bcc	title MICRO-SCHEDULER IMPLEMENTATION		prefix/class-number.revision USI/W-14
checked [illegible]	authors Butler W. Lampson [illegible]	approval date 6/20/69	revision date
checked		classification Working Paper	
approved [illegible]		distribution Company Private	pages 9

ABSTRACT and CONTENTS

The implementation of all the functions performed by the μscheduler is described. These include the handling of wakeup, switching CPUs from one process to another, and managing of the real-time queue.

TABLE OF CONTENTS

Introduction

Most of the functions performed by the μscheduler (US) are described in PRUN/W-8 with little reference to how they are implemented; in that document only the interfaces between the μscheduler and the outside world are specified in detail. Here, on the other hand, all this information is assumed and the microcosmology of the μscheduler is expounded.

Organization

The US runs on its own microprocessor. It communicates with the outside world only through core memory and the attention (ATTN) and protect (PRO) signals between microprocessors. It is programmed to handle the microscheduling of two CPUs. It uses the following objects in core memory for communication with the world.

USIB	the μscheduler input buffer, stack starting at USIBASE and ending at USIBEND ≡ Ø mod 2**n. It has a top pointer USIBTOP used by everyone.
CPUi	Cells which contain the addresses of PRT entries for processes to be run by the CPU.
PRT	The process table, described in
RTQ	The real-time queue, chained through PRRTP in PRT.
RTC	The real-time clock
SWAPRQ	The swapper request queue, and its associated freelist, described in MMI/W-1.
WAKEUPQ	The wakeup queue for the scheduler. This is actually a stack.

It also maintains the following objects for its own use in core and in its scratchpad.

USQj	8 queues, representing different priority levels, of processes which are in core and ready to run
ALi	Activity levels for the CPUs
PRIi	Current priorities for the CPUs
PROCi	Current running processes for the CPU
SCHFLG	Schedule flag - set when the allocation of processors should be reconsidered
WNPFLG	Normally -1. When set, contains a CPU number i if the US is waiting for ALi to drop below P.

The US is normally in an idle state, from which it is roused by the occurrence of an ATTN. It then proceeds as follows:

clears ATTN

checks RTQ and wakes up any processes whose specified real time has elapsed

checks USIB and deals with all the calls it finds there

if any changes have been made to USQj or to the processes running, recomputes the processes which should run and does appropriate switches if necessary

returns to wait for another ATTN, which may, of course, have come in during the processing just described.

We now proceed to describe these operations in detail. Management of USIB is described in PRUN/W-8.

Microscheduler Calls

WAKEUP

Check the process id (PRT address) in the call for validity; ignore it if invalid. Merge the data word in the call into PIW for the process; this is done under a protect. Clear BLK. Now check LDD. If on, check MSQ OR RUN. If true, done. Otherwise, set MSQ, fetch UPRI and put the process on the end of the indicated queue, USQ[UPRI]. Each queue is defined by a word pointing to the last process on the queue. Each process points to its successor with SCHPTR; the last process points to the first. The pointer is Ø if the queue is empty. If the queue was empty, set SCHFLG. This flag is tested at the end of the μscheduler's cycle to determine whether the choice of processes to run should be reconsidered.

If LDD is off, check WAQ OR SCQ OR SWAPIN. If true, done. Otherwise put the process on the wakeup queue. The head of this queue is kept in memory in the form of a pointer to the top entry, or Ø if it is empty. It is actually a stack, since a process is added to the front of the queue and removed the same way.

Before putting a process on WAKEUPQ, whether in WAKEUP or for another reason, the US makes a check to ensure that the scheduler will run to take it off. It tests for the truth of the following conditions both CPUs idle

all USQs empty

the cell RIPQ, which is the head of the swapper's queue of processes being read in, is Ø.

If all hold, it gives the process to the swapper with a SWAPIN call instead of putting it on WAKEUPQ. In this case it sets RSI and SWQ rather than WAQ.

IWAKEUP

This is identical to WAKEUP except that the PRID field of the call is interpreted as a PRT index, i.e. it is converted to the address of a PRT entry by PRID*n+b, where n is the size of a PRT entry and b is the origin of PRT. This option is provided for the convenience of processors which prefer to deal with a process id which contains fewer bits than an 18-bit absolute address. A PRT index is limited to 12 bits.

RETURN

Interpret the data word as a CPU number i. Decrement ALi and set SCHFLG if it is I. Clear RUN. Put the process on the proper USQi as for WAKEUP, and set MSQ.

BLOCK

Interpret the data word as a CPU number i. Examine PIW. If it is non-zero, proceed as for return. Otherwise decrement ALi, clear RUN, set BLK. If ALi = I, set SCHFLG.

BLOCKOUT

Proceed as for BLOCK. If BLK gets set, make a SWAPOUT call as follows: clear LDD, create a swapper request node, put into it a request to swap out the process and put it on SWAPRQ.

UNLOAD

Interpret the data word as a CPU number, i. Decrement ALi and set SCHFLG if it is I, clear RUN. Make a SWAPOUT call. Put the process on WAKEUPQ and set WAQ.

GIVEUP

Look for a process on USQi, i $\geq$ 4, starting with USQ7. If one is found which does not have RES set, remove it from USQi, clear MSQ, make a SWAPOUT call, put the process on WAKEUPQ and set WAQ.

CHANGERT

Described under real-time below.

CPU Scheduling

After emptying USIB, the US checks WNPFLG. If it is $\geq$ Ø, we are waiting for AL[WNPFLG] to drop below P. If it has done so, set WNPFLG ← -1 and proceed. Otherwise loop to wait for ATTN. If WNPFLG < Ø, or AL[WNPFLG] $\neq$ P, check SCHFLG. If it is set, there is a chance that the processors should be re-allocated, and the US proceeds to consider this possibility as follows:

(a) find the highest non-empty USQi (7 is considered highest). Compare its priority i with that of the process running on the CPUs, taking an idle CPU to have priority -1. If

$i > PRI_j$ and $PRI_k \leq PRI_j$ for all $k \neq j$, then the first process on USQi will be given CPU_j; go to (b) to do this. Otherwise clear SCHFLG. If both CPUs are idle, all USQ's empty, RPIQ = Ø and there is a process on WAQ, take it off, clear WAQ, set RSI and SWQ and give it to the swapper with SWAPIN. Then loop waiting for ATTN.

(b) examine ALj. If it is P, we must wait for the CPU to start running the pending process. To do this, set WNPFLG ← j and loop to wait for ATTN.

(c) remove the first process from USQi. Put its PRT index P into CPUj. Send ATTN to CPU j. Increment ALj. Set PRIj ← i. Clear MSQ, set RUN in the PRT for P.

(d) now return to (a).

Relations with the Scheduler

Because we expect to change our minds a number of times about the algorithms for scheduling processes, they are not being implemented in hardware initially. Instead a software scheduler will run whenever a process blocks or suffers a timeout trap. It will be responsible for collecting awakened processes from WAKEUPQ, where they are left by the US, and putting them into its own data structure. It is also responsible for deciding which non-resident processes to run and for giving them to the swapper via SWAPIN calls. It has two words in PRT and the MTC field, in which the CPU records the process' compute timer (8 bits accurate to 1 ms) whenever the process stops running to help it do its job. Its methods will be described elsewhere.

To avoid a situation in which processes are left on WAKEUPQ because the scheduler never runs, the US makes a special check for complete idleness of the hardware mechanisms for running processes. This check is described above under WAKEUP. It ensures that a process will not sit on WAKEUPQ just because there is no other process to run the scheduler.

In order to permit the scheduler to delay the execution of ready processes so as to prevent excessively good service, it will be necessary to have a service process called TICTOC which is woken up by the real timer periodically. This process can give processes to the swapper if they have waited long enough. It can also perform various diagnostic and cleanup functions if that seems appropriate.

The real-time queue

The US is responsible for maintenance of the real-time queue. This is a forward-linked list chained through the PRRTP field of PRT which holds all the processes which are awaiting a signal on the occurrence of a specified real time. The time for which each process is waiting is held in the 23-bit RTT field, accurate to 10 ms. In 23 bits a time of about 1 day can be recorded; it is not expected that RTQ will normally be used for such long delays. The queue is ordered by increasing values of PRRT; the current value of the real-time clock is implicitly considered to be the first entry, and a new entry with timer t is put into the queue between entries with times t1 and t2 provided that

$t1 \leq t < t2$ or $t \geq t1 > t2$ or $t1 > t2 \geq t$

It is put on the end if there is no point where this holds. If the queue starts with only the implicit current-time entry and grows according to this rule, a unique position will always be selected for any entry. The head of the queue is kept in the cell RTQ.

The real-time clock RTC is kept in core and updated once a millisecond by the CHIO, which sends an ATTN to the US whenever it stores a new clock value. The US compares the clock t2 with the PRRT field t for the first process on RTQ and the old clock value t1. When it finds that t satisfies one of the relations above, it removes the process from RTQ and sends it a wakeup with RT turned on.

There is also a call on the US to add processes to RTQ, delete them and move them around. It is called CHANGERT and takes the new value of RTT in the data word; a -1 indicates that the process is to be removed. To process the call, the US scans RTQ to remove any old occurrence of the process and then installs the new occurrence at the proper point. If the desired wakeup time is within 20 ms of the current time it generates the wakeup immediately.

Appendix I: Micro-Scheduler Opcodes

WAKEUP	1
IWAKEUP	2
RETURN	3
BLOCK	4
BLOCKOUT	5
UNLOAD	6
GIVEUP	7
CHANGERT	8