Chapter 14

CMMD Timers

The CMMD library provides a set of timing routines for use by the nodes in message-passing programs. (Timers for the host are not provided.)*

Each node manages its own timers. The timers are called by a single node and record the time spent by that node on whatever block of code the programmer wishes timed. They have no connection with timers used on any other node.

Like other CM timers, these timers measure three values:

- Busy time is the time during which the user program is executing user code.
- Idle time is the time during which the user program is looping in the operating system's dispatch loop. A CMMD program may have very little (if any) idle time.
- Elapsed time is the sum of busy time and idle time; that is, it is the total time during which the program was scheduled for execution on the CM. (It does not include time during which the program was swapped out.)

Each node may have up to 64 timers running simultaneously. The timers are identified by the integers 0 through 63.

Timers can be nested. Thus, one timer can provide timings for an entire program, while a second timer provides timings for a particular routine within the program, and a third timer provides timings for a particular code block — such as an if-then-else block or a case statement — within the routine. Each timer can record timings of up to 43 hours.

---

* CM Fortran and C* programs may use either CMMD timers or CMOST timers, but *not* both in the same program. For nodal programs, the two sets of timers behave alike.

14.1 Using the CMMD Timers

All timers share the same general pattern of use:

- First, you must call **`CMMD_node_timer_clear`** with an integer timer-ID. This creates a timer and initializes it to zero.
- Then you call **`CMMD_node_timer_start`**, to start the timer going.
- The next call to the timer should be **`CMMD_node_timer_stop`**. This records the current values within the timer. The values remain present, to be read, until the next time **`CMMD_node_timer_stop`** is called.
- The call to **`CMMD_node_timer_stop`** can thus be followed by calls to information-reading functions and by a call to **`CMMD_node_timer_start`**; these calls can be issued whenever you like. Timings for this particular timer will be cumulative until you call **`CMMD_node_timer_clear`** again.

Note: The most accurate results are obtained when your program has exclusive use of the partition. System load under timesharing can affect program timings.

14.2 Node Timing Functions

The following functions may be called by any node to manipulate timers for that node. No node can manipulate another node's timers.

Nodes can, however, use timers cooperatively; for example, by performing global operations on the results read from local timers. Cooperative use of local timers can thus achieve global timing across the nodes.

FUNCTION **`CMMD_node_timer_clear`**

C SYNTAX

int **`CMMD_node_timer_clear(`**int *timer***`)`**

FORTRAN SYNTAX

INTEGER FUNCTION **`CMMD_node_timer_clear(`***timer***`)`**
integer *timer*

ARGUMENT

timer	An integer from 0 to 63, identifying the timer to be manipulated.

RETURNS

The function returns 0 for success, –1 for an out-of-range timer number.

DESCRIPTION

This function sets the total elapsed time, total CM busy time, and number of starts for the specified timer to zero. It must be the first function called for any timer.

The function returns –1 if the timer argument is out of range (that is, greater than the allowable number of timers); otherwise, it returns 0.

FUNCTION **CMMD_node_timer_start**

C SYNTAX:

int **CMMD_node_timer_start(**int *timer***)**

FORTRAN SYNTAX:

INTEGER FUNCTION **CMMD_node_timer_start(***timer***)**
integer *timer*

ARGUMENT:

timer An integer from 0 to 63, identifying the timer to be manipulated.

RETURNS:

The function returns 0 for success, –1 for an out-of-range timer number.

DESCRIPTION:

This function starts the clock running for the specified timer. Elapsed time and CM busy time are accumulated. Number of starts is increased by one. (If you wish to zero out these values before restarting a timer, you must first call **CMMD_node_timer_clear**.)

FUNCTION **`CMMD_node_timer_stop`**

C SYNTAX:

int **`CMMD_node_timer_stop(`**int *timer***`)`**

FORTRAN SYNTAX:

INTEGER FUNCTION **`CMMD_node_timer_stop(`***timer***`)`**
integer *timer*

ARGUMENT:

timer An integer from 0 to 63, identifying the timer to be manipulated.

RETURNS:

The function returns 0 for success, –1 for an out-of-range timer number.

DESCRIPTION:

This function stops the clock running for the specified timer and updates the timer's state variables for CM elapsed time and CM busy time. Once the timer is stopped, information can be read from it; the timer can then be cleared and/or restarted. (If it is restarted without being cleared, times will be accumulated. Thus, if you start a particular timer each time a routine begins and stop it each time the routine finishes, you can read, at the end of the program, the total amount of time spent inside the routine over the course of the program.)

FUNCTIONS **CMMD_node_timer_elapsed**
CMMD_node_timer_busy
CMMD_node_timer_idle

C SYNTAX:

double **CMMD_node_timer_elapsed(**int *timer***)**
double **CMMD_node_timer_busy(**int *timer***)**
double **CMMD_node_timer_idle**(int *timer***)**

FORTRAN SYNTAX:

REAL*8 FUNCTION **CMMD_node_timer_elapsed(***timer***)**
integer *timer*

REAL*8 FUNCTION **CMMD_node_timer_busy(***timer***)**
integer *timer*

REAL*8 FUNCTION **CMMD_node_timer_idle(***timer***)**
integer *timer*

ARGUMENT:

timer An integer from 0 to 63, identifying the timer to be read.

RETURNS:

These functions return the elapsed, busy, or idle time, respectively, (in seconds) recorded by the specified timer between the last time the timer was cleared and the last time it was stopped.

Elapsed time is the total time during which this timer was active. Busy time is the time during which the timer was active and the program was actually executing, within its own OS timesharing slice, on the calling node. Idle time is the time during which the timer was active but the program was waiting in the dispatch loop. (There may be very little of this last type of time in many CMMD programs.)