Perché unix (al posto di Windows e MacOS)

- Unix nasce come sistema informatico eterogeneo (in software e hardware)
- Ha un'interfaccia d'uso naturalmente predisposta all'utilizzo in network
- E' uno standard *de-facto* nel mondo scientifico
- Esiste una grande quantità di software di alto livello

Utilizzo di un sistema Unix

Unix è un sistema operativo multiutente e multitasking; un sistema unix è composto da calcolatori interconnessi fra loro da software che permettono di condividere, in modo trasparente all'utente, spazio disco e file amministrativi.

Vantaggi:
accesso a risorse differenziate e servizi usufruibili in un ambiente omogeneo
- Hardware per il calcolo intensivo (CPU+RAM)
- Distribuzione software
- Servizi di backup
- Servizi di code

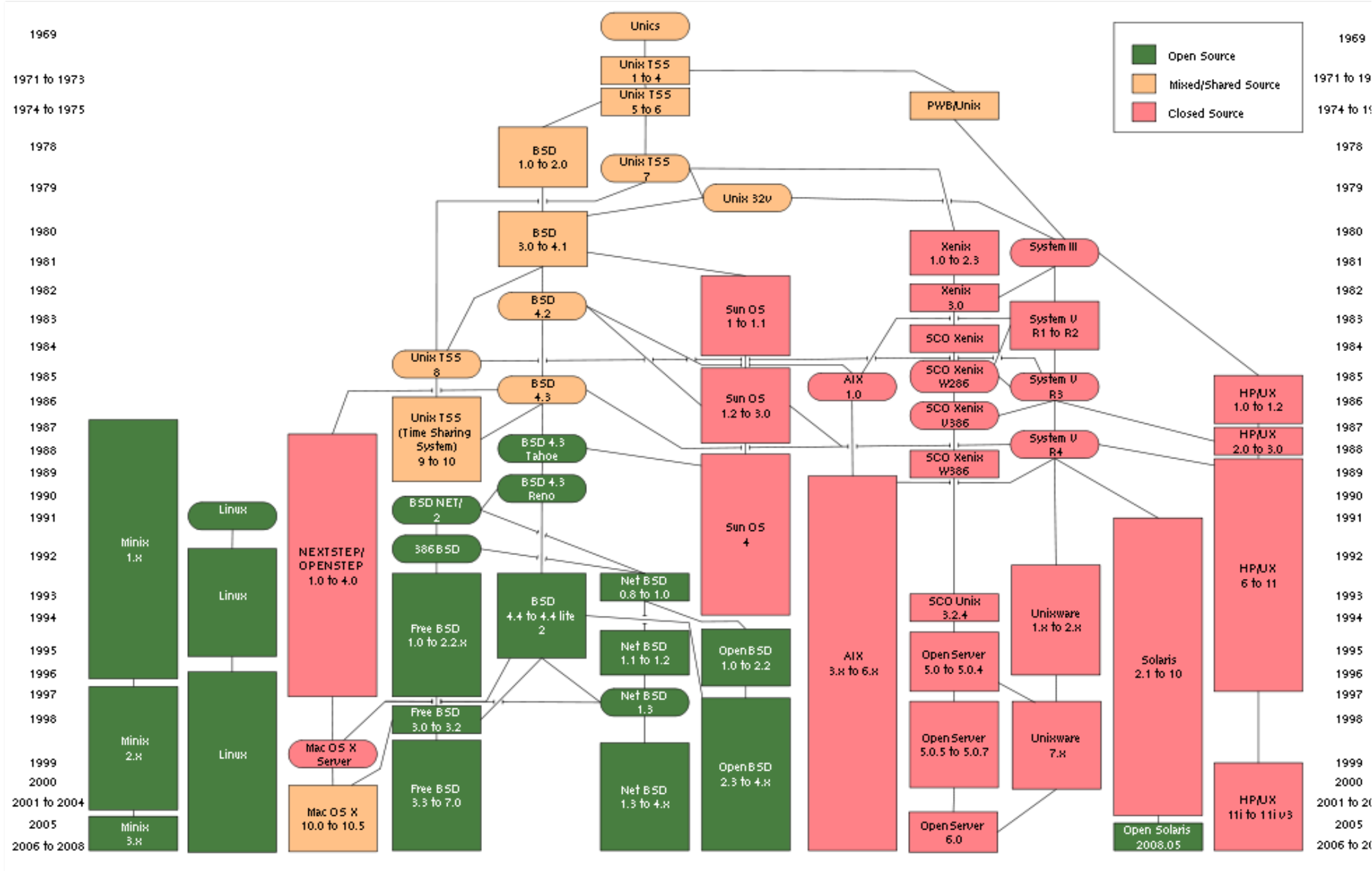
Svantaggi:
tutte le risorse sono condivise --> inevitabili limiti (spazio disco, cpu, ...)
Necessita di una etichetta nell'utilizzo delle risorse (volontaria o forzata)

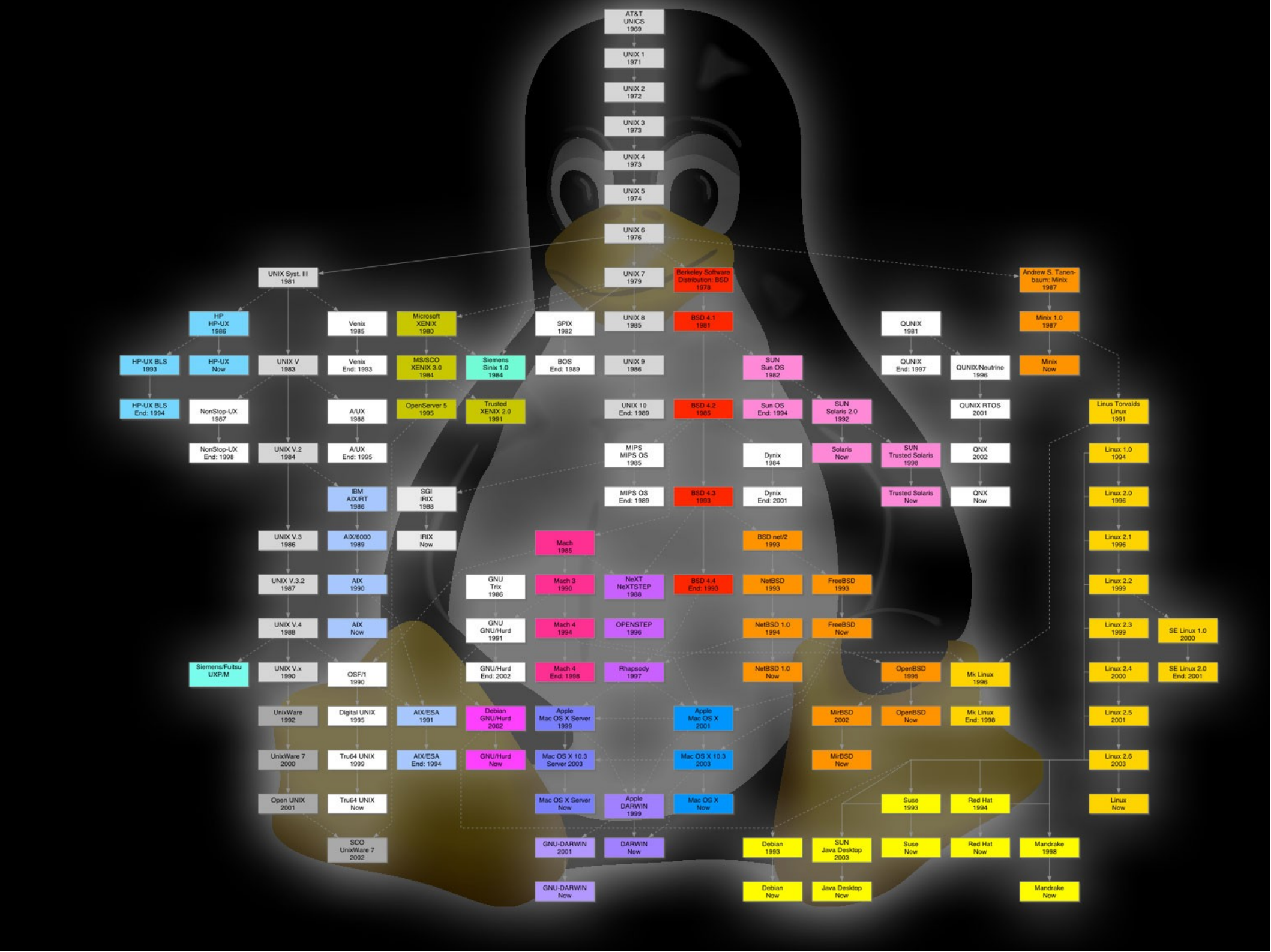
UNIX©

UNIX è un marchio registrato di proprietà della “The Open Group”, http://www.unix.org

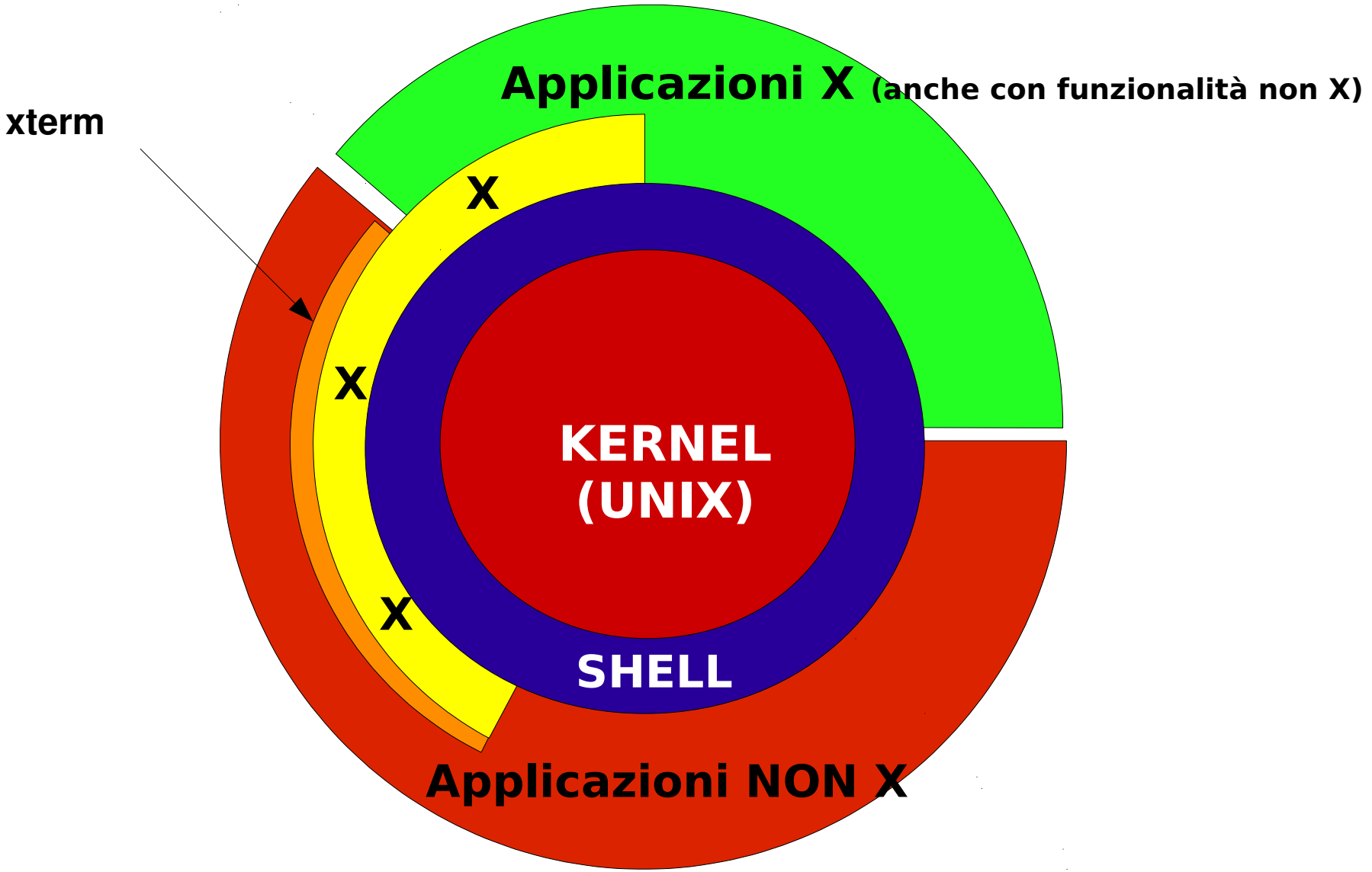
E' anche una serie di specifiche pubbliche che descrivono un sistema operativo (the single Unix specification)

Dal punto di vista pratico, oggi non esiste un sistema operativo di nome “unix”, ma esistono alcune decine di sistemi operativi conformi alla single unix specification: AIX(IBM), IRIX(SGI), HP-UX(HP), Solaris(SUN), TrueUnix(CompaQ-HP), GNU-Linux, ...


1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2008
Open Source
Mixed/Shared Source
Closed Source
Unics
Unix TSS 1 to 4
Unix TSS 5 to 6
Unix TSS 7
PWB/Unix
BSD 1.0 to 2.0
Unix 32v
BSD 3.0 to 4.1
BSD 4.2
Unix TSS 8
BSD 4.3
Unix TSS (Time Sharing System) 9 to 10
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD NET/ 2
386BSD
Free BSD 1.0 to 2.2.x
Free BSD 3.0 to 3.2
Free BSD 3.3 to 7.0
BSD 4.4 to 4.4 lite 2
Net BSD 0.8 to 1.0
Net BSD 1.1 to 1.2
Net BSD 1.3
Net BSD 1.3 to 4.x
Open BSD 1.0 to 2.2
Open BSD 2.3 to 4.x
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
AIX 1.0
AIX 3.x to 6.x
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix V286
SCO Xenix V386
SCO Xenix V386
SCO Unix 3.2.4
Open Server 5.0 to 5.0.4
Open Server 5.0.5 to 5.0.7
Open Server 6.0
System III
System V R1 to R2
System V R3
System V R4
Unixware 1.x to 2.x
Unixware 7.x
Solaris 2.1 to 10
Open Solaris 2008.05
HP/UX 1.0 to 1.2
HP/UX 2.0 to 3.0
HP/UX 6 to 11
HP/UX 11i to 11i v3
Minix 1.x
Minix 2.x
Minix 3.x
Linux
Linux
Linux
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.5


AT&T UNICS 1969
UNIX 1 1971
UNIX 2 1972
UNIX 3 1973
UNIX 4 1973
UNIX 5 1974
UNIX 6 1976
UNIX 7 1979
UNIX 8 1985
UNIX 9 1986
UNIX 10 End: 1989
MIPS MIPS OS 1985
MIPS OS End: 1989
UNIX Syst. III 1981
Berkeley Software Distribution: BSD 1978
BSD 4.1 1981
BSD 4.2 1985
BSD 4.3 1993
BSD 4.4 End: 1993
Andrew S. Tanen-baum: Minix 1987
Minix 1.0 1987
Minix Now
HP HP-UX 1986
HP-UX BLS 1993
HP-UX Now
HP-UX BLS End: 1994
Venix 1985
Venix End: 1993
Microsoft XENIX 1980
MS/SCO XENIX 3.0 1984
OpenServer 5 1995
Siemens Sinix 1.0 1984
Trusted XENIX 2.0 1991
SPIX 1982
BOS End: 1989
UNIX V 1983
NonStop-UX 1987
NonStop-UX End: 1998
A/UX 1988
A/UX End: 1995
UNIX V.2 1984
IBM AIX/RT 1986
SGI IRIX 1988
UNIX V.3 1986
AIX/6000 1989
IRIX Now
UNIX V.3.2 1987
AIX 1990
AIX Now
UNIX V.4 1988
Siemens/Fuitsu UXP/M
UNIX V.x 1990
OSF/1 1990
UnixWare 1992
UnixWare 7 2000
Open UNIX 2001
Digital UNIX 1995
Tru64 UNIX 1999
Tru64 UNIX Now
SCO UnixWare 7 2002
AIX/ESA 1991
AIX/ESA End: 1994
GNU Trix 1986
GNU GNU/Hurd 1991
GNU/Hurd End: 2002
Debian GNU/Hurd 2002
GNU/Hurd Now
Mach 1985
Mach 3 1990
Mach 4 1994
Mach 4 End: 1998
Apple Mac OS X Server 1999
Mac OS X 10.3 Server 2003
Mac OS X Server Now
GNU-DARWIN 2001
GNU-DARWIN Now
NeXT NeXTSTEP 1988
OPENSTEP 1996
Rhapsody 1997
Apple DARWIN 1999
DARWIN Now
Apple Mac OS X 2001
Mac OS X 10.3 2003
Mac OS X Now
SUN Sun OS 1982
Sun OS End: 1994
Dynix 1984
Dynix End: 2001
BSD net/2 1993
NetBSD 1993
NetBSD 1.0 1994
NetBSD 1.0 Now
FreeBSD 1993
FreeBSD Now
MirBSD 2002
MirBSD Now
SUN Solaris 2.0 1992
Solaris Now
SUN Trusted Solaris 1998
Trusted Solaris Now
SUN Java Desktop 2003
Java Desktop Now
Debian 1993
Debian Now
QUNIX 1981
QUNIX End: 1997
QUNIX/Neutrino 1996
QUNIX RTOS 2001
QNX 2002
QNX Now
OpenBSD 1995
OpenBSD Now
Suse 1993
Suse Now
Red Hat 1994
Red Hat Now
Mandrake 1998
Mandrake Now
Linus Torvalds Linux 1991
Linux 1.0 1994
Linux 2.0 1996
Linux 2.1 1996
Linux 2.2 1999
Linux 2.3 1999
Linux 2.4 2000
Linux 2.5 2001
Linux 2.6 2003
Linux Now
SE Linux 1.0 2000
SE Linux 2.0 End: 2001
Mk Linux 1996
Mk Linux End: 1998

Struttura UNIX



Linux©

Linux è un marchio registrato di proprietà di Linus Torvalds, http://www.kernel.org

E' un sistema operativo che “punta” alla conformità con la single Unix specification: in questo senso, Linux è un sistema operativo “unix”. Linux è distribuito sotto licenza GPL

Linux è un kernel; per il suo utilizzo viene completato dalle applicazioni utente, prevalentemente sviluppate da GNU. Tutto il software GNU è distribuito sotto licenza GPL

Distribuzioni linux

Una distribuzione è l'unione di un kernel linux e di una serie di programmi che interagiscono con il kernel, per le più svariate applicazioni e funzionalità.

Esistono distribuzioni mirate a particolari applicazioni (ad es. i linux embedded), oppure a particolari funzioni (ad es. i linux per firewall), oppure generaliste, che realizzano un ambiente di lavoro vario e con un numero di applicazioni potenzialmente infinito.

Distribuzioni linux vs unix

Le distribuzioni linux possono nascere per gemmazione; col tempo si sono quindi create famiglie di distribuzioni:

RedHat (Fedora, Mandriva, ...)
Debian (Ubuntu, ...)
Suse

Gli altri unix, invece, hanno mantenuto una compattezza di distribuzione, e vengono distribuiti come prodotti singoli:
SUN Solaris, SGI IRIX, IBM AIX, HP-UX, ...

UNIX: le identità informatiche

Utente: definito dalla coppia login+password

gruppo: unione di utenti

identità informatica: l'unione utente+gruppo primario+eventuali gruppi secondari

Ogni file sul sistema è associato ad un'identità
Ogni azione del sistema (processo) è associata ad un'identità

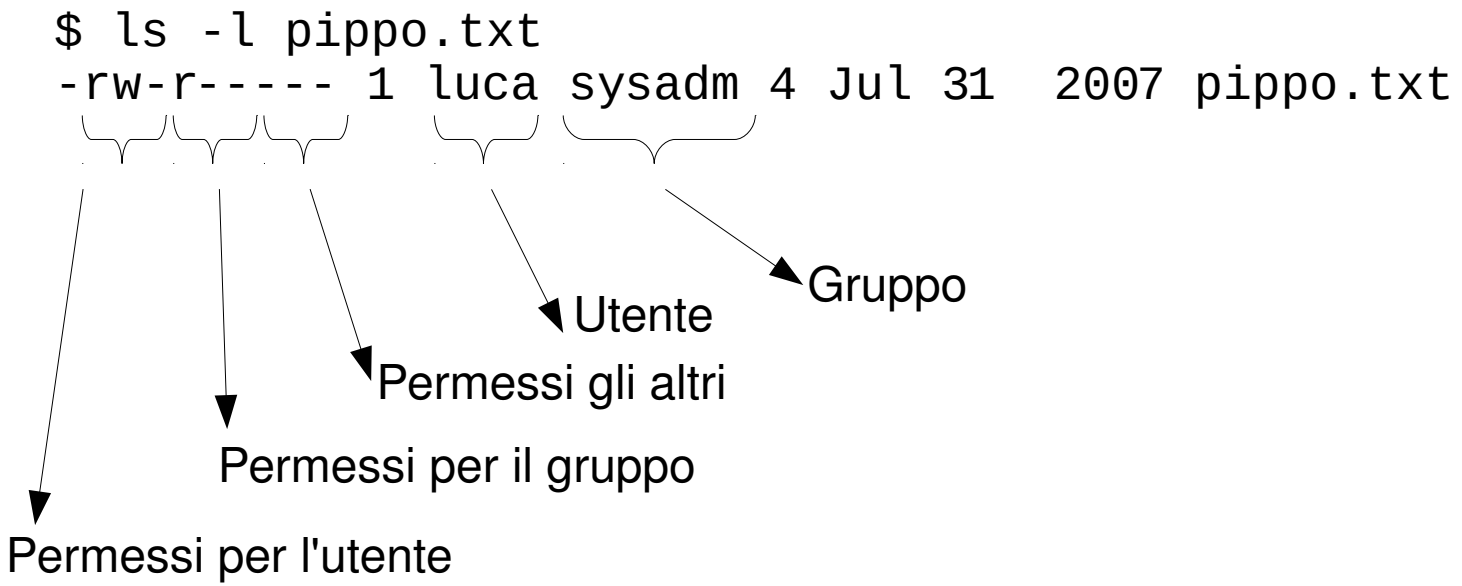
Tutte le identità sono equivalenti, tranne *root* che è l'unica che può porsi al di sopra delle altre (nel senso che vedremo...).

Privacy dei dati

Questo è un problema poco sentito ma estremamente importante:
la sicurezza (integrità) e la privacy dei dati è interamente sotto il controllo dell'identità informatica che li possiede

Ricordiamo che per un'identità unix esistono 4 categorie di utenti:
1) Il superuser
2) l'utente stesso
2) gli utenti che appartengono al suo stesso gruppo
3) tutti gli altri

Dal superuser non c'è modo di difendersi! Per tutti gli altri, si può fare molto!



r = permesso di lettura
w = permesso di modifica-cancellazione
x = permesso di esecuzione (o di accesso, per directory)

u = user
g = gruppo
o = others
a = all

\+ => concessione diritto
\- => negazione diritto

Gestione dei diritti sui file

Metodo 007 – è un metodo assoluto

il comando chmod accetta una tripletta di cifre 0-7 a indicare la tripletta di numeri binari che definisce l'insieme dei diritti (1 diritto concesso, 0 diritto negato). Esempio

```
-rw-r----- 1 luca sysadm 4 Jul 31  2007 pippo.txt
 110100000
```

La tripletta è 110, 100, 000 ossia 6, 4, 0
e il comando per definire questi diritti è

```
chmod 640 pippo.txt
```

Metodo UGO – è un metodo relativo

il comando chmod accetta l'espressione <a_chi><tolgo/concedo><diritto> con la notazione precedentemente introdotta. Esempio

```
chmod ug+rw pippo.txt
```

concede diritto di scrittura e lettura all'utente e al suo gruppo (ma non modifica niente su tutti gli altri)

Definizione del default – la user mask

umask determina il default con cui vengono creati file e directory.
In genere viene eseguito nel file di avvio della propria shell (.bashrc, ...)

il suo argomento è il duale della tripletta di diritti sulle directory
(ossia si indicano i permessi che si vogliono negare)

Es:

```
$ umask 027
$ mkdir pluto
$ touch pippo.txt
$ ls -l
...
drwxr-x---  2 luca sysadm    4096 Mar 16 17:04 pluto
-rw-r-----  1 luca sysadm       0 Mar 16 17:05 pippo.txt
```

Il primo programma: la shell

La shell è il programma di interfaccia fra utilizzatore e kernel.

E' un linguaggio di programmazione interpretato che si compone di diverse decine di comandi di alto livello, assieme agli usuali costrutti dei linguaggi (loop, cicli condizionali).

E' possibile definire variabili: molte di esse sono predefinite al momento dell'avvio (variabili d'ambiente)

http://www.pluto.it/files/ildp/guide/abs/index.html

Il primo programma: la shell

Esistono diverse shell:

sh --> Bourne shell (sempre presente)

bash --> Bourne again shell

ksh --> Korn shell

csh --> C shell

dash --> Debian Almquist shell (Debian, Ubuntu, ...)

Primi comandi: chi sono, dove sono, con chi sono, come chiedo aiuto

Chi sono:
`whoami` restituisce il login name

Dove sono:
`hostname` restituisce il nome del computer

Con chi sono:
`who` mi dice chi è collegato assieme a me

Primi comandi: chi sono, dove sono, con chi sono, come chiedo aiuto

Quale sistema operativo è attivo:
uname restituisce sistema operativo e tipo di cpu

Aiuto!
man <nome_comando> accede alla manualistica
info <nome_comando> alternativo sistema di manualistica

La shell: caratteristiche base

PATH è la lista di cartelle in cui il sistema va a cercare i comandi (viene eseguita la prima istanza trovata)

Visualizzazione
`echo $PATH`

Modifica
`export PATH=$PATH:/mia_cartella` (sh,bash,ksh,...)
`setenv PATH $PATH:/mia_cartella` (C shell)

Dove si trova il comando che invoco?

```
which ls
/usr/bin/ls
```

La shell: caratteristiche base

Gli alias sono comandi composti e personalizzati

```
alias luca='ls –lrt '
```

per visualizzarli:

```
alias
```

E' possibile ridefinire un comando

```
alias ls='ls -lrt'
```

Per recuperare il comando originale è possibile usare il PATH completo

```
/bin/ls
```

Per eliminare un alias:

```
unalias <alias>
```

La shell: caratteristiche base

Tutte le shell utilizzano dei file di configurazione, letti all'avvio della shell stessa.

Es. bash:

.bash_profile .bash_login .profile (shell di login)

.bashrc (shell non di login)

.bash_logout

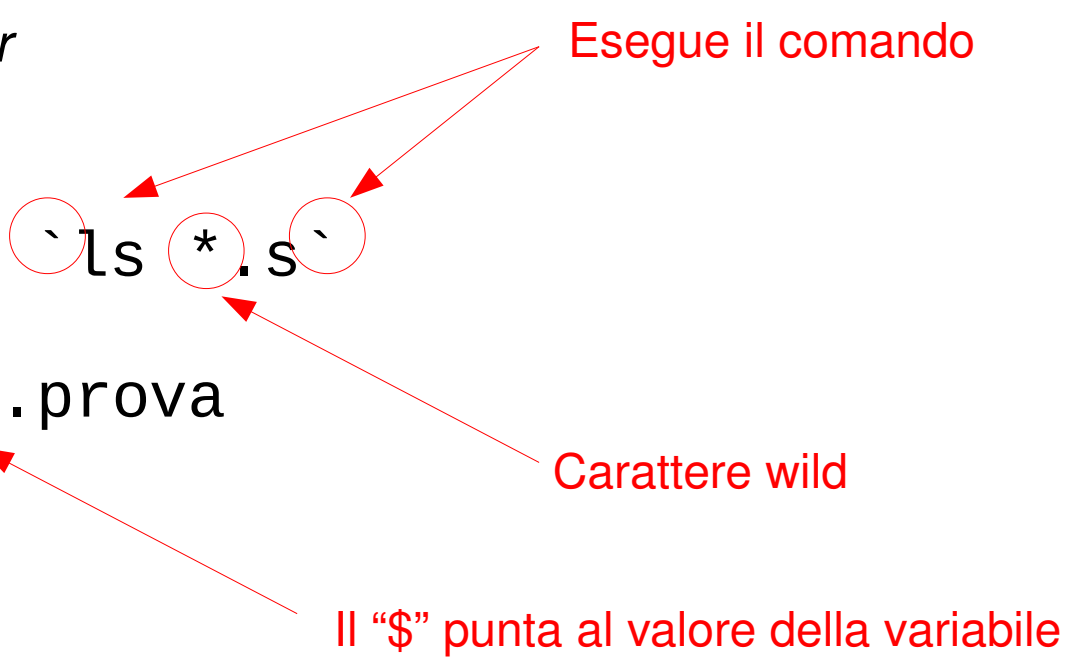
In modo analogo le altre shell
.kshrc ...
.cshrc ...
...

Il ciclo *for*

for *nome_var* in *lista*
do
istruzioni $nome_var
done

Es.

```
for i in `ls *.s`
do
cp $i $i.prova
done
```



Il controllo *if*

if espressione
then
comando
else
comando2
fi

Es.

```
if test -f pippo.txt
then
echo 'Il file pippo esiste'
else
touch pippo.txt
echo 'Ho creato il file pippo.txt'
fi
```

Esegui un test su file o cartelle

Stampa a video

Crea un file (o ne aggiorna la data)

Il controllo *case*

case *$nome_var* in
*) comando ;;
*) comando ;;
esac

Es.

```
for i in `ls *.*`
do
case $i in
    *.txt) echo $i 'sembra un file di testo' ;;
    *.avi) echo $i 'sembra un file video formato AVI' ;;
    *.mpg) echo $i 'sembra un file formato mpeg' ;;
    *.doc) echo $i 'sembra un file di Microsoft Word' ;;
    *)     echo 'Non riconosco il formato di' $i ;;
esac
done
```

STDIN - STDOUT – STDERR

Unix definisce un “device” standard per il passaggio di parametri al processo: lo Standard Input (STDIN) è per default la tastiera.

Esiste anche un “device” standard per l'output dei dati
Standard Output - STDOUT; di default è lo schermo. Il device è bufferizzato

Esiste inoltre un “device” per i messaggi d'errore – Standard Error, STDERR
Non è bufferizzato

Le istruzioni di I/O in ogni linguaggio hanno la sintassi per definire i tre canali standard.

Come ogni device di I/O, anche gli STDIO possono essere rediretti su file, con i comandi “>” e “<”

Esempi di redirezione

```
$ ls > out.txt                    # redirige STDOUT nel file out.txt

$ mioprog < input.txt             # mioprog legge i valori dal file input.txt.
                                  # Eventuali enter saranno indicati nel file con a capo

$ mioprog  2> errore.txt          # solo lo STDERR è rediretto nel file errore.txt

$ mioprog > output.txt 2>&1       # STDOUT e STDERR rediretti nel file output.txt
```

Il meccanismo di concatenazione dello STDOUT di un processo con lo STDIN di un altro processo si chiama “pipeline”

|

(lo STDERR non è rediretto)

Esempio semplice di pipeline

```
ls -la | more
```

Esempio meno semplice di pipeline (da Wikipedia)

```
curl "http://en.wikipedia.org/wiki/Pipeline_(Unix)" | \
sed 's/[^a-zA-Z ]/ /g' | \
tr 'A-Z ' 'a-z\n' | \
grep '[a-z]' | \
sort -u | \
comm -23 - /usr/share/dict/words
```

Variabili di shell (bash)

Come tutti i linguaggi di programmazione, la shell permette di definire delle variabili;

```
MIACARTELLA =  /u/luca/pippo/pluto
```
(negli script)

Per definirle nella sessione di shell corrente, vanno “esportate”

```
export MIACARTELLA = /u/luca/pippo/pluto
```

Molte variabili d'ambiente vengono predefinite tramite gli script iniziali.

Le variabili d'ambiente vengono visualizzate con set

Le variabili vengono referenziate con il simbolo $

```
echo $MIACARTELLA
```

per convenzione si utilizzano nomi con soli caratteri maiuscoli

Risorse condivise: CPU

Esistono modalità e regole per la condivisione delle risorse per l'PC
CPU – RAM - DISCO

Vediamo qualche definizione:
Core : unità singola di elaborazione
Processore : chip composto da uno o più core, una o più memorie RAM (cache di primo livello), et al.
Nodo : scheda che ospita uno o più processori, uno o più moduli di memoria, elementi di interconnessione (bus), et al. La RAM è condivisa fra tutti i processori.
Cluster : insieme di nodi omogenei interconnessi fra loro da una rete “dedicata”.

I server di calcolo e tutte le stazioni di lavoro hanno architettura a 64bit

Modalità di utilizzo:
sequenziale
una sequenza di istruzioni + un flusso dati: processo unico (single core)
SIMD
una sequenza di istruzioni + diversi flussi dati: molti processi (es. openMP)
MIMD
più di una sequenza di istruzioni + diversi flussi dati: molti processi (es. MPI)

Risorse condivise: CPU

Esistono modalità e regole per la condivisione delle risorse per l'PC
CPU – RAM - DISCO

Vediamo qualche definizione:
Core : unità singola di elaborazione
Processore : chip composto da uno o più core, una o più memorie RAM (cache di primo livello), et al.
Nodo : scheda che ospita uno o più processori, uno o più moduli di memoria, elementi di interconnessione (bus), et al. La RAM è condivisa fra tutti i processori.
Cluster : insieme di nodi omogenei interconnessi fra loro da una rete “dedicata”.

I server di calcolo e tutte le stazioni di lavoro hanno architettura a 64bit

Modalità di utilizzo:
sequenziale
una sequenza di istruzioni + un flusso dati: processo unico (single core)
SIMD
una sequenza di istruzioni + diversi flussi dati: molti processi (es. openMP)
MIMD
più di una sequenza di istruzioni + diversi flussi dati: molti processi (es. MPI)

Unix è *multitasking*, ossia è in grado di gestire molti processi concorrenti su un unico processore (core)

Tuttavia, due processi concorrenti *cpu-intensive* si rallenteranno a vicenda maggiormente che se eseguiti in sequenza

$$T(1+2) > T1 + T2$$

La situazione peggiora se i processi sono anche *RAM consuming*

Regola d'oro: **mai** più di un processo “pesante” per core

Ma anche per le esecuzioni multicore, bisogna considerare che i processi contemporanei condividono la memoria RAM e i bus di collegamento CPU-RAM, e quindi posso avere una saturazione delle risorse intra-nodo.

Risorse condivise: CPU

Al MOX abbiamo sistemi “mononodo”:

- ogni nodo è composto da min. 1 – max 8 cpu
- ogni cpu è composta da min. 1 – max 4 core

Controllo della cpu

- controllo velocissimo: *uptime*

```
> uptime
11:42:05 up 100 days, 18:39,  3 users,  load average: 3.10, 3.09, 3.03
```

- ancora più veloce: *w*

- controllo più dettagliato: *top*

E quanti "core" ha la macchina?

```
$ more /proc/cpuinfo
```

```
Tasks: 181 total,   3 running, 164 sleeping,  14 stopped,   0 zombie
Cpu(s): 24.7%us,  0.4%sy,  0.0%ni, 74.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16474700k total, 15908428k used,   566272k free,   222056k buffers
Swap: 16771776k total,    38572k used, 16733204k free, 11370720k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21098 villa     20   0 3233m 1.7g 3160 R  100 10.6 173:33.21 steam3d
21543 villa     20   0 3233m 1.9g 3160 R  100 11.8 129:25.44 steam3d
 2575 root      39  19     0    0    0 S    0  0.0  99:39.51 kipmi0
 6281 root      20   0  208m  11m 2464 S    0  0.1  75:50.25 python
16989 dassi     20   0  164m 9276 6244 S    0  0.1  48:39.03 artsd
20996 villa     20   0  441m  74m  34m S    0  0.5   0:35.28 kdevelop
22701 dassi     20   0 78396  15m 5684 S    0  0.1   0:02.95 emacs
23027 luca      20   0 16892 1328  944 R    0  0.0   0:00.06 top
23066 longoni   20   0 16888 1320  944 S    0  0.0   0:00.07 top
28192 villa     20   0  140m 3196 2220 S    0  0.0 245:03.10 artsd
    1 root      20   0  3792  528  500 S    0  0.0   0:34.94 init
```

Buono a sapersi: Ctrl-M ordina i processi per occupazione di RAM

Regole per la buona amministrazione dei processi

Mai lanciare più processi su un'unica CPU
Mai saturare le CPU di un multiprocessore
Mai esaurire la RAM (ossia affossare un computer)

Strumenti per la buona amministrazione dei processi

Monitorare il computer prima di lanciare i job (comandi *top* e *uptime*)

Prevedere la richiesta di RAM del proprio job

Regolare la priorità di esecuzione...

La priorità di un processo indica quanta attenzione il processo riceve nel programma di scheduling del sistema operativo; più il valore è basso, più il sistema operativo favorirà l'esecuzione del processo.
La priorità è fissata in base a un algoritmo che tiene conto di parecchi fattori (durata trascorsa del processo, I/O, ...); uno solo di questi fattori è definibile dal proprietario del processo: il **NICE**

Un valore di *nice* negativo abbassa il valore di priorità e favorisce il processo, che si vedrà assegnare più risorse CPU

Un valore di *nice* positivo alza il valore di priorità, e il processo verrà sfavorito nello scheduling; in sostanza, andrà più lento

Un utente può solo dare *nice* positivi, e quindi abbassare la priorità dei suoi job

Comandi:
nice -15 nome_programma # in fase di lancio
renice 15 pid_programma # per cambiare la priorità a un job in esecuzione

Job in batch

Per lanciare job in sequenza, in modo che non interferiscano l'uno con l'altro, è possibile creare una sequenza di istruzioni di lancio in un file, e poi lanciare il file stesso

File esegui_job.sh

```
lancio_I_programma
lancio_II programma
lancio_III_programma
```

```
> chmod u+x esegui_job.sh
> ./esegui_job &
```

Il carattere & ritorna il prompt della shell non appena il comando è stato lanciato (e non alla sua terminazione)

MA cosa succede se i miei programmi hanno output a video? E se chiedono un parametro da tastiera? --> Redirigo STDIN e STDOUT

Un modo più sofisticato di gestire le code: *at* e *batch*

`at` permette di far eseguire il job a una data/ora specificata

`batch` permette di far eseguire il job quando il load average scende sotto una determinata soglia (default 0.8)

Entrambi i comandi permettono di definire code di esecuzione...

In buona sostanza: quando si lancia un job in un ambiente multiutente
BE NICE!!

Spazio disco su un sistema condiviso: gioie e dolori

Una delle principali lamentele degli utenti di un sistema unix è lo spazio disco a disposizione

“com'è possibile che io abbia solo mezzo GB quando sul mio portatile da 600€ ho 250 GB?”

Si, ma il tuo portatile ha 150 utenti, che magari fanno I/O tutti assieme? Mostra i tuoi file a 70 computer contemporaneamente? Archivia le modifiche per tre settimane?

Lo spazio disco non manca mai; sono i servizi a essere problematici. Attualmente, il limite più grande è la rete, che mal sopporta i traffici sostenuti (necessari, ad esempio, per il backup)

Work is in progress...

Classificazione dello spazio disco al MOX

• Home: percorso /u/nome_utente
Sono backuppate (~3 settimane), sono in automount (visibili ovunque), sono di dimensione limitata (quota).
Questi spazi sono per i file in sviluppo, particolarmente preziosi

• Aree “di gruppo”: percorso /u/dati/nome_gruppo
NON sono backuppate, sono in automount (visibili ovunque), sono ad occupazione semi-libera (quote dinamiche)
Questi spazi sono per i file in forma definitiva, che sono salvati su qualche altro supporto, ma che è necessario avere anche sottomano, oppure per output di job in corso

• Aree scratch: percorso /scratch
Non sono backuppate, sono visibili solo in locale
Queste aree sono per un veloce I/O con i server di calcolo, e non devono essere considerate aree di archiviazione

Alla ricerca dei file

find (da dove iniziare la ricerca) (criterio di ricerca) (azione sul file trovato)

find / -name core -type f -xdev -exec rm {} \;

/usr/bin/find ./ -name * -mtime +30 -exec /bin/ls -la {} \;

find ./ -name *.html -type f -xdev -exec chmod 600 {} \;

find ./ -name prova -print

/usr/bin/find ./ -name * -user giuseppe -print

/usr/bin/find ./ -name * -group sysadmin -print

IN ROSSO AZIONI PERICOLOSE!

Illustrazione dei flag

-name -> ricerca per nome

-type “d/f/l” ->ricerca solo directory/file/link

-xdev -> ricerca solo nelle sottocartelle appartenenti al medesimo filesystem

-mtime +30 -> ricerca solo i file modificati più di 30 giorni fa

-exec -> esegue un comando

-user -> ricerca solo i file dell'utente giuseppe (volendo si puo' inserire anche id)

-group -> ricerca solo i file del gruppo sysadmin (volendo si puo' inserire anche gid)

-print -> visualizza a video

Creare spazio: la compressione

Alcuni tipi di file si comprimono fino al 90%. Fanno parte di questa categoria i file ASCII (quindi i file di testo, i postscript, i file output “formatted”, ...)

La meraviglia del comando *find*; il massimo con il minimo sforzo

```
$ find ./ -name \*.ps -exec  gzip  {}  \;
```

comprime tutti i file postscript (estensione “ps”) che trova nell'arborescenza
a partire dalla cartella da cui si lancia il programma.
Applicate lo stesso comando a *.eps, *.tar, *.txt ...

Le risorse HPC – definizione

La definizione di risorsa HPC risiede principalmente in caratteristiche non intuitive

- nodi multiprocessore

- interconnessione nodo-memoria

- interconnessione nodo-nodo

- presenza di coprocessori (es. GPU o MIC)

- I/O dedicato

- high availability

...

... e non nella velocità del singolo core di calcolo! (che comunque non è mai indifferente)

Le risorse HPC@MOX: quali sono

Idra: **16 nodi, 4x2 core/nodo Intel Xeon 5560 (speedmark 5539)**
24 GB RAM per nodo x14 + 32 GB RAM x1 + 96 GB RAM x1
dischi locali ai nodi /scratch
disco condiviso /scratch/idra

Cerbero: **4 nodi, 6 core/nodo Intel i7 3930 (speedmark 12093)**
16 GB RAM per nodo
dischi locali ai nodi /scratch
disco condiviso /scratch/idra

Gigat: **5 nodi, 8x4 core/nodo Intel Xeon E5 4610 v2 (speedmark 12807)**
256 GB RAM per nodo

Le risorse HPC@MOX: organizzazione delle code

Torque (clone di PBS) : software per la gestione delle code

Coda = risorsa hpc + disponibilità (ossia quanti nodi posso allocarmi, quanti job posso lanciare, per quanto tempo posso utilizzarli)

Code attive:

Nome	# nodi	# max nodi (nodes)	# max ore (walltime)	# max job
Gigat	5	2	24	1
idra	15	15	24	2
idralong	8	4	-	1
cerbero	4	1	-	2

```
#!/bin/bash
#PBS -S /bin/bash

# Numero di nodi, di core, massimo walltime, e coda richiesta
#PBS -l nodes=1:ppn=1,walltime=00:05:00  -q cerbero

# Nome del job
#PBS -N myjob

# Regirige lo STDOUT su un file e ci unisce anche lo STDERR
#PBS -o out-qsub.txt
#PBS -j oe
#PBS -e err-qsub.txt

# Il job verrà lanciato dalla stessa directory da cui eseguo il comando qsub
cd ${PBS_O_WORKDIR}

# Impostiamo lo STDOUT riportando questo script
cat qsub_script.sub

# Comandi preliminari
hostname
ulimit -s unlimited # make sure we can put big arrays on the stack
date

# Definizione di variabili d'ambiente necessarie al mio job
export LD_LIBRARY_PATH=/u/software/Repository/opt64/any/lib
export PATH=/usr/local/mpich2_1.2.1p1/bin:/bin:/usr/bin:/usr/sbin:/usr/bin/X11:/sbin:/usr/local/bin:.

#-------------------------------------------------------------------#
# Comando di lancio del mio job
time ../../bin/tg_2D_veio

#-------------------------------------------------------------------#

date
```

Le risorse HPC@MOX: comandi Torque

Per sottomettere un job:

qsub *nome_script_di_lancio*

Per controllare lo stato di un job:

qstat

Per cancellare un job in coda o in esecuzione

qdel *PID_del_job*

/scratch è il disco locale (sul nodo e sul front-end)

/scratch/*nome_nodo* è il disco remoto (sul nodo e sul front-end)