5/17/68

TO: F. J. Corbató
R. C. Daley
J. H. Saltzer

FROM: R. Rappaport, M. Spier

SUBJECT: Timing test of New Traffic Controller

An experiment has been performed that allowed us to meter the amount of time spent executing the various traffic controller functions. The experiment consisted of having two processes cooperate in waking up each other, blocking themselves, notifying each other, etc. The two processes followed different scripts which consisted of a number of calls into the traffic controller. The scripts are reporduced below. At the start process A is running and process B is blocked:

Process A script	Process B script
1. call pxss$wakeup (B)	
2. call pxss$block	3. call pxss$wakeup (A)
	4. call pxss$block
5. call pxss$wakeup (b)	
6. call pxss$block	
	7. call pxss$wakeup (A)
	8. call pxss$notify (10)
	9. call pxss$addevent (10)
	10. call pxss$wait (10)
11. call pxss$notify (10)	
12. call pxss$block	
	13. call pxss$notify (11)

14. call pxss$addevent (11)

15. call pxss$delevent (11)

16. call pxss$wait (11)

17. call pxss$wakeup (A)

18. call pxss$block

19. go to 1

20. go to 3

The experiment went as follows. In steps 1 and 2, A wakes up B and then calls block. This results in A giving up the processor and B beginning to run. Steps 3 and 4, 5 and 6 are similar. After step 6, B is running and A is blocked. B then wakes up A (step 7), notifies an event for which no one is waiting (step 8), creates event 10 (step 9) and waits for it (step 10). In waiting for the event, B gives up the process and A (now ready because of step 7) starts running. A now notifies all those waiting for event 10 (i.e., B is now on ready list) and blocks itself causing B to run. B then proceeds to notify an event for which no one is waiting (step 13), create an event (step 14), delete the event (step 15), wait for the deleted event (step 16) which results in an immediate return, wakeup A (step 17) and finally go blocked. The looping instructions cause the scripts to be reexecuted.

In the run extra instructions were placed in the Traffic Controller which had the effect of storing the value of the calendar clock (into an array) upon entry to all the entry points of the traffic controller. As a result, the time to wakeup a blocked process could be obtained by subtracting the time stored at step 1 from that at step 2 or similarly subtracting the time at step 3 from the time at step 4, etc. The storing of the clock times and the maintenance of array pointers added the execution of 15 instructions to each call. The results of the subtractions for three loops through the scripts were as follows. Notice the circled out of place numbers. These were shown to be caused by clock interrupts triggered by the metering routines. Times are in micro seconds.

1. Time to wakeup a blocked process:

	T_2-T_1	T_4-T_3	T_6-T_5	T_8-T_7	T_{18}-T_{17}
loop1	879	908	877	911	904
loop2	882	904	878	911	904
loop3	882	904	1028	911	904

Someone has a grudge against B.

2. Time to block (I.e., to block self, choose a successor to run, switch to him and return to where he blocked himself.)

	T_3-T_2	T_5-T_4	T_7-T_6
loop1	1379	1558	1379
loop2	1389	1407	1379
loop3	1389	1407	1379

3. Time to wait for something (i.e., put self in wait state, choose successor, switch to him, and return to his procedure)

	T_{11}-T_{10}
loop1	1277
loop2	1277
loop3	1277

4. Time to wait for nothing (i.e., an event that doesn't exist, perhaps all ready notified)

	$T_{17}-T_{16}$
loop1	230
loop2	230
loop3	230

5. Time to notify an event for which someone is waiting

	$T_{12}-T_{11}$
loop1	340
loop2	510
loop3	340

6. Time to notify an event for which no one is waiting

	T_9-T_8	$T_{14}-T_{13}$
loop1	237	237
loop2	237	237
loop3	237	237

7. Time to add an event

	$T_{10}-T_9$	$T_{15}-T_{14}$
loop1	231	227
loop2	231	227
loop3	231	227

8. Time to delete an event

	$T_{16}-T_{15}$
loop1	235
loop2	235
loop3	235

9. An additional experiment was performed for each of the processes. They were allowed to run out their time quantuns, reschedule themselves, choose a successor which turned out to be themselves and switch back to themselves. This took 1454 msecs for process A and 1511 msec for B.

We can interpret these figures in the following way. It takes about 1.3-1.4 milliseconds to give a processor away. This comes from looking at the block, wait, and restart times. It takes about 850 msecs to wakeup a blocked process. Total block-wakeup time is about 2 milliseconds. Notifying seems to take about 200 mseconds plus ~ 100 msecs per process waiting. Total addevent, wait, notify time takes about 1.8 milliseconds.