SETL Newsletter 86 November 8, 1972

Proposal for a temporary, but easily implemented, software paging scheme. J. Schwartz

The memory size required for SETLB runs is now becoming so large that some method of reduction is urgently required. This newsletter will propose a scheme, based upon software paging of code (on a subroutine-by-subroutine basis, and at moment of call) which it should be easy to implement. It involves a combination of simple patches at the SETLB and the BALM level. The compilation of subroutines by SETLB must be slightly modified, and two simple primitives added to BALM. At the BALM level, it would also be desirable to make some extensions, probably easy also, to the 'save status' and 'resume' routines. The scheme is easy because it pages principally subroutines created at the SETLB level, rather than paging all the routines in the BALM and BALMSETL libraries.

1. Patches at the SETLB level.

If a subroutine or fuction was to be paged, its SETLB compilation would be modifed as follows:

definef f(arg_1,...,arg_n); ..., end f;

would be compiled as (we write in SETLB, but the transcription to BALMSETL should be obvious; explanations are given following the code text itself)

```
definef setlbfnct(arg_1,...,arg_n);...; end setlfnct;
setlbfnct = record(setlbfnct,savenumber);
/* 'record' is one of the two BALM-level primitives that
     must be created */  compute; do;
definef fname(arg_1,...,arg_n);
sub = bringin(savenumber);
result = sub(argl,...,arg n);
callno(savenumber) =  callno(savenumber)-1;
return result;
end fname;
```

Quite similar code, of obvious form, would be used for subroutines. The auxiliary routine bringin has the following form.

```
definef bringin(subnumber);
callrec(subnumber) = callcyno; callcyno = callcyno+1;
if callcyno  gt maxcyno then callcyno = 1;
sub = pagevect(subnumber); callno(subnumber) = callno(subnumber)+1;
if sub ne 0 then return sub;;
/* otherwise, subroutine must be reloaded */
sub = retrieve(subnumber); /*'retrieve' is one of the
     two BALM-level primitives that must be created */
return sub;
end bringin;
```

The conventions implicit in the above code are as follows:

setlbfnct - is a reserved, global name, used simply as a temporary to hold a pageable procedure until it is 'recorded' by the 'record' primitive.

savenumber - is generated by a counter maintained in the SETLB compiler. A separate number is generated for each pageable subroutine.

callno - this is a BALM vector, whose n-th component records the number of times that the n-th pageable procedure has been called. This information is kept to prevent active routines from being rolled out and retrieved in separate copies.

callcyno - an auxiliary counter, kept cycling from 1 to maxcyno, and used so as to make it possible to tell which routines have been called most recently. this information is of course used when in a space-pinch it must be decided which routines should be rolled out.

callrec - a BALM vector, whose n-th component records the last call-cycle on which the n-th pageable subroutine has been called.

pagevect - a BALM vector, whose components are either procedures or 0. The component pagevect(n) is nonzero if the n-th pageable procedure is present in core; in this case, pagevect(n) equals the n-th pageable procedure.

The BALM-level primitive *record* sets up the information needed to page a procedure; the *retrieve* primitive brings the binary from of a procedure in from external storage. Notice that except during an initial *record* operation, a good binary copy of a pageable procedure will always be available on external storage, so that dynamic 'rollout' operations are unnecessary.

We now go on to describe the innner workings of these two primitives, and of certain associated garbage-collector actions.

2. Patches at the BALM level.

a) The 'record' primitive record(proc,n) has the following internal structure:

```
do;
sizevect(n) = size (in words) of proc;
write proc out to external storage (probably using the
     operating-system WRITEMS) function;
locvect(n) = address of this external copy;
callree(n) = 0;  /* since routine has never been called */
pagevect(n) = proc;
proc = 0;  /* kill to make garbage-collectible */
return;  end;
```

b) The 'retrieve' primitive retrieve(n) has the following internal structure:

```
do:
request a block equal in size to sizevect(n)
     from the garbage collector;
read the piece of code located by locvect(n) into
     this block (probably using the operating system
     READMS function);
return the start address of the block;  end;
```

In the above, sizevect and locvect are probably FORTRAN arrays in the interpreter-code area of BALM. The array locvect may in fact be identical with the 'directory' which the operating system requires for its READMS-WRITMS functions.

c) The garbage-collector must be modified in the following rather trivial way.

ci) Immediately before the issuance of the 'can retrieve less than 5% of storage; goodbye' termination insert the following line (transposed to FORTRAN)

```
 if canget(tenpercent * memorysize + blockneeded) then
     go to garbstart;;
```

here

memorysize - size of memory in which currently running

blocksize - size of block requested from garbage collector

tenpercent - 0.1 or some other reasonable constant

blockneeded- block requested from garbage collector

garbstart - first statement of garbage collector. By transferring to it, we start up the garbage collector compaction procedures, after the routine 'canget' has dumped enough code blocks to free the space which is required.

The 'canget' routine canget(nwds) acts as follows:

do:
insert n into an unused field of the n-th word of the
callno vector
sort (i.e. heapsort) callno on the values of
maxcyno + callcyno - callno(n) (mod maxcyno)-
this gives an ordering of pageable procedures from least to most recently called;
put accumulated_space = freespace collected on last garbage collector run;
work thru the sequence of sorted routines thus obtained, beginning with the least recently called, the and doing the following:
if callno of routine k is positive, bypass it;
otherwise add sizevect(k) to accumulated_space;
if accumulated_space is no less than nwords, put result = true, and go to restore;
end of loop;
result = false;
restore: re-sort the callno vector into its original order;
return;
end of *canget* procedure;

A count of the total number of paging operations might be kept by the *retrieve* routine, and 'excess paging' error termination enforced.

3. Generalization to other BALM procedures; SAVE; RESUME.

a) The following scheme, based on the above, could be used to make routines of the BALM or BALMSETL library pageable.

If *procd* is a procedure with n arguments, execute:
makepage(procd(arg_1,...,arg_n)),

where

```
makepage(procd(x)) means
do procd = record(procd,global ctr), globalctr=globalctr+1,
 procd   = proc(x), sub = bringin([present value of globalctr]),
     result = sub(x),
 callno([present value of globalctr]) = callno([present value
                                        of globalctr])+1,
 return(result) end end .
```

By calling makepage to a suitable list of present library routines we might get down to smaller sizes.

b) The SAVE routine should be modified to save not only core status (after compression) but also the set of disc records set up by the record routine. This would merely append a string of records (separated by end-record marks) to the end of the present BALM save file format. During the save operation, locvect(n) would be changed to give the order, in this sequence of records, of the binary copy representing the n-th pageable procedure. The RESUME routine would then, in addition to its present actions, read this sequential sequence of records and write them (probably using WRITMS) to randomly-accessible disc records, reinitializing corresponding locvect entries during this process.

4. Bells and whistles.

The scheme suggested might be adequate to support operation on the 7600. For this use, it should be generalized to support two levels of paging, one to masscore, the other to disc. Presumably appropriate routines akin to READMS - WRITMS are already available for this purpose.

To give convenient user control over the choice of subroutines and functions to be paged, a declaration could be added to SETLB, having the syntax

page ($name_1$,...,$name_k$);

This would stash away a set of subroutine names. Subroutines not appearining in this collection would be compiled normally; subroutines in the collection would be compiled with the modifications indicated above. Even better, since it would permit incremental definition of the set of pageable routines, would be the slightly more general syntactic form

page (n, $name_1$, ..., $name_k$);

where n is an integer. This would set the *savenumber* counter (cf. section 1 above) to n, and hence make the procedures $name_1$,...,$name_k$ pageable as procedures numbered n, n+1, n+2, ...