

ATTENZIONE: Anche se ho impiegato cura ed impegno nella realizzazione di questo documento, consiglio vivamente chiunque voglia applicare i suggerimenti da me proposti, di eseguire tutte le opportune prove in un ambiente di test, prima di effettuare eventuali modifiche in produzione. Questa convinzione nasce dal fatto che nonostante mi sia documentato attraverso libri di testo, manuali e documenti in rete, esiste sempre la probabilità di eventuali errori nella stesura ed imprecisioni nella esposizione.

ATTENZIONE: Quanto scritto in questo documento può essere utilizzato in modo del tutto libero. Pregherei comunque tutti coloro che desiderano delle modificarlo, di comunicarlo al [mio](mailto:) indirizzo di posta in modo tale da permettermi di aggiornare e/o correggere eventuali errori.

<http://www.oral.com> Giugno 2005

Di [Andrea Salzano](#) e [Federico Proietti](#)

V\$SESSION_WAIT (draft)

Le tabelle dinamiche di performance sono tabelle in cui Oracle storicizza le statistiche di sistema. ATTENZIONE: Le tabelle V\$ sono suscettibili a frequenti cambiamenti. Se si desidera una lista completa di tutte le V\$tables, è possibile accedere alla vista V\$FIXED_VIEW_DEFINITION¹. Il seguente documento ha come scopo solo quello di riassumere una particolare vista di sistema, la V\$SESSION_WAIT, spiegandone il significato e come interpretarla correttamente, anche se vengono fatte alcune considerazioni di tuning.

1. Definizione

La V\$SESSION_WAIT, visibile in tabella 1, mostra quale evento (di attesa) ogni sessione sta aspettando, o qual era l'ultimo evento per cui la sessione ha atteso. A partire da Oracle 10g², le informazioni di V\$SESSION_WAIT sono espone all'interno della V\$SESSION consentendo in tal modo di evitare join per reperire informazioni.

- P1RAW, P2RAW, e P3RAW mostrano gli stessi valori delle colonne P1, P2, e P3, eccetto il fatto che sono espressi in esadecimale.
- La colonna WAIT_TIME contiene un valore di -2 su piattaforme che non supportano il meccanismo di fast timing. Se il tuo db sta girando su una di queste piattaforme e desideri che questa colonna rifletta il vero tempo di attesa (il wait time), allora devi impostare il parametro TIMED_STATISTICS

¹ Questa vista contiene la definizione di tutte le "fixed views". Come vedremo più avanti, le tabelle V\$ sono basate sulle viste GV\$ che lo sono a loro volta sulle X\$. Le tabelle X\$ sono istanze o strutture di sessione in memoria presentate sottoforma di tabelle (servono come base per le V\$). I nomi di molte tabelle X\$ iniziano con X\$K. La lettera dopo "K" indica a quale strato di kernel questa struttura dati appartiene. Controlla il capitolo 1 del libro di Steve Adams, *Oracle8i Internal Services*, per trovare la funzionalità di ogni strato. Di solito questa conoscenza ci può dare una mano per capire a cosa è dovuto l'errore interno di Oracle. Se la lettera "K" è preceduta da una "S", come in SKGXP, allora è un'entità OSD (Operating System Dependent).

² In Oracle 10g, è stata introdotta una nuova vista: la V\$SESSION_WAIT_HISTORY. Questa mostra gli ultimi 10 wait events per ogni sessione attiva.

a TRUE. Ricorda che facendo questo hai un piccolo effetto negativo sulle performance del sistema.

Su alcune release, la colonna `WAIT_TIME` conteneva un valore arbitrariamente grande invece di uno negativo per indicare che la piattaforma non aveva il meccanismo fast timing.

- La colonna `STATE` interpreta il valore di `WAIT_TIME` e descrive lo stato corrente o della più recente attesa.

WAIT_TIME

All'inizio di un evento associato ad un'attesa, il kernel di Oracle imposta il valore di `WAIT_TIME` a zero. Tale valore resta a zero fino a che l'evento d'attesa è completato, dopo di che il kernel imposta questo valore ad uno di quelli mostrati in Tabella 2. Nota che l'unità di misura è il centesimo di secondo, anche in Oracle9i. Non c'è una colonna tipo `WAIT_TIME_MICRO` almeno fino ad Oracle 9.2.0.2.1, anche se il valore di `WAIT_TIME` deriva dalle viste X\$ sottostanti, a partire da un valore in microsecondi.

SECONDS_IN_WAIT

All'inizio di un evento associato ad un'attesa, il kernel di Oracle imposta il valore di `SECONDS_IN_WAIT` a zero. Tale sessione non aggiorna mai questo valore fino al successivo evento d'attesa, dopo di che la sessione reimposta il valore a zero. Il valore della colonna `SECONDS_IN_WAIT` è incrementato di 3 approssimativamente ogni tre secondi dal processo log writer (LGWR). Nota che l'unità di misura è in secondi, non in centesimi o millesimi di secondo.

Eventi che vanno in "time out" complicano il problema. Per esempio, un evento d'attesa "enqueue" va in time out approssimativamente ogni due secondi, anche per quegli eventi di "enqueue" che durerebbero considerevolmente di più. Ad ogni time out, il kernel di Oracle incrementa `SEQ#`, ma non reimposta il valore di `SECONDS_IN_WAIT`.

STATE

All'inizio di un evento associato ad un'attesa, il valore della colonna `STATE` diventa `WAITING`. Tale valore resta `WAITING` fino a che l'evento di attesa è completato, dopo di che il kernel imposta questo valore ad uno di quelli descritti in Tabella 2.

Table 1

Column	Datatype	Description
SID	NUMBER	Identificativo della sessione
SEQ#	NUMBER	Sequenza che identifica univocamente questa attesa. Incrementata ad ogni evento d'attesa
EVENT	VARCHAR2(64)	Risorsa o evento di attesa della sessione. Vedi anche: "Oracle Wait Events"
P1TEXT	VARCHAR2	Descrizione del primo parametro
P1	NUMBER	Primo parametro
P1RAW	RAW(4)	Primo parametro
P2TEXT	VARCHAR2	Descrizione del secondo parametro
P2	NUMBER	Secondo parametro
P2RAW	RAW(4)	Secondo parametro
P3TEXT	VARCHAR2	Descrizione del terzo parametro
P3	NUMBER	Terzo parametro
P3RAW	RAW(4)	Terzo parametro
WAIT_TIME	NUMBER	Un valore diverso da zero rappresenta l'ultimo evento di attesa della sessione. Un valore zero significa che la sessione sta correntemente aspettando.
SECONDS_IN_WAIT	NUMBER	I secondi di attesa. Se WAIT_TIME = 0, allora SECONDS_IN_WAIT sono i secondi spesi nella corrente condizione di attesa. Se WAIT_TIME > 0, allora SECONDS_IN_WAIT rappresenta i secondi a partire dall'inizio dell'ultimo evento di attesa, e SECONDS_IN_WAIT - WAIT_TIME / 100 rappresenta il numero di secondi trascorsi dall'ultimo evento d'attesa terminato.
STATE	VARCHAR2	Stato dell'attesa: 0: WAITING (la sessione sta aspettando (adesso)) -2: WAITED UNKNOWN TIME (la durata dell'ultima attesa è sconosciuta) -1: WAITED SHORT TIME (l'ultima attesa è stata <1/100th di secondo) >0: WAITED KNOWN TIME (WAIT_TIME = durata dell'ultima attesa)

Table 2

STATE	WAIT_TIME	Implication
WAITED UNKNOWN TIME	-2	Quando l'evento si è concluso, il valore di TIMED_STATISTICS era FALSE, perciò la durata attuale è sconosciuta.
WAITED SHORT TIME	-1	L'evento d'attesa si è concluso, ma è iniziato e terminato all'interno dello stesso "clock tick" della funzione gettimeofday ³ .
WAITING	0	L'evento è in fase di elaborazione, e sta per essere concluso.
WAITED KNOWN TIME	$t \geq 0$	L'evento è terminato, ed ha impiegato $t = t1 - t0$ centesimi di secondo per essere concluso.

Non scrivere la queries sulla V\$SESSION_WAIT con WAIT_TIME=0 nella tua clausola where se ciò che realmente intendi è STATE='WAITING'. Alcuni analisti hanno l'abitudine di assumere che i predicati WAIT_TIME=0 e STATE='WAITING' sono equivalenti, poiché nel kernel Oracle7 and Oracle8i, essi lo sono. Comunque in Oracle9i, non lo sono più.

Il kernel di Oracle9i calcola la colonna WAIT_TIME come round(ksusstim/10000), sulla x\$ksusecst, ma il valore di SATAE è calcolato come DECODE di un valore non arrotondato di KSUSSTIM⁴. Per tale motivo, WAIT_TIME può risultare zero mentre il

³ La funzione gettimeofday è una system call di quei sistemi operativi compatibili con lo standard POSIX. Essa restituisce una struttura dati contenente il numero di secondi e microsecondi misurati a partire dalla mezzanotte (ora 0) del 1° gennaio del 1970 (istante chiamato anche Epoch).

⁴ Se la funzione ROUND applicata ad un numero non le viene passato il secondo parametro, restituisce la parte intera del valore decimale.

```
SQL> select KSUSSTIM, KSUSSTIM/10000 div1,round(KSUSSTIM/10000) round,
2      decode(ksusstim, 0, 'WAITING', -2, 'WAITED UNKNOWN TIME',
3             -1, 'WAITED SHORT TIME', 'WAITED KNOWN TIME')
4      from x$ksusecst where KSUSSTIM >0;
```

KSUSSTIM	x1000	WAIT_TIME	STATE
173655	17.3655	17	WAITED KNOWN TIME
165217	16.5217	17	WAITED KNOWN TIME
3	.0003	0	WAITED KNOWN TIME
169824	16.9824	17	WAITED KNOWN TIME
176399	17.6399	18	WAITED KNOWN TIME
128433	12.8433	13	WAITED KNOWN TIME
170464	17.0464	17	WAITED KNOWN TIME

suo valore di base non lo è. Quindi, Oracle 9i produce situazioni in cui `WAIT_TIME` è zero, mentre `STATE` è qualcosa di diverso da `WAITING`⁵.

2. Come usare `V$SESSION_WAIT`

Scoprire dove una sessione è bloccata

Di tanto in tanto, ci tocca la prendere la seguente chiamata: “Ciao sono Andra. La mia sessione è bloccata”. Lui ha già chiesto ai suoi colleghi se i sistemi erano giù ed ognuno gli ha risposto che tutto sembra funzionare correttamente. Se sai come trovare l’ID della sessione di Andrea, allora è facile determinare cosa sta succedendo. Supponiamo di aver scoperto che il Session ID (SID) di Andrea è il 42. Usa allora la seguente query per determinare perché è bloccata:

```
SQL> col sid format 999990
SQL> col seq# format 999,990
SQL> col event format a26
SQL> select sid, seq#, event, state,
           2 seconds_in_wait seconds from v$session_wait
           3 where sid=42;
```

SID	SEQ#	EVENT	STATE	SECONDS
42	29,786	db file sequential read	WAITED SHORT TIME	174

Questo vuol dire forse che la sessione di Andrea è bloccata in attesa di un’operazione di I/O? No, anzi, vuol dire esattamente il contrario (rivedi il significato della colonna `STATE` in tabella 2). L’evento d’attesa più recente che il processo kernel di Oracle di Andrea ha eseguito è stato infatti un’operazione di “file read”, ma l’operazione si è conclusa approssimativamente 174 secondi fa (grossolanamente ogni 3 secondi). Inoltre, l’operazione si è conclusa in meno di un’unità di risoluzione del timer di Oracle (su un sistema Oracle 8i, questo vuol dire che il tempo dell’operazione di lettura trascorso è meno di 0.01 (un centesimo) secondi). Quindi, la sessione di Andrea cosa sta aspettando?

La risposta è che la sua sessione (in verità, il suo processo kernel Oracle) o sta lavorando internamente, consumando CPU, oppure sta aspettando il suo turno, nella coda "ready to run" per la sua successiva elaborazione, consumando comunque una quantità elevata di CPU. Puoi osservare cosa la sessione stia facendo, eseguendo queries successive sulla vista `V$SESS_IO`:

Come si vede, la colonna `WAIT_TIME`, calcolato come valore arrotondato alla parte intera del valore decimale, è zero, mentre la colonna `STATE` è diversa da `WAITING`.

⁵ Vedi il capitolo “Conclusioni”

```

SQL> col block_gets format 999,999,999,990
SQL> col consistent_gets format 999,999,999,990
SQL> select to_char(sysdate, 'hh:mi:ss') "TIME",
2  block_gets, consistent_gets from v$$sess_io
3  where sid=42;

```

TIME	BLOCK_GETS	CONSISTENT_GETS
05:20:27	2,224	22,647,561

```
SQL> /
```

TIME	BLOCK_GETS	CONSISTENT_GETS
05:20:44	2,296	23,382,994

Incorporando un timestamp all'interno della tua query, puoi percepire la rapidità con cui il tuo sistema Oracle può processare le chiamate LIO (Logical IO). Il sistema mostra che sono processate 735,433 (23,382,994-22,647,561) chiamate LIO in circa 17 (05:20:44-05:20:27) secondi, che porta ad un rapporto di 43,261 LIOs per secondo. Con queste informazioni, sei in grado di capire il problema di Andrea. Le oltre 22,000,000 LIO eseguite dal suo programma quando inizi ad osservarlo, hanno già consumato quasi nove minuti di tempo di esecuzione⁶. E' quindi arrivato il momento di scoprire qual è l'SQL che Andrea sta eseguendo in modo tale da perfezionarlo. Potresti utilizzare l'"extended SQL trace" per questa sessione, ma se non ti è possibile, allora l'utilizzo delle viste di sistema può aiutarti a risolvere il problema, facendo ad esempio un join tra la V\$OPEN_CURSOR e la V\$SQL.

Scoprire dove il sistema è bloccato

Qualche volta il telefono squilla, e prima che Andrea termina di descrivere il suo problema, noti un'altra chiamata sulla linea due. Praticamente, in due minuti devi ascoltare le lamentele di cinque utenti ed hai altrettanti messaggi in segreteria telefonica. Che fare? Un buon punto di partenza può essere la seguente query:

⁶ Se 43,261 sono le LIO per secondo, allora 22,647,561/43,261 = 523.1 è il numero di secondi impiegato dall'inizio dell'operazioni fino al momento della misura. Dimensionalmente infatti abbiamo:

$$[\text{LIO}]/([\text{LIO}/\text{sec}]) = [\text{sec}]$$

Dividendo il numero di secondi precedentemente ottenuti, per 60, esprimiamo il risultato in minuti:

$$523.1/60 = 8.7 \text{ min} \sim 9 \text{ min}$$

```

SQL> break on report
SQL> compute sum of sessions on report
SQL> select event, count(*) sessions from v$session_wait
  2  where state='WAITING' group by event order by 2 desc;

```

EVENT	SESSIONS
SQL*Net message from client	211
log file switch (archiving needed)	187
db file sequential read	27
db file scattered read	9
rdbms ipc message	4
smon timer	1
pmon timer	1
sum	440

Il report, mostra 440 sessioni connesse. All'istante di esecuzione della query, gli oltre 200 processi Oracle sono bloccati su una read sul socket SQL*Net mentre le corrispondenti applicazioni eseguono codice tra "database calls". Molti di questi 211 sono probabilmente inattivi mentre i loro utenti usano applicazioni non Oracle o parlano con colleghi. Ciò che preoccupa è che 187 sessioni sono bloccate, in attesa che "log file switch (archiving needed)" sia completato. Questo messaggio indica che il processo ARCH non riesce ad andare di pari passo con la generazione dei redo log online.

A few other users on the system are actually getting work done (36 are engaged in reading database files), but as each user attempts to execute a database COMMIT call, she'll get caught waiting for a log file switch (archiving needed) event. Più il problema non viene corretto, più gli utenti si bloccheranno aspettando per l'evento. Sul sistema dove questo output è stato ottenuto, il dba ha trascurato di dire che in questo particolare giorno, il file system del processo ARCH si sarebbe potuto riempire.

3. Conclusioni

3.1. V\$FIXED_VIEW_DEFINITION

Può essere difficile trovare le informazioni, di cui necessiti, su una vista V\$ dalle sole informazioni date da Oracle: talvolta ciò che cerchi semplicemente non è stato pubblicato. Altre volte trovi le informazioni che pensavi di cercare, ma invece erano sbagliate. Le pubblicazioni su Oracle sono inaffidabili per quanto riguarda il kernel, in quanto la sua evoluzione è molto veloce. Fortunatamente, il kernel è qualche volta auto documentato nell'ambito delle "fixed views". Un segreto risiede nel sapere come usare V\$FIXED_VIEW_DEFINITION. The hardest part is knowing its name:

```
SQL> desc v$fixed_view_definition
```

Name	Null?	Type
VIEW_NAME		VARCHAR2(30)
VIEW_DEFINITION		VARCHAR2(4000)

V\$FIXED_VIEW_DEFINITION è il mezzo attraverso cui è possibile imparare, per esempio le definizioni dettagliate delle colonne STATE e WAIT_TIME della V\$SESSION_WAIT. Puoi riprodurre il risultato attraverso pochi semplici passi. Inizia con l'eseguire la seguente query per ricevere la definizione della vista V\$SESSION_WAIT:

```
SQL> select * from v$fixed_view_definition
2 where view_name='V$SESSION_WAIT';
```

```
VIEW_NAME
-----
VIEW_DEFINITION
-----
V$SESSION_WAIT
select sid,seq#,event,p1text,p1,p1raw,p2text,p2,p2raw,p3text,
p3,p3raw,wait_time,seconds_in_wait,state from gv$session_wait
where inst_id = USERENV('Instance')
```

Così adesso sai che la V\$SESSION_WAIT è semplicemente una proiezione della GV\$SESSION_WAIT. Ovviamente questo non ti dice ancora molto. Il successivo passo è quello di visualizzare la definizione di GV\$SESSION_WAIT:

```
SQL> desc gv$session_wait
```

Name	Null?	Type
INST_ID		NUMBER
SID		NUMBER
SEQ#		NUMBER
EVENT		VARCHAR2(64)
P1TEXT		VARCHAR2(64)
P1		NUMBER
P1RAW		RAW(4)
P2TEXT		VARCHAR2(64)
P2		NUMBER
P2RAW		RAW(4)
P3TEXT		VARCHAR2(64)
P3		NUMBER
P3RAW		RAW(4)
WAIT_TIME		NUMBER
SECONDS_IN_WAIT		NUMBER
STATE		VARCHAR2(19)

```
SQL> select * from v$fixed_view_definition
  2  where view_name='GV$SESSION_WAIT';
```

```
VIEW_NAME
```

```
VIEW_DEFINITION
```

```
GV$SESSION_WAIT
select s.inst_id,s.indx,s.ksusseq,e.kslednam,
e.ksledp1,s.ksussp1,s.ksussplr,e.ksledp2,
s.ksussp2,s.ksussp2r,e.ksledp3,s.ksussp3,s.ksussp3r,
round(s.ksusstim / 10000), s.ksusewtm, decode(s.ksusstim,
0, 'WAITING', -2, 'WAITED UNKNOWN TIME', -1, 'WAITED SHORT
TIME', 'WAITED KNOWN TIME') from x$ksusecst s, x$ksled e
where bitand(s.ksspaflg,1)!=0 and bitand(s.ksuseflg,1)!=0
and s.ksusseq!=0 and s.ksussopc=e.indx
```

Qui puoi vedere l'operazione di arrotondamento utilizzato per calcolare WAIT_TIME. Da ciò che vedi, puoi anche determinare l'unità di misura nella quale, la cosa detta X\$KSUSECST.KSUSSTIM, è espressa. Sappiamo che WAIT_TIME è riportata in centesimi di secondo, e sappiamo anche che il kernel divide questo valore per 10^4 (10000) per produrre un valore in centesimi di secondo. Perciò, ci sono 10^k KSUSSTIM unità in un secondo, dove $10^k/10^4 = 10^2$. Quindi, ci sono 10^6 KSUSSTIM in un secondo. In alter parole, il kernel calcola il tempo di attesa in microsecondi, ma la API pubblica (la V\$SESSION_WAIT) fornisce il risultato in centesimi di secondo.

3.2. gettimeofday(): descrizione

Se la portabilità è l'obiettivo principale, allora devi utilizzare la funzione *gettimeofday*: questa fornisce il "wall clock time" e può essere utilizzata come misuratore del trascorrere del tempo.

Al di là del fatto che tu acceda alle statistiche di timing del kernel di Oracle attraverso le "extended SQL trace data", le viste V\$, o puntando direttamente ai segmenti di memoria condivisa sottesi dalle V\$, le statistiche di timing a cui accedi sono ottenute utilizzando un semplice insieme di funzioni del sistema operativo. Al di là poi di quale interfaccia usi per accedervi, le stative di timing sono soggette alle limitazioni insite nei timers del sistema operativo, utilizzati per produrli. Per apprezzare appieno i dati temporali che Oracle rivela, è necessario comprendere i sevizi che un sistema operative fornisce al kernel di Oracle.

Partiamo allora dalla definizione della funzione *gettimeofday*, osservando direttamente le pagine del man (quì viene fatto riferimento anche alla funzione *settimeofday*() a cui tuttavia non siamo particolarmente interessati):

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

La funzione `gettimeofday` può ottenere sia un tempo che un `timezone`. L'argomento "tv" è una struttura di tipo "timeval, così come specificato in `/usr/include/sys/time.h`:

```
struct timeval {
    long tv_sec;           /* seconds since Jan. 1, 1970 */
    long tv_usec;        /* and microseconds */
};
```

e restituisce il numero di secondi e microsecondi a partire da Epoch (vedi anche `time(2)`). L'argomento "tz" invece è un `timezone`:

```
struct timezone {
    int tz_minuteswest; /* minutes W of Greenwich */
    int tz_dsttime;     /* type of dst correction to apply */
};
```

Descrizione

Il corrente "Greenwich time" ed il corrente "time zone" sono mostrati con la funzione `gettimeofday` (ed impostati dalla funzione `settimeofday`). Il tempo è espresso in secondi e microsecondi a partire dalla mezzanotte (ora 0), del Gennaio 1970. La risoluzione del clock di sistema è hardware-dipendente, ed il tempo può essere aggiornato o in modo continuo a in "ticks"⁷. Se il parametro `tz` o `tv` ha un valore nullo (NULL), la corrispondente l'informazione di tempo non è restituita o impostata. Il parametro `tv` restituisce un puntatore ad una struttura **timeval** che contiene il tempo, in secondi e microsecondi, misurato dall'inizio di epoch.

Parameteri

`tv` punta ad una structure **timeval**, definite nel file `sys/time.h`

`tz` punta ad una structure **timezone**, definite nel file `sys/time.h`

Vaolori restituiti

Se la funzione ha successo, viene restituito 0 (zero), mentre se c'è un errore, alloare viene restituito -1 ed **errno** è impostato per indicare tale errore.

⁷ In inglese ticks viene tradotto come: tic tac, ticchettio. In questo caso si potrebbe anche tradurre come "istante discreto"

Codici di errore

Se la funzione `settimeofday` non ha successo, il valore di `errno` è impostato a `EPERM` per indicare che l'effettivo user ID del processo non ha i privilegi `ri root`; è impostato a `EINVAL` se la struct puntata da `tp` specifica un tempo invalido.

Implementazione

Il kernel di Oracle ricava tutte le sue statistiche sul tempo come risultato di una chiamata di sistema eseguita dal sistema operativo stesso. L'esempio che segue, mostra come un programma del kernel di Oracle, conta e calcola la durata delle proprie azioni:

```
t0=gettimeofday;          /* marca il tempo immediatamente
                           prima di fare qualcosa */
doSomething();
t1 = gettimeofday;       /* marca il tempo immediatamente
                           dopo aver fatto qualcosa */
t = t1 - t0;             /* è approssimativamente la durata
                           di doSomething */
```

Esempio 3.1

Nel paragrafo 3.3 riporto un esempio in codice C. Se non sei interessato ai dettagli puoi tranquillamente saltare tale sezione.

Immaginiamo l'esecuzione dell'esempio precedente su linea temporale.

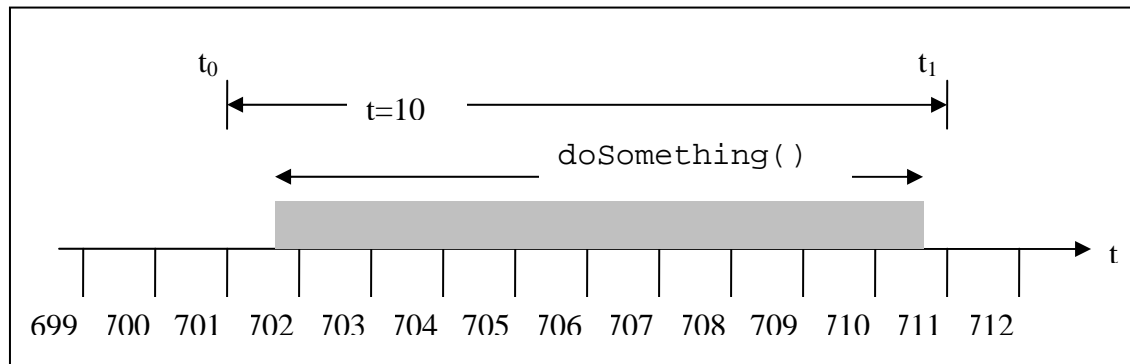


Figura 3.1

Come mostrato, quando la funzione `doSomething()` inizia, il valore del "clock tick" di `gettimeofday` è $t_0=701$. Quando invece `doSomething()` termina, il valore del "clock tick" di `gettimeofday` è $t_1=711$. Questo vuol dire che la durata della funzione `doSomething()` è $t = t_1 - t_0 = 10$ "clock ticks".

Il kernel di Oracle pubblica solo alcuni differenti tipi di informazione sul timing. L'output dell'"extended SQL trace" ne contiene 4 molto importanti che sono

visibili nell'esempio generato da un Oracle 9.0.1.2.0 su piattaforma SUN con Solaris 5.6:

```
WAIT #9: nam='db file sequential write' ela= 17378 p1=7 p2=3422 p3=2
FETCH #9:c=0,e=18112,p=1,cr=4,cu=0,mis=0,r=1,dep=2,og=4,tim=1010033275346418
```

Le due righe mostrate descrivono una singola "db call", FETCH. Il significato delle statistiche di tempo sono descritte di seguito:

ela= 17378 Il kernel di Oracle ha consumato un periodo di tempo pari a 17378 microsecondi nell'esecuzione di una "system call" chiamata "db file sequential write"

c=0 Il kernel di Oracle riporta che la fetch della "database call" ha consumato 0ms della capacità totale della CPU

e=18112 Il kernel di Oracle riporta che la fetch ha consumato 13112ms del tempo trascorso

tim=1010033275346418 Il tempo di sistema quando la fetch si è conclusa era di 1010033275346418 (espresso in microsecondi, trascorsi a partire dalla mezzanotte UTC del 1 gennaio 1970).

La durata totale della fetch include sia il consume della capacità totale della CPU che qualunque periodo di tempo che la fetch ha consumato durante l'esecuzione dell'evento di attesa. Guardando il l'esempio, le statistiche di tempo sono legate attraverso l'approssimazione:

$$e \approx c + ela$$

In questo caso, l'approssimazione è molto buona:

$$18112 \approx 0 + 17378$$

con uno scarto di 0.000734 (18112-17378) secondi.

Ovviamente in questo caso sono state riportate solo due righe, ma nella realtà una singola "database call" emette diverse linee che contengono eventi di attesa. Per tale motivo, la relazione che lega i valori di *e*, *c* ed *ela* per una "database call" deve essere la somma dei valori di *ela* prodotti dentro il contesto di una data "database call". Ad esempio se abbiamo:

```
WAIT #9: nam='db file sequential write' ela=900 p1=50 p2=9052 p3=1
WAIT #9: nam='db file sequential write' ela=1500 p1=43 p2=5673 p3=1
WAIT #9: nam='db file sequential write' ela=15378 p1=7 p2=3422 p3=2
FETCH #9:c=0,e=18112,p=1,cr=4,cu=0,mis=0,r=1,dep=2,og=4,tim=1010033275346418
```

allora:

$$e \approx c + \Sigma ela$$

$$18112 \approx 0 + 15378 + 1500 + 900$$

con uno scarto di 0.000334 secondi.

Oracle kernel riporta due tipi di tempo trascorso: la statistica “*e*” denota la durata di una singola “database call”, mentre la statistica “*ela*” denota la durata di una sequenza di istruzioni (spesso una system call) eseguita da un processo Oracle. Il kernel esegue questi calcoli eseguendo il codice strutturato grossolanamente come di seguito. Nota che il kernel usa il metodo mostrato nell’esempio 3.1 come blocco base per la costruzione delle metriche “*e*” ed “*ela*”.

```
procedure dbcall {
e0 = gettimeofday; # marca il wall time
... # esegue la db call (può chiamare wevent)
e1 = gettimeofday; # marca il wall time
e = e1 - e0; # tempo trascorso della db call
print(TRC, ...); # emette la linea PARSE, EXEC, FETCH
}

procedure wevent {
ela0 = gettimeofday; # marca il wall time
... # esegue il wait event
ela1 = gettimeofday; # marca il wall time
ela = ela1 - ela0; # ela è la durata del wait event
print(TRC, "WAIT..."); # emette la linea di WAIT
}
```

Esempio 3.2

Il kernel non solo riporta il tempo trascorso “*e*” per ogni database call ed “*ela*” per ogni system call, ma anche la quantità “*c*” di capacità totale di CPU consumata da ogni database call. Sui sistemi operativi POSIX-compliant, il kernel ottiene informazioni dell’uso di CPU da una funzione chiamata *getrusage*⁸ (su Linux e molti altri sistemi operativi) o una funzione simile detta *times* su HP-UX e pochi altri sistemi.

Il kernel calcola “*c*”, “*e*” ed “*ela*” eseguendo il codice rappresentato sommariamente nell’esempio 3.3. Nota che questo esempio è costruito sopra quello dell’esempio 3.2 includendo l’esecuzione della chiamata di sistema *getrusage*. In modo analogo al calcolo di *gettimeofday*, il kernel marca la quantità di tempo di CPU in user-mode consumato dal processo all’inizio della “database call” ed alla fine. La differenza tra i due marchi (*c0* e *c1*) è quantità approssimata della capacità di CPU in user-mode consumata dalla “database call”.

⁸ Vedi paragrafo 3.4

```

procedure dbcall {
e0 = gettimeofday; # marca il wall time
c0 = getrusage; # ottiene la statistica dell'uso della
risorsa
... # esegue la db call (può chiamare wevent)
c1 = getrusage; # ottiene la statistica dell'uso della
risorsa
e1 = gettimeofday; # marca il wall time
e = e1 - e0; # tempo trascorso della db call
c = (c1.utime + c1.stime) - (c0.utime + c0.stime);
# tempo totale di CPU consumata dalla
dbcall
print(TRC, ...); # emette la linea PARSE, EXEC, FETCH
}

procedure wevent {
ela0 = gettimeofday; # marca il wall time
... # esegue il wait event
ela1 = gettimeofday; # marca il wall time
ela = ela1 - ela0; # ela è la durata del wait event
print(TRC, "WAIT..."); # emette la linea di WAIT
}

```

Esempio 3.3

3.3. gettimeofday(): codice C

Nel precedente paragrafo, è stato fatto come esempio in linguaggio naturale, un'applicazione della funzione gettimeofday(). Il codice era:

```

t0 = gettimeofday;
doSomething();
t1 = gettimeofday;
t = t1 - t0;

```

Adesso invece, mostro il codice C testato su una macchina Linux con kernel 2.4.18-24.8.0. Riporto due esempi che fanno esattamente la stessa cosa, ma lo fanno in modo leggermente diverso: nel primo caso, il tempo misurato prima e dopo il "fare qualche cosa" (doSomething) viene memorizzato in variabili di appoggio e successivamente ne viene fatta la differenza per calcolare il tempo trascorso. Nel secondo caso invece, non viene utilizzata alcuna variabile di appoggio, ma viene sfruttata la macro *timersub* definita in /usr/include/sys/time.h:

Prima versione

```

#include <stddef.h> /* definizione di NULL */
#include <sys/time.h> /* definizione della struct timeval */
#include <stdio.h>

```

```

main(){
double t1,t2,elapsed;
struct timeval tp;
int rtn;

rtn=gettimeofday(&tp, NULL);
t1=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;
printf("The value of t1 is: %lf\n",t1);

printf ("I'm writing something\n");

rtn=gettimeofday(&tp, NULL);
t2=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;
printf("The value of t2 is: %lf\n\n",t2);

elapsed=t2-t1;

printf("The elapsed time is: %lf\n", elapsed);
}

```

Il codice scritto sopra non è nulla di complesso: è abbastanza autoesplicativo. Forse quello che c'è da spiegare è come vengono calcolati i valori di t1 e t2. Sappiamo che &tp altro non è che l'indirizzo di una struttura di tipo timeval. Questa struttura è costituita da due parti: la prima esprime i secondi, mentre la seconda i microsecondi. Per poter ottenere quindi il tempo complessivo, dobbiamo sommare le due quantità:

$$\begin{aligned} \text{timeTOT [sec]} &= \text{tv_sec [sec]} + \text{tv_usec [\mu\text{sec}]} \\ 1 \mu\text{sec} &= 10^{-6} \text{ sec} \\ \text{timeTOT [sec]} &= \text{tv_sec [sec]} + \text{tv_usec} * 10^{-6} [\text{sec}] \end{aligned}$$

e poiché in C, 10^{-6} (10 alla meno sei) si esprime come 1.e-6, ecco spiegata l'espressione per t1 e t2. Il cast a double in questo caso è necessario per come sono definiti tv_sec e tv_usec.

Seconda versione

```

#include <stddef.h>      /* definizione di NULL */
#include <sys/time.h>    /* definizione della struct timeval */
#include <stdio.h>

main(){
double t1,t2,elapsed;
struct timeval tpstart, tpend, tpdiff;
int rtn;

rtn=gettimeofday(&tpstart, NULL);

```

```

printf("The value of t1 is: %lf\n",
      (double)tpstart.tv_sec+(1.e-6)*tpstart.tv_usec);

printf ("I'm writing something\n");

rtn=gettimeofday(&tpend, NULL);
t2=;
printf("The value of t2 is: %lf\n",
      (double)tpend.tv_sec+(1.e-6)*tpend.tv_usec);

timersub (&tpend, &tpstart, &tpdiff);

elapsed=(double)tpdiff.tv_sec+(1.e-6)*tpdiff.tv_usec;

printf("The elapsed time is: %lf\n", elapsed);
}

```

Rispetto a quanto già detto per la prima versione, non c'è molto da aggiungere. La differenza con la prima risiede nel fatto che non vengono utilizzate le variabili di appoggio t1 e t2. Piuttosto utilizzato la macro `timersub` che ha come parametri 3 indirizzi: i primi due puntano a quanto valorizzato dalle due `gettimeofday` (l'ultimo ed il primo nell'ordine), mentre il terzo memorizza il risultato della differenza dei primi due. Tale differenza viene poi memorizzata nella variabile `elapsed` (per la quale vale quanto già detto per i valori di t1 e t2) utilizzata poi per stamparne il risultato.

Output

Una volta scritto il codice, basta compilare il programma con il gcc e lanciare l'eseguibile ottenuto. Per entrambe le versioni, l'output generato è il seguente:

```

The value of t1 is: 1114697995.804346
I'm writing something
The value of t2 is: 1114697995.804497

The elapsed time is: 0.000151

```

3.4. `getrusage()`: descrizione

Come per la funzione `gettimeofday`, anche per `getrusage` riporto del codice C di esempio. Prima tuttavia conviene darne la descrizione e lo facciamo partendo anche in questo caso dalle pagine del man.

```

#include <sys/resource.h>
int getrusage(int who, struct rusage *r_usage);

```

DESCRIPTION

The `getrusage()` function provides measures of the resources used by the current process or its terminated and waited-for child processes. Se il valore dell'argomento `who` è `RUSAGE_CHILDREN`, information is returned about resources used by the current process. If the value of the `who` argument is `RUSAGE_CHILDREN`, information is returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for instance, if the parent has `SA_NOCLDWAIT` set or sets `SIGCHLD` to `SIG_IGN`), the resource information for the child process is discarded and not included in the resource information provided by `getrusage()`.

The `r_usage` argument is a pointer to an object of type `struct rusage` in which the returned information is stored. The members of `rusage` are as follows:

```

struct timeval ru_utime; /* tempo usato dall'utente */
struct timeval ru_stime; /* tempo usato dal sistema */
long          ru_maxrss; /* maximum resident set size */
long          ru_idrss; /* integral resident set size */
long          ru_minflt; /* page faults not requiring
                        physical I/O */
long          ru_majflt; /* page faults requiring physical
                        I/O */
long          ru_nswap; /* swaps */
long          ru_inblock; /* block input operations */
long          ru_oublock; /* block output operations */
long          ru_msgsnd; /* messages sent */
long          ru_msgrcv; /* messages received */
long          ru_nsignals; /* signals received */
long          ru_nvcsw; /* voluntary context switches */
long          ru_nivcsw; /* involuntary context switches */

```

I membri della struttura hanno il seguente significato:

`ru_utime`

La quantità totale di tempo speso nell'esecuzione in user mode. Il tempo è dato in secondi e microsecondi.

`ru_stime`

La quantità totale di tempo speso nell'esecuzione in system mode. Il tempo è dato in secondi e microsecondi.

3.5. `getrusage()`: codice C

4. Ringraziamenti

Un particolare grazie va a Federico Proietti che mi ha aiutato sapientemente nella stesura di questo documenti: da solo non sarei stato in grado di completarlo correttamente.

5. Riferimenti

✓ Libri

[Optimizing Oracle Performance](#), di Cary Millsap & Jeff Holt.

[Oracle Wait Interface: A Practical Guide to Performance Diagnostics & Tuning](#), di Richmond Shee, Kirtikumar Deshpande, K. Gopalakrishnan

[Oracle8i Internal Services](#) di Steve Adams

V\$SESSION_WAIT (draft)