Clocks

Three uses:

1. Timer runout on long-running programs; used by scheduler to switch processes.

2. Time-of-day and interval calculations for time-accounting

3. Wake-up of sleeping processes ~~on a timeout~~ as on alarm clock.

Possibilities

1. Use internal timer/CPU for 1,
Integrate " " for 2,
Simulate with " " for 3
Difficulties:
a. must send in an initialization time.
b. will lose ~3 min./day in breakage from lost ticks.
c. Any given processor may drift away from another — creating time accounting problems.
d. Simulation of wakeup requires considerable overhead coding.

2. A synchronized time-of-day register / processor, read only, guaranteed identical in all processors. Ticks every 15.625 μs, just like interval timer.

Problems: Suppose clock is manually reset by operator?

Still doing wake-up simulation.

3. Wake-up clock in ~~[illegible]~~ G1ΦC

allows sloppy simulation or a backup.

(Sloppy means wakeup at nearest timer reset.)

Ideal: Send a time-interval + data word (process id) at any time.

When time-interval elapses, sends back data word and an interrupt; can stack any number of requests.

Satisfactory: One time-interval at a time, no data word.

Program must handle wakeups + stacking.

Need: ~~the~~ ability to kill earlier request to slip in new one.

VI CPU Time Keeping

1. There is an Interval Timer per processor named CLOCK(I)

2. Each processor maintains a cell STOP_TIME(I) which is the time-of-day of the next ~~trap~~ clock trap. This cell is always updated simultaneously with the clock of the processor.

3.

3. The exact time of day is then

   TIME_NOW = STOP_TIME(I) - CLOCK(I)

   even if the clock trap is being inhibited.

4. For accounting purposes, the cell START_TIME(I) is always the time that accounting was last done for this processor. It is updated by storing TIME_NOW, computed as in 3) above. Time used is ~~→~~ TIME_USED = ~~T~~IMENOW - START_TIME(I)
   (the same technique is used in term-accounting

VII Alarm Clock Time-Keeping

1. Master program keeps track of all wake-up requests.

   call WAKE_ME (INTERVAL)

   puts an entry into a sorted table to wake this process at TIME_NOW + INTERVAL. It blocks the process.

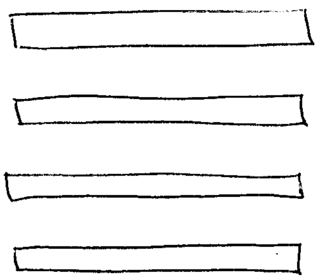
2. On every Interrupt, every processor calls

   call AWAKE (run-interval, wake-interval, sleeper) ~~(wake-interval)~~

   which returns WAKE_TIME - TIME_NOW of the least sleeping ~~wor~~ process in the list

3. If wake-interval is smaller than the assigned portion of the process about to be run, the interval timer is set for the shorter period, the difference is stored in ~~overtime(I)~~ OVER_TIME(I), and the controlling process for a clock-fault is set to be the sleeper.

4. If not, waketime is set to zero, and the controlling process for a clock-fault is set to be the cpu-process.

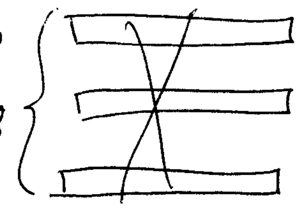


master clock cell

1 clock / processor = time of ~~last~~ next clock trap

2 ~~Time of next trap~~

3



for any processor,

TIME NOW = TIME OF NEXT TRAP + c(Interval timer)

rule: at every trap

1. Compute TIME NOW
2. Compare with master clock.

TIME NOW $\geq$ master clock	time now < master clock
reset master clock to TIME NOW	reset time of next trap to ~~processor clock to~~ master clock + c(Interval timer)

Is Interval Timer still 24 bits?

this will overflow in 16 sec. if increment every μsec.