April 1981

INTERNAL SYSTEM SUBROUTINES

This section contains descriptions of internal system subroutines that are callable only from programs running in system mode. These subroutines generally are of use only to systems programmers.

MTS 3: System Subroutine Descriptions

April 1981

April 1981

M T S

The Michigan Terminal System

Volume 3: System Subroutine Descriptions

(System Edition)

April 1981

Updated March 1982 (Update 1)
Updated February 1983 (Update 2)
Updated January 1984 (Update 3)
Updated September 1984 (Update 4)
Updated April 1985 (Update 5)
Updated September 1985 (Update 6)
Updated July 1987 (Update 7)

The University of Michigan Computing Center
Ann Arbor, Michigan

****
*
* This obsoletes the October 1976 edition. *
*
****

April 1981

April 1981

Contents

April 1981

ATTNTRP

Subroutine Description

Purpose: To allow control to be returned to the user on an attention interrupt from a terminal.

Location: Resident System

Alt. Entry: ATTNT

Calling Sequences:

Assembly:
```
LM   0,1,=A(exit,region)
CALL ATTNTRP
```

Parameters:

GR0 should contain zero or the location to transfer control to if an attention interrupt occurs.
GR1 should contain the location of a 72-byte save region for storing pertinent information.

Return Codes:

0 Successful return.
4 Illegal parameter specified.

Description: A call on the subroutine ATTNTRP sets up an attention interrupt intercept for one interrupt only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of an attention interrupt. When an interrupt occurs and the exit is taken, the intercept is cleared so that another call to ATTNTRP is necessary to intercept the next attention interrupt. When an attention interrupt occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location previously specified. If the exit subroutine returns to MTS (BR 14), MTS will handle the interrupt as if ATTNTRP had not been called originally. This feature allows the user to take brief control of the interrupt before MTS takes complete control of the interrupt. When MTS takes control of the interrupt, execution of the program will be terminated and a message will be printed providing the location of the interrupt.

If GR0 is zero on a call to ATTNTRP, the attention interrupt intercept is disabled. GR1 should be zero, or it should point to a valid save region.

When the attention interrupt exit is taken, the first eight bytes of the save region contain the attention interrupt PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region. GR2 contains bits indicating attributes of the interrupted program:

bit 28: 1 - (X'00000008') if program was executing in user execution mode.
bit 27: 1 - (X'00000010') if program was executing in virtual-machine mode.
bit 25: 1 - (X'00000040') if program was executing with SVC trapping enabled.

The remaining bits in GR2 are unpredictable. The contents of GR0 and GR3 through GR12 are unpredictable.

If on a call to ATTNTRP the first byte of the save region is X'FF', ATTNTRP does not return to the calling program; rather, the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on ATTNTRP, after the first attention interrupt exit is taken, allows the user to set a switch (for example) and to return to the point at which he was interrupted with the attention interrupt intercept again enabled.

Routines called from within an attention interrupt exit routine must be recursive if execution is to be resumed after interrupt processing. The MTS I/O subroutines READ, WRITE, SCARDS, SPRINT, SPUNCH, SERCOM, and GUSER are recursive; the FORTRAN I/O subroutines are not.

The ATTNTRP item of the GUINFO/CUINFO subroutine may be used to save a previous exit address and associated region so that it may be later restored.

April 1981

CMDSTATS

System Subroutine Description

Purpose: To record the text and type of MTS commands, CLS commands, machine checks, unit checks, and other system events that may be useful either individually or in a statistical context.

Location: Resident System

Calling Sequences:

Assembly: CALL CMDSTATS,(text,length,type,abbr)

Parameters:

text is the location of the data portion of the CMDSTAT record. This is the ADATA portion of the dsect described in COPY:CMDAREADSECT.

length is the location of a fullword giving the length of the text that is to be recorded.

type is the location of a fullword CMDSTAT record type:

1 = Command
2 = Overflow
3 = Reload
4 = Old style unit-check data (obsolete)
5 = End-of-User (user has been signed-off)
6 = 3270 DSR Statistics
7 = HASP Log
8 = JOBS (SYSTEMSTATUS) statistics
9 = TSFO DSR Statistics
10 = Machine-check records
11 = Operator console log
12 = Editor statistics
13 = Loader statistics
14 = Plot statistics
15 = SPIRES statistics
15 = Unused (was SPIRES usage data)
17 = Merit statistics
18 = Operator plotter statistics
19 = SSRTN summary statistics
20 = Unused
21 = HASP device statistics (e.g., OMR cards)
22 = PW failure statistics
23 = *STATISTICS records
24 = System software errors
25 = Unit-check data

26 = Reserved for use at WSU (VM data)
27 = Help statistics
28 = Response time data collection single-job item

abbr is the location of the four-letter official command name for the command that is being recorded, if the type code is 1. For type codes other than 1, this parameter is ignored.

Return Codes:

0 The item was successfully recorded.
4 The item could not successfully be recorded and an overflow item has been generated.

April 1981

FREED

System Subroutine Description

Purpose: To release a FDUB acquired by the GETD subroutine.

Location: Resident System

Calling Sequences:

Assembly:
```
L    0,fdub
CALL FREED
```

Parameters:

GR0 contains a FDUB-pointer obtained by a call to the GETD subroutine.

Return Codes:

0 Successful return.
4 Invalid FDUB-pointer.

Description: The FDUB for the device obtained by a call to GETD is released. When all the FDUBs for the device have been released, the device and associated logical device name (LDN) are released.

April 1981

April 1981

GETACCRE

System Subroutine Description

Purpose: To read an accounting record for a given signon ID.

Location: Resident System

Calling Sequences:

Assembly: CALL GETACCRE,(ccid,region,sws)

Parameters:

ccid is the location of the 4-character signon ID whose accounting record is to be read.

region is the location of the region into which the accounting record is to be placed. (This should be at least LACCR bytes in length).

sws is the location of a switch controlling the locking of the appropriate accounting file (*ACCOUNTING1 - *ACCOUNTING5).

0 - lock *ACCOUNTINGn for modification and leave it locked after the record is read.
1 - lock *ACCOUNTINGn for read and unlock it before returning.

Values Returned:

GR0 contains a FDUB-pointer for *ACCOUNTINGn if sws is zero and the return code is zero.

Return Codes:

0 Successful return.
4 Nonexistent signon ID.
8 Error.
12 Invalid parameter.

Description: The format of the accounting record is defined by the ACCDCT macro in COPY:MTS.MACROS.

Only programs with a specific set of program keys may call GETACCRE, and only some of those programs may lock the file *ACCOUNTINGn for modification. Calls from programs with the correct program keys may be made from either user or system mode.

April 1981

April 1981

GETD

System Subroutine Description

Purpose: To obtain a device and/or reserve a logical device number (LDN) for use.

Location: Resident System

Calling Sequences:

```
Assembly: L    2,name
          L    1,type
          CALL GETD
```

Parameters:

name is the location of a 4-character UMMPS device name.

type is the location of a 4-character UMMPS device type, zero if the type should not be checked, or C'NONE' if a logical device name (LDN) is to be reserved but no device is to be attached.

Values Returned:

GR0 contains a FDUB-pointer which may not be used in standard MTS I/O calls, but which will contain the LDN for the device if the return code is zero.

Return Codes:

```
 0 Successful return.
 4 Device is busy.
 8 Device is logically off-line.
12 Device does not exist.
```

Description: If the device requested is already attached to the MTS job (through a previous call to GETD, GETFD, GETFDN, etc., but not via direct use of the SVC GTUNIT), the logical device number (LDN) for that device is placed in the FDUB and returned. If the device is not currently in use and the device type in GR1 is "NONE", an unused LDN is reserved and returned in the FDUB that is returned in GR0. Otherwise, an unused LDN is reserved and an attempt is made to obtain the device using SVC GTUNIT.

There are two advantages to using this subroutine instead of the direct use of the SVC GTUNIT. The first is that

there will be no conflicts in the use of LDNs between MTS and user programs. The second is that the FDUB is automatically released when the program is unloaded, and when all FDUBs for the device are released, the device and LDN are released. If the SVC GTUNIT is used directly, a device may end up attached to the task with no method of releasing it without providing a short program to release it, or stopping the job (a signoff is insufficient).

April 1981

GETFD

System Subroutine Description

Purpose: To obtain a file or device.

Location: Resident System

Entries: GETFDN, GETFD1, GETFD5, GETFD6

Calling Sequence:

Assembly:
```
LA   1,FDname
CALL GETFD
```

Parameters:

GR1 contains the location of the first character of the FDname of the file or device wanted. The complete name must be terminated by a blank. The name does not have to be aligned.

Return Codes:

```
 0 Successful return.
 4 Invalid address.
 8 Device is busy.
12 Device is not operational.
```

GETFD will give a zero return code for nonexistent, nonaccessible, or invalid file or device names. The type code given by word 2 of the information area from GDINFO, GDINFO2, or GDINFO3 can be used to check for the status of the file or device. This type code should always be tested for the validity of the result from GETFD since nonzero return codes are rarely returned by GETFD. A type code of "NONE" will indicate a nonvalid result from GETFD.

Values Returned:

GR0 contains the FDUB-pointer if a successful return is made.

Notes: FORTRAN users can call this subroutine by using the RCALL subroutine.

Normally, calls to GETFD assign the FDUB to the correct FDUB chain for the CLS. Calls to GETFD5 place the FDUB on chain 5. Calls to GETFD6 place

the FDUB on chain 6. Calls to GETFDN place the FDUB on the chain specified by the chain number given in GR0. GETFD1 is an alternate name for GETFD5. The calling sequence for the above entry points is the same as for GETFD.

The FDUB chain numbers are assigned as follows:

0 - MSOURCE and MSINK
1 - SOURCE, previous SOURCE, SINK, previous SINK, and $LOGged files and devices
2 - AFD
3 - MTS CLS
4 - USER CLS
5 - Mounted devices, held *...*'s, and $LOCKED files
6 - Device support routines

The chain numbers for the other CLS's can be determined by adding 5 to the CLS number (see the description of the GUINFO/CUINFO subroutine in this volume). There is no special CLS chain for the MTS(0) and USER(1) CLS's.

Description: If the name is a device, the device is acquired. If the name is a file, the file is not opened until the first usage. Thus this subroutine cannot determine whether or not a file exists. The caller can determine whether the file exists by calling GDINFO. The name may be a concatenation of file or device names each followed by modifiers or a line number range as described in "Files and Devices" in MTS Volume 1, The Michigan Terminal System. If the FDUB-pointer returned is used in a call to READ or WRITE, the modifiers or line number ranges will be used, and if a concatenation was specified, the usual sequencing through the concatenation will take place.

Example: Assembly:

```
        LA   1,FNAME
        CALL GETFD
         .
         .
FNAME DC   C'DATAFILE '
```

FORTRAN:

```
      INTEGER ADROF
      EXTERNAL GETFD
      LOGICAL*1 FNAME(9)/'DATAFILE '/
       ...
      CALL RCALL(GETFD,2,DUMMY,ADROF(FNAME),1,FDUB)
```

The above examples call GETFD to obtain a FDUB-pointer for the file DATAFILE.

GETSPACE

System Subroutine Description

Purpose: To acquire storage.

Location: Resident System

Alt. Entry: GETSPA

Calling Sequences:

Assembly:
```
L     0,switch
L     1,length
CALL  GETSPACE

L     0,switch
L     1,length
L     2,index
CALL  GETSPACE
```

```
GETSPACE [length][,T=switch][,EXIT=err]
```

Parameters:

GR0 contains switches:

Bit 31 = 1 Return not made unless space is available.
0 Return always made with return code indicating whether space is available.

30 = 1 Storage acquired is associated with the current level of LINK so that it is released at the next return from a LINK, or the next XCTL. This bit is ignored if bit 28 is set.
0 Storage acquired is associated with the highest level program so that it is not released until execution terminates.

29 = 1 Allocate storage in the system segment. Bit 29 is ignored if bit 28 is set or if bits 0-7 of general register 1 are not zero. This option is valid only in system mode.

28 = 1 Use storage index number in general register 2.

27 = 1 Allocate storage in the virtual machine segment (ignored if an explic-

it segment number is given in general register 1).

26 = 1 Allocate protectable storage (half-page multiples, half-page aligned).

25 = 1 (privileged) Allocate storage in the system segment which will be released when the current CLS is unloaded. This is the same as setting bit 24 when GETSPACE is called in the MTS CLS.

24 = 1 (privileged) Allocate storage in the system segment which will be released when the current MTS command is finished. (This is known internally as SFTC storage.)

Other bits in GR0 must be zero.

GR1 contains the length (in bytes) of storage desired. If this is not a multiple of 8, the next largest multiple of 8 will be used. The upper limit for a storage request is 1,048,576 bytes (1 segment).

Normally space will be allocated wherever available in virtual memory. However, if the first byte (byte 0) of GR1 is nonzero, it is assumed to be the number of the segment in which the storage is to be allocated. If this is an invalid number [is less than 6, or is greater than the maximum (currently 12)], or if this space request cannot be allocated in this segment, a return is made with a return code of 4.

GR2 contains the storage index number to use if bit 28 of GR0 is 1; otherwise, GR2 is ignored.

A GR13 save area is not required for a call to this subroutine.

Values Returned:

GR1 contains the location of the first byte of the storage region acquired. The first word of this region is set to the length (in bytes) of the region.

Return Codes:

0 Successful return. Storage has been acquired.
4 Space is not available.

Notes: FORTRAN users can call this subroutine by using the RCALL subroutine and giving GETSPA as the entry point.

The complete description for using the GETSPACE macro is given in MTS Volume 14, 360/370 Assemblers in MTS.

Storage located in system segments may only be allocated or released from programs executing in system mode.

If GETSPACE is called from a nonexecution CLS through the CLS transfer vector or through the V-con GETSPCLS, the high-order bit of GR0 is not 1, and the current CLS is not the USER CLS, then the call is made as if GR0 bit 29 was set. Unless bit 30 is set, this storage is assigned the storage index number for the current CLS or, in the case of MTS, is assigned to "storage for this command."

Description: See the "Virtual Memory Management" section in MTS Volume 5, System Services, for further details on storage allocation and storage index numbers.

Examples: Assembly:

```
          L    0,SWITCH
          L    1,LENGTH
          CALL GETSPACE
           .
           .
SWITCH DC   F'0'
LENGTH DC   F'256'
```

FORTRAN:

```
      INTEGER SPACE
      EXTERNAL GETSPA
        ...
      CALL RCALL(GETSPA,2,0,256,2,DUMMY,SPACE)
```

The above two examples call GETSPACE to acquire 256 bytes of storage. The storage will be associated with the highest level program.

April 1981

GPKSPACE

System Subroutine Description

Purpose: To return a storage protection key for a storage region in either the virtual or real machine environment.

Location: Resident System

Calling Sequences:

Assembly:
```
L    1,region
CALL GPKSPACE
```

Parameters:

GR1 contains the address of the region for which the storage key is to be returned.

Values Returned:

GR0 Bits 0-15: Unused
16-23: Storage key
24-30: Unused
31: 0, if real machine key
1, if virtual machine key

Return Codes:

0 Successful return.
4 Storage not allocated to this task or not allocated within a single GETSPACE allocation. GR0 is still valid.
8 Invalid region address.

Notes: For use in virtual-machine mode, the storage key is specified in bits 16-22; bit 23 is unused and should be zero.

For use in real-machine mode, bits 16-23 should have one of the following values:

0 = reset to normal
1 = store protect
2 = fetch protect

April 1981

April 1981

GPSECT, QPSECT, FPSECT

System Subroutine Description

Purpose: To acquire, query, and release psect (dsect) storage allocations.

Location: Resident System

Calling Sequences:

Assembly:
```
L     0,id
L     1,length
CALL  GPSECT

L     0,id
CALL  QPSECT

L     0,id
CALL  FPSECT
```

Parameters:

GR0 contains an unique fullword identifier for the psect (i.e., a fixed address within the calling program could be used as such an identifier).
GR1 (GPSECT only) contains the length of the psect to be allocated.

A GR13 save area is not required for a call to the GPSECT, QPSECT, or FPSECT subroutines.

Values Returned:

GR1 (GPSECT only) contains the address of the psect allocated.

GR1 (QPSECT only) contains the address of the psect if found, otherwise zero.

Return Codes:

GPSECT:

0 Psect found.
4 Psect not found but allocated.
8 Error return from GETSPACE subroutine.
12 Internal error in GPSECT.

QPSECT:

0 Psect found.
4 Psect not found.
8 Not used.
12 Internal error in QPSECT.

FPSECT:

0 Psect released.
4 Psect not found.
8 Error return from FREESPAC subroutine.
12 Internal error in FPSECT.

Note: A separate collection of psect ids is maintained by each CLS. Storage allocated for CLS's other than the USER CLS is located in the system segment and is assigned the storage index number used by the CLS.

Description: The GPSECT, QPSECT, and FPSECT subroutines are used to acquire, query, and release storage to be used for psects (dsects) in the calling program. An identifier for the psect and the length of the psect are specified in id and length.

The GPSECT subroutine is used to allocate storage for the psect. If a psect with the identifier id already exists, its address is returned and a new psect is not allocated.

The QPSECT subroutine is used to query the existence of a psect with the identifier id. A new psect is not allocated.

The FPSECT subroutine is used to release the storage for the psect with identifier id.

Example: Assembly:

```
         L     0,ID
         L     1,LEN
         CALL  GPSECT
          .
          .
         L     0,ID
         CALL  FPSECT
          .
          .
ID       DC    A(ID)
LEN      DC    F'4096'
```

The example allocates a psect of 4096 bytes with the identifier which is an address contained within the calling program (e.g., the address of ID). The psect is then released later in the program.

April 1981

GUINFO/CUINFO

System Subroutine Description

Table of System Items Arranged by Index

Index	Name	Size	Description
34	LDTLOCN	Fullword	Address of table of components dynamically loaded by MTS (LOADT table)
53*	UNLODOFF	Fullword	1 -> $SET UNLOAD=OFF
97	FILETV	Fullword	Address of file transfer vector used by MTS
101	CATBUFF	Fullword	Address of catalog buffer used by MTS
102	BTCHMORE	Fullword	1 -> MORE specified on MTS job request
103	DEVTBLS	Fullword	Address of device tables used by MTS
105			Used for array-processor billing (UQV)
111	ASYNCCTL	Fullword	Asynchronous event control switch[13]
115	RUNETIME	Dblword	Cumulative real time for CLS[5]
123	USERGONE	Fullword	User is being signed off
132	LOADT	Fullword	Address of load tables used by MTS
140	SCRFILES	Fullword	Address of table of scratch files owned by task
142	PDNTBL	Fullword	Address of pseudodevice name table
143	CMDSEEN	Fullword	1 -> Command "seen" by user
144	SCRFNAME	8 Bytes	Internal scratch-file prefix (replaces SCRFCHAR to yield scratch-file name)
153*	CNT9TP	Fullword	Count of 9-track tapes (UQV)
155*	CNT7TP	Fullword	Count of 7-track tapes (UQV)
160	BILLCLAS	Fullword	Billing Class[14]
161	RTDCA	Fullword	Address of data area used by RESPONSE-time data collection facility
165	CLSNAME	4 Bytes	Name of CLS currently in control (characters)
198	RATENBR	Fullword	Number determining rate set in use
200-225			Reserved for local installations (UBC starts at 200)
233*	NOSDS	Fullword	1 -> $SET SDSMSG=OFF (default is ON)
235	NOFILERE	Fullword	1 -> $SET FILEREF=OFF
243	UNPROT	Fullword	1 -> User may set $SET PROT=OFF (logical OR of items 246 and 250)
244	PRIVPROG	Fullword	1 -> A system-mode program is loaded as the USER CLS
245	PROTOFF	Fullword	1 -> $SET PROT=OFF
246	ACCPRIV	Fullword	1 -> The accounting privileged bit (ACCPRIV in ACCSWS1) is set
248	ACCCCPF	Fullword	1 -> User may create public files (ACCCCLF in ACCSWS1)
250	ACCPUSE	Fullword	1 -> A protection-privileged user (ACCPUSE in ACCSWS2)
256	SIGNEDON	Fullword	1 -> User is fully signed on.
261-273			Reserved for use by UBC
274	AUTOSIG	Fullword	1 -> Automatic signoff after "n" idle minutes

			may not be suppressed by user
275			Reserved for use by UQV
278-282			Used for Xerox 9700 billing (WSU)
283-290			Used for floppy-disk and array-processor bill-ing (UQV)
291			Used for Xerox 9700 billing (WSU)
292			Used for surcharge billing (WSU)
297			Used for Xerox 9700 billing (UQV)
299	USERNMID	4 Bytes	Name ID set by $SET NAME=name (characters)
317-319			Reserved for future INITFILEs
330	MACRO	Fullword	$SET MACRO={OFF|ON|BROKEN} (0|1|2) (default OFF)
331	MACVFDST	Fullword	Address of virtual file dsect (used by macro processor)
332-333			Reserved for UQV (special forms)
337	LOCPPGS	Fullword	Local page-printer page limit
338	GLOBPPGS	Fullword	Global page-printer page limit
339-352			Reserved for UBC (resource manager)
353	PSWGRFR	104 Byts	PSW, GRS, FRS for specific CLS (used by SDS and $DIS)
354	LDDCTPTR	Fullword	Address of loader setup area for EXEC CLS (used by SDS)
355	APSWGRFR	Fullword	Address of PSW, GRS, FRS for specific CLS (used by SDS)
356	SUBTASK	Fullword	1 -> This is a subtask
357	STSKMSTR	Fullword	Task number of subtask's "master" task
358	STSKNAME	4 Bytes	Subtask name (left-justified with trailing blanks)
359	SRCINVID	Fullword	Macro processor invocation ID
360-363			Reserved for RPI
364	EDSYMTAB	Fullword	Address of editor symbol table
365-374			Reserved for UQV
376	FCBCHAIN	Fullword	Address of file control block chain
378-380			Reserved for RPI
381	SOUFDUB	Fullword	FDUB for *SOURCE*
382	PSOUFDUB	Fullword	FDUB for previous $SOURCE
383	SINKFDUB	Fullword	FDUB for *SINK*
384	PSINFDUB	Fullword	FDUB for previous $SINK
385	AFDFDUB	Fullword	FDUB for *AFD*
389	RUNCSTRT	Fullword	Head of chain of previous $RUN commands (used by $DIS RUNS)
390	RUNCEND	Fullword	End of chain of previous $RUN commands (used by $DIS RERUN)
394-399			Reserved for UBC
401-410			Reserved for UQV (phototypesetter and "new" array processor)
411	USERDEF1	Fullword	Used for program surcharging (add only)
412	USERDEF2	Fullword	Used for program surcharging (add only)
413	USERDEF3	Fullword	Used for program surcharging (add only)
414	USERDEF4	Fullword	Used for program surcharging (add only)
415	USERDEF5	Fullword	Used for program surcharging (add only)
419-428			Reserved for UBC
429*	TYPEPTUS	Fullword	Cum. phototypesetter units for task

430*	TYPEPAPR	Fullword	Cum. phototypesetter media for task (cm²)
431	SINUMBR	Fullword	???????
434-439			Reserved for UQV
441-446			Reserved for UQV
447	EDITPSCT	Fullword	Global storage pointer for EDIT subroutine
448			Reserved for UQV
449	AMNTRLCK	Fullword	Address of subtasking monitor coroutine services lock byte
450			Reserved for UNE

---

[13]Asynchronous event control
    Bit 31: 1 -> Stack attention interrupts
        30: 1 -> Stack attention interrupts unless CLS has an ATTNTRP exit enabled
        29: 1 -> Stack timer interrupts while CLS is active
        28: 1 -> Stack interrupts that cause control to be taken away from this CLS and given to another CLS (e.g., the MTS out-of-money check)
        27   1 -> Don't quit even if the job STOPped or DSROFF is called

[14]Billing class
    0=University/Government
    1=Commercial
    2=Exchange
    3=Non-University
    4=Special University/Government (EPA)
    5=Research
    6=Course

April 1981

April 1981

KWSCANNM

System Subroutine Description

Purpose: To perform keyword processing on a character string. This subroutine is designed for use by non-MTS jobs or MTS jobs where it is not safe to call standard system subroutines such as GETSPACE and GETFD.

Location: Resident System

Calling Sequence:

The calling sequence for KWSCANNM is the same as for KWSCAN except that the rvec parameter (word 8 in the parameter list) specifies a region of 1024 bytes of storage for use by KWSCANNM. The first 27 words of rvec are used as a return area as described for KWSCAN.

Description: KWSCANNM provides the same features as KWSCAN except that some of the sws bits are not available since they require features available only in the MTS environment. For example, prompting is not available, the FDname right-hand side type is not available, and features which require the use of FDnames or GETSPACE calls are not available. Specifically, only the following sws bits are available for KWSCANNM; all other bits are ignored:

15	Execute subroutines instead of instructions.
16-17	Perform spelling correction.
18	Formatting of return vector.
19	Allow parenthesized left-hand sides.
20-21	Special separators between LHS and RHS.
22	Special delimiter characters.
23	Left-hand side initial substrings.
27	2-byte lengths used in left-hand table.

April 1981

April 1981

PGNTTRP

Subroutine Description

Purpose: To allow control to be returned to the user on a program interrupt.

Location: Resident System

Alt. Entry: PGNTT

Calling Sequences:

Assembly:
```
LM   0,1,=A(exit,region)
CALL PGNTTRP
```

Parameters:

GR0 should contain zero or the location to transfer control to if a program interrupt occurs.
GR1 should contain the location of a 72-byte save region for storing pertinent information.

Return Codes:

0 Successful return.
4 Illegal parameter specified.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine specifying PGNTT as the entry point.

Description: A call on the subroutine PGNTTRP sets up a program interrupt intercept for one interrupt only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of a program interrupt. When an interrupt occurs and the exit is taken, the intercept is cleared so that another call to PGNTTRP is necessary to intercept the next program interrupt. When a program interrupt occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location previously specified. If the exit subroutine returns to MTS (BR 14), MTS will handle the interrupt as if PGNTTRP had not been called originally. This feature allows the user to take brief control of the interrupt before MTS takes complete control of the interrupt. When MTS takes control of the interrupt, execution of the program will be terminated and a message will be printed providing the location of the interrupt.

If GR0 is zero on a call to PGNTTRP, the program interrupt intercept is disabled. GR1 should be zero or point to a valid save region.

When the program interrupt exit is taken, the first eight bytes of the save region contain the program interrupt PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the interrupt. The floating-point registers remain as they were at the time of the interrupt. GR1 will contain the location of the save region. GR2 contains bits indicating attributes of the interrupted program:

bit 28: 1 - (X'00000008') if program was executing in user execution mode.
bit 27: 1 - (X'00000010') if program was executing in virtual-machine mode.
bit 25: 1 - (X'00000040') if program was executing with SVC trapping enabled.

The remaining bits of GR2 are unpredictable. The contents of GR0 and GR3 through GR12 are unpredictable.

If, on a call to PGNTTRP, the first byte of the save region is X'FF', PGNTTRP does not return to the calling program; rather the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on PGNTTRP, after the first program interrupt exit is taken, allows the user to set a switch (for example) and to return to the point at which he was interrupted with the program interrupt intercept again enabled.

The PGNTTRP item of the GUINFO/CUINFO subroutine may be used to save a previous exit address and associated region so that it may be later restored.

April 1981

PKEY

System Subroutine Description

Purpose: To push and pop program keys.

Location: Resident System

Calling Sequences:

Assembly: CALL PKEY,(string,pkey),VL

FORTRAN: CALL PKEY(string,pkey,&rc4,&rc8,&rc12,&rc16)

Parameters:

string is the location of a command (see below) terminated with a trailing blank.

pkey (optional) is the location of a new program key terminated with a trailing blank.

rc4,...,rc16 (optional) are statement labels to transfer to if a nonzero return code occurs.

VL must be specified even if both parameters are given in order to facilitate the addition of new parameters.

Return Codes:

0 Successful return.
4 Invalid command string.
8 Invalid program key.
12 Attempt to push or pop too many times.
16 Invalid parameters.

Description: The legal command strings are:

PUSH The current program key is pushed onto the stack of program keys (one stack per CLS) and the new program key is made the current key. The new key must be in the table of program keys. If no new program key is specified, the current program key is pushed onto the stack, but remains the current key.

POP The program key on the top of the stack is made the current program key and is removed from the stack. The pkey parameter is not required.

SET The new program key is made the current program key. The old program key is not pushed onto the stack of program keys.

RESET The stack of program keys is cleared, and the current program key is reset to its original value.

Subject to restrictions on which program keys are valid, calls to PKEY made be made in either user or system mode.

Currently, user programs may only specify the program key *EXEC in addition to the program key assigned to the file being executed. This will be expanded to include other program keys in the future.

Example:

```
FORTRAN:        CALL FTNCMD('ASSIGN 99=WXYZ:LOGFILE; ')
                CALL PKEY('PUSH ','*EXEC ')
                 ...
                CALL PKEY('POP ')
                WRITE(99,100) A,B,C
          100   FORMAT(3F10.2)
                CALL PKEY('PUSH ','*EXEC ')
                 ...
```

The above example assigns FORTRAN I/O unit 99 to the file WXYZ:LOGFILE, which is permitted to a program key. When the program writes into this file, the PKEY subroutine is called to switch the program key from *EXEC to the program key of the file, and subsequently is called to restore the program key back to *EXEC.

April 1981

RERUN

System Subroutine Description

Purpose: To force a HASP batch job to be rerun. The current job is not charged.

Location: Resident System

Calling Sequences:

Assembly: CALL RERUN

Return Codes:

4 Job is not a HASP batch job.

If the job can be rerun, the subroutine does not return.

Description: This subroutine can only be executed from system mode. Programs calling RERUN should write a message to the operator's console explaining why the job is being rerun.

April 1981

April 1981

SETFPRIV

System Subroutine Description

Purpose: To make a file privileged.

Location: Resident System

Alt. Entry SETFP

Calling Sequences:

Assembly: CALL SETFPRIV,(what,onoff,info,errcode,errmsg)

Parameters:

what is the location of either
(a) a file name with a trailing blank (if info=0),
(b) a fullword-integer FDUB pointer (such as returned by GETFD) (if info=1),
(c) a fullword-integer logical I/O unit number (if info=1), or
(d) an 8-character, left-justified logical I/O unit name (e.g., SCARDS) (if info=1).

onoff is the location of a fullword-integer 0 or 1. If 1, the file is made privileged.

info is the location of a fullword-integer 0 or 1 which identifies the type of what parameter.

errcode (optional) is the location of a fullword in which SETFPRIV will place the error number if an error return (return code 4) is made. If errcode is omitted, the errmsg parameter must also be omitted. Assembly language users who wish to omit this parameter should either follow the variable parameter list convention (high-order bit of the previous parameter adcon is set to 1) or supply an adcon which is zero (rather than pointing to a zero).

Error numbers less than 100 indicate an error in the mechanics of the call or in the value of the parameters.

Number	Message
1	Illegal parameter list pointer
2	Illegal "what" parameter address
3	Illegal "ONOFF" parameter address
4	"ONOFF" parameter value not 0 or 1

5 Illegal "info" parameter address
7 Only privileged users may change a file's privileged status
11 "info" parameter value not 0 or 1

Error numbers between 100 and 105 indicate errors that occur in accessing the file.

101 Illegal file name
102 File not found - file "xxxx"
103 Access not allowed to file "xxxx" (Permit access is required to set the privileged status.)
104 Deadlock situation, try later - file "xxxx"
105 Interrupted out of wait for locked file "xxxx"

Error numbers 201 and above indicate a system error.

errmsg (optional) is the location of a 20-fullword (80-character) region in which SETFPRIV will place the corresponding error message if an error occurs.

rc4 is the statement label to transfer to if the corresponding return code occurs.

Return Codes:

0 The privileged status of the file has been changed.
4 Error return. The privileged status has not been changed, but the errcode and errmsg values have been set, if specified.

April 1981

SPKSPACE

System Subroutine Description

Purpose: To set storage protection keys for a storage region in either the virtual or real machine environment.

Location: Resident System

Calling Sequences:

Assembly:
```
LM   0,2,pars
CALL SPKSPACE
```

Parameters:

GR0 contains the length in bytes of the region for which storage keys are to be set.

GR1 contains the address of the region for which the storage keys are to be set.

GR2
```
Bits  0-15:  Unused (must be zero)
     16-23:  Storage key
     24-30:  Unused (must be zero)
        31:  0, if real machine key
             1, if virtual machine key
```

Return Codes:

0 Successful return.

4 Storage region not contained within a single GETSPACE allocation.

8 Invalid parameters, i.e., length not a multiple of 2048, region not half-page aligned, or user-mode call attempting to change keys for a nonuser page.

Notes: For use in virtual-machine mode, the storage key is specified in bits 16-22; bit 23 is unused and should be zero.

For use in real-machine mode, bits 16-23 should have one of the following values:

```
0 = reset to normal
1 = store protect
2 = fetch protect
```

April 1981

April 1981

STARTJOB

System Subroutine Description

Purpose: To allow a user-mode program to issue the STARTJOB SVC with certain predefined argument strings.

Location: Resident System

Calling Sequences:

Assembly: CALL STARTJOB,(name,tasknum),VL

Parameters:

name is an 8-character, left-justified name that must match a name in the internal STARTJOB table. This table includes the argument string that is to be used with the STARTJOB SVC as well as a program key that must match the caller's current program key before the call will be allowed.

tasknum (optional) is the task number of the task started by the STARTJOB SVC.

Return Codes:

0 Successful return.
4 Unable to start the job after three attempts.
8 name not in the table.
12 Program key not correct for the name specified.
16 Invalid parameters.

April 1981

April 1981

SVCTRP

Subroutine Description

Purpose: To suspend program execution whenever an SVC instruction is executed by a user program.

Location: Resident System

Calling Sequences:

Assembly:
```
LM   0,1,=A(exit,region)
CALL SVCTRP
```

Parameters:

GR0 should contain zero or the location to transfer control to if an SVC instruction is executed.
GR1 should contain the location of a 72-byte save region for storing pertinent information.

Return Codes:

0 Successful return.
4 Illegal parameter specified.

Note: FORTRAN users can call this subroutine by using the RCALL subroutine specifying SVCTRP as the entry point.

Description: A call on the subroutine SVCTRP sets up an SVC intercept for one SVC instruction only. The calling sequence specifies the save region for storing information and a location to transfer to upon the next occurrence of an SVC instruction in the user program. When an SVC instruction is encountered and the exit is taken, the intercept is cleared so that another call to SVCTRP is necessary to intercept the next SVC instruction. When a SVC instruction occurs, the exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save region provided) to the location specified by the GR0 value in the call to SVCTRP. If the exit subroutine returns to MTS (BR 14), MTS will declare the SVC instruction invalid, suspend program execution, and print a message providing the location of the intercept.

If GR0 is zero on a call to SVCTRP, the SVC intercept is disabled. GR1 should point to a valid save region in this case also.

When the SVC intercept exit is taken, the first eight bytes of the save region contain the PSW, and the remainder contains the contents of general registers 0 through 15 (in that order) at the time of the intercept. The floating-point registers remain as they were at the time of the intercept. GR1 will contain the location of the save region. GR2 contains bits indicating attributes of the interrupted program:

bit 28: 1 - (X'00000008') if program was executing in user execution mode.
bit 27: 1 - (X'00000010') if program was executing in virtual-machine mode.
bit 26: 1 - (X'00000020') if program was executing with SVC trapping enabled.

The remaining bits of GR2 are unpredictable. The contents of GR0 and GR3 through GR12 are unpredictable.

If, on a call to SVCTRP, the first byte of the save region is X'FF', SVCTRP does not return to the calling program; rather the right-hand half of the PSW and the general registers are immediately restored from the save region and a branch is made to the location specified in the second word of the region. This type of call on SVCTRP, after the first SVC instruction has been intercepted, allows the user to set a switch (for example) and to return to the point following the SVC instruction with the intercept again enabled.

The SVCTRP item of the GUINFO/CUINFO subroutine may be used to save a previously set exit to allow nesting of SVC intercepts.

Example: In this example, the location of the first SVC instruction in a user program is recorded and execution is resumed with at the SVC instruction.

```
        LM   0,1,=A(EXIT,REGION)
        CALL SVCTRP     The intercept is enabled.
         .
         .
        USING EXIT,15
EXIT    L    0,4(,1)    Get address of SVC
        SL   0,=F'2'    Back up to SVC instruction
        ST   0,FIRSVC   Remember location
        ST   0,4(,1)    Restart at SVC
        MVI  0(1),X'FF'
        SR   0,0        Disable further intercepts
        CALL SVCTRP     Note GR1 points to REGION
REGION  DS   18F
FIRSVC  DS   A
```

April 1981

TIMNTRP

Subroutine Description

Purpose: To enable, disable, or return from timer interrupts set by the SETIME subroutine.

Location: Resident System

Calling Sequences:

Assembly:
```
LM   0,1,=A(exit,region)
CALL TIMNTRP
```

Parameters:

GR0 should contain zero or the location of the exit routine to transfer control to when a timer interrupt occurs.
GR1 should contain the location of a 76-byte exit region for storing pertinent information.

Return Codes:

0 Successful return.
4 Illegal parameter specified.

Description: A call on the TIMNTRP subroutine sets up an exit for one timer interrupt only. The calling sequence specifies the location of an exit routine to transfer control to when the next timer interrupt occurs and an exit region for storing information. The timer interrupts themselves are set up by calls to the SETIME subroutine.

TIMNTRP may be called several times with different exit regions and different exit routines specified. Each call on SETIME must also specify the exit region to be used when the interrupt occurs. This "subsetting" capability allows separate parts of large programs to use the timer interrupt facility independently.

If GR0 is zero, timer interrupt exits for the specified exit region are disabled. If, when a timer interrupt occurs, its exit is disabled, the interrupt will remain pending until the next call on TIMNTRP which enables the exit, and the exit will be taken immediately following the call.

When a timer interrupt exit is taken, the exit is disabled, so that further timer interrupts which specify

this exit region will remain pending while the current one is being processed. The exit is taken in the form of a subroutine call (BALR 14,15 with a GR13 save area provided). At the time of this call, GR1 will point to the exit region, whose contents will be

Word 1:	the identifier passed to SETIME when the interrupt was set up.
Words 2-3:	the PSW at the time of the interrupt.
Words 4-19:	GR0-GR15 (in that order) at the time of the interrupt.

GR2 contains bits indicating attributes of the interrupted program:

bit 28:	1 - (X'00000008') if program was executing in user execution mode.
bit 27:	1 - (X'00000010') if program was executing in virtual-machine mode.
bit 25:	1 - (X'00000040') if program was executing with SVC trapping enabled.

If the exit routine returns to MTS (BR 14), the user's program will be restarted at the point of the interrupt. The exit will be reenabled if the return code in GR15 is zero; otherwise, the exit will remain disabled until another call on TIMNTRP. The registers must be restored in the standard fashion when the exit routine returns.

For further details, see also the GETIME, RSTIME, and SETIME subroutine descriptions.