6g Esercizi (laboratorio lez 10) Progr. Concorrente e Vari

Contenuti:

SIMULAZIONE PROVA PRATICA 0a

- **Esercizio 1031 Programmazione concorrente semplice**
- **Esercizio 1032 Programmazione concorrente complicato**
- **Esercizio 1035 script bash (stdout, espressioni condizionali if e text utils (sort etc etc))**

- **Soluzioni esercizi 1031, 1032, 1035**

ESERCIZI vari, per casa

- **Esercizio 42 - script , processi in background , pid, variabili .**
- **Esercizio 43 - script e stdout**
- **Esercizio 44 - script , stdout, e espressioni condizionali if**
- **Esercizio 46 - esecuzione comandi, precedenze operatori e raggruppamento comandi**
- **Esercizio 47 - espressioni condizionali**

- **Soluzioni esercizi 42, 43, 44, 46, 47**

Simulazione Prova Pratica 0a

Download Materiale:

Scaricare il file con le **dispense** e gli **esempi** svolti a lezione

wget http://www.cs.unibo.it/~ghini/didattica/TREE4OS1718.tgz

Decomprimere l'archivio scaricato: tar xvzf TREE4OS1718.tgz

Viene creata una directory **TREE4OS1718** con dentro una sottodirectory **sistemioperativi** con dentro tutto il **materiale**.

Potete navigare tra il materiale con un normale browser aprendo l' URL

file://home/studente/VOSTRADIRECTORY/TREE4OS1617/sistemioperativi/dispenseSistOp2017-18_ordineCronologico.html

Esercizi d'esame: per chi ha difficoltà a superare la prova pratica, ho previsto due tipi di prove:

A. **una prova COMPLICATA**, e' la modalità normale che vi permette di raggiungere un **voto massimo** (nella prova pratica stessa) di **30Lode** ,
B. ed **una prova SEMPLICE**, un po' **meno complicata**, che però vi permette di raggiungere un **voto massimo di 24** perché l'esercizio di programmazione concorrente é meno difficile.

Scegliete voi quale prova svolgere in funzione della vostra preparazione e del grado di angoscia e panico che vi sommerge.

La prova COMPLICATA è composta dagli esercizi **1032 e 1035**,

La prova SEMPLICE è composta dagli esercizi **1031 e 1035**.

Come vedere l'esercizio 1035 è comune alle due prove.

Svolgete SOLO gli esercizi della prova che vi interessa.

I file da consegnare **devono** essere collocati nella directory **CONSEGNA** dentro la home directory dell'utente studente.

Esercizio 1031 - **I Flintstones semplice (ma non troppo)**

Nella preistoria, nel villaggio dei Flintstones, un enorme dinosauro svolge il ruolo di traghetto attraverso un canyon. Il dinosauro sta fermo su un lato del canyon. I cavernicoli salgono sulla coda in 2. Il dinosauro sposta la punta della coda continuamente da un lato all'altro del canyon trasportando i cavernicoli **due alla volta** da un lato all'altro.

Quando la coda arriva su un lato, il dinosauro 1) avvisa i passeggeri che devono scendere, 2) aspetta che siano scesi tutti, 3) avvisa i nuovi passeggeri che devono salire, 4) aspetta che siano saliti esattamente **due** passeggeri, 5) poi sposta la coda sull'altro lato del canyon impiegando 2 secondi.

A questo punto ripete la procedura da 1) a 4) per far fare l'attraversamento nel senso opposto.

I passeggeri dopo che sono scesi dalla coda fanno un giretto di 4 secondi e poi si rimettono in coda per tornare sull'altra riva, e così all'infinito.

All'inizio, la coda del dinosauro è sul lato A in attesa di passeggeri.

Ci sono 3 cavernicoli in totale.

All'inizio 2 cavernicoli sono sul lato A ed 1 sul lato B del canyon.

Modellare ed implementare il sistema descritto, **utilizzando dei thread POSIX** per ciascuna figura (dinosauro, cavernicolo) ed avvalendosi delle opportune strutture dati per la sincronizzazione.

Scrivere il Makefile per compilare e linkare i sorgenti. La mancanza del Makefile viene considerato un errore grave.

Occorre inserire il controllo di errore nelle chiamate a funzione delle librerie dei pthread. In caso di errore grave, terminare il programma producendo un avviso a video.

Esercizio 1032 - **I Flintstones complicato**

Nella preistoria, nel villaggio dei Flintstones, un enorme dinosauro svolge il ruolo di traghetto attraverso un canyon. Il dinosauro sta fermo su un lato del canyon. I cavernicoli salgono sulla coda in 2. Il dinosauro sposta la punta della coda continuamente da un lato all'altro del canyon trasportando i cavernicoli **due alla volta** da un lato all'altro.

Quando la coda arriva su un lato, il dinosauro 1) avvisa i passeggeri che devono scendere, 2) aspetta che siano scesi tutti, 3) avvisa i nuovi passeggeri che devono salire, 4) aspetta che siano saliti esattamente **due** passeggeri, 5) poi sposta la coda sull'altro lato del canyon impiegando 2 secondi.

A questo punto ripete la procedura da 1) a 4) per far fare l'attraversamento nel senso opposto.

I passeggeri dopo che sono scesi dalla coda fanno un giretto di 4 secondi e poi si rimettono in coda per tornare sull'altra riva, e così all'infinito.

All'inizio, la coda del dinosauro è sul lato A in attesa di passeggeri.

Ci sono 3 cavernicoli in totale.

All'inizio 2 cavernicoli sono sul lato A ed 1 sul lato B del canyon.

Modellare ed implementare il sistema descritto, **utilizzando dei PROCESSI** per ciascuna figura (dinosauro, cavernicolo) ed avvalendosi delle opportune strutture dati per la sincronizzazione.

Scrivere il Makefile per compilare e linkare i sorgenti. La mancanza del Makefile viene considerato un errore grave.

Occorre inserire il controllo di errore nelle chiamate a funzione per le mutex e condition variable. In caso di errore grave, terminare il programma producendo un avviso a video.

Esercizio 1035 somma_h_ifdef10.sh

(script, stdout, espressioni condizionali if e text utils (sort etc etc))

Realizzare uno script bash che considera i file con estensione .h contenuti direttamente nella directory /usr/include/ (non nelle sottodirectory) e che al loro interno abbiano almeno 10 righe che contengono la parola ifdef
Da questi file prendere le prime 5 righe che contengono la parola ifdef.

Creare un primo file che contiene tutte le righe cosi' selezionate.
Usando il comando **sort**, creare un secondo file, chiamandolo FINALE.txt, che contiene le righe del primo file ma **ordinate** secondo l'ordine lessicografico crescente.

SOLUZIONI

Soluzione Esercizio 1031 es1031_flintstones_semplice.c
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es1031_flintstones_semplice.tgz

Soluzione Esercizio 1032 es1032_flintstones_complicato.c
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es1032_flintstones_complicato.tgz

Soluzione Esercizio 1035 somma_h_ifdef10.sh

```
#!/bin/bash
if [[ -e PARZIALE.txt ]] ; then
        rm -f PARZIALE.txt
fi
for nomefile in `ls /usr/include/*.h` ; do
        OUT=`cat  ${nomefile}  |  grep ifdef  | wc -l  `
                LINES=${OUT%% *}
                if ((  "${LINES}" >= "10" )) ; then
                                cat  ${nomefile}  |  grep ifdef  |  head -n 5 >> PARZIALE.txt
                fi
done

sort  PARZIALE.txt  >  FINALE.txt
exit 0
```

FINE SIMULAZIONE PROVA PRATICA

da qui in avanti altri esercizi per casa

Esercizio 42 - script , processi, background e pid.

Realizzare uno script bash **aspetta.sh** che prende come argomento un numero intero.
Lo script deve attendere un numero di secondi pari al numero passato come argomento allo script, e poi terminare restituendo 0.

Scrivere poi uno script principale, **lanciaekilla.sh** che lancia 10 volte direttamente in background lo script aspetta.sh, passando come argomento 1000.
Lo script lanciaekilla.sh, ogni volta che lancia lo script aspetta.sh, deve prendere il pid del processo messo in background.
Man mano che crea figli, lo script lanciaekilla.sh memorizza i pid dei figli creati in una variabile **PIDFIGLI** che contiene uno dopo l'altro, separati da uno spazio bianco, i pid dei diversi figli.
Ad esempio, dopo la creazione di 10 figli, il contenuto della variabile PIDFIGLI potrebbe essere
101 43 1002 67 98 303 204 510 77 654
Dopo avere creato i 10 figli, il processo padre lancia un ultimo script, **killa.sh**.

Lo script **killa.sh**, al proprio interno, legge la variabile PIDFIGLI, separa i pid dei diversi processi, e usa uno per uno quei pid per eliminare uno per uno i processi, mediante il comando
kill -9 *pidprocessodaeliminare*

Esercizio 43 - script e stdout

Realizzare uno script bash che considera tutti i file con estensione .h contenuti direttamente nella directory /usr/include/ (non nelle sottodirectory).
Lo script deve stampare sullo standard output il numero intero dato dalla **somma** del numero delle righe di ciascuno dei file considerati.
Suggerimento: utilizzare il comando wc per ottenere il numero di righe di cui e' formato ciascun file.

Esercizio 44 - script , stdout, e espressioni condizionali if

Realizzare uno script bash che considera i file con estensione .h contenuti direttamente nella directory /usr/include/ (non nelle sottodirectory) e che abbiano al loro interno almeno 100 righe.
Lo script deve stampare sullo standard output il numero intero dato dalla **somma** del numero delle righe di ciascuno dei file considerati.
Suggerimento: utilizzare il comando wc per ottenere il numero di righe di cui e' formato ciascun file.

Esercizio 46 - esecuzione comandi, precedenze operatori e raggruppamento comandi

Realizzare uno script bash che:

esegue il comando echo ciao
se tale comando va a buon fine esegue il comando cat /usr/include/stdlib.h
se tale comando va a buon fine esegue il comando sleep 1
se tale comando va a buon fine esegue il comando echo puzzola

Se uno dei 4 comandi specificati non ha ottenuto il risultato indicato, allora occorre eseguire il comando cat /usr/include/stdio.h

tutto lo standard output prodotto dai comandi precedenti deve essere processato dal comando
grep -i -v int

Esercizio 47 - espressioni condizionali

realizzare uno script che mette in output il numero di directory contenute nella directory /usr/include (non considerare le directory contenute nelle sottodirectory).

soluzione Esercizio 42

aspetta.sh

```
#/bin/bash
if (( "$#" != 1 )) ; then echo "num arg errato" ; exit 1 ; fi
sleep $1
exit 0
```

killa.sh

```
#/bin/bash
for pidfiglio in ${PIDFIGLI} ; do
        kill -9 ${pidfiglio}
done
exit 0
```

lanciaekilla.sh

```
#/bin/bash
for (( num=0; ${num}<10; num=${num}+1 )) ; do
      ./aspetta.sh  1000 &
      PIDFIGLI="${PIDFIGLI} $!" ;
done
export PIDFIGLI
./killa.sh
exit 0
```

NB: for pid in `ps aux | grep sleep | cut --bytes=3-10 ` ; do kill -9 $pid ; done

soluzione Esercizio 43

somma_h.sh

```
#/bin/bash
SOMMA=
for nomefile in `ls /usr/include/*.h` ; do
      OUT=`wc -l  ${nomefile}`
      (( SOMMA=${SOMMA}+${OUT%% *} ))
done
echo "SOMMA=${SOMMA}"
exit 0
```

soluzione Esercizio 44

somma_h_minori100.sh

```
#/bin/bash
SOMMA=
for nomefile in `ls /usr/include/*.h` ; do
       OUT=`wc -l  ${nomefile}`
       LINES=${OUT%% *}
       if ((  "${LINES}" >= "100" )) ; then
                    (( SOMMA=${SOMMA}+${LINES} ))
       fi
done
echo "SOMMA=${SOMMA}"
exit 0
```

soluzione Esercizio 46

```
#!/bin/bash
(   echo ciao && cat /usr/include/stdlib.h &&
    sleep 1 && echo puzzola ;
    if (( $? != 0 )) ; then
        cat /usr/include/stdio.h
    fi
)  | grep -i -v int
```

soluzione Esercizio 47

```
find /usr/include/ -maxdepth 1 -mindepth 1 -type d |  wc -l
```