Preview Compito pari - turno 1

[redacted]

1

Dati studente

Inserisci qui i tuoi dati, **compila subito questa parte.**

Cognome

Nome

Matricola

2

Memory management

Considera un sistema con page buffering e considera il problema del trattamento di una pagina X appartenente ad un processo P. Supponi che X venga modificata da P e poi venga tolta dal suo resident set.

1. Possiamo riassegnare subito il frame di X al processo P? spiega.
2. Possiamo riassegnare subito il frame di X ad un processo diverso da P? spiega.
3. Descrivi una politica di cleaning che pensi possa essere conveniente e motiva la risposta.

Answer:

1. Si nel caso in cui P abbia bisogno della pagina contenuta in X poiché "page buffering" non modifica X finché non è indispensabile. Se a P ha bisogno di una pagina differente allora ciò non è possibile poiché prima X va "pulita", cioé scritta su disco
2. No, prima X va scritta su disco.
3. Una possibile politica di cleaning è quella di scrivere su disco i frame sporchi quando il disco non ha nulla da fare, oppure quando i frame puliti sono sotto una certa soglia.

3

I/O

Descrivi l'algoritmo di disk scheduling **cyclic-elevator** e metti in evidenza i vantaggi rispetto all'algoritmo elevator semplice?

Answer:

Cyclic-elevator schedula le richieste in attesa di essere esguite spostando la testina in una direzione ed eseguendo le richieste indipendentemente dall'ordine di arrivo. Quando la testina giunge all'ultima traccia, ritorna all'inizio senza esguire nulla e ricomincia. Elevator, invece, esegue le richieste anche durante la fase di ritorno. Elevator è unfair nel senso che il tempo massimo di attesa di una richiesta per una traccia esterna è doppia rispetto al tempo massimo di attesa di una traccia centrale. Per Cyclic-elevator tale tempo è uguale per tutte le richieste indipendentemente dalla traccia.

4

Scheduling delle attivita' all'interno del SO

Considera un sistema con architettura del kernel “execution within user process”. In tale sistema sono presenti tre processi: A, B, C, inizialmente tutti e tre ready nell'ordine A in testa, poi B, C in coda. La politica di scheduling è **round robin** con quanto di tempo pari a 60ms.

- **A** è cpu bound: genera un page fault ogni 20 ms e ciascun page fault è servito in 30ms.
- **B** è cpu bound: nessun page fault.
- **C** è cpu bound: genera page faults ogni 300ms.

Il processore esegue di volta in volta A, B, C, e inoltre, con tempi trascurabili, mode switching, dispatching, system call e interrupt handlers. Mostra schematicamente, nella seguente tabella, l'ordine con cui tali attività vengono eseguite (una sola croce per ciascuna colonna). Indica anche quali processi sono running, quali ready e quali bloccati in ciascun istante come indicato nell'esempio.

user mode	A	X																		X
	B						X				X									
	C														X					
mode switch			X			X		X		X		X			X		X			X
kernel mode	disptatching				X									X					X		
	system call per I/O																			
	interrupt handler per page fault			X																
	interrupt handler per I/O								X											
	interrupt handler per quanto scaduto												X					X		
stati processi	running	A	A	A	A	B	B	B	B	B	B	B	B	C	C	C	C	C	A	A	A
	ready	B C	B C	B C	B C	C	C	C	C	C	C	C	C	A B	A B	A B	A B	B A	B C	B C	B C
	block					A	A	A	A												
note tempi																					
altre note																					

5

Pratica Unix

Considera la directory /usr/include che contiene gli header delle librerie C.
Il comando

`find` *directory* `-name` "*pattern*"

permette di cercare file il cui nome sia conforme al pattern specificato.

- Mostra un comando che dia la lista dei file che iniziano per la lettera "s" contenuti in un punto qualsiasi del sottoalbero /usr/include (consulta la documentazione

Answer:

```
find /usr/include -name "s*"
grep '^#define' `find /usr/include -name "s*"`
```

Scripting

Il file router_configuration.txt contiene il dump di una configurazione di un router. Il file si compone di vari blocchi (pensali come record) separati da due o piu' linee vuote.

Per svolgere l'esercizio non è necessario conoscere il significato di tutti i campi. Suggerimenti: alcune volte, ma non sempre, conviene processare tale file con awk usando RS="" (stringa vuota) e FS="\n"; ricorda che, in awk, gsub() e' un efficace strumento di sostituzione.

6 Una parte del file contiene la specifica di rotte, inserite a mano dall'amministratore, dette *rotte statiche*. Tali rotte statiche sono esplicitate nelle righe del file di configurazione della forma

`ip route` *<indirizzo-IP-destinazione> <netmask> <indirizzo-interfaccia-router>*

Seleziona le righe del file di configurazione che iniziano con "`ip route`" e in cui l'indirizzo-IP-destinazione ha il terzo byte con un valore compreso tra 7 e 11.

esempio: `ip route 20.30.10.2 255.255.255.0 35.1.1.1`

Answer:

```
cat router_configuration.txt | egrep "^ip route [0-9]*\.[0-9]*\.([7-9]|10|11)\."

ip route 10.0.7.0 255.255.240.0 192.168.2.2
ip route 10.0.8.0 255.255.0.0 192.168.2.2
ip route 10.6.9.0 255.248.0.0 192.168.2.2
ip route 10.0.10.0 255.255.63.0 192.168.4.2
ip route 10.1.7.0 255.255.127.0 192.168.3.2
ip route 10.1.8.0 255.255.0.0 192.168.3.2
ip route 10.1.9.0 255.255.0.0 192.168.2.2
ip route 10.1.10.0 255.255.255.0 192.168.2.2
ip route 10.1.10.0 255.255.255.0 192.168.4.2
```

7 Una parte del file contiene la configurazione delle interfacce del router, su più righe, che inizia con

`interface` *<nome-interfaccia>*

Le righe seguenti nel file (fino alla linea vuota) contengono alcune informazioni sulla configurazione dell'interfaccia specificata.

```
speed 1000M
media-type rj45
negotiation auto
```

in tabella apparirà
GigabitEthernet0/0 rj45

Answer:

```
$ cat router_configuration.txt | awk -v FS="\n" -v RS="" '/media-type/ {print $
0 ; print"" }' | egrep '(^interface|^media-type|^$)' | awk -v FS="\n" -v RS=""
'{print $1, $2}' | awk '{gsub(/interface /,""); gsub(/media-type /, ""); print $0}'
FastEthernet0/1 sfp
GigabitEthernet0/0 rj45
GigabitEthernet0/1 rj45
GigabitEthernet0/3 sfp
```

8 Hai di fronte a te un file list.h contenente esclusivamente il seguente codice.

```
struct element
{
struct element* next;
double num;
};


struct list
{
struct element* first;
struct element* last;
};
```

List.h viene incluso da vari altri .h nel progetto. Pensi che questo sia un corretto stile di impostare il progetto? Discuti quello che può succedere lasciando le cose come stanno e suggerisci le opportune modifiche .

Answer:

No, se una unità di compilazione include due volte list.h, direttamente o indirettamente, il compilatore da un errore di ridefinizione delle strutture. Si può evitare ciò mediante l'uso della compilazione condizionale, inserendo all'inizio e alla fine del file le seguenti direttive.

#ifndef __LIST_H__
#define __LIST_H__
.....resto del file.....
#endif

9 Considera il codice del seguente progetto prj.tar.gz. Compila tutti i file con il comando

```
gcc -g *.c -lm -o fib
```

Considera una esecuzione di fib con parametro **21**. Considera il 300-esima volta che la funzione **fib()** sta per ritornare. Quale è il valore che fib ritorna? Sempre in

[redacted] [redacted]

Moodle Docs for this page

You are logged in as Admin User (Logout)

SO_2010.02.03

```
utente@campus22:~/Scrivania/prj$ gdb fib
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu"...
(gdb) b fib
Breakpoint 1 at 0x400781: file init_list.c, line 8.
(gdb) r 21
Starting program: /home/utente/Scrivania/prj/fib 21

Breakpoint 1, fib (f1=1) at init_list.c:8
8 if ( f1==1 || f1==2 )
(gdb) l
3 #include "list.h"
4 #include <math.h>
5 long int fib( long int f1 )
6 {
7 long int f;
8 if ( f1==1 || f1==2 )
9 f = 1;
10 else
11 f = fib(f1-1) + fib(f1-2);
12 return f;
(gdb) b 12
Breakpoint 2 at 0x4007be: file init_list.c, line 12.
(gdb) ig 2 299
Will ignore next 299 crossings of breakpoint 2.
(gdb) dis 1
(gdb) r 21
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/utente/Scrivania/prj/fib 21

Breakpoint 2, fib (f1=5) at init_list.c:12
12 return f;
(gdb) info b
Num Type Disp Enb Address What
1 breakpoint keep n 0x0000000000400781 in fib at init_list.c:8
2 breakpoint keep y 0x00000000004007be in fib at init_list.c:12
breakpoint already hit 300 times
(gdb) p f
$1 = 5
(gdb) bt
#0 fib (f1=5) at init_list.c:12
#1 0x00000000004007b6 in fib (f1=7) at init_list.c:11
#2 0x00000000004007a6 in fib (f1=8) at init_list.c:11
#3 0x00000000004007a6 in fib (f1=9) at init_list.c:11
#4 0x00000000004007a6 in fib (f1=10) at init_list.c:11
#5 0x00000000004007a6 in fib (f1=11) at init_list.c:11
#6 0x00000000004007fb in init_list (L=0x601010, n=21) at init_list.c:20
#7 0x000000000040089d in main (argc=2, argv=0x7fffea9551c8) at main.c:21
(gdb) up 7
#7 0x000000000040089d in main (argc=2, argv=0x7fffea9551c8) at main.c:21
21 init_list(L, n);
(gdb) p L.last
$2 = (struct element *) 0x601150
(gdb) p *L.last
$3 = {next = 0x0, num = 10170.28645158315}
```