Laboratorio Sistemi Operativi

Armir Bujari, Ph.D.

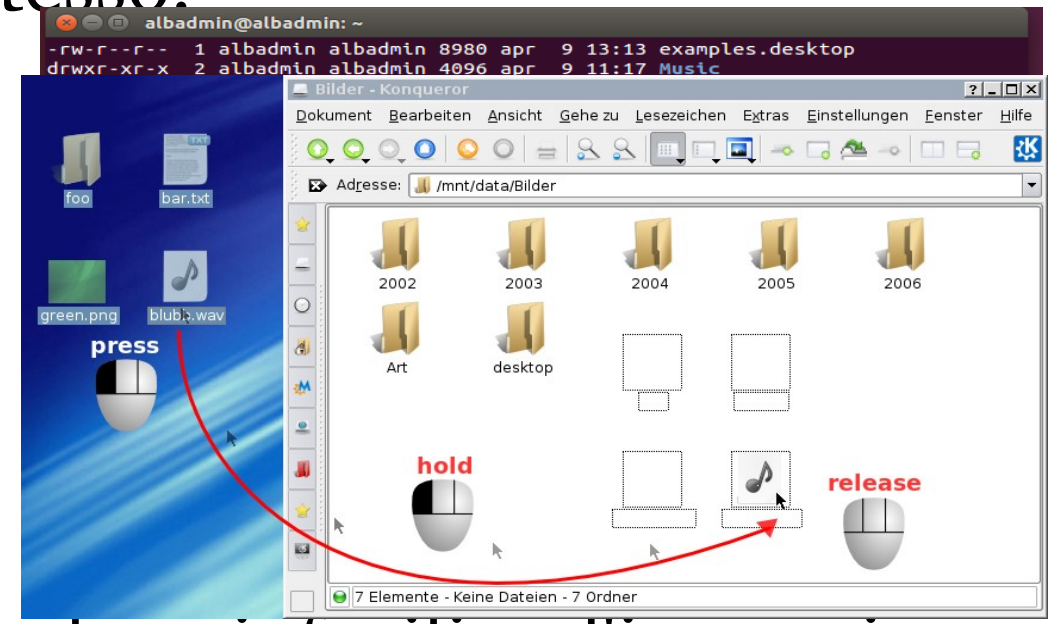
(University of Padua)

Email: abujari@math.unipd.it

Padova, Italia - Maggio 6, 2016

Cosa è il Shell ?

Parte del sistema operativo che permette all'utente di interagire con il sistema stesso.



Può essere

- Testuale (Terminale)
- Grafico (GUIs)

Un involucro (wrapper) che ci facilita l'interazione con il sistema (nucleo), nascondendone I dettagli

Il Shell Unix

Non solo permette all'utente di lanciare programmi e operare all'interno del sistema, ma è anche dotato di un suo **linguaggio di programmazione** vero e proprio.

Ci sono diversi Unix shell disponibili:

- **Bourne Shell** (sh, bash): il primo shell è il più usato
- **C-Shell** (csh): creato in Unix-BSD, ha un linguaggio di programmazione differente da sh
- **Tenex C Shell** (tcsh): evoluzione di csh
- **Korn Shell** (ksh): un altro shell sviluppato da David Korn (AT&T labs), con il proprio linguaggio di programmazione

Il Shell Unix

Dal punto di vista dell'utente, il shell è caratterizzato da un **prompt**, dal quale l'utente può inserire dei comandi

albadmin@albadmin:~/Desktop/SistemiOperativi$ command

Il **prompt** fornisce all'utente alcune informazioni utili:

- Il nome del computer
- Nome dell'utente attuale
- Directory di lavoro
- ...

Il Shell Unix

albadmin@albadmin:~/Desktop/SistemiOperativi$ command

Il commando può essere:

- *Interno (built-in):* viene interpretato ed eseguito internamente dallo shell stesso. Esempio: cd, exit ...

- *Esterno*: denota il nome di un programma eseguibile che viene eseguito dello shell lanciando un altro processo; quando il programma termina, il controllo ritorna alla shell. Esempio: ls, cat ...

Il Shell Unix

albadmin@albadmin:~/Desktop/SistemiOperativi$ command arg1 arg2

- Il **command** viene eseguito e l'intera linea di commando viene passata a una funzione main come un array di stringe
- Il valore di ritorno della funzione main denota lo stato di uscita del commando

*int main(int argc, char **argv)*

Ad un valore di uscita:

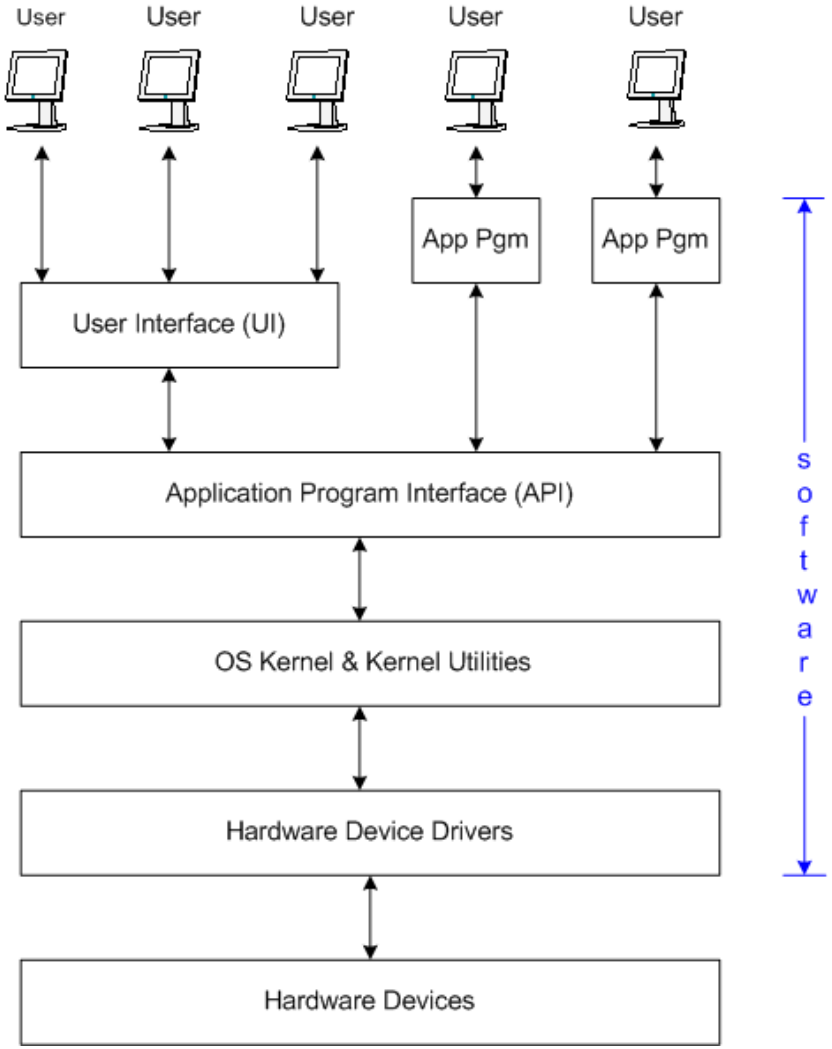
- **0**: esecuzione avvenuta con successo senza alcun errore
- **Non zero**: uno o più errori sono avvenuti durante l'esecuzione (in generale, ad ogni errore ha un valore di uscita associato)

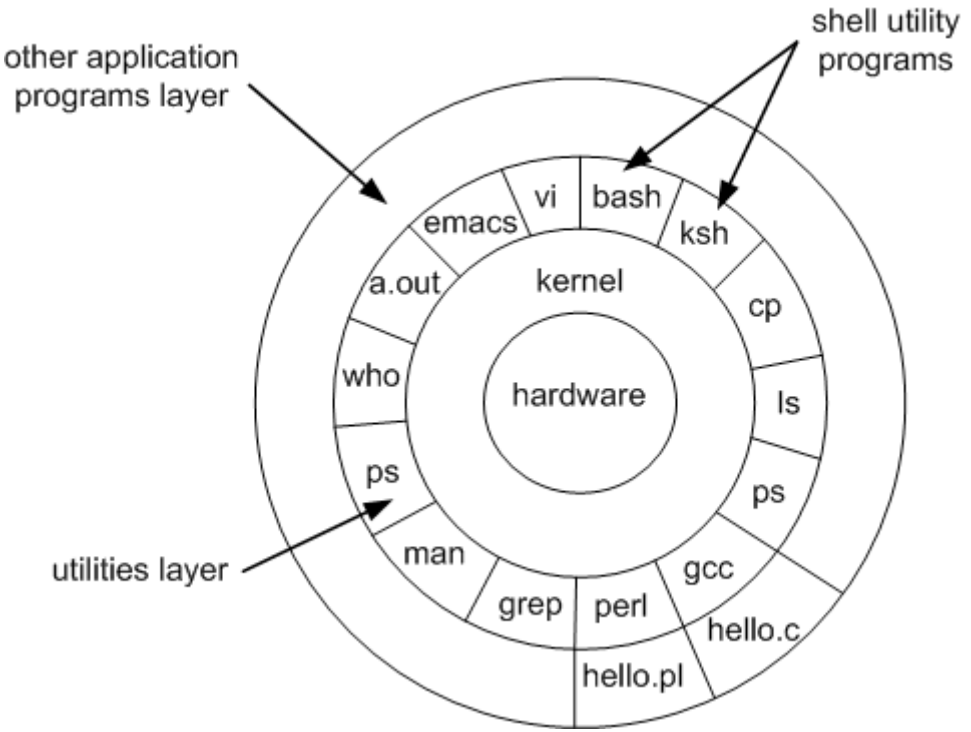
Il Shell Unix

albadmin@albadmin:~/Desktop/SistemiOperativi$ ls /

- Il file eseguibile **ls** viene cercato nel sistema (si trova in /bin/ls)
- La shell crea un processo figlio che a sua volta lancia in esecuzione l'eseguibile (fork + exec)
- La linea di commando viene passata alla funzione main come un array di stringe: ["ls", "/"]
- Il shell fa una chiamata alla funzione wait, mettendosi in attesa di fine esecuzione del processo figlio appena lanciato
- Quando **ls** ha terminato, il controllo viene di nuovo dato alla shell che verifica il codice di uscita (valore ritorno) di **ls**

Collochiamo Il Shell nel SO





Tempo di vedere alcuni comandi in azione !!

Creiamo il primo file

- Apriamo una sessione terminal
  - Attraverso GUI | (CTRL+ALT+T)
- Creiamo il file
  - Editor testo: **nano** | **pico** | **vi**
  - Sintassi: nano | pico | vi <nomefile.txt>
- Verificare che il file è stato creato
  - **ls**: mostra la lista dei contenuti directory
  - Opzione **-i** mostra l'inode number

Creare una directory (mkdir)

- Sintassi: **mkdir** <nome_directory>
  - **cd** <nome_directory>
  - cd .. (directory padre)
  - **pwd** (print work directory)
  - Directory speciali: / | ~
- Creiamo un file all'interno della directory e verifichiamo la sua creazione
- Verifichiamo che il contenuto del file sia quello atteso
  - **cat** <nome_file>
- Per vedere tutti le opzioni di un comando e la sua funzione
  - **man** <nome_commando>

Attributi file

- Eseguiamo **ls -l** su una directory a scelta

```
drwx------ 5 albadmin albadmin 4096 nov  4  2006 Dueffe

-rw-rw-r-- 1 albadmin albadmin    8 mag  4 14:14 file_esempio.txt

drwx------ 4 albadmin albadmin 4096 mag  3 12:18 SiGeM
```

- Output formattato come segue:
  - Permessi del file/directory
  - Numero di coppie (link)
  - Nome proprietario
  - Nome gruppo utente
  - Peso in byte
  - Timestamp (by default, the modification time).

Attributi file: Permessi

```
-rw-rw-r-- 1 albadmin albadmin    8 mag  4 14:14 file_esempio.txt
```

- Organizzati in gruppi di blocchi da (4-3-3) caratteri ciascuno
- Blocchi di permessi per
  - Utente (user, u)
  - Gruppo (g)
  - Tutti gli altri utenti (o)
- Permessi di
  - Lettura (read, r) | int 4
  - Scrittura (write, w) | int 2
  - Esecuzione (execute, x) | int 1

Attributi file: Cambiare Permessi

-rw-rw-r-- 1 albadmin albadmin 8 mag 4 14:14 file_esempio.txt

- Change file mode bits (chmod)
  - **chmod** [OPTION]... MODE[,MODE]... FILE...
  - **chmod** [OPTION]... OCTAL-MODE FILE...

- Esempio: chmod o+w file_esempio.txt
  - Attribuisce permessi in scrittura al file a tutti gli altri utenti sistema
- Esercizio: Attribuire permessi in lettura e scrittura per il file a tutti I gruppi
  - Ricordate la modalità numerica (read – 4), (write – 2), (execute – 1)
  - Per info. ulteriori: man chmod

- **ls -li**: lista I contenuti directory corrente assieme al numero inode appartenente

Copiare un file (cp)

- Formato: **cp** <file_sorgente> <file_destinazione>
- Vediamo il comando in azione
  - cp esempio_file.txt c_esempio_file.txt
  - Verifichiamo la sua creazione: ls | ls file* (suffix, prefix)
- Spostare un file all'interno del FS
  - **mv** <file_sorgente> <file_destinazione>
  - Sposta da una cartella all'altra rinominando (se si vuole) il file

- Esercizio: svuotiamo la directory principale, portando il suo contenuto in una sua sotto directory

Rimozione file e Linking

- Rimuovere un file
  - **rm** <file_sorgente>
  - **rm** *pattern
- Rimuovere una directory
  - **rmdir** <nome_directory>
  - Directory non vuota ?
- Linking: situazioni in cui viene richiesta la duplicazione di un file in altre directory
  - Gestire tante coppie è oneroso; richiede l'aggiornamento di tutte le sorgenti
  - **cp -l** <f_sorgente> <f_destinazione> | **ln** <f_sorgente> <f_destinazione>
- Esercizio: Creiamo una sotto directory della directory principale LabSistemiOperativi che è un mirror della prima
  - Eseguite **ls -l** nella sotto directory appena creato
  - E' cambiata qualche colona nel output del commando ?
- Rimozione link: **rm** <linked_file> | ls -l

Reindirizzamento Shell

- Ogni processo Unix caratterizzato da:
  - Linea comando
  - Valore di ritorno
  - Variabili ambiente
  - Stream strandard (stdin, stdout, stderr)

- Reindirizzamento dei stream
  - ls -l > ls_output.txt: se il file ls_output.txt esiste lo rimpiazza, creandone una nuovo, inserendo l'output del comando eseguito
  - ls -l >> ls_output.txt: concatena al contenuto del file ls_output.txt (se esiste), l'output del comando eseguito

- Esercizio: concatenate l'output di 2 comandi a un file esistente

Shell scripting

- Racchiudere diversi comandi in un singolo file che rappresenta il vostro programma
- Il nostro primo script

```
echo "My first bash script which hopefully will execute"
echo "Fine."
exit 0
```

- Per eseguirlo
  - bash <nome_script>
  - chmod u+x <nome_script> & ./nome_script

Shell scripting

- L'esempio di prima senza il costrutto di raggruppamento comandi
- Possibile soluzione:

```
echo “ls -l > ls_output.txt” > program.sh
echo “ls -i > lsi_output.txt” >> program.sh
echo “cat ls_output.txt lsi_output.txt > output_2_cmd.txt” >> program.sh
#rimozione file ausiliari output
echo “rm ls_output.txt” >> program.sh
echo “rm lsi_output.txt” >> program.sh
echo “exit 0” >> program.sh
```

- Eseguiamo il programma e verifichiamo se il risultato e' quello atteso:
  - chmod u+x program.sh & ./program.sh