

Lezione 6 in laboratorio - parte 2

processi, stringhe

hic sunt
canes stercore



```
for cane in
'pastore maremmano'
'rottweiler' 'pittbull'
do
    kill - 9 ${cane};
done
```

NOTA BENE:

A questo punto abbiamo già visto ed usato anche i comandi:

ps nohup disown bg fg kill wait

Usare il comando man nomecomando per ottenere informazioni sull'uso di uno specifico comando di nome nomecomando.

Familiarizzarsi con processi PID e signal (1)

1. Scrivere tre script bash **lanciaPipeSignal.sh** **uno.sh** e **due.sh**.

- Lo script **lanciaPipeSignal.sh** mette in esecuzione in una pipe i due script uno.sh e due.sh facendo sì che l'input di uno.sh sia preso dal contenuto del file /usr/include/stdio.h e che l'output di uno.sh venga usato come input di due.sh.
- Lo script **uno.sh** inizia scrivendo in un file uno.pid.txt il proprio process identifier. Poi aspetta 2 secondi. Poi legge dal file due.pid.txt il process identifier del processo bash che sta eseguendo due.sh. Poi uno.sh legge la prima riga dal suo standard input, la mette in output aggiungendovi all'inizio la stringa "UNO ", poi manda il segnale SIGUSR2 al processo due.sh di cui possiede il pid. Poi si mette in attesa in un loop infinito contenente sleep 1.

Successivamente, ogni volta che lo script uno.sh riceve il segnale SIGUSR2 da due.sh, uno.sh deve leggere una riga dal proprio standard input, la deve mettere in output, aggiungendovi all'inizio la stringa "UNO ", poi manda il segnale SIGUSR2 al processo due.sh di cui possiede il pid. E così via. Leggendo la fine file lo script uno.sh termina.

- Lo script **due.sh** inizia scrivendo in un file due.pid.txt il proprio process identifier. Poi aspetta 2 secondi. Poi legge dal file uno.pid.txt il process identifier del processo bash che sta eseguendo due.sh. Poi entra in un loop infinito contenente sleep 1. Successivamente, ogni volta che lo script due.sh riceve il segnale SIGUSR2 da uno.sh, due.sh deve leggere una riga dal proprio standard input, la deve mettere in output, aggiungendovi all'inizio la stringa "DUE ", poi manda il segnale SIGUSR2 al processo uno.sh di cui possiede il pid. E così via. Leggendo la fine file lo script due.sh termina.

Familiarizzarsi con processi PID e signal (1)

Soluzione

3. lanciaPipeSignal.sh

```
#!/bin/bash
cat /usr/include/stdio.h | \
./uno.sh | ./due.sh
```

due.sh

```
#!/bin/bash
ricevutoSIGUSR2() {
    if read RIGA ; then
        echo "DUE ${RIGA}";
        kill -SIGUSR2 ${PIDuno}
    else
        kill -SIGUSR2 ${PIDuno}
        exit 1
    fi
}

trap ricevutoSIGUSR2 SIGUSR2

echo ${BASHPID} > due.pid.txt
sleep 2
read PIDuno < uno.pid.txt
while true; do
    sleep 1
done
```

Familiarizzarsi con processi PID e signal (1)

Continuazione Soluzione

uno.sh

```
#!/bin/bash
ricevutoSIGUSR2() {
    if read RIGA ; then
        echo "UNO ${RIGA}";
        kill -s SIGUSR2 ${PIDdue}
    else
        kill -s SIGUSR2 ${PIDdue}
        exit 0
    fi
}
trap ricevutoSIGUSR2 SIGUSR2
echo ${BASHPID} > uno.pid.txt
sleep 2
read PIDdue < due.pid.txt

# leggo prima riga e mando segnale
continua qui a destra
```

continuazione di uno.sh

```
# leggo la prima riga e mando
# il primo segnale

if read RIGA ; then
    echo "UNO ${RIGA}";
    kill -s SIGUSR2 ${PIDdue}
else
    kill -s SIGUSR2 ${PIDdue}
    exit 1
fi

while true; do
    sleep 1
done
```

comandi condizionali (1)

2. Capire che exit status viene restituito dal seguente script **bastardo.sh**

```
( sleep 2; ls -d /usr/include/ ) && { [[ (! ( "false" > "true" )) ||  
      ( 3 -le 5 ) ]] ; } && if [[ $? < "01" ]] ; then exit 0; else exit 1 ; fi ;  
exit $!
```

comandi condizionali (1)

Soluzioni di esercizi slide precedente

2. bastardo.sh

l'esecuzione produce exit status 0

Processi e altro (1)

3. SOLO PER FUORI DI TESTA: Scrivere un nuovo script **lanciapuntini.sh** trova il modo di: a) lanciare in foreground lo script `puntini.sh` (già realizzato in una lezione precedente e descritto qui sotto) passandogli come parametro intero il numero 30, di farlo eseguire per circa 5 secondi e di sospenderne l'esecuzione senza usare la tastiera. b) poi stampa in output "sospeso". c) poi attende circa 3 secondi. d) poi riporta in esecuzione in foreground lo script `puntini.sh` e trova il modo di stampare a video, dopo circa 4 secondi, la stringa hello. Lo script `puntini` non deve essere modificato.

USARE LO SCRIPT `puntini.sh` **GIA PROPOSTO IN LEZIONE PRECEDENTE.**

Scrivere uno script **puntini.sh** che prende come argomento a riga di comando un intero positivo che rappresenta un certo numero di secondi. Lo script deve rimanere in esecuzione per quel numero di secondi e, ad ogni secondo, stampare a video un punto . seguito dal proprio PID. Ma senza andare a capo.

51.puntini.sh

```
#!/bin/bash
NUM=0
while (( ${NUM} <= $1 )) ; do
    sleep 1; echo -n ". ${BASHPID}";
    ((NUM=${NUM}+1))
done
```

Processi e altro (1)'

Soluzione di esercizio fuori di testa

3. lanciapuntini.sh

```
#!/bin/bash
( sleep 5 ;
RIGA=`ps | grep puntini.sh | grep -v lanciapuntini.sh` 
# tolgo eventuali spazi iniziali prima del pid nella riga di testo
while [[ "${RIGA}" != "${RIGA# }" ]]; do RIGA="${RIGA# }" ; done
PIDPUNTINI=${RIGA%% *}
kill -s SIGTSTP ${PIDPUNTINI} ;
echo lanciata sospensione
sleep 4
kill -s SIGCONT ${PIDPUNTINI} ;
echo lanciata ripresa puntini
(sleep 4 ; echo hello ) &
) &
./puntini.sh 15
```

1. puntini.sh

```
#!/bin/bash
NUM=0
while (( ${NUM} <= $1 )) ; do
    sleep 1
    echo -n ". ${BASHPID}"
    ((NUM=${NUM}+1))
done
```

usare iterazioni bash e file

Esercizio1: piu' for per tutti

In una propria directory, creare 10 directory avente nome

1.0 1.1 1.2 1.3 1.4 1.9

Utilizzare il comando **for** ed il comando **mv** della bash, per cambiare i nomi delle directory rispettivamente in :

2.0 2.1 2.2 2.3 2.4 2.9

Suggerimento:guardare le slide su bash scripting, dove si parla di Estrazione di sottostringhe da variabili.

Esercizio2: e ancora un po' piu' di for per tutti

In una propria directory, creare 10 directory avente nome

1.0 1.1 1.2 1.3 1.4 1.9

Utilizzare il comando **for** ed il comando **mv** della bash, per cambiare i nomi delle directory rispettivamente in

2.9 2.8 2.7 2.6 2.5 2.0

Notare che, ad esempio, 1.1 deve diventare 2.8 e 1.3 deve diventare 2.6

In generale, 1.X deve diventare 2.(9-X)

usare iterazioni bash e file

Soluzione Esercizio1: piu' for per tutti

```
for (( NUM=0 ; ${NUM}<10 ; NUM=${NUM}+1 )) ; do mv 1.${NUM} 2.${NUM} ; done
```

Soluzione Esercizio2: : e ancora un po' piu' di for per tutti

```
for (( NUM=0 ; ${NUM}<10 ; NUM=${NUM}+1 )) ; do ((NEWNUM=9-${NUM})) ; mv 1.${NUM} 2.${NEWNUM} ; done
```

usare moduli, man, gcc, Makefile

Esercizio3: correggere errori nei Makefile e nei moduli C

All'indirizzo

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/ESERCIZI_CORREGGERE_ERRORI_1.tgz

c'e' un archivio in formato tar gz contenente una directory 1 che a sua volta contiene delle sottodirectory 1.1 1.2 1.3 1.4 1.8 1.9

In ciascuna sottodirectory c'e' il necessario per creare un eseguibile, ovvero i codici sorgenti in linguaggio ANSI C (esageratamente semplici) ed un Makefile.

Purtroppo 😊 sorgenti e Makefiles possono contenere degli errori.

I sorgenti e i makefile sono talmente semplici che DOVETE essere in grado di capire cosa fanno, anche se contengono errori.

Quindi, scaricate l'archivio, decomprimetelo in una vostra directory, e poi entrate in ciascuna delle directory in ordine crescente di secondo indice, cioe' prima 1.1 poi 1.2 poi 1.3

In ciascuna directory provate a generare l'eseguibile, correggendo gli eventuali errori.

Poi provate ad eseguire l'eseguibile, correggendo eventuali errori.

NB: decomprimere l'archivio fa parte dell'esercizio.