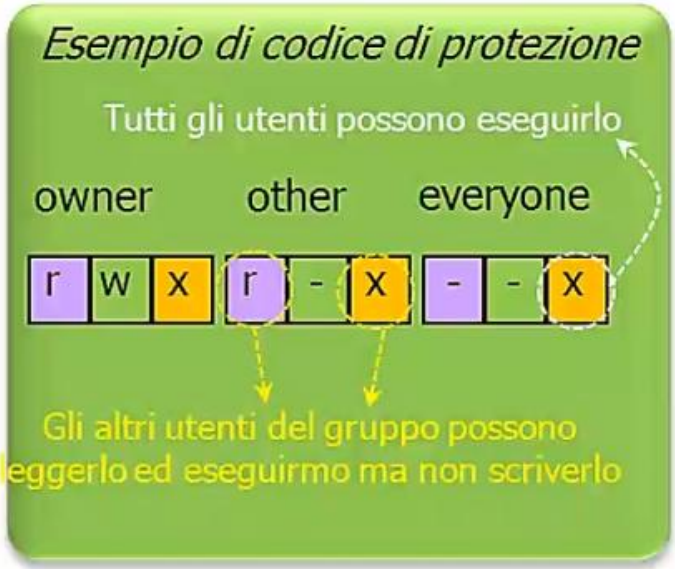
Protezione

- La protezione dei dati è una caratteristica importante dei SO.

- Considerando come esempio UNIX, i file sono protetti mediante l'assegnazione di un **codice di protezione** a 9 bit costituito da tre ambiti:
  - Il proprietario (user are organized in groups by the system administrator);
  - Gli utenti del medesimo gruppo del proprietario (gli utenti sono organizzati in gruppi definiti dall'amministratore del Sistema);
  - Tutti gli altri utenti.

Protezione

- Ogni ambito ha tre bit (rwx bits) che definiscono le operazioni consentite:
  - lettura (r)
  - scrittura (w)
  - esecuzione (x)



Shell

- Il SO è il codice che realizza le chiamate di Sistema.
- Gli editor, i compilatori, gli assemblatori, i linker e l'interprete dei comandi (la **shell** in UNIX) non fanno parte del SO.
- La GUI (Graphic User Interface) è un'applicazione che sta sopra al SO proprio come l'interprete dei comandi, ma non fa parte del SO.

System call

- I SO hanno due funzioni principali:
  - Forniscono astrazioni ai programmi utente.
  - Gestiscono le risorse del computer.

- Qualsiasi computer monoprocessore può eseguire una istruzione per volta. Se un processo utente ha bisogno di leggere un file deve chiamare un servizio di sistema, che restituirà poi il controllo al chiamate una volta eseguito il servizio.

System call

- Una **chiamata di sistema** o **system call** è una procedura speciale che viene eseguita in modalità kernel, ad esempio:

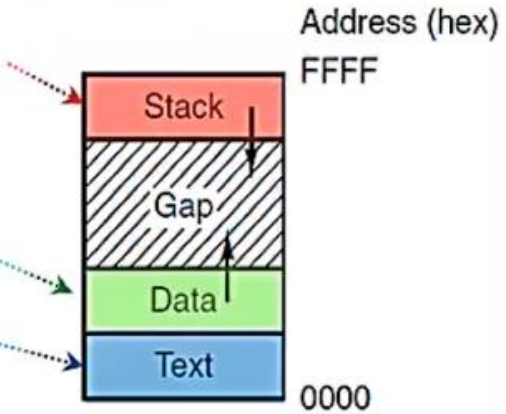
  count = read(fd, buffer, nbytes)

  La chiamata di sistema *read()* restituisce il numero di byte realmente letti nel *buffer* : non è detto che si riesca a leggere *nbytes*.

System call per la gestione dei processi

- In POSIX (lo standard per UNIX) la chiamata **fork**() permette di creare una replica del processo (chiamato processo figlio), a questo punto i due processi avranno una vite separate.
- In UNIX ad ogni processo è assegnato uno spazio di indirizzi suddiviso in tre segmenti disgiunti:

– **stack**, utilizzato per gestire le attivazioni delle procedure;
– **dati**, lo spazio per memorizzare i dati
– **testo**, il codice del programma

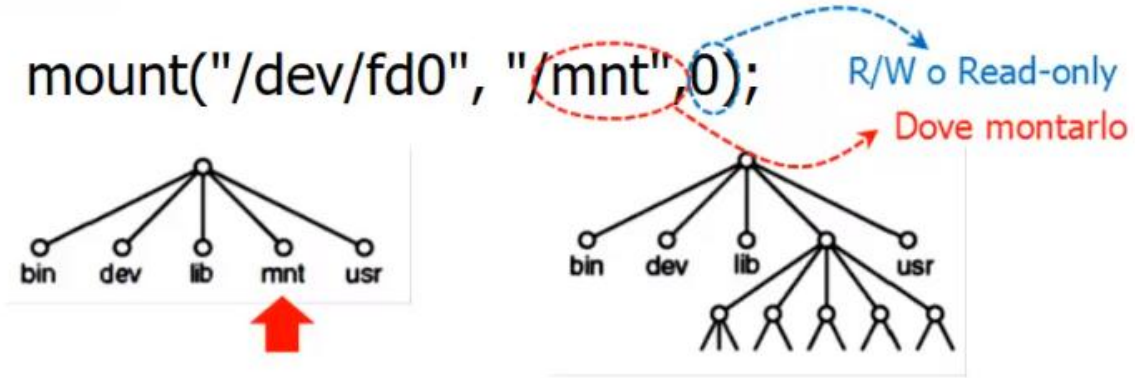


25

System Call per la gestione delle directory

- Le chiamate **mkdir**() e **rmdir**() servono rispettivamente a creare una directory e cancellarne una vuota.
- **link**() permette di creare un riferimento ad un file o una directory.
- **mount**() permette di fondere insieme due file system, è utilizzato quando si vuole «montare» il file system di un dispositivo (es. CD-ROM) all'interno del file system del sistema:

mount("/dev/fd0", "/mnt",0);



Le API di Windows

- Windows e UNIX differiscono per il modello di programmazione: un programma UNIX richiede dei servizi attraverso le chiamate di sistema, un programma Windows invece è guidato dagli eventi.

- ...anche Windows ha le chiamate di sistema.

- Microsoft ha definito un insieme di procedure chiamate le **API Win32** (Application Program Interface) che i programmatori possono utilizzare per ottenere i servizi del SO.

Confronto tra system call

UNIX	Win32	Descrizione
fork	CreateProcess	Crea un nuovo processo
waitpid	WaitForSingleObject	Aspetta la terminazione di un processo
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Termina esecuzione
open	CreateFile	Crea un file o apre un file esistente
close	CloseHandle	Chiude il file
read	ReadFile	Legge dati da un file
write	WriteFile	Scrivi dati in un file
lseek	SetFilePointer	Sposta il puntatore nel file
stat	GetFileAttributesEx	Recupera gli attributi del file
mkdir	Create Directory C	Crea una nuova directory
rmdir	Remove Directory	Rimuove una directory vuota
link	(none)	Win32 non supporta i collegamenti
unlink	DeleteFile	Cancella un file esistente
mount	(none)	In Win32 non esiste
umount	(none)	In Win32 non esiste
chdir	SetCurrentDirectory	Cambia la directory corrente di lavoro
chmod	(none)	Win32 non supporta la protezione (NT sì)
kill	(none)	Win32 non supporta questi segnali
time	GetlocalTime	Restituisce l'orario corrente

Struttura di un sistema operativo

- Dopo aver analizzato i sistemi operativi dal punto di vista esterno (ovvero l'interfaccia per l'utente e il programmatore), è il momento di dare un'occhiata all'interno.

- Nelle slide, esamineremo sei differenti strutture che sono state utilizzate in vari ambiti: **sistemi monolitici**, **sistemi a livelli**, **sistemi a microkernel**, **sistemi client-server**, **virtuali macchine** e **exokernels**.

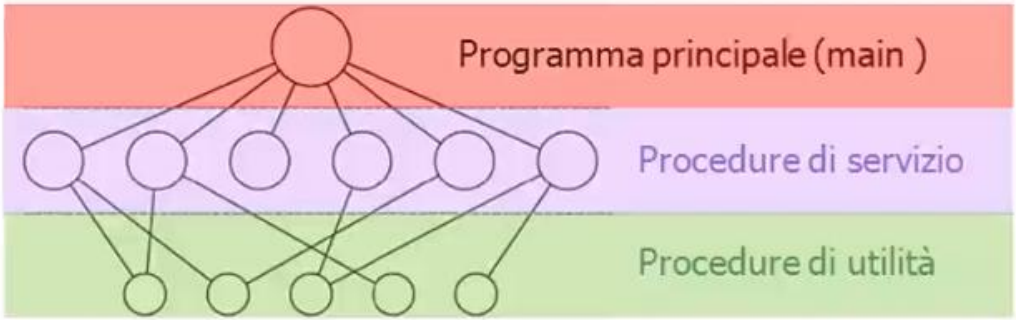
- Le strutture che seguono non costituiscono una panoramica esaustiva della realtà, ma danno un'idea di alcuni approcci utilizzati in pratica.

Sistemi monolitici

- È l' approccio più comune, il sistema operativo viene eseguito come un unico programma in modalità kernel.
- Il sistema operativo è un unico **grande programma binario eseguibile** che contiene all'interno un insieme di procedure collegate tra loro.
- Ogni procedura nel sistema è libera di chiamare qualsiasi altra in funzione del proprio compiti e del risultato che le altre possono offrire.
- La presenza di migliaia di procedure che si possono chiamare reciprocamente senza limiti rende il sistema di difficile comprensione.

Sistemi monolitici

- Questa organizzazione suggerisce una struttura di base per il sistema operativo:
  - un programma principale che richiama la procedura di servizio richiesto;
  - un insieme di procedure di servizio che svolgono le chiamate di sistema;
  - un insieme di procedure di utilità che aiutano le procedure di servizio.

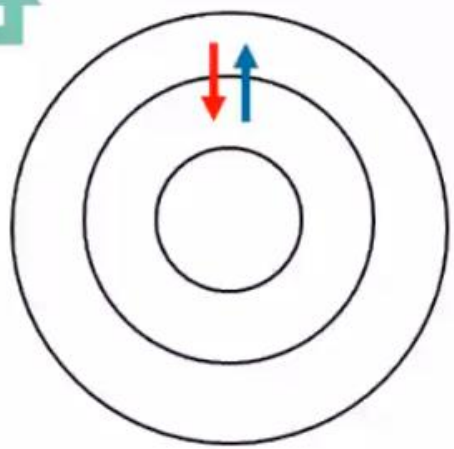


Un **modello strutturale** semplice per un sistema monolitico

Sistemi a livelli

- Una generalizzazione del modello strutturale dei sistemi monolitici è il sistema operativo **a livelli**, ciascuno costruito sopra quello sottostante.
- Il primo sistema così realizzato fu il THE (Technische Hogeschool Eindhoven) sviluppato nei Paesi Bassi da E.W. Dijkstra nel 1968 ma era ancora un unico eseguibile.
- Il MULTICS fu invece realizzato come una serie di anelli concentrici, quelli interni avevano maggior privilegi rispetto a quelli esterni.

Livello	Funzione
5	L'operatore
4	Programmi utente
3	Gestione dell'I/O
2	Comunicazione operatore-processo
1	Gestione della memoria
0	Allocazione del processore

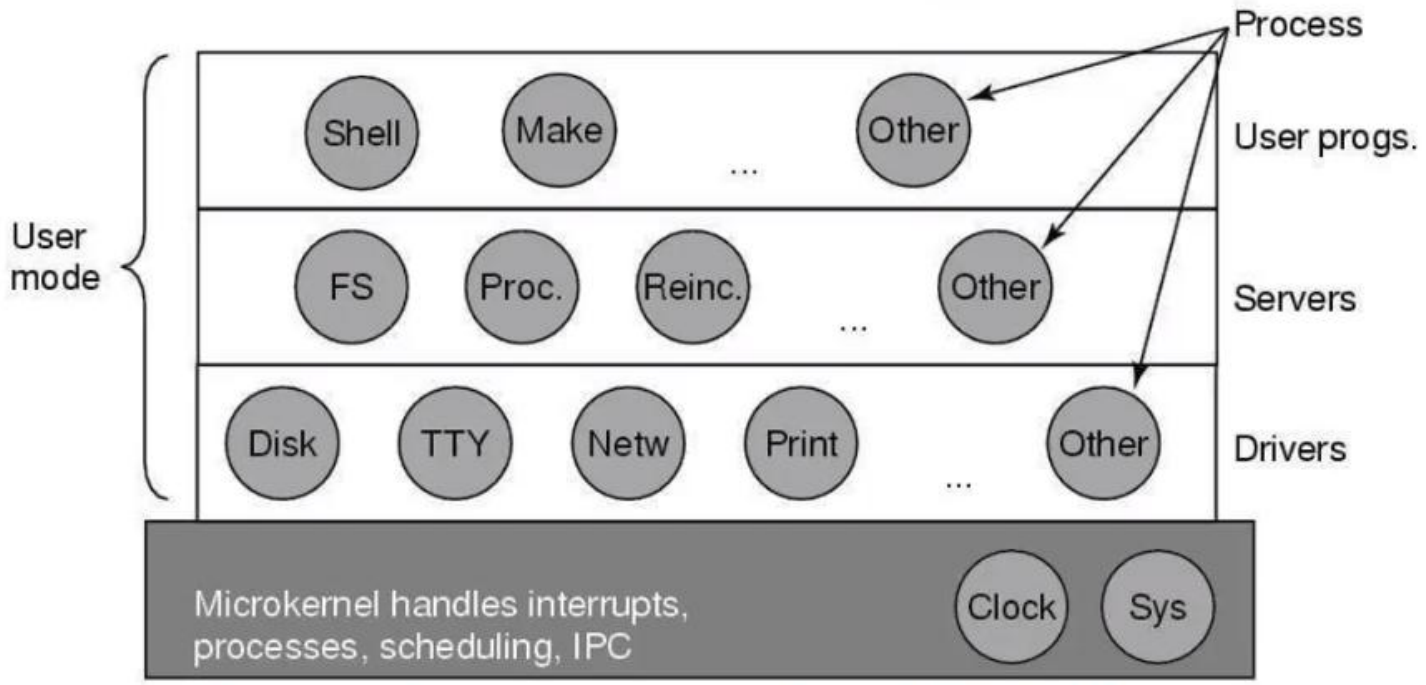


Microkernel

- Con l'approccio a strati, i progettisti potevano scegliere dove definire il confine tra kernel e utente.
- Inizialmente tutti i livelli erano nel kernel, invece bisognava ridurlo al minimo poiché un errore lì era catastrofico.
- La difettosità di un codice dipende dalla **dimensione** del modulo, dalla sua **età** e altri fattori, statisticamente 10 bug in 1000 righe di codice.
- L'idea di base della **struttura microkernel** è di incrementare l'affidabilità suddividendo il sistema operativo in piccoli moduli collaudati e di avere uno solo modulo eseguito in modalità kernel (il **microkernel**).

Microkernel

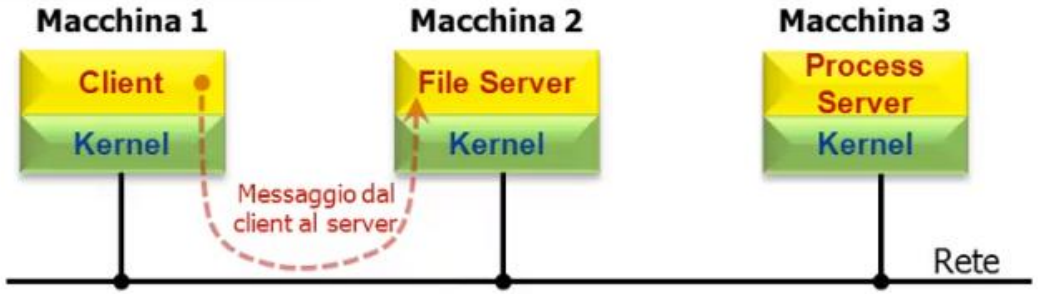
- I sistemi operativi a microkernel sono utilizzati in quelle applicazioni mission critical che hanno requisiti di **alta affidabilità** (come gli ambienti real-time).



Struttura di un sistema a microkernel (MINIX 3)

Modello client-server

- Una piccola variante all'idea microkernel è distinguere due classi di processi:
  - i **server**, ciascuno dei quali fornisce un servizio;
  - i **clienti**, che utilizzano questi servizi.
- Questo modello è noto in letteratura come modello client-server.
- Spesso il livello più basso è un microkernel, ma ciò non è necessario.
- La comunicazione tra client e server avviene spesso attraverso lo scambio di messaggi.
- Per ottenere un servizio, un processo client costruisce un messaggio indicando la propria richiesta e lo invia al servizio appropriato.



Macchine virtuali

- Il sistema di timesharing TSS/360 di IBM (1967) è stato il primo tentativo di virtualizzazione: più utenti potevano lavorare sui terminali collegati alla stessa macchina:
  - Il sistema **z/VM**, utilizzato sugli attuali mainframe IBM **zSeries**, è il discendente diretto del TSS.
- Mentre IBM ha avuto un prodotto di macchina virtuale disponibile per **quattro decenni**, l'idea della virtualizzazione è stata ignorata nel mondo dei PC/server fino a pochi anni fa.
- La virtualizzazione è divenuta popolare anche nel mondo del **web hosting**.
- Oggi anche gli utenti finali possono eseguire due o più sistemi operativi contemporaneamente sulla stessa macchina (Virtual Box, VMware,...).
- Quando Sun Microsystem (oggi Oracle) inventò il linguaggio di programmazione Java, creò anche un'architettura di computer chiamata **Java Virtual Machine**.

Exokernel

- Piuttosto che clonare una macchina reale un'altra strategia consiste nel partizionare macchina dando a ciascun utente un sottoinsieme delle risorse.

- Ad esempio si possono assegnare ad una macchina virtuale i blocchi disco da 0 a 1023, ad un'altra quelli successivi (da 1024 a 2047) e così via.

- Al livello più basso troviamo in esecuzione in modalità kernel l'**exokernel**. Il suo compito è quello di allocare le risorse alle macchine virtuali e di controllare i tentativi di impiego, in modo che ciascuna VM utilizzi le proprie risorse.

Il linguaggio C

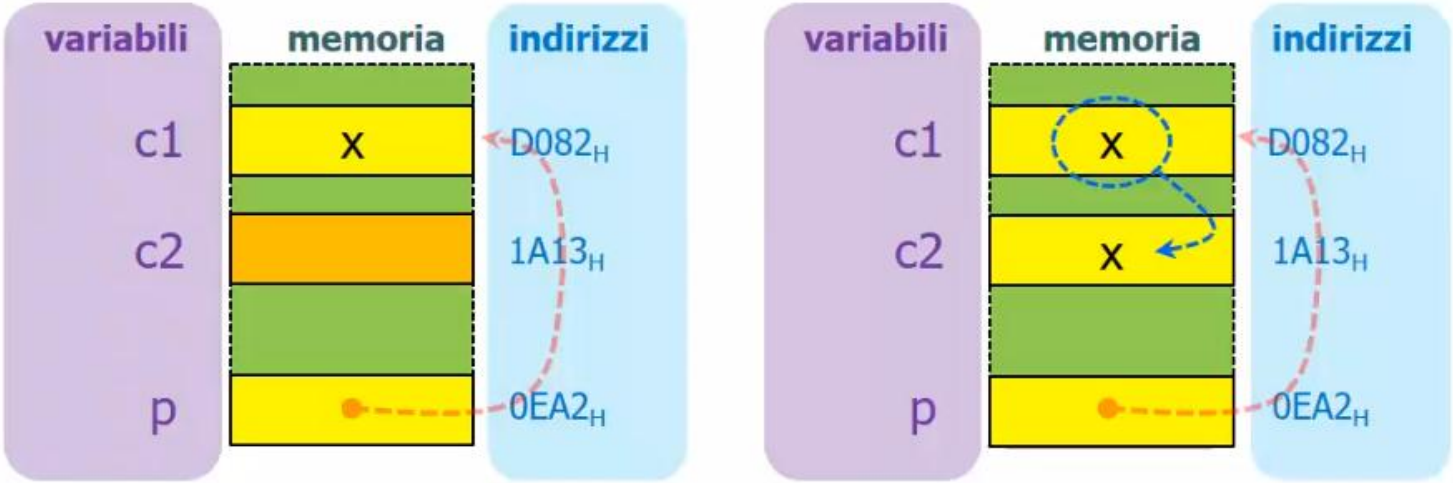
- I sistemi operativi sono normalmente scritti in C (o, talvolta, in C++).

- Un **puntatore** è una variabile che contiene l'indirizzo di memoria di un'altra variabile o l'inizio di una struttura di dati.

Il linguaggio C

- I programmatori usano l'espressione la variabile «punta a» per indicare il riferimento ad un altro oggetto, ad esempio:

```
char c1, c2, *p;// p punta/referenzia una locazione che contiene un carattere
c1 = 'x';          // in c1 è inserito il carattere 'x'
p  =  &c1;         // in p è memorizzato l'indirizzo di memoria di c1 (p punta a c1)
c2 = *p;           // in c2 è inserito il contenuto puntato da p
```



Perché utilizzare il linguaggio C per scrivere SO?

- I puntatori sono costrutti estremamente potenti ma allo stesso tempo molto delicati: non avendo controlli è facile indirizzare la memoria in modo improprio e commettere errori (motivo per il quale un programmatore java non può utilizzarli, se non implicitamente).
- La gestione della memoria in C è statica o allocata in modo esplicito (con la funzione *malloc()*) e rilasciata dal programmatore al termine del lavoro svolto (con la funzione *free()*).
- I sistemi operativi sono dei veri e propri **sistemi real-time utilizzati per scopi generali**, quindi l'esistenza di un garbage collector che entra in azione in modo arbitrario non è ammissibile.
- Inoltre è un linguaggio che ha un livello concettuale non troppo distante dalla macchina fisica.
- Tutte queste considerazioni rendono il C il linguaggio ideale per scrivere i sistemi operativi.

Il linguaggio C – header file

- I file d'intestazione (o **header**) hanno estensione **.h** e contengono: dichiarzioni, definizioni e **macro**.
- Le macro sono dichiarate attraverso la parola chiave **#define**.
- Una macro è un pezzo di codice a cui è assegnato un nome: in fase di pre-processamento è sostituito dal frammento di codice corrispondente.

```
#define BUFFER_SIZE 4096
```

- Le macro possono contenere dei parametri:

```
#define max( a, b) (a > b ? a : b)
```

- I file di instestazione possono contenere compilazioni condizionali:

```
#ifdef PENTIUM
intel_int_ack();
#end if
```

La funzione è compilata solo se è definita la macro PENTIUM.
Il codice si adatta all'architettura di destinazione.

- Un file sorgente C ha estensione **.c** e può includere uno o più file intestazione che utilizzano la direttiva **#include**.

Conclusioni

- Analizzato il sistema operativo come estenzione della macchina e come gestore delle risorse fisiche.
- Introdotto le principali astrazioni che il sistema operativo fornisce.
- Le principali architetture interne dei sistema operativi.
- Appreso perché il linguaggio C è il linguaggio ideale per scrivere sistemi operativi.