Sistemi Operativi 2010.07.07 (computer pari)

Preview Compito pari - turno 1

Start again

1

Dati studente

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

Cognome

Nome

Matricola

2

Memory management

Con riferimento alla tecnica nota come page buffering descrivi cosa succede quando...

1. ...una pagina X di un processo P viene tolta dal resident set di P
2. ...una pagina X di un processo P, appena tolta dal resident set, è di nuovo acceduta da P
3. ...un processo P' ha bisogno di un frame aggiuntivo e il frame che prima conteneva la pagina X del processo P viene scelto dal sistema per essere assegnato a P'.

Answer:

1
- il frame contenente X viene messo nel page buffer. Se X e' stata modificata il frame è considerato "dirty" e va nella coda delle pagine dirty.

2
- si genera un minor page fault, il frame contiene ancora X e ridato al processo senza alcuna oprazione di input/output

3
- se il frame non è dirty viene assegnato a P' senza problemi, se è dirty il suo contenuto deve essere prima scritto su disco e poi assegnato a P'. In altre parole le sole pagine che possono essere assegnate a processi diversi da P sono quelle pulite.

3

I/O

Descrivi la configurazione raid 10.
Se dovessi scegliere tra una configurazione raid 10 e una raid 01 quale sceglieresti? perche?

Answer:

in 10 i dischi mirrorati (a coppie) vengono usati per fare striping.
es. con 6 dischi
1 1 2 2 3 3
4 4 5 5 6 6
7 7 8 8 9 9

01 i dischi in striping vengono usati per fare mirroring

E' meglio 10 poiché se c'è il fault di un disco in 10 solo la coppia in mirroring è
degradata, in 01 tutto l'array è degradato.

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 50ms.

- **A** è cpu bound: nessun page fault.
- **B** è I/O bound: cpu burst trascurabili, I/O servito in 20ms, nessun page fault.
- **C** è cpu bound: genera page faults ogni 40ms.

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	A
	B						X															B
	C											X				X						C
mode switch			X			X		X			X		X		X		X			X		mode switch
kernel mode	disptatching				X					X									X			disptatching
	system call per I/O								X													system call
	interrupt handler per page fault																	X				interrupt handler per page fault
	interrupt handler per I/O												X									interrupt handler per I/O
	interrupt handler per quanto scaduto			X																		interrupt handler per quanto scaduto
stati processi	running	A	A	A	B	B	B	B	B	C	C	C	C	C	C	C	C	C	A	A	A	running
	ready	B C	B C	B C	C A	C A	C A	C A	C A	A	A	A	A	A B	A B	A B	A B	A B	B	B	B	ready
	block									B	B	B	B						C	C	C	block
note tempi				50					50					70				90				
altre note																						

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.

5

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 la netmask ha il terzo byte con un valore compreso tra 240 e 255 oppure pari a 63.

esempi:

```
ip route 20.30.3.2 255.255.248.0 35.1.0.1
ip route 20.30.3.2 255.255.63.0 35.1.1.1
```

Answer:

```
cat router_configuration.txt | egrep "^ip route [0-9]*\.[0-9]*\.[0-9]*\.[0-9]
[0-9]*\.[0-9]*\.((2(4[0-9]|5[0-5]))|63)"
```

6

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.

Mostra un comando che produca una tabella con una riga per ciascuna interfaccia **con ip configurato** (quando l'ip non è configurato compare "no ip address"), in cui il primo campo sia il nome dell'interfaccia, il secondo l'indirizzo IP configurato per essa.

Esempio

```
interface GigabitEthernet0/0
ip address 192.168.3.1 255.255.255.0
duplex half
speed 1000M
media-type rj45
negotiation auto
```

in tabella apparirà

```
GigabitEthernet0/0 192.168.3.1
```

Answer:

```
cat router_configuration.txt | egrep '^$|interface|(ip address)' |awk -v RS="" -v FS='\n' '!/no/ {gsub(/interface/,""); gsub(/ip address/,""); print $1, $2;}'
```

7

Pratica Unix

Supponi che la tua directory corrente contenga 10000 file di questo tipo

```
pippo1.txt
pippo2.txt
....
pippo10000.txt
```

Vuoi rinominare ciascun file nel corrispondente come segue

```
pluto1.csv
pluto2.csv
....

pluto10000.csv
```

Dai una riga di comando che ti permette di fare ciò velocemente.

Answer:

```
for i in `seq 1 10000`; do mv pippo${i}.txt pluto${i}.csv ; done
```

8

Supponi di avere un progetto con un makefile contenente una sola riga, la seguente

```
CFLAGS=-g
```

A cosa serve? mostrane un utilizzo e spiega i meccanismi sottostanti.

Answer:

quando nella directory corrente è presente un siffatto makefile si può eseguire il comando
make nomeeseguibile
e il make compilerà nomeseguibile.c e creereà nomeseguibile usando le sue regole predefiite (si possono vedere con "make -np", in particolare %: %.c che richiama LINK.c = $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
Nel fare questo, la compilazione avverrà con i simboli di debug.

infatti l'unica riga del makefile CFLAGS=-g configura la variabile del make CFLAGS con la stringa "-g". Tale variabile è particolare poiché è una variabile usata nelle regole predefinite del make
In particolare è usata nella regola predefinita relativa alla compilazione o alla compilazione + link come in questo caso.

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.

- Mostra lo **stack** in quell'istante.

Sempre in quell'istante, considera la corrente invocazione di **init_list()**

- mostra nel contesto di **init_list()** il valore ha la variabile **i** in quell'istante
- mostra il contenuto dell'ultimo elemento della lista **L** in quell'istante.

Answer:

```
pizzonia@pisolo:~/tmp/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 "i486-linux-gnu"...
(gdb) b fib
Breakpoint 1 at 0x804859b: file init_list.c, line 8.
(gdb) r 21
Starting program: /home/pizzonia/tmp/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 0x80485d4: file init_list.c, line 12.
(gdb) del 1
(gdb) ig 2
Second argument (specified ignore-count) is missing.
(gdb) ig 2 299
```

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/pizzonia/tmp/prj/fib 21

Breakpoint 2, fib (f1=5) at init_list.c:12
12         return f;
(gdb) bt
#0  fib (f1=5) at init_list.c:12
#1  0x080485ce in fib (f1=7) at init_list.c:11
#2  0x080485be in fib (f1=8) at init_list.c:11
#3  0x080485be in fib (f1=9) at init_list.c:11
#4  0x080485be in fib (f1=10) at init_list.c:11
#5  0x080485be in fib (f1=11) at init_list.c:11
#6  0x08048605 in init_list (L=0x804a008, n=21) at
init_list.c:20
#7  0x080486ba in main (argc=2, argv=0xbf806bb4) at
main.c:21
(gdb) fr 6
#6  0x08048605 in init_list (L=0x804a008, n=21) at
init_list.c:20
20           double x=pow(fib(i), log(i));
(gdb) p i
$1 = 11
(gdb) p L
$2 = (struct list *) 0x804a008
(gdb) ptype L
type = struct list {
    struct element *first;
    struct element *last;
} *
(gdb) p L->last
$3 = (struct element *) 0x804a0a8
(gdb) p *(L->last)
$4 = {next = 0x0, num = 10170.28645158315}
(gdb)
```

Save without submitting | Submit all and finish