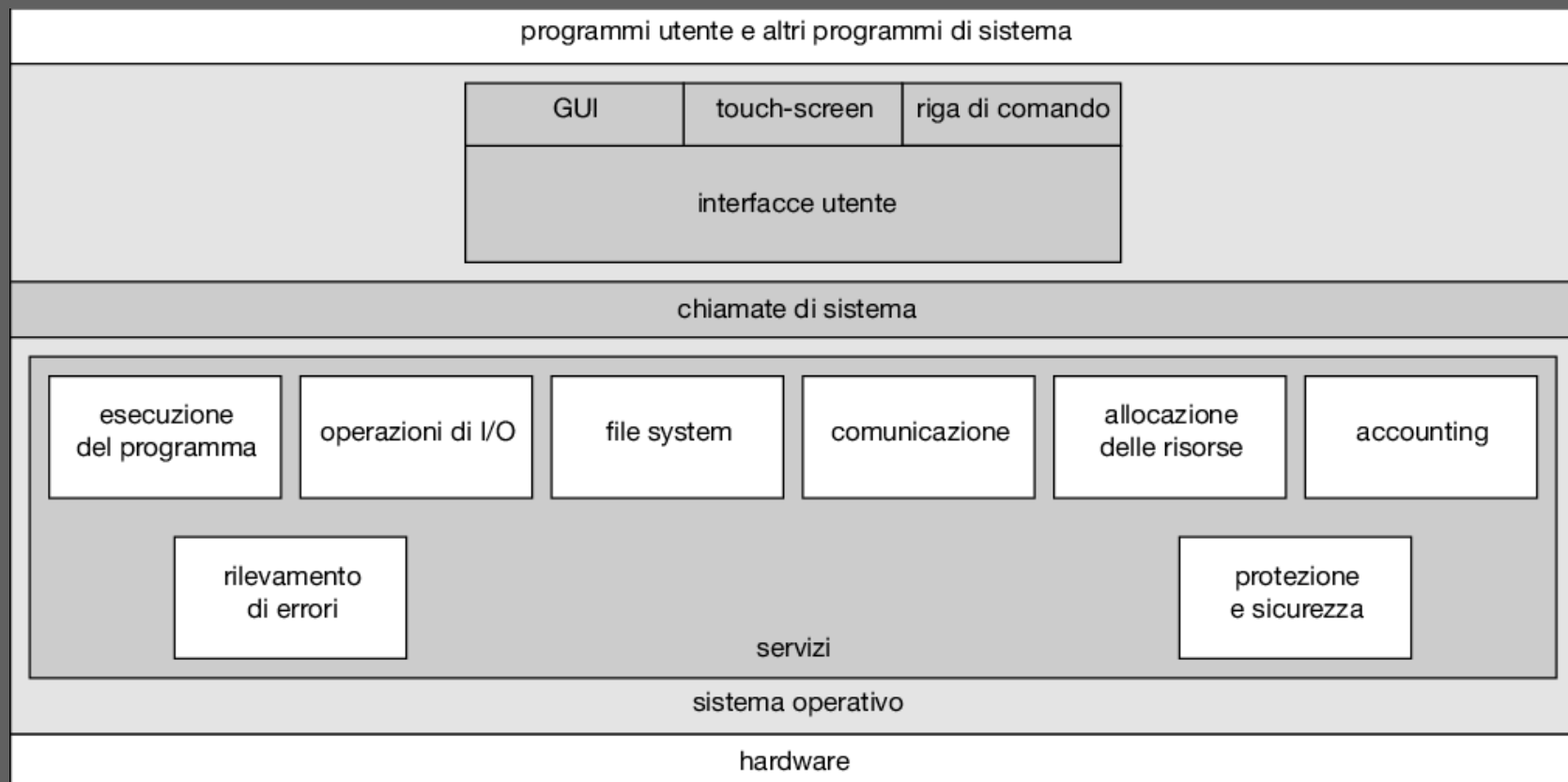


2. Strutture dei Sistemi Operativi

- Un sistema operativo mette a disposizione degli utenti (e dei loro programmi) molti servizi (figura 2.1):



2. Strutture dei Sistemi Operativi

- Alcuni di questi servizi sono completamente invisibili agli utenti, altri sono solo parzialmente visibili, e altri sono direttamente usati dagli utenti. *Ma il grado di visibilità dipende anche dal tipo di utente.* Ad esempio:

- interfaccia col sistema operativo stesso (visibile)
- Programmi/servizi di sistema (visibili)
- chiamate di sistema (quasi sempre invisibili)
- gestione dei processi, della memoria primaria e secondaria (praticamente invisibili)
- Protezione e sicurezza (parzialmente visibili)

2.2 Interfaccia col Sistema Operativo

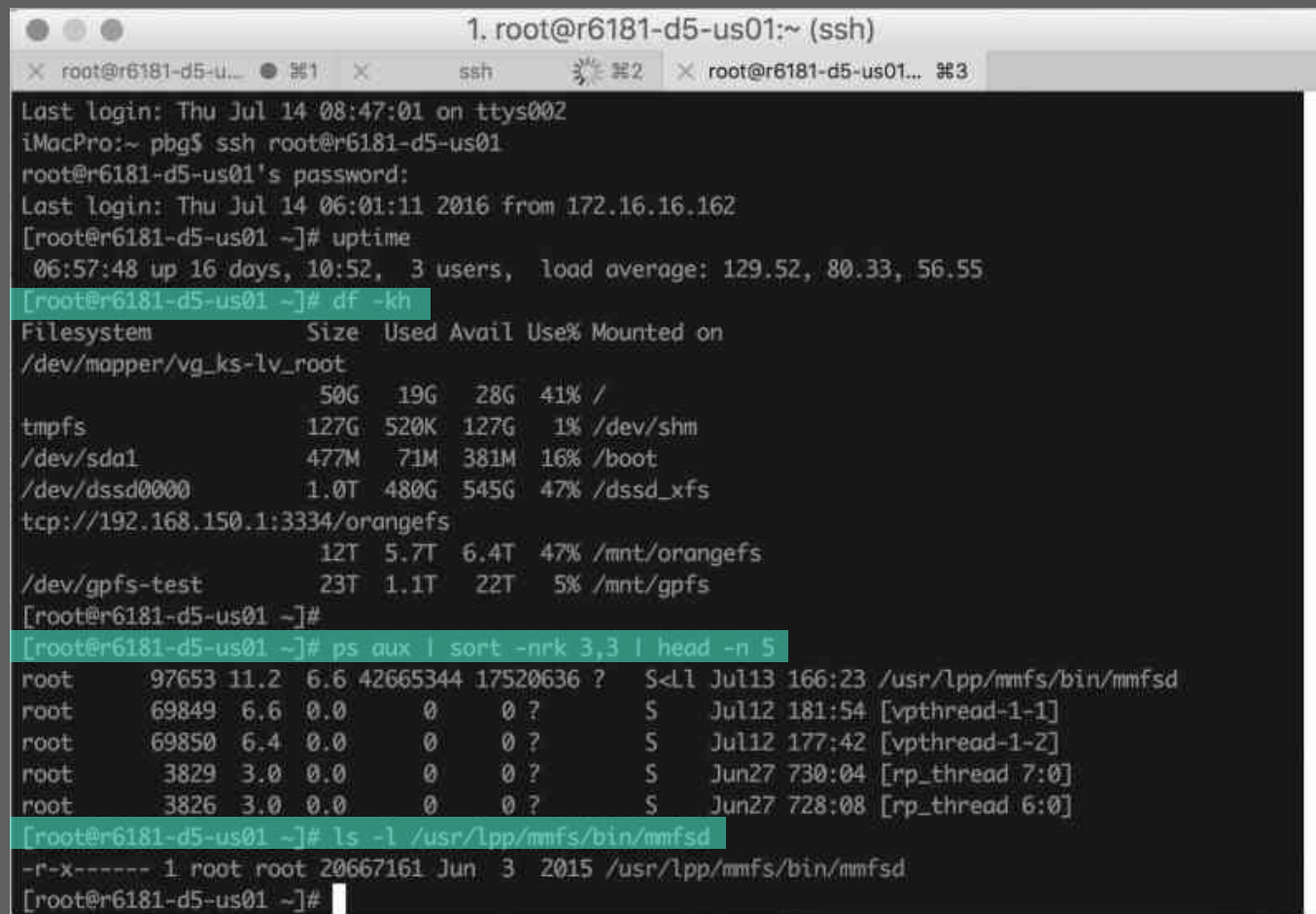
- È lo strumento con il quale gli utenti interagiscono con il SO, e ne sfruttano i servizi offerti
- Può essere un **interprete di comandi**, o un'**interfaccia grafica** con finestre e menù, ma di solito è possibile usare una combinazione di entrambi.

2.2.1 Interprete dei comandi

- Normalmente non fa parte del kernel SO, ma è un programma speciale fornito insieme al SO.
- Esempi di interprete dei comandi sono la **shell dell'Ms-Dos** e la **shell Unix**.
- una shell rimane semplicemente in attesa di ciò che l'utente scrive da linea di comando, ed esegue il comando stesso.
- spesso, i comandi che possono essere usati dagli utenti del SO sono dei normali eseguibili. L'interprete dei comandi si occupa di trovare sull'hard disk e lanciare il codice dell'eseguibile passando eventuali argomenti specificati.

2.2.1 Interprete dei comandi

- Un interprete dei comandi con alcuni comandi evidenziati (figura 2.2)



```

1. root@r6181-d5-us01:~ (ssh)
root@r6181-d5-us01's password:
Last login: Thu Jul 14 08:47:01 on ttys002
iMacPro:~ pbg$ ssh root@r6181-d5-us01
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 06:57:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem                Size      Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root   50G       19G   28G   41% /
tmpfs                     127G      520K   127G    1% /dev/shm
/dev/sda1                  477M       71M   381M   16% /boot
/dev/dssd0000              1.0T     480G   545G   47% /dssd_xfs
tcp://192.168.150.1:3334/orangefs 12T     5.7T   6.4T   47% /mnt/orangefs
/dev/gpfs-test             23T      1.1T   22T    5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root      97653 11.2  6.6 42665344 17520636 ?    S<  Ll  Jul13 166:23 /usr/lpp/mmfs/bin/mmfsd
root      69849  6.6  0.0      0      0 ?        S    Jul12 181:54 [vpthread-1-1]
root      69850  6.4  0.0      0      0 ?        S    Jul12 177:42 [vpthread-1-2]
root       3829  3.0  0.0      0      0 ?        S    Jun27 730:04 [rp_thread 7:0]
root       3826  3.0  0.0      0      0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ls -l /usr/lpp/mmfs/bin/mmfsd
-r-x----- 1 root root 20667161 Jun  3  2015 /usr/lpp/mmfs/bin/mmfsd
[root@r6181-d5-us01 ~]#
  
```

2.2.1 Interprete dei comandi

- ad esempio, in un sistema operativo Unix-like, se da linea di comando l'utente scrive:

```
$> rm myfile
```

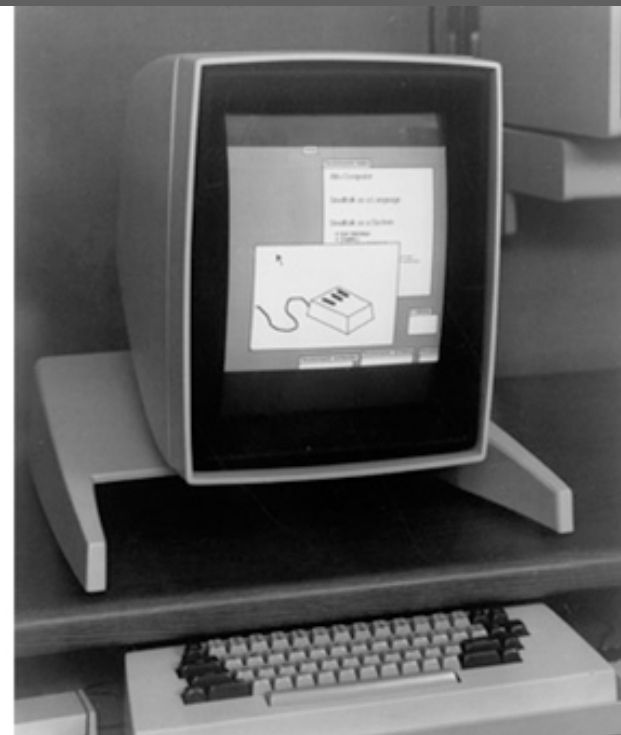
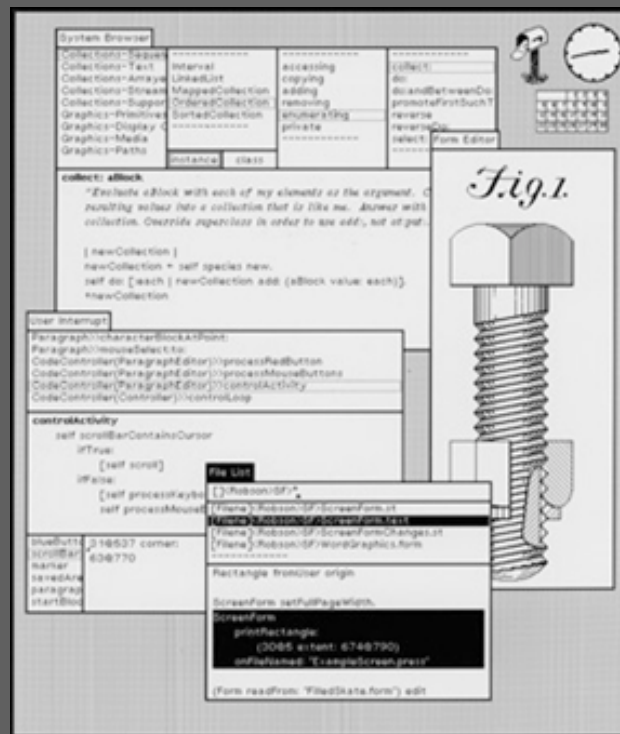
- l'interprete cerca un file eseguibile di nome “rm” e lo lancia, passandogli come parametro “myfile”
- Notate che possiamo estendere il set di comandi disponibili, semplicemente aggiungendo nuovi eseguibili...
- in Unix, sono a disposizione più shell, con funzionalità simili fra loro: shell, c-shell, bourne-shell, bourne-again-shell, korn-shell, bash-shell,...

2.2.2 Interfaccia grafica

- I moderni SO offrono anche una interfaccia grafica (GUI) per gli utenti, spesso più facile da imparare ed usare.
- Le GUI sono state inventate dalla Xerox a Palo Alto (in California, nella *Silicon Valley*) nel 1973, ma la prima diffusione commerciale di successo si deve al SO Macintosh del 1984.
- Windows nasce qualche anno dopo come GUI da lanciare sopra Ms-Dos, e solo nel 1995 diventa parte del SO.
- Unix offre varie interfacce grafiche, sia proprietarie che open-source, come **KDE** e **GNOME**, e ogni utente del SO può scegliersi quella che preferisce (o anche nessuna...)

2.2.2 Interfaccia grafica

- In questa figura l'aspetto della prima interfaccia grafica, e il computer "Alto" della Xerox, su cui era implementata.



2.4 Programmi/servizi di Sistema

- Non fanno parte del kernel del SO, ma vengono forniti insieme al SO, e rendono più facile, comodo e conveniente l'uso del Sistema.
- E' di solito facile arricchire una data installazione di un SO con nuovi programmi di sistema, mentre non è banale modificare e installare un nuovo kernel...
- Gli **interpreti dei comandi** e le **interfacce grafiche** sono gli esempi più evidenti di programmi di sistema, ma ve ne sono di molti altri tipi:

2.4 Programmi/servizi di Sistema

- per modificare i file: ossia gli **editor** di vario tipo
- per programmare: ossia **compilatori, assembleri, debugger, interpreti, ambienti di sviluppo**
- per comunicare: ossia i programmi di **posta elettronica e di navigazione in rete (browser)**
- per monitorare lo stato del sistema: come il task manager di Windows

2.3 Chiamate di sistema (System Call)

- Conveniamo, d'ora in poi, di **chiamare processo un programma in “esecuzione”** (notate le virgolette).
- Qualsiasi (quasi) cosa venga **fatta in un computer, avviene attraverso un processo.**
- Come abbiamo già osservato, quando un processo utente deve compiere qualche operazione delicata, deve chiedere aiuto al SO, attraverso una **Chiamata di Sistema, o System Call**
- In un sistema con più utenti (e quindi con più processi attivi contemporaneamente), ogni secondo possono verificarsi centinaia o migliaia di system call

2.3 Chiamate di sistema (System Call)

- Le system call costituiscono la vera e propria interfaccia tra i processi degli utenti e il Sistema Operativo
- Ad esempio, in Unix assumono la forma di procedure che possono essere inserite direttamente in programmi scritti con linguaggi ad alto livello (C, C++, ...)
- Sembra di usare una subroutine, ma l'esecuzione della system call trasferisce il controllo al SO, e in particolare alla porzione di codice del SO che implementa la particolare System Call invocata.

2.3 Chiamate di sistema (System Call)

- Ad esempio, in un programma C, per scrivere dentro ad un file (nel SO Unix):

```
Int fd, i;
```

```
fd = open("nomefile", O_WRONLY);
```

```
i = write(fd, "ciao!", sizeof("ciao!"));
```



```
close(fd);
```

- Esercizio: provate a dire di quali cose si deve occupare il codice della system call "open"

2.3.2 Chiamate di sistema: le “API”

- I programmatori possono anche non usare direttamente le system call, ma una **interfaccia per la programmazione di applicazioni**
- Ossia **le API** (**A**pplication **P**rogramming **I**nterface)
- le API non sono altro che uno *strato intermedio* tra le **applicazioni sviluppate dai programmatori e le system call, per rendere più facile l'uso e migliorare la portabilità** delle applicazioni.
- Naturalmente esistono API per i vari sistemi operativi: **API Windows** per i sistemi Windows e **API POSIX** per le varie versioni di Unix, Linux e Mac OS X.

2.3.2 Chiamate di sistema: le “API”

- Quindi, le funzioni API **invocano a loro volta opportune system call e** spesso vi è una corrispondenza diretta tra una funzione API ed una corrispondente system call
- Ad esempio, la libreria C dell’ambiente Unix è una semplice forma di API. In questa libreria esiste la funzione per aprire un file:

```
FILE *fp;  
fp = fopen("myfile", "W");           // usa la system call "open"  
fprintf(fp, "ciao");                 //usa la system call "write"  
fclose(fp);                         //usa la system call "close"
```

2.3.3 Categorie di system call

- Esistono tipi di System Call per ogni possibile operazione, ecco alcuni esempi per Windows e Unix (figura 2.8a):

	Windows	UNIX
Controllo dei processi	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
Gestione dei file	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Gestione dei dispositivi	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Gestione delle informazioni	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Comunicazione	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protezione	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Gestione dei processi

17

- In un dato istante, all'interno di un SO sono attivi più processi (anche se uno solo è in esecuzione, in un dato istante)
- si parla allora di **Processi Concorrenti**, perché questi processi competono per usare le risorse hardware della macchina:
 - la CPU, innanzi tutto,
 - lo spazio in memoria primaria e secondaria
 - i dispositivi di input e output
- Il SO ha la responsabilità di fare in modo che ogni processo abbia la sua parte di risorse, senza danneggiare e venire danneggiato dagli altri processi.

Gestione dei processi

18

responsabilità SO

- Il SO deve quindi gestire tutti gli aspetti che riguardano la vita dei processi:
 - **Creazione** di nuovi processi e **cancellazione** dei processi la cui esecuzione è terminata
 - **Sospensione** temporanea dell'esecuzione, e **riavvio** dei processi
 - Fornire meccanismi per la **sincronizzazione** tra i processi (ad esempio, un processo deve potersi fermare per aspettare che un altro processo gli fornisca una informazione)
 - Fornire meccanismi per la **comunicazione** tra processi

Gestione della memoria primaria

- Per eseguire un programma questo deve essere caricato in **memoria primaria** (correggeremo questa affermazione quando parleremo della Memoria Virtuale).
- In un sistema **a time-sharing, più processi** possono essere contemporaneamente attivi: **il loro codice** e i loro dati sono caricati in qualche area della **RAM**. Il SO deve quindi:
 - tenere traccia di quali **i parti della RAM** sono utilizzate e da quale processo
 - decidere come la RAM **va distribuita** fra i vari processi
 - Gestire la RAM in base alle necessità e all'evoluzione dello stato della computazione dei vari processi.

Gestione dei file e del file system

- Quasi ogni informazione presente in un sistema è contenuta **in un file**: una raccolta di informazioni denotata da un nome (e di solito da altre proprietà)
- I file sono organizzati in una struttura gerarchica detta **File System**, mediante le **cartelle** (o **directory**, o **folder**)
- Il SO e' responsabile della:
 - **Creazione e cancellazione** di file e directory
 - Fornitura di **strumenti per gestire file e directory**
 - Memorizzazione efficiente del file system in memoria secondaria

Gestione della memoria secondaria

- I file sono memorizzati permanentemente in memoria secondaria: di solito implementata su un hard disk
- Il SO deve:
 - decidere dove e come memorizzare i file su disco, ed essere in grado di ritrovarli in modo veloce, quando questi devono essere caricati in RAM
 - Trovare velocemente spazio libero quando un file viene creato o la sua dimensione aumenta, e recuperare lo spazio che si è liberato alla rimozione di un file
 - Gestire in modo efficiente accessi concorrenti ai file da parte dei vari processi correntemente attivi.

Gestione di un sistema di protezione

- Il SO deve fare in modo che:
- Ogni processo sia protetto dalle attività improprie degli altri processi, cioè:
 - nessun processo deve potersi impadronire di una qualsiasi risorsa in modo esclusivo, e/o per un tempo eccessivo
 - nessun processo deve poter accedere alle aree di codice e dati di un altro processo.
- In un sistema multi-utente, nessun utente può accedere (leggere, spostare, modificare, cancellare) i file di un altro utente, se non secondo quanto stabilito dall'utente che “possiede” i file.

Macchine Virtuali

- Un moderno SO trasforma una macchina reale in una sorta di **macchina virtuale (MV)**:
 - Questo permette all'utente di usare la MV **indipendentemente dall'hardware sottostante**
 - L'utente sembra avere a disposizione una CPU, un File System e altre risorse private, anche se in realtà sono condivise con altri utenti e con il SO stesso
 - la portabilità delle applicazioni tra macchine diverse aumenta (di quanto, dipende dal tipo di SO, dal tipo di applicazione e dall'ambiente/linguaggio in cui è stata sviluppata)

Per chi vuole approfondire:

- Sezione 2.6: perché le applicazioni dipendono dal sistema operativo
- Sezione 2.8: struttura del sistema operativo