hphcs_ : Undocumented Entries

add_pd_records
delete_pd_records
quota_get_reset
quota_reset
sysem_opr_con_read
sysem_opr_con_write
sysem_store_event_channel
tdcm_add_drive
tdcm_priv_attach

hphcs_ : Obsolete Entries

usercode_add
usercode_del
usercode_spacl_add

hphcs_$add_acl_entries

This entry is the same as hcs_$add_acl_entries except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$add_dir_acl_entries

This entry is the same as hcs_$add_dir_acl_entries except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$add_dsu_270_chan

Name: dsu270_reconfig

This module allows the dynamic addition or removal of an additional DSU-270 data channel in the Multics configuration.

Entry: dsu270_reconfig$add_dsu270_chan

This entry adds the additional channel and prints a message on the operator's typewriter.

Usage

```
declare dsu270_reconfig$add_dsu270_chan entry;

call dsu270_reconfig$add_dsu270_chan;
```

There are no arguments.

hphcs_$call_bos

Internal Interface
Hardcore Ring
3/31/72
System: 15.30

Name: call_bos

This procedure transfers to BOS. It will return to its caller when GO or CONTIN is typed on the operator's console. It makes sure that the free storage map on secondary storage is up-to-date by calling pc$fsout before calling BOS.

Usage

```
declare call_bos entry;

call call_bos;
```

There are no arguments.

(END)

hphcs_$copy_fdump

Name: copy_fdump

This module provides facilities for handling BOS FDUMPs.

The main entry constructs a segment pointing to the dump partition and copies the information contained there into the Multics hierarchy. Up to 10 segments will be created in the directory >dumps to hold the information found in the dump partition. The names of the segments will be of the following form:

mmddyy.tttt.s.eee

1) mmddyy is the date the dump was taken.

2) tttt is the time the dump was taken.

3) s is the segment sequence number (from 0 to 9).

4) eee is the Error Report Form (ERF) number used by operations in reporting the system crash for which the dump was taken.

Usage

```
declare copy_fdump entry (fixed bin);

call copy_fdump (code);
```

1) code is a status code. (Output)

hphcs_$create_proc

Name: act_proc

The act_proc procedure is that part of the traffic controller which creates and activates processes. The act_proc$create entry is callable from outside the Hardcore Ring by administrative programs.

Entry: act_proc$create

This entry is called to create a new process. The process directory for the new process is created; the segments Known Segment Table (KST), and Process Data Segment (PDS) are created and initialized; and the process is entered into the traffic control data bases.

Usage

```
declare act_proc$create entry (ptr, fixed bin);

call act_proc$create (cp, code);
```

1) cp is a pointer to a caller-supplied structure containing information needed by act_proc. (Input)

2) code is a status code. (Output)

Note

The structure pointed to by cp is declared as follows:

```
declare 1 create_info based (cp) aligned,
          2 processid,
            3 rel_apte bit(18) unaligned,
            3 unique_index bit(18) unaligned,
          2 version fixed bin,
          2 term_channel fixed bin(71),
          2 term_processid bit(36),
          2 words_of_pit fixed bin,
          2 record_quota fixed bin,
          2 ppml fixed bin,
          2 initial_ring fixed bin(3),
          2 timax fixed bin,
          2 account_ptr ptr,
          2 pit_ptr ptr,
          2 process_group_id char(32),
          2 user_processid char(32),
          2 account_id char(32),
          2 homedir bit(18),
          2 lot_size fixed bin,
          2 cls_size fixed bin,
          2 kst_size fixed bin,
          2 dont_call_init_admin bit(1) aligned,
          2 lot_in_stack bit(1) aligned,
```

1) processid — is the process ID of the newly created process.

2) version — is the version number of the create_info structure.

3) term_channel — is the event channel to use when the process is to be terminated.

4) term_processid — is the process ID of the process to receive the terminate request.

5) words_of_pit — is the number of words to be copied from the Process Initialization Table (PIT) pointed at by pit_ptr into the newly created PIT.

6) record_quota — is the record quota to be set on the new process directory.

7) ppml — is the per-process master limit to be used for the new process.

8) initial_ring — is the ring in which the new process should first run after leaving ring 0.

9) timax — is a scheduling parameter to be assigned to the new process.

10) account_ptr — points to accounting information for the new process.

11) pit_ptr — points to a copy of the PIT to be used for the new process.

12) process_group_id — is the process group ID of the new process.

13) user_processid — is the user process ID of the new process.

14) account_id — is the account ID of the new process.

15) homedir — is the offset, from the start of the PIT, of the home directory name.

16) lot_size — is the size of the Linkage Offset Table (LOT).

17) cls_size — is the size of the initial combined linkage segment.

18) kst_size — is the number of entries in each KST array.

19) dont_call_init_admin — is "1"b if the process overseer should be called directly from ring 0.

20) lot_in_stack — is "1"b if the LOT should go into the stack.

21) cls_in_stack — is "1"b if the initial combined linkage segment should go into the stack.

hphcs_$del_dsu270_chan

Entry: dsu270_reconfig$del_dsu270_chan

This entry removes the additional channel and prints a message on the operator's typewriter.

Usage

```
declare dsu270_reconfig$del_dsu270_chan entry;

call dsu270_reconfig$del_dsu270_chan;
```

There are no arguments.

hphcs_$delete_acl_entries

This entry is the same as hcs_$delete_acl_entries. except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$delete_dir_acl_entries

This entry is the same as hcs_$delete_dir_acl_entries except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$destroy_proc

Name: deact_proc

This procedure is called by the answering service to destroy the specified process. The traffic controller is called to stop the process and when that is completed, the process is destroyed and its process directory deleted.

Entry: deact_proc$destroy

This entry is called to destroy a process.

Usage

```
declare deact_proc$destroy entry (bit(36) aligned, fixed
        bin, fixed bin(71));

call deact_proc$destroy (id, code, cpu_time);
```

1) id — is the process id of the process to be destroyed. (Input)

2) code — is a file system error code reflecting any problems in the attempt to delete the process directory of the process being destroyed. (Output)

3) cpu_time — is the CPU time which this process used while it was in existence. (Output)

hphcs_$fs_get_trans_sw

Entry: fs_get$trans_sw

This entry inputs the new transparent usage value desired. The old value is returned. This is generally called through the hphcs_ gate.

Usage

```
declare fs_get_$trans_sw entry (fixed bin, fixed bin);

call fs_get_$trans_sw (newsw, oldsw);
```

1) newsw — newsw>3 means return oldsw but do not reset pdf$transparant. (Input)

2) oldws — is the old transparent usage value. (Output)

Notes

newsw and oldsw = 0 for opaque to usage and modification; = 1 for transparent to usage only; = 2 or 3 for transparent to usage and modification.

hphcs_$ioam_status_priv

- Entry: ioam_$ioam_status_priv

This is the same as the ioam_$ioam_status entry except that if the device is assigned, the message is set to the process group id of the assigning process.

Usage

```
declare ioam_$ioam_status_priv entry (char(*),
        char(*), fixed bin);

call ioam_$ioam_status_priv (device_name, message, code);
```

Same arguments as in ioam_$ioam_status.

1) device_name — is the name of the device in question. (Input)

2) message — is a character string description of the status of the device. It can have a maximum length of 32 characters. (Output)

3) code — is a numerical representation of the device status. (Output)

Notes

The status of the device and the returned values are as follows:

1) No meaning can be attached to the given device_name.

   message = "unknown"; code = error_table_$ioname_not_found;

2) Device not assigned to any process.

   message = "unassigned"; code = error_table_$dev_nt_assnd;

3) Device assigned to this process.

   message = "assigned"; code = 0;

4) Device assigned to some other process.

   message = "assigned to other process";
   code = error_table_$already_assigned;

hphcs_$ioam_release_priv

Entry: ioam_$ioam_release_priv

This is the same as ioam_$ioam_release except that it will free a device assigned to any process.

Usage

```
declare ioam_$ioam_release_priv entry (char(*), fixed bin);

call ioam_$ioam_release_priv (device_name, code);
```

Same arguments as in ioam_$ioam_release.

1) device_name — is the name of the device to be released. (input)

2) code — is a returned error code:

error_table_$ioname_not_found: device name unknown;

error_table_$io_no_permission: device not assigned to this process;

hphcs_$ips_wakeup

Entry: pxss$ips_wakeup

This entry is called to cause a process to be interrupted when it leaves ring 0. When the process is interrupted, it will signal the message set up by this call.

Usage

```
declare pxss$ips_wakeup entry (bit(36) aligned, char(4)
        aligned);

call pxss$ips_wakeup (processid, message);
```

1) processid — is the process ID of the process to receive the wake-up. (Input)

2) message — is the character string to be signalled in the User Ring. (Input)

hphcs_$move_device

Name: move_device

Entry: $move_device

This entry will move a segment onto a specified device.

Usage

```
declare move_device entry (char(*), char(*) fixed bin(4),
        fixed bin);

call move_device (dirname, ename, did, code);
```

1) dirname — is the pathname of the directory containing the segment to be moved. (Input)

2) ename — is the entry name of the segment to be moved. (Input)

3) did — is the device ID of the device to which the segment is to be moved. (Input)

4) code — is a standard file system error code. (Output)

Notes

The following device ID's are valid:

1 drum
2 dsu270
3 dsu170

hphcs_$process_status

4/29/72
System: 17.1

Entry: pxss$process_status

This entry may be called from outside the Hardcore Ring, but only by privileged users. It returns several items of interest about a specified process.

Usage

```
declare pxss$process_status (ptr);

call pxss$process_status (info_pointer);
```

1) info_pointer — is a pointer to a structure (see Notes below) in which the process ID is specified and information is returned. (Input)

Notes

The format of the structure pointed to by pxss$process_status is:

```
declare 1 process_status_info aligned,
        2 processid bit(36),
        2 padding bit(36),
        2 apt_ptr ptr,
        2 ex_state fixed bin,
        2 mp_state fixed bin,
        2 state_change_time fixed bin(71),
        2 cpu_time fixed bin(71);
        2 paging_measure fixed bin(71);
```

1) processid — is the process ID of the process for which status is requested.

2) padding — is currently unused.

3) apt_ptr — is a pointer to the APT entry of the process.

4) ex_state — is the returned execution state of the process.

5) mp_state — is the returned multi-programming state of the process.

6) state_change_time — is the time the execution state for this process was last changed.

7) cpu_time — is the total CPU time charged to this process.

8) paging_measure — is the total memory usage of this process.

hphcs_$process_status

4/29/72
System: 17.1

Entry: pxss$process_status

This entry may be called from outside the Hardcore Ring, but only by privileged users. It returns several items of interest about a specified process.

Usage

```
declare pxss$process_status (ptr);

call pxss$process_status (info_pointer);
```

1) info_pointer — is a pointer to a structure (see Notes below) in which the process ID is specified and information is returned. (Input)

Notes

The format of the structure pointed to by pxss$process_status is:

```
declare  1  process_status_info aligned,
            2 processid bit(36),
            2 padding bit(36),
            2 apt_ptr ptr,
            2 ex_state fixed bin,
            2 mp_state fixed bin,
            2 state_change_time fixed bin(71),
            2 cpu_time fixed bin(71),
            2 paging_measure fixed bin(71);
```

1) processid — is the process ID of the process for which status is requested.

2) padding — is currently unused.

3) apt_ptr — is a pointer to the APT entry of the process.

4) ex_state — is the returned execution state of the process.

5) mp_state — is the returned multi-programming state of the process.

6) state_change_time — is the time the execution state for this process was last changed.

7) cpu_time — is the total CPU time charged to this process.

8) paging_measure — is the total memory usage of this process.

hphcs_$pxss_set_timax

Entry: pxss$set_timax

The entry may be called from outside the Hardcore Ring but only by privileged users. It will assign a new value to timax for the specified process.

Usage

```
declare pxss$set_timax entry (bit(36) aligned, fixed bin);

call pxss$set_timax (processid, timax);
```

1) processid — is the process- ID of the process whose timax is to be changed. (Input)

2) timax — is the new value of timax to assign (in microseconds). If timax <= 0, then the process' timax will be set to <tc_data>|timax . (Input)

hphcs_$pxss_status

Entry: pxss$status

This entry is obsolete. Use pxss$process_status below instead.

This entry may be called from outside the Hardcore Ring but only by privileged users. It returns several items of interest about a specified process.

Usage

```
declare pxss$status entry (bit(36) aligned, fixed bin, fixed
     bin, ptr, fixed bin(71), fixed bin(71));

call pxss$status (processid, ex_state, mp_state, apt_ptr,
     state_change_time, cpu_time);
```

1) processid	is the process ID of the process for which status is requested. (Input)
2) ex_state	is the returned execution state of the process. (Output)
3) mp_state	is the returned multiprogramming state of the process. (Output)
4) apt_ptr	is a pointer to the APT entry of the process. (Output)
5) state_change_time	is the time the execution state for this process was last changed. (Output)
6) cpu_time	is the total CPU time charged to this process. (Output)

hphcs_$quota_reload

Entry: quota$qreload

This entry is privileged and is intended for use only by the reloader. It is used to completely restore the quota information on a directory.

Usage

```
declare quota$qreload entry (char(*), fixed bin,
        fixed bin, fixed bin, fixed bin,
        fixed bin);

call quota$qreload (dirname, quota, trp, tup, infcnt,
     tmqsw, code);
```

1) dirname	is the pathname of the directory whose quota is being reloaded. (Input)
2) quota	is the value of quota to be set. (Input)
3) trp	is the time-record product in page-seconds to be set. (Input)
4) tup	is the time that trp was last updated. (Input)
5) infcnt	is the inferior account count. (Input)
6) tmqsw	is the terminal account switch. (Input)
7) code	is a file system error code. (Output)

Notes

This entry resets all the account items in the directory header to the input values. If the value of the terminal quota switch is changed, the pages used are either added or subtracted from the superior account depending on the new value of the terminal quota switch. No permission in the directory is needed and the quota need not be greater than the pages used.

hphcs_$quota_set

Name: quota

This segment contains the various unwired procedures to manipulate secondary storage quotas on directories. Some entries are callable by a regular user, some by privileged users, and some from ring 0 only.

Entry: quota$qset

This entry is privileged and is used to set arbitrary quotas on directories.

Usage

```
declare quota$qset entry (char(*), fixed bin, fixed bin);

call quota$qset (dirname, quota, code);
```

1) dirname — is the pathname of the directory whose quota will be changed. (Input)

2) quota — is the number of 1024 word pages of non-master device secondary storage in the new quota. (Input)

3) code — is a file system error code. (Output)

Notes

A terminal quota of the input value is set on the directory dirname. If dirname did not previously have a terminal quota, its pages used are subtracted from whatever quota they are previously charging against. No permission is required in the directory and the quota need not be large enough to cover the pages used.

hphcs_$reconfig

Internal Interface
Hardcore Ring
11/30/70

Name: reconfig

This procedure provides the hardcore ring functions needed by the command "reconfigure". This routine copies hardcore ring data for use by its caller, updates hardcore ring data bases, and calls hardcore ring primitives. This routine is called through the gate hphcs_.

Usage

```
declare reconfig entry (ptr, fixed bin, fixed bin);

call reconfig (data_ptr, entry_vector, error_code);
```

1) data_ptr — is a pointer to a data structure used to communicate (input and output) with the caller. (Input)

2) entry_vector — is a code that specifies which function is to be performed. This code is used to avoid multiple entry points in the hardcore ring gate. Only meaningful sequences of calls are allowed, and the hardcore ring data in the System Communication Segment (SCS) remains locked until the sequence is completed. (Input)

3) error_code — is an error_code that is interpreted by the command "reconfigure" and used to print error messages at the console. (Output)

(END)

hphcs_$replace_acl

This entry is the same as hcs_$replace_acl except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$replace_dir_acl

This entry is the same as hcs_$replace_dir_acl except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hohcs_$restore_quota

Entry: $restore_quota

This entry point sets "pds$quota_inhib" to 0 so that quota checking will again be in force.

Usage

```
declare quota_util$restore_quota entry;
call quota_util$restore_quota;
```

There are no arguments.

Note

The above entries are accessed through a privileged gate.

hphcs_$ring_0_patch

Entry: ring_0_peek$patch

This entry will move data from any location readable in ring 0 to any location writeable in ring 0. If p2 points to a segment that does not have write permission at the user's validation level, a message will be printed on the 645 operator's console stating that this user has made a patch into ring 0.

Usage

```
call ring_0_peek$patch (p1, p2, n);
```

or

```
call hphcs_$ring_0_patch (p1, p2, n);
```

hphcs_$salvage_dir

Entry: on_line_salvager$salvage_dir

This entry is accessed via a highly privileged gate entry from the user ring. It is used both for testing and salvaging a directory if the system desires.

Usage

```
declare on_line_salvager$salvage_dir (char(*), fixed bin);

call on_line_salvager$salvage_dir (dirname, code);
```

1) dirname — is the absolute pathname of the directory to be salvaged. (Input)

2) code — is a standard file system error code. (Output)

hphcs_$set_act

Entry: set$act

This entry point, used only by highly privileged users, sets the activity of a branch.

Usage

```
declare set$act entry (char(*), char(*), bit(18) aligned,
        bit(36), fixed bin);

call set$act (parent, ename, ind, time, code);
```

Arguments 1), 2), and 5) as above

3) ind is the activity indicator. (Input)

4) time is the activity time. (Input)

hphcs_$set_auth

Entry: set$auth

This entry point, used only by highly privileged users, sets the author of a branch.

Usage

```
declare set$auth entry (char(*), char(*), fixed bin,
        char(*), fixed bin);

call set$auth (parent, ename, chasesw, auth, code);
```

Arguments 1), 2), and 5) as above.

3) chasesw is a switch which, if nonzero, indicates that the caller wants to chase a link to its terminal branch. (Input)

4) auth is the new author for the branch. (Input)

hphcs_$set_bc_auth

This entry allows one to set the value of the bitcount author.

Usage: dcl hphcs_$set_bc_auth entry (char(*), char(*), char(*), fixed bin(35))

call hphcs_$set_bc_auth (dname, ename, auth, code)

1) dname is the parent directory of the entry whose bitcount author is to be set.

2) ename is the entry whose bitcount author is to be set.

3) auth is the new bitcount author

4) code is a status code.

hphcs_$set_dates

Entry: set$dates

This entry is used during reload and retrieval to set the dates as shown below.

Usage

```
declare set$dates entry (char(*), char(*), ptr, fixed bin);

call set$dates (parent, ename, dateptr, code);
```

Arguments 1), 2), and 4) as above.

3) dateptr — is a pointer to a structure containing new values for the dates. (Input)

Notes

The structure pointed to by dateptr is declared as follows:

```
declare 1 time based (dateptr) aligned,
        (2 dtem bit(36),
         2 dtd bit(36),
         2 dtu bit(36),
         2 dtm bit(36)) unaligned;
```

1) dtem — is the date and time the entry was modified. (Input)

2) dtd — is the date and time the segment was dumped. (Input)

3) dtu — is the date and time the segment was used. (Input)

4) dtm — is the date and time the segment was modified. (Input)

hphcs_$set_dir_ring_brackets

This entry is the same as hcs_$set_dir_ring_brackets except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$set_fdump_num

Entry: copy_fdump$set_erf_no

This entry constructs a one page "window" into the dump partition to set the ERF number of the next dump to be taken.

Usage

```
declare copy_fdump$set_erf_no entry (fixed bin, fixed bin);

call copy_fdump$set_erf_no (erf_no, code);
```

1) erf_no is the ERF number to be associated with the next dump. (Input)

2) code is a status code. (Output)

hphcs_$set_mask_ring

This entry point is provided for the use of the initializer process only. An explicit check is performed to insure that the process id of the calling process is that of the initializer process.

The entry changes the value of pds$initial_ring to 4 so that ips interrupts will be masked in lower rings. ~~On exit the initialization process~~ It is called by the initializer process after it has completed its ring 1 initialization.

Usage
```
dcl hphcs_$set_mask_ring entry;
call hphcs_$set_mask_ring();
```

There are no arguments.

hphcs_$set_ring_brackets

This entry is the same as hcs_$set_ring_brackets except that it sets its validation level to zero before proceeding, so that the access information of segments whose first ring bracket is zero can be manipulated

hphcs_$shutdown

Name: shutdown

The shutdown procedure is used to bring Multics to an orderly end. It first calls tc_shutdown to make sure all users are logged out and the traffic control data bases are in a consistent state. It then deactivates all active segments and updates all file maps.

Finally, wired_shutdown is called to delete all hardcore segments and write out the updated File System Device Configuration Table (FSDCT).

Usage

```
declare shutdown entry;

call shutdown;
```

There are no arguments.

ncs_$star_no_acc_ck

Entry: star_$no_acc_ck

This entry point is the same as the star_ entry point except that no access checking is performed, and therefore is highly privledged.

Usage

```
declare star_$no_acc_ck entry (char(*), char(*), fixed
        bin(3), ptr, fixed bin(17), ptr, ptr, fixed bin(17));

call star_$no_acc_ck (dirname, star_name, bl, areap, ecount,
     eptr, nptr, code);
```

1) dirname — is the path name of the directory to be searched. (Input)

2) star_name — is the possibly star-laden entry name. (Input)

3) bl — is a two bit switch set on to select branches (b) and/or links (l).

   3 = both,
   2 = b only,
   1 = l only,
   0 = error. (Input)

4) areap — is a pointer to a caller-provided area into which the structures for return information will be allocated and written. If null, ecount is set to the total number of entries only. (Input)

5) ecount — is a count of the number of selected entries. (Output)

6) eptr — is a ptr to the allocated entry structure (See MPM) (OUTPUT)

7) nptr — is a pointer to the allocated name structure (See MPM) (OUTPUT)

8) code — is a status code

hphcs_$start_disk_meters

Entry: start_dm

This entry wires the segment disk_traffic_data, allowing meter_disk to accumulate data in it.

Usage

```
declare get_disk_meters$start_dm entry;

call get_disk_meters$start_dm;
```

There are no arguments.

hphcs_$start_process

Entry: pxss$start

This entry is called to restart a stopped process.

Usage

```
declare pxss$start entry (bit(36) aligned, fixed bin);

call pxss$start (processid, state);
```

1) processid — is the process ID of the process being started. (Input)

2) state — is the returned execution state of the process being started. (Output)

hphcs_$status_backup_branch_info

Entry: status_$backup_branch_info

This entry is used to return information necessary for static migration of segments and is highly privileged as it does not perform access checking.

Usage

```
declare status_$backup_branch_info entry (char(*), char(*),
        bit(36), bit(18), bit(18), bit(4), bit(1), bit(36),
        bit(4), bit(18), fixed bin);

call status_$backup_branch_info (dirname, ename,
     actime, actind, astep, did, dirsw, dtu, move,
     records, code)
```

1-2) are as above. (Input)

3-5) are as above. (Output)

6) did is the device ID. (Output)

7) dirsw is "1"b if this is a directory, otherwise it is 0"b". (Output)

8) dtu is the date-time the segment was last used. (Output)

9) move is the ID of the target device if the segment is being moved. (Output)

10) records is the number of records used by this segment. (Output)

11) code is as above. (Output)

hphcs_$stop_disk_meters

Entry: stop_dm

This entry unwires the segment disk_traffic_data, prohibiting meter_disk from accumulating any more data in it.

Usage

```
declare get_disk_meters$stop_dm entry;

call get_disk_meters$stop_dm;
```

There are no arguments.

hphcs_$stop_process

Internal Interface
Hardcore Ring
03/15/71

Name: stop_process

This procedure is used to initiate the stopping of a process.

Usage

```
declare stop_process entry (bit(36) aligned);

call stop_process (process_id);
```

1) process_id is the id of the process to be stopped. (Input)

(END)

hphcs_$suspend_quota

Name: quota_util

This procedure is used to set or clear the flag "pds$quota_inhib" which is checked by page control to see whether quota checking for a process is suspended.

Entry: $suspend_quota

This entry point sets "pds$quota_inhib" to 1 so that quotas will not be checked for this process.

Usage

```
declare quota_util$suspend_quota entry;

call quota_util$suspend_quota;
```

There are no arguments.

hphcs_$tty_write_force

This entry is called by priveleged processes (or ring 0 procedures) when they cannot go blocked to do a limited amount of output. This entry will almost always return nelemt = nelem

Usage: The calling sequence is the same as that of hcs_$tty_write