CAPITOLO 10

Gestione del Real-Time Clock

Un clock è un congegno hardware che emette impulsi, di solito onde quadrate, a regolari intervalli con alta precisione. Oltre al sistema centrale del clock che controlla la frequenza al quale la CPU esegue istruzioni, i sistemi del computer possono avere un real-time clock e un time-of-day clock, i due sono connessi ma non identici. Come un orologio da polso digitale, un time-of-day clock è un cronometro. Esso consiste in un accurato clock che pulsa un intero numero di volte per secondo e un contatore per segnare gli impulsi. Dei programmi leggono il contatore per determinare il tempo e data corrente, e dei privilegiati programmi scrivono nel contatore per determinare il tempo. Bisogna raramente azzerarlo perché i time-of-day clock continuano a contare correttamente sino a quando essi ricevono potenza, indipendentemente se la CPU è pesantemente caricata o ferma (storie abbondano riguardo la confusione introdotta dagli operatori del computer che regolano il time-of-day clock inaccuratamente dopo una abbassamento di potenza).

10.1 I meccanismi del Real-Time Clock

Diversamente dal time.of-day clock il real-time clock non registra impulsi o conserva le tracce della data. Invece pulsa regolarmente un numero di volte ogni secondo segnalando la CPU ogni volta che avviene un impulso segnalando un interrupt. Quindi, una distinzione fra i due clock è basata sia se il clock controlla la CPU o la CPU controlla il clock:

> *La CPU legge il time-of-day clock ogni volta che si vuole conoscere la data e il tempo corrente; Il real-time clock costringe la CPU a sviluppare un interrupt ad ogni impulso.*

I due clock sono inoltre distinti dal fatto che essi contano gli impulsi. Il real-time clock non comprende un contatore, e non accumula interrupt. La responsabilità di contare gli interrupt ricade sul sistema:

> *Se la CPU prende troppo tempo per servire un interrupt del real-time clock o se essa opera con gli interrupt disabilitati per più di un ciclo del clock, essa perderà l'interrupt.*

Evidentemente i sistemi devono essere designati per servire l'interrupt del clock velocemente. L'hardware aiuta dando più alta priorità agli interrupt del clock. Alcuni clock hardware memorizzano pochi interrupts permettendo alla CPU di ritardare più di un ciclo. Ciò nonostante alcuni processori lenti chiamano una procedura C ad ogni clock interrupt, o essi spenderebbero la maggior parte del loro tempo gestendo gli interrupt del clock

10.2 PC Real-Time Clock Interrupts

Il tipico clock opera a 4.77272 MHz. Un circuito del processore divide la frequenza per 4, producendo un segnale di 1.19318 MHz per gli 8.253 chip timer. Il chip timer, a sua volta, divide il segnale in 65536 per produrre una frequenza di 18.20648 Hz, la frequenza dell' interrupt del real-time clock. Il clock interrupt attraversa il vettore CLK_INT (tipo 08 H) ed è inviato attraverso l'INTMAP tavola descritta nel capitolo 9.
Certi sistemi hanno un real-time clock più veloce - cioè una frequenza di interruzione più frequente. Tali sistemi non possono permettersi di chiamare una procedura C su ogni interrupt di clock, o essi spenderebbero la maggior parte del loro tempo gestendo l'interrupt del clock. Come può un tale

sistema evitare di spendere tutto il suo tempo sviluppando interrupt del real-time clock? La risposta è che la frequenza del clock dovrebbe essere adattata per combinarsi al sistema. Rallentare il clock è un importante ottimizzazione perché permette al progettista di costruire ricche funzionalità pur sacrificando la precisione.

Idealmente l'hardware del clock dovrebbe essere rallentato, ma questo è di solito inconveniente o impossibile. Invece un gestore degli interrupt del clock a scopo speciale può essere designato per simulare una frequenza più lenta del clock. Il modo più semplice per simulare un lento clock consiste nel dividere la frequenza del clock. Per esempio, il real-time clock nel Digital Equipement Corporation LSI 11/2 - l'originale processore Xinu era designato per - genera 60 impulsi per secondo, frequenza che richiederebbe più tempo da gestire. Per simulare un lento clock il gestore del clock può essere programmato a ignorare 5 clock interrupt prima di svilupparne uno, effettivamente dividendo la frequenza per 6. In pratica, il gestore del clock deve accettare tutti i clock interrupts, ma esso semplicemente decrementa un contatore e ritorna velocemente su 5 fuori di 6. Come risultato, la parte principale del codice del gestore degli interrupt di clock dell' LSI 11/2 è eseguito solo 10 volte al secondo.

La frequenza al quale è eseguito il codice principale del gestore di interrupt del clock è affettuosamente conosciuta come il tick rate, e noi diciamo che un tick avviene ogni qual volta la parte principale del codice del gestore del clock interrupt è chiamata. In PC Xinu, il codice principale del clock interrupt è eseguito su ogni clock interrupt, così il suo tick rate è lo stesso della frequenza degli interrupt del real-time clock - approssimativamente 18.2 Hz. Mentre questa frequenza è lenta abbastanza da evitare un significante sovraccarico del *interrupt-handling*, la frequenza è una sfortunata scelta dal momento che essa non è un numero intero di tick al secondo. Questa frequenza rende difficile gestire attività che richiedono una soluzione, misurata in secondi o multipli di secondi.

10.3 L'utilizzazione del Real-Time Clock

I sistemi operativi utilizzano il real-time clock internamente per limitare l'ammontare del tempo che un processo può eseguire, come esternamente, per fornire programmi utenti con servizi- come ritardi di tempo. Di solito, i sistemi mantengono una lista di "eventi", ordinati in base al tempo al quale avverrebbero. In qualsiasi momento gli interrupt del real-time clock, esaminano la lista degli eventi ed esegue quegli eventi per il quale il ritardo si è finito. Il nostro progetto permette due tipi di eventi scheduled per il futuro. Il primo è preemption: quando concedendo il servizio della CPU al processo, il sistema schedula una preemption evento per prevenire ai processi di girare per sempre. Il secondo è timed delay (ritardo di tempo): quando un processore richiede una timed delay, rimuove il processo dallo stato corrente e schedula un evento in attesa (wakeup) per rieseguirlo al tempo corretto.

Il sistema usa preemption per garantire che processi di uguale priorità ricevono servizi round-robin (specificato dal programma di schedulazione nel capitolo 4). Ricordiamo che quando resched chiama context switches, esso ripristina la variabile preempt al QUANTUM (QUANTUM è una costante simbolica definita nel file KERNEL.H). Il gestore degli interrupt del clock decrementa preempt ad ogni tick di clock chiamando resched quando esso raggiunge zero. Il corrente processo in esecuzione è sempre il processo di più alta priorità che è adatto per il servizio della CPU ma altri di uguale priorità possono attendere nella lista pronta. Qualora lo sono, resched deposita il processo corrente nella lista dei pronti dietro altri processi con uguale priorità e esso esegue il primo processo della lista. Così qualora tutti i processi di uguale priorità k necessitano il servizio della CPU, tutti verranno eseguiti, al massimo, per k QUANTUM clock tick prima che ognuno di loro riceveranno più servizi.

Il valore della costante simbolica QUANTUM dà la *granularity of preemption*; può essere modificato prima che il sistema sia compilato. Stabilendo un QUANTUM piccolo, diciamo 2 o 3, rendiamo la granulosità piccola, la schedulazione avverrà ogni qualche decimo di secondo. La

granulosità piccola tende a mantenere tutti i processi di uguale priorità sviluppando approssimativamente la stessa velocità. Ma una piccola granulosità introduce molte spese di gestione perché costringe spesso la routine del clock interrupt a chiamare resched. Determinando un QUANTUM grande, diciamo 100, riduce le spese dello scambio di contesto, ma rende la granulosità grande. Il potenziale svantaggio della granulosità grande è che un processo può rimanere in esecuzione molti secondi prima di scambiarsi con un altro di uguale priorità. Raramente i processi usano la CPU a lungo abbastanza da garantire preemption. Un processo volontariamente chiama resched eseguendo routines di sistemi come wait o attività input ed output (I/O). Perché input ed output leggermente paragonati al processo, trascorrono il massimo del loro tempo aspettando i dispositivi. Tuttavia, il sistema non può mai riprendere il controllo del processo che esegue un ciclo infinito senza una preemption, così è importante concluderlo nei sistemi che sostengono la multiprogrammazione. Alcuni sistemi usano anche real-time clock per rispondere alle richieste per i ritardi di tempo. Per esempio, quando il processo in esecuzione richiede un ritardo, PC-Xinu lo trasferisce a una lista di processi “sleeping” , organizzando il loro risveglio dopo l’appropriato numero di clock ticks. In ogni clock tick la routine di interrupt del clock controlla i processi sleeping trasferendo quelli che hanno ritardato l’ora specificata alla lista dei pronti. Routine per gestire simili ritardi saranno considerati in seguito.

10.4 Delta List Processing

Il sistema tiene processi sleeping in una struttura dati chiamata ***delta list***, perché non può permettersi di cercare attraverso lunghe liste arbitrarie di processi sleeping per trovare quelli che dovrebbero svegliarsi su ciascun tick dell'orologio. Come le altre liste dei processi, la *delta list* dei processi sleeping risiede nella struttura q. La Variabile Clockq contiene l’indice q della sua testa. Il gestore di interrupt del clock esamina il primo processo in clockq e chiama la routin di interruzione *wakeup* di alto-livello per svegliare i processi se la loro dilazione del tempo è cessata. Diversamente dalle altre liste nella struttura q, la delta list né è ordinata per chiave crescente, né FIFO. Invece, il campo key registra delta successive (differenze) in ritardi:

> *Processi su clockq sono ordinati in base al tempo in cui si sveglieranno; ciascuna chiave dice il numero di clock tick che il processo deve differire da quello che lo precede sull'elenco.*

Il primo processo sulla lista è quello con meno ritardo, e la sua chiave dà la dilazione che rimane in clock tick finché deve svegliarsi. L'organizzazione *delta* permette alla routine di interrupt del clock di diminuire la prima chiave su ciascun clock tick senza analizzare la lista perché le dilazioni che rimangono sono relativo a esso. Per esempio, se quattro processi hanno bisogno di rimandare 17,27,28, e 32 tick, allora le loro chiavi sul delta list contengono 17, 10, 1 e 4. Data solamente la delta list, la somma parziale delle chiavi dà la dilazione totale di fronte a processi svegli. La dilazione totale di fronte al primo processo sveglio è 17, il totale per il secondo è 17+10, il totale per il terzo è 17+10+1, e il totale per l'ultimo è 17+10+1+4.
Le routine per manipolare le delta list sono facili da progettare, ma i dettagli possono essere ingannevoli; vale la pena dare uno sguardo meticoloso al codice. La procedura *insertd*, mostrata sotto, inserisce il pid del processo in clockq, data la sua dilazione nel parametro key. Come le code con priorità, i campi qkey di ogni nodo in ciascuna lista registra il valore della chiave per quel nodo. Nel codice, la variabile *next* analizza la lista cercando il posto per inserire il nuovo processo.
Le chiavi nella delta list specificano dilazioni relative al loro predecessore; loro non possono essere comparati direttamente al valore iniziale di key, che specifica una dilazione relativa al tempo corrente. Per tenere ritardi paragonabili *insertd* sottrae le dilazioni relative dalla chiave come la ricerca procede, mantenendo il seguente invariante:

A qualsiasi tempo durante la ricerca, sia key che q[next].qkey specificano una dilazione relativa al tempo per svegliare il predecessore di "next".

Insertd inserisce il nuovo processo al punto dove la sua dilazione relativa è meno della dilazione relativa lasciata sulla lista. Nota che insertd non deve controllare esplicitamente la fine dell'elenco, perché il valore della chiave nella coda costringe un'inserzione. Dopo avere collegato il pid del processo nella lista, insertd sottrae la dilazione extra che presenta dalla dilazione del prossimo processo.

```
/* insertd.c - insertd */

#include <conf.h>
#include <kernel.h>
#include <q.h>

/*------------------------------------------------------------------------
 *  insertd  --  inserisce pid del processo in delta list "head", data la chiave
 *------------------------------------------------------------------------
 */
INTPROC insertd(pid, head, key)
      int   pid;
      int   head;
      int   key;
{
      int   next;             /* runs through list          */
      int   prev;             /* follows next through list  */

      for(prev=head,next=q[head].qnext ;
          q[next].qkey < key ; prev=next,next=q[next].qnext)
            key -= q[next].qkey;
      q[pid].qnext = next;
      q[pid].qprev = prev;
      q[pid].qkey  = key;
      q[prev].qnext = pid;
      q[next].qprev = pid;
      if (next < NPROC)
            q[next].qkey -= key;
      return(OK);
}
```

10.5 Mettere un processo a sleep

Programmi utente di solito non accedono direttamente alla coda del real-time clock; chiamano routine di sistema che provvedono ritardi. La chiamata di sistema *sleept(n)* rimanda il processo di n tick. Fa così inserendo il processo nella delta list dei processi sleeping.
Quando un processo si è trasferito nella lista dei processi sleeping, non è più *ready* o *current*. In quale stato dovrebbe essere messo? Processi sleeping differiscono da processi che sono sospesi, aspettando di ricevere messaggi o aspettando su semafori, così nessuno di questi stati basta. È tempo di aggiungere un nuovo stato del processo al progetto; noi lo chiameremo e lo denoteremo con la costante simbolica PRSLEEP. Il nuovo diagramma delle transizioni di stato dei processi è mostrato in Figura:

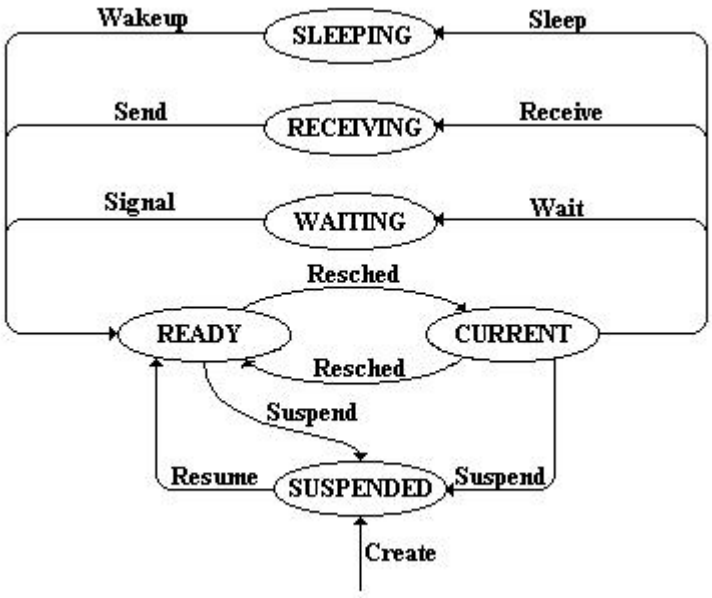

Figura: il diagramma delle transizioni dello stato dei processi per lo stato ‘sleep’

La realizzazione di sleept è lineare. Come mostrato sotto, usa insertd per trasportare il processo corrente nella lista dei processi sleeping, cambia il suo stato a sleeping, e poi chiama *resched* per permettere gli altri processi di eseguire.

```
/* sleept.c - sleept */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------------------------------
 *  sleept  --  delay the caller for a time specified in system ticks
 *------------------------------------------------------------------------
 */
SYSCALL     sleept(n)
int n;
{
      int   ps;

      if ( n<0 )
            return(SYSERR);
      if (n == 0)
            return(OK);
        disable(ps);
        insertd(currpid,clockq,n);
      slnempty = TRUE;
      sltop = & q[q[clockq].qnext].qkey;
        proctab[currpid].pstate = PRSLEEP;
      resched();
        restore(ps);
      return(OK);
}
```

Sleept cita tre variabili esterne definite nel file sleep.h (sotto): *clockq*, *sltop*, e *slnempty*.Clockq

contiene i l’indice in q della lista dei processi sleeping.

```
/* sleep.h */

#define     TICSN 19663            /* numero di tich per 1080 sec. */
#define     TICSD 1080
#define TODQ      54
#define TODR      18198

extern      int   clockq;     /* indice in q della lista dei proc. sleeping */
extern      int   *sltop;     /* indirizzo della prima chiave in clockq*/
extern      int   slnempty;   /* 1 sse clockq è nonempty          */
extern      long  tod;        /* time of day (tick dallo startup)*/

extern      int   defclk;     /* >0 sse clock interrupts differito     */
extern      int   clkdiff;    /* number of clock ticks differiti  */

extern      int   count10;    /* used to ignore 9 of 10 ticks          */
extern      int   clmutex;    /* mutual exclusion sem. for clock  */
```

Anche se la frequenza di tick per PC-Xinu è relativamente lenta, sviluppare clock interrupt è costoso. Le variabili *Sltop* e *slnempty* aiutano l’ottimizzazione dell’interruzione dei processi facilitando a determinare se processi sleeping dovrebbero essere svegliati. *Slnempty* dice se la lista clockq è attualmente nonempty. Se dei processi rimangono su *clockq*, *sltop* dà l'indirizzo della chiave del primo. La routine dell'interrupt salta completamente il codice per la gestione dei processi sleeping quando *slnempty* è zero, e usa *sltop* per localizzare velocemente la chiave-delta del primo processo sleeping quando *slnempty* è diversa da zero.

10.6 Ritardi misurati in secondi

La dimensione di un integer, 16 bits, limita il ritardo permesso da *sleept* a (2^15) –1 ticks che è circa 1800 secondi, o 30 minuti. La chiamata di sistema *sleep* procura un modo ai processi per differire oltre 9 ore perché il suo argomento specifica una dilazione misurata in secondi piuttosto che in ticks. *Sleep* usa ripetutamente *sleept* per elencare dilazioni più brevi fino a che il tempo della dilazione totale è passato.

```
/* sleep.c - sleep */

#include <conf.h>
#include <kernel.h>
#include <sleep.h>

/*------------------------------------------------------------------------
 *  sleep  --  delay the calling process n seconds
 *------------------------------------------------------------------------
 */
SYSCALL     sleep(n)
int n;
{
      int   ps;

      if ( n<0 )
            return(SYSERR);
      if (n == 0) {
            disable(ps);
```

```
            resched();
            restore(ps);
            return(OK);
      }
      while (n >= TICSD) {
            sleept(TICSN);
            n -= TICSD;
      }
      if (n > 0)
            sleept( (int)( ((long)n*(long)TICSN) / (long)TICSD ) );
      return(OK);
}
```

I numeri TICSN e TICSD (avendo rispettivamente valori 19663 e 1080) definiti in *sleep.h* meritano un commento speciale. Risulta che il quoziente TICSN/TICSD approssima molto da vicino la frequenza dei tick , con un errore di circa 25 parti per miliardo. Sotto un altro aspetto, TICSN ticks uguagliano i TICSD secondi. Se scheduliamo una dilazione calcolata in secondi, dilazioni eccedenti TICSD secondi possono essere schedulati come (possibilmente ripetuti) dilazioni di TICSN ticks usando sleept. Quando la dilazione che rimane in secondi è meno di TICSD, esso è usato per determinare quanti ticks dovrebbero essere schedulati come una frazione di TICSN per realizzare la dilazione richiesta in secondi. Per esempio, se “n” è il numero di secondi per ritardo, il quoziente

n*TICSN/TICSD

sarà il numero di ticks a schedulare per *sleept*. Osserviamo che i n*TICSN della moltiplicazione dovrebbero essere fatti prima di dividere per TICSD per evitare errori integer round-off. In vista delle dimensioni di questi numeri, questo prodotto deve essere portato fuori usando l’aritmetica del numero intero lungo.
Notiamo che una richiesta per una dilazione di uno secondo vorranno dire elencare una dilazione di precisamente 18 ticks e produrranno una dilazione attuale di approssimativamente 0.988659 secondi, dando un errore di circa il 1.1341 percento. Questi errori si possono accumulare quando tali dilazioni sono elencate ripetutamente su periodi lunghi di tempo. Inoltre osserviamo che *sleep* è designato per causare un reschedule immediato se il tempo del sleep è zero. Poichè *resched* non può essere chiamato con interrupt inabilitati, questa caratteristica procura un semplice modo per permettere ad un processo di rischedularsi da solo.

10.7 Clock interrupt processing

Il gestore d’interrupt *intcom* chiama la routine di servizio clock interrupt *clkint* su ciascun tick del clock. Poiché è stato chiamato da un gestore d'interrupt, *clkint* presume che gli interrupt sono stati disabilitati all’entrata; la keyword INTPROC ricorda al lettore di questa assunzione. Il parametro a *clkint* è un parametro finto passato automaticamente per *intcom*. *Clkint* prima incrementa il valore di tempo-del-giorno globale tod che conta il numero di ticks sin dallo startup del sistema. Se il processo di differimento clock è in effetto, *clkdiff* è incrementata e *clkint* ritorna immediatamente.
Se la lista dei processi sleeping non è vuota, *clkint* diminuisce la prima chiave su *clockq,* e se la dilazione è arrivata a zero, *wakeup* è chiamato per svegliare i processi il cui ritardo è terminato. Nota come *slnempty* e *sltop* eliminano il calcolo di indici inferiori a tempo dell'interrupt. Finalmente, il contatore preemption è decrementato che produrrà la rescheduling del processo corrente se la sua fetta di tempo è scaduta.

```
/* clkint.c - clkint */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <sleep.h>
#include <io.h>


/*------------------------------------------------------------------------
 *  clkint  --  clock service routine
 *  called at every clock tick and when starting the deferred clock
 *------------------------------------------------------------------------
 */
INTPROC clkint(mdevno)
int mdevno;                  /* minor device number       */
{
      int   i;


    tod++;

      if (defclk) {
            clkdiff++;
        return;
      }
    if (slnempty)
        if ( (--*sltop) <= 0 )
            wakeup();
    if ( (--preempt) <= 0 )
        resched();
    return;
}
```

10.8 Svegliare processi sleeping

Wakeup rende pronti tutti i processi il cui ritardo è terminato. Il contatore della prima chiave su *clockq* prende in considerazione qualsiasi ticks accumulati dall'ultima chiamata a *wakeup*. *Wakeup* rimuove e rende pronto il primo processo da *clockq* il cui ritardo è terminato, e propaga qualsiasi eccedenza di ticks accumulati fino al prossimo item nella lista di attesa. Finalmente *wakeup* azzera *sltop* e *slnempty* per riflettere il nuovo stato della coda prima di chiamare *resched*, perché le rescheduling possono permettere l'esecuzione di un altro processo (e il clock di interrupt).

```
/* wakeup.c - wakeup */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------------------------------
 *  wakeup  --  called by clock interrupt dispatcher to awaken processes
 *------------------------------------------------------------------------
 */
wakeup()
{
      register int      makeup;           /* makeup for lost time      */
```

```
        register int      k;          /* key value                  */

        makeup = 0;
        while ( nonempty(clockq) && (k=firstkey(clockq)) <= makeup ) {
              makeup -= k;
              ready(getfirst(clockq));
        }
        if ( slnempty = nonempty(clockq) ) {
              sltop = &firstkey(clockq);
              *sltop -= makeup;
        }
        resched();
}
```

10.9 I processi dei clock differiti

Il gestore del clock include una caratteristica supplementare che lo complica: *processo del clock differito*. In essenza, il modo del clock differito permette al sistema di accumulare tick di clock senza iniziare gli eventi. La differenza tra il clock che ignora interrupts e il clock che differisce gli interrupt è che chi gestisce il clock può schedulare eventi che "sarebbero dovuti accadere" quando lascia il modo differito e ritorna alla maniera normale. Se il clock rimane solamente differito per alcuni ticks a tempo, il sistema apparirà operare normalmente.
La motivazione per sviluppare il clock differito è che il sistema operativo mantiene le interruzioni disabilitate mentre cambia contesto. Disabilitare le interruzioni non pone problemi quando l'input avviene da tastiera perché gli utenti dattilografano lentamente, paragonati alla velocità alla quale i computer consumano dati. Ma per alcune comunicazioni computer a computer, i dati sono spediti e ricevuti ad alta velocità. Se un context switch accade mentre riceve un blocco di caratteri, alcuni dei caratteri possono essere persi.
Per risolvere questo problema il sistema di I/O ha bisogno di proibire context switch, per periodi brevi di tempo anche se le interruzioni rimangono abilitati. Idealmente, il sistema dovrebbe essere in grado di riguadagnare il tempo perduto quando il context switch è di nuovo riabilitato. Anche se è impossibile prevenire il context switch senza cambiare il comportamento del sistema, l'idea è di trovare una soluzione che ha il minimo impatto senza corrompere il progetto di base. Il processo del clock differito consiste nel posticipare, ma non nell'ignorare, contex switches. Durante un periodo di clock differito, il contex switch è sospeso per interrupts clock differiti. Quando il differimento termina, il processo normale riprende.

10.9.1 Procedure per cambiare a /e dal modo differito

Un processo può mettere il clock nel modo differito chiamando *stopclk* e può ritornare il clock nel modo real-time chiamando *strtclk*. Qualsiasi numero di processi può richiedere che il clock sia differito - esso rimane differito finché tutti hanno chiamato *strtclk*. *Stopclk* conta le richieste dei differimenti incrementando *defclk*, e *strtclk* conta le richieste di restart decrementandolo. Finchè *defclk* rimane positivo, il gestore dell'interrupt conta i tick di clock in *clkdiff*.
Strtclk riguadagna il tempo perduto quando *defclk* raggiunge zero di nuovo, prendendo tutti gli eventi che sarebbero dovuti accadere mentre il clock è rimasto differito. Per fare così, *strtclk* aggiorna il contatore preemption e sottrae i ticks accumulati dalla dilazione di processi a riposo, svegliando processi i cui tempi a riposo sono terminati. Il codice per entrambe le procedure *strtclk* e *stopclk* è contenuto nel file *ssclock.c*, mostrato sotto.

```
/* ssclock.c - stopclk, strtclk */

#include <conf.h>
#include <kernel.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------------------------------
 *  stopclk  --  put the clock in defer mode
 *------------------------------------------------------------------------
 */
stopclk()
{
      int   ps;
      disable(ps);
      defclk++;
      restore(ps);
}

/*------------------------------------------------------------------------
 *  strtclk  --  take the clock out of defer mode
 *------------------------------------------------------------------------
 */
strtclk()
{
      int   ps;
      int makeup;
      int next;
      disable(ps);
      if ( defclk<=0 || --defclk>0 ) {
            restore(ps);
            return;
      }
      makeup = clkdiff;
      clkdiff = 0;
      if (slnempty)
            if ( (*sltop -= makeup) <= 0 )
                  wakeup();
      if ( (preempt -= makeup) <= 0 )
            resched();
      restore(ps);
}
```

10.10 Inizializzazione del clock

La procedura *Clkinit* , mostrata sotto, compie quanto necessario per l'inizializzazione.

```
/* clkinit.c - clkinit */

#include <conf.h>
#include <kernel.h>
#include <q.h>
#include <sleep.h>

/*------------------------------------------------------------------------
 *  clkinit  --  initialize the clock and sleep queue (called at startup)
 *------------------------------------------------------------------------
 */
clkinit()
{
      tod = 0L;               /* seconds since startup    */
      preempt = QUANTUM;           /* initial time quantum      */
      slnempty = FALSE;      /* initially, no process asleep    */
      clkdiff = 0;                 /* zero deferred ticks       */
    defclk = 0;             /* clock is not deferred    */
    clockq = newqueue();    /* allocate clock queue in q    */
}
```