Sistemi Operativi 2010.07.07 (computer pari)

Preview Compito dispari - turno 1

Start again

1

Dati studente

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

Cognome

Nome

Matricola

2

Memory management

Rispondi brevemente alle seguenti domande sul page fault

1. Quali sono i motivi che possono dar luogo ad un page fault?
2. Qual'è il ruolo degli interrupt nel momento in cui avviene il "page fault"?
3. Qual'è il ruolo degli interrupt nel momento in cui il "page fault" viene servito?

Answer:

1 Si ha page fault in uno dei seguenti casi.
- la pagina è in una regione di memoria virtuale legale ma non è nel resident set (ad esempio sta su disco, ma potrebbe anche essere una regione di memoria appena allocata per contenere nuovi dati)
- la pagina è in una regione di memoria virtuale illegale (accesso non consentito)

2 il page fault genera un interrupt. Può dare luogo a un kill del processo (accesso a regione illegale), ad una riconfigurazione della page table (es. minor page fault, nuove allocazioni o pagina nel page buffer) o ad un i/o bloccante (pagina da caricare da disco, major page fault)

3 i minor page fault vengono serviti nell'interrupt relativo al page fault.
i major page fault sono trattati come richieste di input e il processo viene messo in blocco. Quando l'input arriva, si genera un interrupt. Durante tale interrupt il processo che era in blocco a causa del pf viene messo ready, il processo running può rimanere running o meno (dipendentemente dalla politica (preemptive con priorità o no).

3

I/O

Supponi che ci siano due processi **R** e **W** in esecuzione nello stesso sistema. **R** effettua consecutivamente tante letture, **W** effettua consecutivamente tante scritture. Rispondi brevemente alle seguenti domande.

- In un sistema senza particolari accortezze pensi che i due processi avanzerebbero alla stessa velocità? Descrivi il problema "write-starving-reads"
- che tipo di accortezze possono essere prese, a livello di sistema operativo per mitigare il problema?

Answer:

- no W sarebbe molto piu' veloce poiche' le scritture non sono bloccanti e quindi W sarebbe in grado di mettere in coda un gran numero di attivita' per il disco e le letture di R dovrebbero aspettare molte scritture che gia' sono presenti in coda quando le letture di R arrivano.

- si puo' avere un approccio "anticipatorio" cioe' dopo ogni lettura si attende un po' per vedere se ci sono altre letture in arrivo prima di servire le scritture, oppure
si puo' vare una gestione di tempo disco per processo (ad esempio round robin)

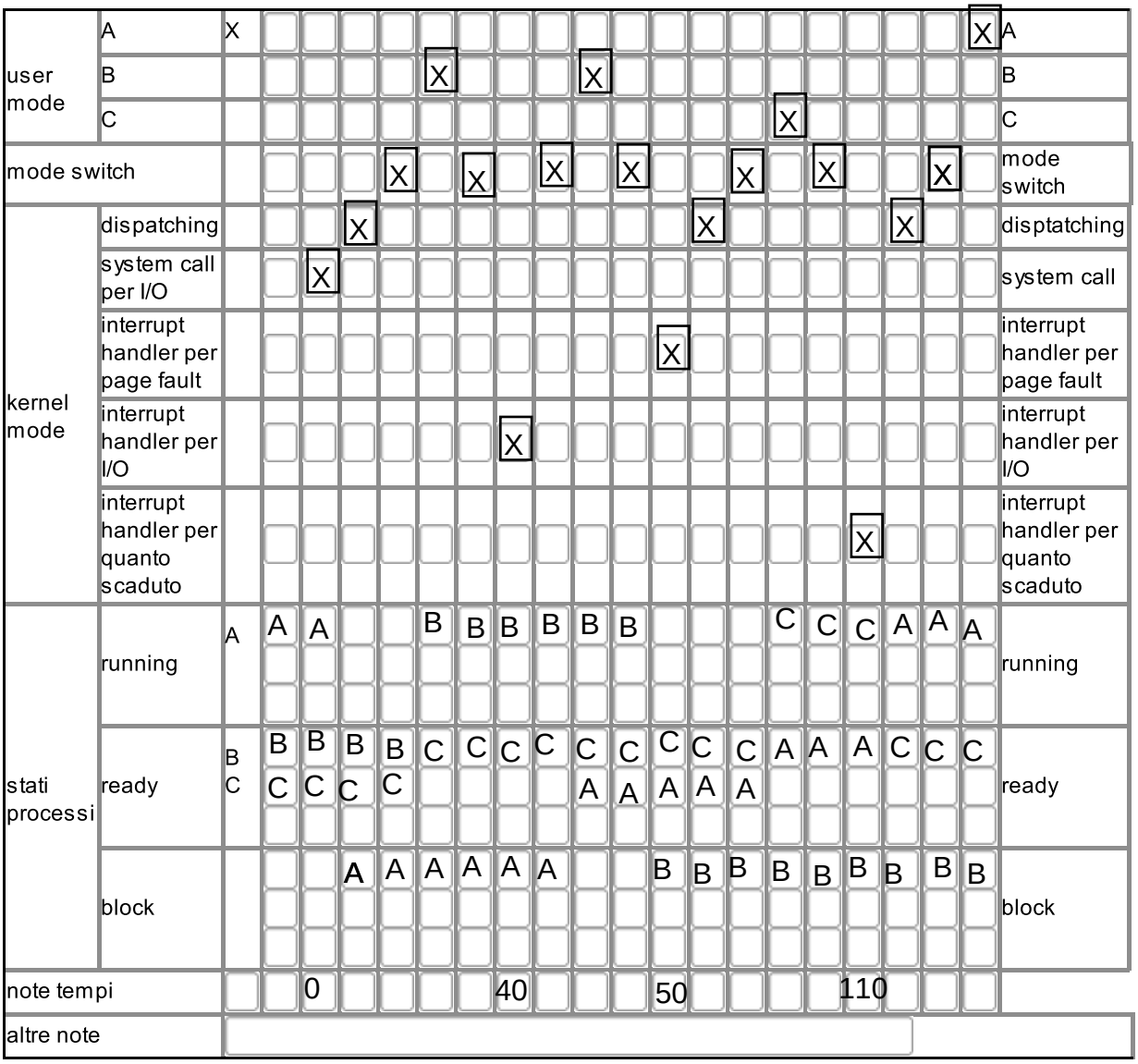
4

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 I/O bound, cpu burst trascurabili, I/O burst di durata 40ms, nessun page fault.
- **B** è cpu bound: genera page fault ogni 50 ms e ciascun page fault è servito in 300ms.
- **C** è puramente cpu bound e non provoca page faults.

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.



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 che iniziano per `ip route` e in cui il terzo byte dell'indirizzo destinazione è pari a 3 o 4 e contemporaneamente il terzo byte dell'indirizzo-interfaccia-router è pari a 2 o 4.

esempi:

```
ip route 20.30.4.2 255.255.255.0 35.1.2.1
ip route 20.30.3.2 255.255.255.0 35.1.4.1
```

Answer:

```
cat router_configuration.txt | egrep "^ip route [0-9]*\.[0-9]*\.[3-4]\
.[0-9].*[0-9]*\.[0-9]*\.[24]*\.[0-9]"
```

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 il campo speed configurato**, in cui il primo campo sia il nome dell'interfaccia e il secondo il valore del campo speed.

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 1000M
```

Answer:

```
cat router_configuration.txt | egrep '^$|interface|speed' |awk -v RS="" -v FS='\n' '/speed/ {gsub(/interface/,""); gsub(/speed/,""); print $1, $2;}'
```

7

Pratica Unix

Esegui le seguenti operazioni su una shell unix e fai copia di ciò che vedi sul terminale (comandi e output).

1. crea una variabile di ambiente X, con contenuto arbitrario, locale alla tua shell
2. mostra che la variabile esiste
3. crea una subshell
4. mostra che X non sia stata ereditata
5. esci dalla subshell
6. rendi X accessibile ai processi figli
7. crea una subshell
8. mostra che X è stata ereditata
9. esci dalla subshell

Answer:

```
pizzonia@pisolo:~$ X=ciao
pizzonia@pisolo:~$ echo $X
ciao
pizzonia@pisolo:~$ bash
pizzonia@pisolo:~$ echo $X

pizzonia@pisolo:~$ exit
exit
pizzonia@pisolo:~$ export X
pizzonia@pisolo:~$ bash
pizzonia@pisolo:~$ echo $X
ciao
pizzonia@pisolo:~$ exit
exit
pizzonia@pisolo:~$
```

8 Spiega perché **make** legge le date di ultima modifica dei file e come questa influenza l'esecuzione dei comandi.

Answer:

il make non ricrea sempre tutti i target. Se deve creare un target T da dipendenze D1 D2 .... Dn esegue il comando della regola solo se la data di T è precedente ad almento una delle date di creazione di D1 D2 ... Dn. Tale verifica è fatta in maniera ricorsiva. Uno tra D1 ... Dn, es. Di, può essere non "up to date" e quindi va ricreato se esiste una regola che lega Di ad altre dipendenze E1... En e uno tra E1... En ha una data di modifica più recente di Di.

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

Moodle Docs for this page

You are logged in as Maurizio Pizzonia (Logout)

SOpari20100707