Lezione 11
Processi

Sistemi Operativi (9 CFU), CdL Informatica, A. A. 2022/2023
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Università di Modena e Reggio Emilia
http://weblab.ing.unimo.it/people/andreolini/didattica/sistemi-operativi

Quote of the day

(Meditate, gente, meditate...)

“This is the UNIX philosophy: write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.”

Douglas McIlroy (1932-)
Matematico, Ingegnere, Programmatore
Inventore delle pipe e delle utility UNIX

SOLUZIONI DEGLI ESERCIZI

Esercizio 1 (1 min.)

Qual è il PID del processo **`init`** (il gestore dei servizi UNIX)?

Identificazione esatta con **`pidof`**

Il PID del processo **`init`** può essere individuato a partire dal suo nome esatto con il comando **`pidof`**:

```
pidof init
```

Si dovrebbe ottenere il PID 1.

Esercizio 2 (2 min.)

I processi del desktop “GNOME” iniziano con la stringa “gnome”. Come si possono individuare i processi del desktop in esecuzione?

Individuazione tramite **pgrep**

I processi dell'ambiente desktop GNOME possono essere individuati cercando l'espressione regolare **^gnome.*$** con il comando **pgrep**:

```
pgrep ^gnome.*$
```

Per stampare anche i nomi dei comandi si usa l'opzione **-l**:

```
pgrep -l ^gnome.*$
```

Esercizio 3 (2 min.)

Elencate tutti i processi (e relativi PID) eseguiti per conto del vostro username.
Leggete la pagina di manuale del comando opportuno per capire come fare.

Lettura pagina manuale **`pgrep`**

Si apra la pagina di manuale del comando **`pgrep`**:

```
man pgrep
```

Scorrendo la pagina, si dovrebbero trovare le due opzioni **`-u`** e **`-U`**, che svolgono il compito richiesto.
Si usi l'opzione **`-u`**. Il perché sarà chiaro in una lezione successiva.

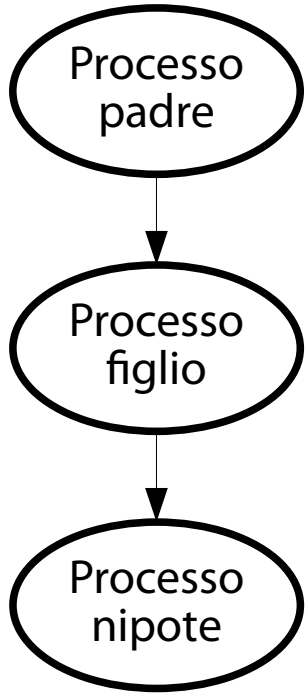
Individuazione tramite **pgrep**

I processi dell'utente **studente** possono essere individuati desktop con l'opzione **-u** del comando **pgrep**:

```
pgrep -l -u studente
```

Esercizio 4 (2 min.)

Aprite un terminale. Con gli strumenti a vostra disposizione, individuate un modo per creare una gerarchia di processi come quella mostrata in figura.

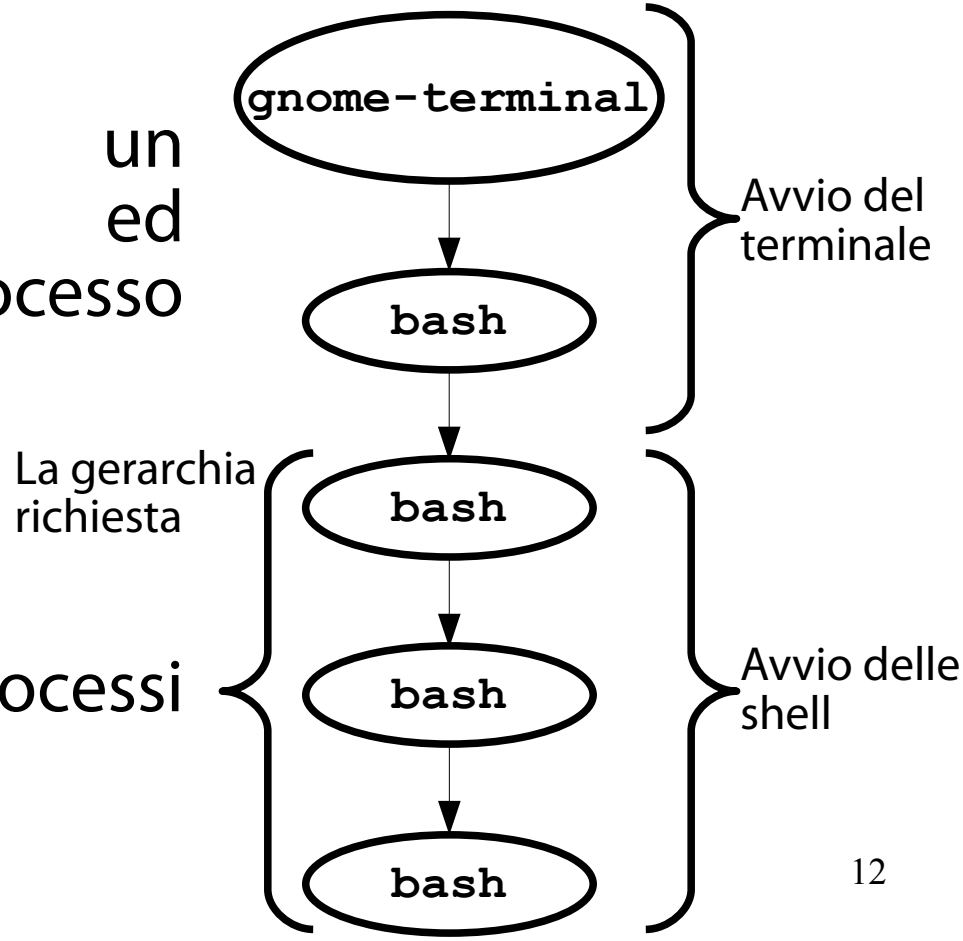


Esecuzione di shell

È sufficiente lanciare un emulatore di terminale ed eseguire per tre volte il processo **`bash`**:

`bash`
`bash`
`bash`

Si ottiene l'albero dei processi riportato a fianco.



Esercizio 5 (2 min.)

Mostrate la catena di processi da **`init`** ad uno qualunque dei processi **`bash`**.

Individuazione processo con **pgrep**

Si invidivua una istanza di BASH con il comando **pgrep**:

```
pgrep bash
1234
2345
3456
```

Si sceglie uno dei PID, ad esempio **1234**.

Visualizzazione tramite **`pstree`**

Per visualizzare i processi richiesti si può usare il comando **`pstree`**. Si usano le opzioni seguenti:

- **`-a`** → stampa dei comandi
- **`-p`** → stampa dei PID
- **`-H`** → evidenziazione di un processo specifico (e antenati)

Il comando richiesto è il seguente:

```
pstree -a –p -H 1234
```

Per visualizzare solo la catena di processi, si può sostituire l'opzione **`-H`** con l'opzione **`-s`**:

```
pstree -a –p -s 1234
```

Esercizio 6 (1 min.)

Avviate il comando **`top`**.
Terminate il processo appena creato, inviandogli un segnale opportuno.

Esecuzione del comando `top`

Si apra un nuovo terminale e si esegua il comando seguente:

```
top
```

Individuazione del PID con **pgrep**

Si individua il PID del processo **top** con il comando **pgrep**:

```
pgrep top
1234
```

Terminazione con `kill`

Si invia il segnale di terminazione anomala SIGTERM al PID:

```
kill 1234
```

Non è necessario esplicitare il segnale in quanto SIGTERM è il segnale di default inviato al processo.
Se proprio lo si vuole esplicitare:

```
kill -15 1234
```

Esercizio 7 (3 min.)

Eseguite un emulatore di terminale.
Leggete la pagina di manuale di **`pkill`** e individuate un modo per terminare l'istanza più recente del processo **`bash`**.

Lettura pagina manuale **`pgrep`**

Si apra la pagina di manuale del comando **`pkill`**:

```
man pkill
```

Scorrendo la pagina, si dovrebbe trovare l'opzione **`-n`**, che svolge il compito richiesto.

Uccisione del processo con **`pkill`**

Si termina il processo con il comando **`pkill`**, usando l'opzione **`-n`**:

```
pkill -n bash
```

ATTENZIONE! Può succedere che questo comando non funzioni. Ciò capita quando si prova ad uccidere il terminale da cui si sta digitando il comando. BASH ignora per motivi di sicurezza tutti i segnali ricevuti dai processi figli.
Per garantire la chiusura, si usi SIGKILL (non gestibile) al posto di SIGTERM:

```
pkill -KILL -n bash
```

Esercizio 8 (2 min.)

Visualizzate i trenta file più grandi nell'intero file system.

Stampa di file e dimensioni con **`find`**

Si stampano i file e le relative dimensioni con il comando **`find`**:

```
find / -printf "%s %p\n"
```

Ordinamento con **`sort`**

L'output del comando precedente viene dato in pasto al comando di ordinamento **`sort -nrk1`**, che ordina le righe ricevute da STDIN in modo numerico e decrescente sul primo campo:

```
find / -printf "%s %p\n" | sort -nrk1
```

Selezione prime righe con **head**

L'output del comando precedente viene dato in pasto al comando di **head -n 30**, che seleziona le prime trenta righe (ovvero i primi trenta file):

```
find / -printf "%s %p\n" | sort -nrk1 | head -n 30
```

Esercizio 9 (3 min.)

Il comando seguente estrae le colonne “username” e “user ID” di **`/etc/passwd`** e stampa il valore delle ultime dieci righe:

```
cut -f1,4 /etc/passwd | head -10
```

Verificate la correttezza del comando.

Debug del comando con **tee**

Si logga ogni output intermedio aggiungendo una istanza di **tee** dopo ogni pipe:

```
cut -f1,4 /etc/passwd | tee cut.txt | head -10 | tee head.txt
```

Analisi output **`cut`**

L'output di **`cut`** mostra tutti i campi di **`/etc/passwd`**, non solo il primo e il quarto.
Il separatore non è stato impostato; lo si imposta con l'opzione **`-d:`** di **`cut`**:

```
cut -d: -f1,4 /etc/passwd
```

Debug del comando con **tee**

Si logga ogni output intermedio aggiungendo una istanza di **tee** dopo ogni pipe:

```
cut -f1,4 /etc/passwd | tee cut.txt | head -10 | tee head.txt
```

Analisi dell'output di **`cut`**

L'output di **`cut`** mostra uno user ID pari a 65534. Ciò è molto sospetto, poiché gli user ID dei primi utenti sono solitamente vicini allo zero.

Il quarto campo del record di **`/etc/passwd`** è il group ID, non lo user ID. Si imposta il terzo campo, ovvero lo user ID:

```
cut -d: -f1,3 /etc/passwd
```

Debug del comando con **tee**

Si logga ogni output intermedio aggiungendo una istanza di **tee** dopo ogni pipe:

```
cut -f1,3 /etc/passwd | tee cut.txt | head -10 | tee head.txt
```

Analisi dell'output di **`cut`**

L'output di **`cut`** è corretto. Si passa al comando successivo.

Analisi dell'output di **`head`**

L'output di **`head`** mostra la stampa dei primi dieci utenti in **`/etc/passwd`**. In realtà, si vogliono gli ultimi dieci utenti.

Si corregge il comando usando **`tail`** al posto di **`head`**:

```
tail
```

Debug del comando con **tee**

Si logga ogni output intermedio aggiungendo una istanza di **tee** dopo ogni pipe:

```
cut -f1,3 /etc/passwd | tee cut.txt | tail | tee tail.txt
```

Analisi dell'output di **`tail`**

L'output di **`tail`** è corretto. In definitiva, il comando corretto è:

```
cut -f1,3 /etc/passwd | tail
```

Esercizio 10 (2 min.)

Visualizzate i trenta file più grandi nell'intero file system.
Usate le named pipe per la comunicazione tra processi.
Spiegate il blocco dei vari processi coinvolti.

Creazione named pipe con **`mkfifo`**

Il comando finale richiede la collaborazione di tre processi distinti. Pertanto, sono necessarie due named pipe per la comunicazione.

Si creano le named pipe con il comando **`mkfifo`**:

```
mkfifo fifo1 fifo2
```

Stampa di file e dimensioni con **`find`**

Si riversa nella named pipe **`fifo1`** lo STDOUT del comando seguente basato su **`find`**, che stampa i file e le relative dimensioni:

```
find / -printf "%s %p\n" > fifo1
```

Il processo che esegue il comando si blocca al momento della scrittura, poiché non esiste ancora un lettore disponibile a consumare l'output prodotto.

Ordinamento con **`sort`**

Si esegue il comando di ordinamento **`sort -nrk1`**, leggendo le righe dalla named pipe **`fifo1`** e scrivendo l'output sulla named pipe **`fifo2`**:

```
sort -nrk1 < fifo1 > fifo2
```

Il processo che esegue il comando si blocca al momento della scrittura, poiché non esiste ancora un lettore disponibile a consumare l'output prodotto.
Il blocco di **`sort`** implica ancora il blocco di **`find`**, purtroppo.

Ordinamento con **`sort`**

Si esegue il comando di ordinamento **`sort -nrk1`**, leggendo le righe dalla named pipe **`fifo1`** e scrivendo l'output sulla named pipe **`fifo2`**:

```
sort -nrk1 < fifo1 > fifo2
```

Non appena si esegue questo comando, **`find`** si sblocca e comincia a riversare il suo output. Il comando **`sort`** lo legge tutto, e solo dopo inizia l'ordinamento.
Al termine dell'ordinamento, sort riversa il suo output sulla named pipe **`fifo2`**, bloccandosi se non trova un lettore disponibile a consumarlo.

Selezione prime righe con **head**

Si prelevano le prime trenta righe dalla named pipe **fifo2** con il comando **head -n 30**:

```
head -n 30 < fifo2
```

Questo comando sblocca **sort**, che a sua volta sblocca **find**.

Distruzione named pipe con **`rm`**

Infine si cancellano le named pipe **`fifo1`** e **`fifo2`** con il comando **`rm`**:

```
rm fifo1 fifo2
```

Esercizio 11 (2 min.)

Lanciate sullo sfondo un comando che legge un valore da terminale e lo memorizza nella variabile **`a`**.
Visualizzate l'elenco dei job.
Notate qualcosa di strano?
Come è possibile portare a termine l'operazione?

Lettura di una variabile con **`read`**

Si lancia sullo sfondo il comando **`read a`**, che legge un valore e lo associa alla variabile **`a`**:

```
read a &
```

Analisi dei job con **`jobs`**

Si elencano i job con il comando **`jobs -l`**:

```
jobs -l
[1]+ 1234 Fermato (input da teminale) read a
```

Il processo **`bash`** che esegue **`jobs -l`** è stato sospeso poiché ha provato a leggere STDIN mentre era sullo sfondo.

Ripristino in primo piano con **fg**

Si ripristina il job in primo piano con il comando interno **fg**:

```
fg %1
```

A questo punto, il processo che esegue **jobs -l** ha pieno controllo del terminale e può ricevere input da STDIN.

Immissione e verifica esecuzione

Si immette un valore qualunque (ad esempio, **`10`**) e si preme **`<INVIO>`**.

Si stampa il valore immesso:

```
echo $a
```

Perché non è stato stampato alcun valore?

Troubleshooting

L'esecuzione sullo sfondo **`read a &`** forza BASH a creare un sottoprocesso apposito per l'esecuzione del comando.

Le modifiche alla variabile **`a`** sono visibili solo in quel sottoprocesso. L'interprete interattivo BASH di partenza è isolato da quello lanciato per l'esecuzione del comando in background.

Il comando **`export`**, purtroppo, non aiuta. **`Export`** esporta la definizione di **`a`** alle shell figlie, ma non permette la propagazione delle modifiche alla shell madre.

Esercizio 12 (2 min.)

Cercate un comando che stampa una sequenza di numeri.
Usate tale comando per implementare un ciclo for che stampa i numeri da 1 a 100.

Individuazione comando con **apropos**

Si cerca la parola chiave “sequence” in tutti i comandi nella Sezione 1 del manuale, usando il comando **apropos** con l’opzione **-s 1**.

```
apropos -s 1 sequence
```

Si dovrebbe poter individuare il comando **seq**, che stampa sequenze di numeri.

Lettura pagina manuale **seq**

Si apra la pagina di manuale del comando **seq**:

```
man seq
```

Il comando **seq** può ricevere uno, due o tre argomenti:

```
seq FIRST
seq FIRST LAST
seq FIRST INCREMENT LAST
```

Per stampare i numeri da 1 a 100 si può eseguire il comando seguente:

```
seq 100
```

Ciclo for con sostituzione di comando

Si può scrivere un ciclo for che, al posto di un insieme di valori scritti esplicitamente, contenga una sostituzione di comando che li genera dinamicamente.

```
for n in $(seq 1 100); do
   echo $n
done
```

Esercizio 13 (4 min.)

Il comando **`diff`** confronta due file e stampa una rappresentazione efficiente delle loro differenze.
A partire da questa informazione, individuate un metodo per trovare le differenze nei contenuti di due pagine Web distinte.

Ricerca comando con **apropos**

Bisogna individuare un programma da linea di comando per scaricare pagine Web. A tal scopo, si può usare il comando **apropos**:

```
apropos -s 1 http
```

Non si trova nulla. Si effettua un altro tentativo:

```
apropos -s 1 client
```

Non si trova nulla. Si effettua un altro tentativo:

```
apropos -s 1 download
```

Si dovrebbe individuare il comando **wget**.

Lettura pagina manuale **wget**

Si apra la pagina di manuale del comando **wget**:

```
man wget
```

Si può scaricare una pagina Web a partire dall'URL con il comando seguente:

```
wget URL
```

Ad esempio:

```
wget http://weblab.ing.unimo.it
```

Problemi

L'uso di default di **`wget`** comporta due problemi.
Wget salva la pagina su file e non la stampa su STDOUT (non desiderabile se si vuole usare una sostituzione di processo).
Wget stampa anche un indicatore di progresso (non desiderabile se si vuole fare la differenza con un altro file).

Lettura pagina manuale **wget**

Si apra la pagina di manuale del comando **wget**:

```
man wget
```

L'opzione **--quiet** disattiva l'indicatore di progresso.
L'opzione **-O** redirige l'output su un file arbitrario. Se il nome del file è **-**, l'output è rediretto su STDOUT.
Pertanto, il comando seguente stampa sul terminale il sorgente della pagina Web puntata da URL:

```
wget --quiet -O - URL
```

Lettura pagina manuale **`diff`**

Si apra la pagina di manuale del comando **`diff`**:

```
man diff
```

Il comando **`diff`** riceve in ingresso due argomenti corrispondenti ai due file da confrontare:

```
diff FILE1 FILE2
```

L'output di **`diff`** è una rappresentazione sintetica delle differenze dei due file.

Uso di una sostituzione di processo

Per confrontare il contenuto di due pagine Web, si può eseguire **`diff`** con due sostituzioni di processo che le scaricano:

```
diff <(wget --quiet -O - URL1)
  <(wget --quiet -O - URL2)
```