PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
M.I.T.
CAMBRIDGE, MASSACHUSETTS 02139

PDP-39

DRUM SCHEDULING TECHNIQUES

in the PDP-1-X

June 8, 1966

DRUM SCHEDULING TECHNIQUES in the PDP-1-X

In the following we assume that the job mix for the drum consists of 400_8-word transfers intended to move blocks of a file, and 10000_8-word transfers called for by the computation scheduling algorithm as it moves program image sections in and out of core. We assume that the drum has a "swap" instruction, which causes the entire contents of a specified field on the drum to be written from, or read into, a specified field of core. This transfer starts as soon as possible after the instruction is given, independent of the current drum address, and occupies one drum revolution.

The drum mover maintains a drum job queue of 400_8-word jobs and the scheduling algorithm maintains the queue of 10000_8-word jobs. The drum mover process, on being restarted when the drum completes a transfer, first checks the scheduling algorithm's queue, and performs a 10000_8-word job if one is there. If not, it performs the first job in its queue.

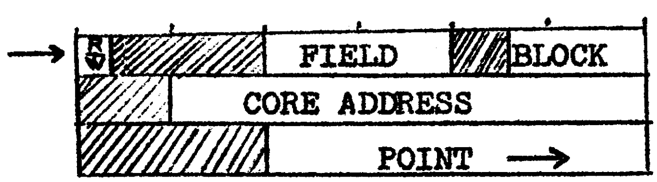
The essence of the drum scheduling policy is therefore implemented in the algorithm which inserts drum jobs in the queue: for once the queue is set up, it can only be modified by insertion of a new job or deletion of a finished job, and neither of these mechanisms will re-order the queue. Any scheduling policy we design will not be dependent on the scheduling of 10000_8-word jobs, since these are almost transparent with respect to drum position.

Since 400_8-word file blocks all start at drum addresses divisible by 400_8, the best performance we can expect from the drum is 4000_8 words transferred per revolution. This is obtained by placing the drum in operation for all even (odd) numbered blocks, while the odd (even) numbered blocks are passed over as the program restarts the drum. This is the best possible performance because delays in the drum field selection hardware make it impossible to set up a new drum transfer within the $8.6\mu s$ spacing between words on the drum.

Hence, the object of drum scheduling is to pack the requested transfers into as little time as possible, while preserving a block of do-nothing time before and after every block of transfer time.

Method

The drum queue is composed of three data objects: a *drum job queue head table*, 100_8 words; the *drum job space*, 140_8 words; and the *overflow table*, 20_8 words. The queue head table and the overflow table both contain pointers to drum jobs stored in the drum job space. A drum job entity requires three words with format as follows:



R/W gives the mode of transfer, FIELD and BLOCK specify the drum address, CORE ADDRESS is the translated memory address, and POINT is a pointer to a list of processes to be made active when the job is completed. Unused drum job entities are held on a list.

The queue head table is the scheduling data: the 100_8 words allow us to schedule 100_8 blocks or 4 drum revolutions into the future. There will be a pointer to "now" in the queue (the queue is a ring buffer). If a word in the queue points to a drum job, then it represents that drum job. If a word in the queue has value 400000_8, then it represents "occupied time". A zero word in the queue represents "available time".

The drum job scheduling algorithm is as follows: first, a drum job entity for the job is created. Then we attempt to schedule the job by testing the four words in the queue capable of holding this job. We pick the earliest zero word, and load it with a pointer to the job entity. Then we set the two words on either side to 400000_8. If all four slots are filled, we place a pointer to the job entity in the overflow table. About once every drum revolution we try to reschedule the overflow jobs.

When a drum transfer is actually started, the job entity is returned to the free list, and the associated words in the queue head table are zeroed. When the transfer has completed, we search the queue (linearly) for the next job.