1.5 Host file server library

Tony Debling, SGS-Thomson Microelectronics Ltd, 1000 Aztec West, Almondsbury, Bristol BS12 4SQ, England. June 7, 1995.
Converted to HTML via RTFtoHTML, PERL5 and by hand by Dave Beckett <*D.J.Beckett@ukc.ac.uk*>.

WARNING: This has been converted from RTF format and may have mistakes compared to the printed version. If you find any in this file, please send them to me, and I will update it.

Dave Beckett <*D.J.Beckett@ukc.ac.uk*>

Library: `hostio.lib`

The host file server library contains routines that are used to communicate with the host file server. The routines are independent of the host on which the server is running. Using routines from this library you can guarantee that programs will be portable across all implementations of the toolset.

Constant and protocol definitions for the hostio library, including error and return codes, are provided in the include file `hostio.inc`.

The result value from many of the routines in this library can take the value w `spr.operation.failed` which is a server dependent failure result. It has been left open with the use of w because future server implementations may give more information back via this byte.

1.5.1 Errors and the server run time library

The hostio routines use functions provided by the host file server. These are defined in Appendix C in the *Toolset Reference Manual*. The server is implemented in C and uses routines in a C run time library, some of which are implementation dependent.

In particular, the hostio routines do not check the validity of stream identifiers, and the consequences of specifying an incorrect streamid may differ from system to system. For example, some systems may return an error tag, some may return a text message. If you use only those stream ids returned by the hostio routines that open files (`so.open`, `so.open.temp`, and `so.popen.read`), invalid ids are unlikely to occur. It is also possible in rare circumstances for a program to fail altogether with an invalid streamid because of the way the C library is implemented on the system. This error can only occur if direct use of the library to perform the operation would produce the same error.

1.5.2 Inputting real numbers

Routines for inputting real numbers only accept numbers in the standard occam format for REAL numbers. Programs that allow other ways of specifying real numbers must convert to the occam format before presenting them to the library procedure.

For details of the occam syntax for real numbers see the *occam 2 Reference Manual*.

1.5.3 Procedure descriptions

In the procedure descriptions, **fs** is the channel *from* the host file server, and **ts**is the channel *to* the host file server. The **SP** protocol used by the host file server channels is defined in the include file **hostio.inc**. The hostio routines are divided into six groups: five groups that reflect function and use, and a sixth miscellaneous group. The five specific groups are:

- File access and management
- General host access
- Keyboard input
- Screen output
- File output

Each group of routines is described in a separate section. Each section begins with a list of the routines in the group with their formal parameters. This is followed by a description of each routine in turn. **Note**: for those routines which write data to a stream (including the screen), if the data is not sent as an entire block then it cannot be guaranteed that the data arrives contiguously at its destination. This is because another process writing to the same destination may interleave its server request(s) with those of these routines.

1.5.4 File access

This group includes routines for managing file streams, for opening and closing files, and for reading and writing blocks of data.

```
Procedure           Parameter Specifiers
so.open             CHAN OF SP fs, ts, VAL []BYTE name,
                    VAL BYTE type, mode, INT32 streamid,
                    BYTE result
so.open.temp        CHAN OF SP fs, ts, VAL BYTE type,
                    [so.temp.filename.length]BYTE filename,
                    INT32 streamid, BYTE result
so.popen.read       CHAN OF SP fs, ts, VAL []BYTE filename,
                    VAL []BYTE path.variable.name,
                    VAL BYTE open.type, INT full.len,
                    []BYTE full.name, INT32 streamid,
                    BYTE result
so.close            CHAN OF SP fs, ts, VAL INT32 streamid,
                    BYTE result
so.read             CHAN OF SP fs, ts, VAL INT32 streamid,
                    INT length, []BYTE data
so.write            CHAN OF SP fs, ts, VAL INT32 streamid,
                    VAL []BYTE data, INT length
so.gets             CHAN OF SP fs, ts, VAL INT32 streamid,
                    INT length, []BYTE data, BYTE result
so.puts             CHAN OF SP fs, ts, VAL INT32 streamid,
                    VAL []BYTE data, BYTE result
so.flush            CHAN OF SP fs, ts, VAL INT32 streamid,
                    BYTE result
so.seek             CHAN OF SP fs, ts, VAL INT32 streamid,
                    VAL INT32 offset, origin, BYTE result
so.tell             CHAN OF SP fs, ts, VAL INT32 streamid,
                    INT32 position, BYTE result
so.eof              CHAN OF SP fs, ts, VAL INT32 streamid, BYTE result
so.ferror           CHAN OF SP fs, ts, VAL INT32 streamid, INT32 error.no,
                    INT length,
                    []BYTE message, BYTE result
so.remove           CHAN OF SP fs, ts, VAL []BYTE name,
                    BYTE result
so.rename           CHAN OF SP fs, ts,
```

```
                    VAL []BYTE oldname, newname, BYTE result
so.test.exists      CHAN OF SP fs, ts,
                    VAL []BYTE filename, BOOL exists
```

Procedure definitions

so.open

```
PROC so.open  (CHAN OF SP fs, ts,
               VAL []BYTE name,
               VAL BYTE type, mode,
               INT32 streamid, BYTE result)
```

Opens the file given by **name** and returns a stream identifier **streamid** for all future operations on the file until it is closed. If **name** does not include a directory then the file is searched for in the current directory. File type is specified by **type** and the mode of opening by **mode**. **type** can take the following values:

```
spt.binary                  File contains raw bytes only.Binary byte
                            stream
spt.text                    File contains text records separated by
                            newline sequences.Text:stream
```

mode can take the following values:

```
spm.input                   Open existing file for reading.
spm.output                  Open new file, or truncate an existing
                            one, for writing.
spm.append                  Open a new file, or append to an
                            existing one, for writing.
spm.existing.update         Open an existing file for update
                            (reading and writing), starting at
                            beginning of the file.
spm.new.update              Open new file, or truncate existing one,
                            for update.
spm.append.update           Open new file, or append to an existing
                            one, for update.
```

result can take the following values:

```
spr.ok                        The open was successful.
spr.bad.name                  Null file name supplied.
spr.bad.type                  Invalid file type.
spr.bad.mode                  Invalid open mode.
spr.bad.packet.size           File name too large
                              (i.e.> sp.max.openname.size).
wspr.operation.failed         If result w  spr.operation.failed then
                              this denotes a server returned
                              failure. (See section C.2 in the
                              Toolset Reference Manual.)
```

so.open.temp

```
PROC so.open.temp
        (CHAN OF SP fs, ts,
         VAL BYTE type,
```

```
            [so.temp.filename.length]BYTE filename,
            INT32 streamid, BYTE result)
```

Opens a temporary file in **spm.new.update** mode. The first filename tried is **temp**00. If the file already exists the **nn** suffix on the name **temp***nn* is incremented up to a maximum of 9999 until an unused number is found. If the number exceeds 2 digits the last character of **temp** is overwritten. For example: if the number exceeds 99 the **p** is overwritten, as in **tem999**; if the number exceeds 999, the **m** is overwritten, as in **te9999**. File type can be **spt.binary** or **spt.text**, as with **so.open**. The name of the file actually opened is returned in filename. The result returned can take any of the following values:

```
spr.ok                      The open was successful.
spr.notok                   There are already 10,000 temporary
                            files.
spr.bad.type                Invalid file type specified.
wspr.operation.failed       If result w  spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

so.popen.read

```
PROC so.popen.read
              (CHAN OF SP fs, ts,
               VAL []BYTE filename,
               VAL []BYTE path.variable.name,
               VAL BYTE open.type,
               INT full.len, []BYTE full.name,
               INT32 streamid, BYTE result)
```

As for **so.open**, but if the file is not found and the filename does not include a directory, the routine uses the directory path string associated with the host environment variable, given in **path.variable.name**, and performs a search in each directory in the path in turn. This corresponds to the searching rules used by the toolset, using the environment variable ISEARCH, see section in the *occam 2 Toolset User Guide* . File type can be **spt.binary** or **spt.text**, as with **so.open**. The mode of opening is always **spm.input**. The name of the file opened is returned in **full.name**, and the length of the file name is returned in **full.len**. If no file is opened, **full.len** and **full.name** are undefined, and the result will not be **spr.ok**. The result returned can take any of the following values:

```
spr.ok                      The open was successful.
spr.bad.name                Null name supplied.
spr.bad.type                Invalid file type specified.
spr.bad.packet.size         File name is too large
                            (i.e. > sp.max.openname.size) or
                            path.variable.name is too large (i.e.>
                            sp.max.getenvname.size).
spr.buffer.overflow         The environment string referenced by
                            path.variable.name is longer than 507
                            characters.
wspr.operation.failed       If result w  spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual).
```

so.close

```
PROC so.close (CHAN OF SP fs, ts,
               VAL INT32 streamid,
               BYTE result)
```

Closes the stream identified by streamid. The result returned can take any of the following values:

```
spr.ok                      The close was successful.
wspr.operation.failed       If result w  spr.operation.failed then
                            this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

so.read

```
PROC so.read  (CHAN OF SP fs, ts,
               VAL INT32 streamid,
               INT length, []BYTE data)
```

Reads a block of bytes from the specified stream up to a maximum given by the size of the array **data**. If **length** returned is not the same as the size of **data** then the end of the file has been reached or an error has occurred. **Note:** **so.read** reads in multiples of the packet size defined by **sp.max.readbuffer.size**. For greatest efficiency, read requests should be made in multiples of this size.

so.write

```
PROC so.write (CHAN OF SP fs, ts,
               VAL INT32 streamid,
               VAL []BYTE data,
               INT length)
```

Writes a block of data to the specified stream. If **length** is less than the size of **data** then an error has occurred. **Note:** **so.write** writes in multiples of the packet size defined by **sp.max.writebuffer.size**. For greatest efficiency, write requests should be made in multiples of this size.

so.gets

```
PROC so.gets (CHAN OF SP fs, ts,
              VAL INT32 streamid,
              INT length, []BYTE data,
              BYTE result)
```

Reads a line from the specified input stream. Characters are read until a newline sequence is found, the end of the file is reached, or **sp.max.readbuffer.size** characters have been read. The characters read are in the first **length** bytes of **data**. The newline sequence is not included in the returned array. If the read fails then either the end of file has been reached or an error has occurred. The result returned can take any of the following values:

```
spr.ok                      The read was successful.
spr.bad.packet.size         Data is too large
                            (> sp.max.readbuffer.size).
spr.buffer.overflow         The line was larger than the buffer
                            data and has been truncated to fit.
wspr.operation.failed       If     result w spr.operation.failed
```

```
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.puts

```
PROC so.puts (CHAN OF SP fs, ts,
              VAL INT32 streamid,
              VAL []BYTE data, BYTE result)
```

Writes a line to the specified output stream. A newline sequence is added to the end of the line. The size of **data** must be less than or equal to the hostio constant **sp.max.writebuffer.size**. The result returned can take any of the following values:

```
spr.ok                          The write was successful.
spr.bad.packet.size             SIZE data is too large
                                ( > sp.max.writebuffer.size).
wspr.operation.failed           If result wspr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.flush

```
PROC so.flush (CHAN OF SP fs, ts,
               VAL INT32 streamid,
               BYTE result)
```

Flushes the specified output stream. All internally buffered data is written to the stream. Write and put operations that are directed to standard output are flushed automatically. The stream remains open. The result returned can take any of the following values:

```
spr.ok                          The flush was successful.
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.seek

```
PROC so.seek (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT32 offset, origin,
BYTE result)
```

Sets the file position for the specified stream. A subsequent read or write will access data at the new position. For a binary file the new position will be **offset** bytes from the position defined by **origin**. For a text file **offset** must be zero or a value returned by **so.tell**, in which case **origin** must be **spo.start**. **origin** may take the following values:

```
spo.start                   The start of the file.
spo.current                 The current position in the file.
spo.end                     The end of the file.
```

The result returned can take any of the following values:

```
spr.ok                          The operation was successful.
spr.bad.origin                  Invalid origin.
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.tell

```
PROC so.tell (CHAN OF SP fs, ts,
VAL INT32 streamid,
INT32 position, BYTE result)
```

Returns the current file position for the specified stream. The result returned can take any of the following values:

```
spr.ok                          The operation was successful.
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.eof

```
PROC so.eof (CHAN OF SP fs, ts,
VAL INT32 streamid, BYTE result)
```

Tests whether the specified stream has reached the end of a file. The end of file is reached when a read operation attempts to read past the end of file. The result returned can take any of the following values:

```
spr.ok                          End of file has been reached.
.spr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.) This result
                                will also be obtained if eof has not
                                been reached.
```

so.ferror

```
PROC so.ferror (CHAN OF SP fs, ts,
VAL INT32 streamid,
INT32 error.no, INT length,
[]BYTE message, BYTE result)
```

Indicates whether an error has occurred on the specified stream. The integer **error.no** is a host defined error number. The returned message is in the first **length** bytes of **message**. **length** will be zero if no message can be provided. If the returned message is longer than 503 bytes then it is truncated to this size. The result returned can take any of the following values:

```
spr.ok                          An error has occurred on the specified
                                stream.
spr.buffer.overflow             An error has occurred but the message
                                is too large for message and has been
                                truncated to fit.
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
```

```
                                Toolset Reference Manual.)  This
                                result will also be obtained if no
                                error has occurred on the specified
                                stream.
```

so.remove

```
PROC so.remove (CHAN OF SP fs, ts,
VAL []BYTE name, BYTE result)
```

Deletes the specified file. The result returned can take any of the following values:

```
spr.ok                          The delete was successful.
spr.bad.name                    Null name supplied.
spr.bad.packet.size             SIZE name is too large
( >
                                sp.max.removename.size).
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.rename

```
PROC so.rename (CHAN OF SP fs, ts,
VAL []BYTE oldname, newname,
BYTE result)
```

Renames the specified file. The result returned can take any of the following values:

```
spr.ok                          The operation was successful.
spr.bad.name                    Null name supplied.
spr.bad.packet.size             File names are too large
((SIZE
                                oldname + SIZE newname)
>
                                sp.max.renamename.size).
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.test.exists

```
PROC so.test.exists (CHAN OF SP fs, ts,
VAL []BYTE filename,
BOOL exists)
```

Tests if the specified file exists. The value of **exists** is **TRUE** if the file exists, otherwise it is **FALSE**.

1.5.5 General host access

This group contains routines to access the host computer for system information and services.

```
Procedure                          Parameter Specifiers
so.commandline                     CHAN OF SP fs, ts,
VAL BYTE all, INT
                                   length,
[]BYTE string, BYTE result
```

```
so.parse.command.line              CHAN OF SP fs, ts,
VAL [][]BYTE
                                   option.strings,
VAL
                                   []INT
option.parameters.required,
[]BOOL
                                   option.exists,
[][2]INT
                                   option.parameters,
INT error.len, []BYTE
                                   line
so.getenv                          CHAN OF SP fs, ts,
VAL []BYTE name, INT
                                   length,
[]BYTE value, BYTE result
so.system                          CHAN OF SP fs, ts,
VAL []BYTE
                                   command,
INT32 status, BYTE result
so.exit                            CHAN OF SP fs, ts,
VAL INT32 status
so.core                            CHAN OF SP fs, ts
VAL INT32 offset,
INT
                                   bytes.read,
[]BYTE data, BYTE result
so.version                         CHAN OF SP fs, ts,
BYTE version, host,
                                   os, board
```

Procedure definitions

so.commandline

```
PROC so.commandline (CHAN OF SP fs, ts,
VAL BYTE all, INT length,
[]BYTE string, BYTE result)
```

Returns the command line passed to the server when it was invoked. If **all** has the value **sp.short.commandline** then all valid server options and their arguments are stripped from the command line, as is the server command name. If **all** is **sp.whole.commandline** then the command line is returned exactly as it was invoked. The returned command line is in the first **length** bytes of **string**. If the command line string is longer than 507 bytes then it is truncated to this size. The result returned can take any of the following values:

```
spr.ok                         The operation was successful.
spr.buffer.overflow            Command line too long for string and
                               has been truncated to fit.
wspr.operation.failed          If     result w spr.operation.failed
                               then this denotes a server returned
                               failure. (See section C.2 in the
                               Toolset Reference Manual.)
```

so.parse.command.line

```
PROC so.parse.command.line
(CHAN OF SP fs, ts,
VAL []]BYTE option.strings,
```

```
VAL []INT option.parameters.required,
[]BOOL option.exists,
[][2]INT option.parameters,
INT error.len, []BYTE line)
```

This procedure reads the server command line and parses it for specified options and associated parameters. The parameter **option.strings** contains a list of all the possible options and must be in upper case. Options may be any length up to 256 bytes and when entered on the command line may be either upper or lower case. Because all of the strings in **option.strings** must be the same length, trailing spaces should be used to pad. To read a parameter that has no preceding option (such as a file name) then the first option string should be empty (contain only spaces). For example, consider a program to be supplied with a file name, and any of three options `A', 'B' and `C'. The array **option.strings** would look like this: **VAL option.strings IS [" ", "A", "B", "C"]:** The parameter **option.parameters.required** indicates if the corresponding option (in **option.strings**) requires a parameter. The values it may take are:

```
spopt.never                 Never takes a parameter.
spopt.maybe                 Optionally takes a parameter.
spopt.always                Must take a parameter.
```

Continuing the above example, if the file name must be supplied and none of the options take parameters, except for `C', which may or may not have a parameter, then **option.parameters.required** would look like this:

```
VAL option.parameters.required IS
[spopt.always, spopt.never,
spopt.never, spopt.maybe]:
```

If an option was present on the command line the corresponding element of **option.exists** is set to **TRUE**, otherwise it is set to **FALSE**. If an option was followed by a parameter then the position in the array **line** where the parameter starts and the length of the parameter are given by the first and second elements respectively in the corresponding element in **option.parameters**. If an error occurs whilst the command line is being parsed then **error.len** will be greater than zero and **line** will contain an error message of the given length. If no error occurs then **line** will contain the command line as supplied by the host file server. Most of the possible error messages are self explanatory, however, it is worth noting the meaning of the error `**Command line error: called incorrectly**'. This error means that either:

- **option.strings** was null, or
- **SIZE option.exists**, **SIZE option.parameters** or **SIZE option.parameters.required** does not equal **SIZE option.strings**.

so.getenv

```
PROC so.getenv (CHAN OF SP fs, ts,
VAL []BYTE name,
INT length, []BYTE value,
BYTE result)
```

Returns the string defined for the host environment variable **name**. The returned string is in the first **length** bytes of **value**. If **name** is not defined on the system **result** takes the value . **spr.operation.failed**. If the environment variable's string is longer than 507 bytes then it is truncated to this size. The result returned can take any of the following

values:

```
spr.ok                          The operation was successful.
spr.bad.name                    The specified name is a null string.
spr.bad.packet.size             SIZE name is too large
( >
                                sp.max.getenvname.size).
spr.buffer.overflow             Environment string too large for value
                                but has been truncated to fit.
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.system

```
PROC so.system (CHAN OF SP fs, ts,
VAL []BYTE command,
INT32 status, BYTE result)
```

Passes the string **command** to the host command processor for execution. If the command string is of zero length **result** takes the value **spr.ok** if there is a host command processor, otherwise an error is returned. If **command** is non zero in length then **status** contains the host specified value of the command, otherwise it is undefined. The result returned can take any of the following values:

```
spr.ok                          Host command processor exists.
spr.bad.packet.size             The array command is too large
(>
                                sp.max.systemcommand.size).
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

so.exit

```
PROC so.exit (CHAN OF SP fs, ts,
VAL INT32 status)
```

Terminates the server, which returns the value of **status** to its caller. If **status** has the special value **sps.success** then the server will terminate with a host specific `success' result. If **status** has the special value **sps.failure** then the server will terminate with a host specific `failure' result.

so.core

```
PROC so.core (CHAN OF SP fs, ts,
VAL INT32 offset, INT bytes.read,
[]BYTE data, BYTE result)
```

Returns the contents of the root transputer's memory as peeked from the transputer when **iserver** is invoked with the analyze (`**sA**') option. The start of the memory segment is given by **offset** which is an offset from the base of memory (and is therefore positive). The number of bytes to be read is given by the size of the **data** vector. The number of bytes actually read into **data** is given by **bytes.read**. An error is returned if **offset** is larger than the total amount of peeked memory. The result returned can take any of the following values:

```
spr.ok                          The operation was successful.
spr.bad.packet.size             The array data is too large
(>
                                sp.max.corerequest.size).
wspr.operation.failed           If     result w spr.operation.failed
                                then this denotes a server returned
                                failure. (See section C.2 in the
                                Toolset Reference Manual.)
```

This procedure can also be used to determine whether the memory was peeked (whether the server was invoked with the `**SA**' option), by specifying a size of zero for **data** and **offset**. If the result returned is **spr.ok** the memory was peeked.

so.version

```
PROC so.version (CHAN OF SP fs, ts,
BYTE version, host, os, board)
```

Returns version information about the server and the host on which it is running. A value of zero for any of the items indicates that the information is unavailable. The version of the server is given by **version**. The value should be divided by ten to yield the true version number. For example, a value of 15 means version 1.5. The host machine type is given by **host**, and can take any of the following values:

```
sph.unknown                 unknown host type
sph.PC                      IBM PC
IBM 370sph.S370             IBM 370 Architecture
NEC PCsph.NECPC             NEC PC
DEC VAXsph.VAX              DEC VAX
sph.SUN3                    Sun Microsystems Sun 3
Sun:host typessph.BOX.SUN4  Sun Microsystems Sun 4
sph.BOX.SUN386              Sun Microsystems Sun 386i
Apollosph.BOX.APOLLO        Apollo
Apollosph.BOX.ATARI         Atari ST or TT
```

Values up to 127 are reserved for use by INMOS. The host operating system is given by **os**, and can take any of the following values:

```
spo.unknown                 unknown OS type
DOSspo.DOS                  DOS
HELIOSspo.HELIOS            HELIOS
VMSspo.VMS                  VMS
SunOSspo.SUNOS              SunOS
CMSspo.CMS                  CMS
CMSspo.TOS                  TOS
```

Values up to 127 are reserved for use by INMOS. The interface board type is given by **board**, and can take any of the following values:

```
spb.unknown                 unknown board type
IMS B004spb.B004            IMS B004
IMS B008spb.B008            IMS B008
IMS B010spb.B010            IMS B010
IMS B011spb.B011            IMS B011
IMS B014spb.B014            IMS B014
IMS B015spb.B015            IMS B015
IMS B016spb.B016            IMS B016
DRX-11spb.DRX11             DRX 11
CATspb.IBMCAT               CAT
Caplin QT0spb.QT0           Caplin QT0
UDPlinkspb.UDPLINK          UDP link
```

```
UDPlinkspb.TCPLINK          TCP link
UDPlinkspb.ACSILA           ACSILA
```

Values up to 127 are reserved for use by INMOS.

1.5.6 Keyboard input

```
Procedure                   Parameter Specifiers
so.pollkey                  CHAN OF SP fs, ts,
BYTE key, result
so.getkey                   CHAN OF SP fs, ts,
BYTE key, result
so.read.line                CHAN OF SP fs, ts,
INT len, [] BYTE
                            line,
BYTE result
so.read.echo.line           CHAN OF SP fs, ts,
INT len, [] BYTE
                            line,
BYTE result
so.ask                      CHAN OF SP fs, ts,
VAL [] BYTE prompt,
                            replies,
VAL BOOL
                            display.possible.replies,
VAL BOOL
                            echo.reply,
INT reply.number
so.read.echo.int            CHAN OF SP fs, ts, INT n,
BOOL error
so.read.echo.int32          CHAN OF SP fs, ts, INT32 n,
BOOL error
so.read.echo.int64          CHAN OF SP fs, ts, INT64 n,
BOOL error
so.read.echo.hex.int        CHAN OF SP fs, ts, INT n,
BOOL error
so.read.echo.hex.int32      CHAN OF SP fs, ts, INT32 n,
BOOL error
so.read.echo.hex.int64      CHAN OF SP fs, ts, INT64 n,
BOOL error
so.read.echo.any.int        CHAN OF SP fs, ts, INT n,
BOOL error
so.read.echo.real32         CHAN OF SP fs, ts, REAL32 n,
BOOL error
so.read.echo.real64         CHAN OF SP fs, ts, REAL64 n,
BOOL error
```

Procedure definitions

so.pollkey

```
PROC so.pollkey (CHAN OF SP fs, ts,
BYTE key, result)
```

Reads a single character from the keyboard. If no key is available then it returns immediately with . **spr.operation.failed**. The key is not echoed on the screen. The result returned can take any of the following values:

```
spr.ok                      A key was available and has been
                            returned in key.
wspr.operation.failed       If     result w spr.operation.failed
```

```
                              then this denotes a server returned
                              failure. (See section C.2 in the
                              Toolset Reference Manual.)
```

so.getkey

```
PROC so.getkey (CHAN OF SP fs, ts,
BYTE key, result)
```

As **so.pollkey** but waits for a key if none is available.

so.read.line

```
PROC so.read.line (CHAN OF SP fs, ts, INT len,
[]BYTE line, BYTE result)
```

Reads a line of text from the keyboard, without echoing it on the screen. The characters read are in the first **len** bytes of **line**. The line is read until `RETURN' is pressed at the keyboard. The line is truncated if **line** is not large enough. A newline or carriage return is not included in **line**. The result returned can take any of the following values:

```
spr.ok                        The read was successful.
wspr.operation.failed         If     result w spr.operation.failed
                              then this denotes a server returned
                              failure. (See section C.2 in the
                              Toolset Reference Manual.)
```

so.read.echo.line

```
PROC so.read.echo.line (CHAN OF SP fs, ts,
INT len, []BYTE line,
BYTE result)
```

As **so.read.line**, but user input (except newline or carriage return) is echoed on the screen.

so.ask

```
PROC so.ask (CHAN OF SP fs, ts,
VAL []BYTE prompt, replies,
VAL BOOL display.possible.replies,
VAL BOOL echo.reply,
INT reply.number)
```

Prompts on the screen for a user response on the keyboard. The prompt is specified by the string **prompt**, and the list of permitted relies by the string **replies**. Only single character responses are permitted, and alphabetic characters are *not* case sensitive. For example if the permitted responses are `Y', `N' and `Q' then the **replies** string would contain the characters `YNQ', and `**y**', `**n**' and `**q**' would also be accepted. **reply.number** indicates which response was typed, numbered from zero. ` ? ' is automatically output at the end of the prompt. If **display.possible.replies** is **TRUE** the permitted replies are displayed on the screen. If **echo.reply** is **TRUE** the user's response is displayed. The procedure will not return until a valid response has been typed.

so.read.echo.int

```
PROC so.read.echo.int (CHAN OF SP fs, ts, INT n,
BOOL error)
```

Reads a decimal integer typed at the keyboard and displays it on the screen. The number must be terminated by `RETURN'. The boolean **error** is set to **TRUE** if an invalid integer is typed, **FALSE** otherwise.

so.read.echo.int32

```
PROC so.read.echo.int32 (CHAN OF SP fs, ts,
INT32 n, BOOL error)
```

As **so.read.echo.int** but reads 32 bit numbers.

so.read.echo.int64

```
PROC so.read.echo.int64 (CHAN OF SP fs, ts,
INT64 n, BOOL error)
```

As **so.read.echo.int** but reads 64 bit numbers.

so.read.echo.hex.int

```
PROC so.read.echo.hex.int (CHAN OF SP fs, ts,
INT n, BOOL error)
```

As **so.read.echo.int** but reads a number in hexadecimal format. The number may be in lower or upper case but must be prefixed with either `**#**', or `**$**' which directly indicates a hexadecimal number, or `**%**', which means add **MOSTNEG INT** to the given hex (using modulo arithmetic). For example, on a 32 bit transputer **%70** is interpreted as **#80000070**, and on a 16 bit transputer as **#8070**. This is useful when specifying transputer addresses, which are signed and start at **MOSTNEG INT**.

so.read.echo.hex.int32

```
PROC so.read.echo.hex.int32 (CHAN OF SP fs, ts,
INT32 n, BOOL error)
```

As **so.read.echo.hex.int** but reads 32 bit numbers.

so.read.echo.hex.int64

```
PROC so.read.echo.hex.int64 (CHAN OF SP fs, ts,
INT64 n, BOOL error)
```

As **so.read.echo.hex.int** but reads 64 bit numbers.

so.read.echo.any.int

```
PROC so.read.echo.any.int (CHAN OF SP fs, ts,
INT n, BOOL error)
```

As **so.read.echo.int** but accepts numbers in either decimal or hexadecimal format. Hexadecimal numbers may be lower or upper case but must be prefixed with either `**#**' or `**$**' which specifies the number directly, or `**%**', which means add **MOSTNEG INT** to the given hex (using modulo arithmetic). For example, on a 32 bit transputer **%70** is interpreted as **#80000070**, and on a 16 bit transputer as **#8070**. This is useful when specifying transputer addresses, which are signed and start at **MOSTNEG INT**.

so.read.echo.real32

```
PROC so.read.echo.real32 (CHAN OF SP fs, ts,
```

```
REAL32 n, BOOL error)
```

Reads a real number typed at the keyboard and displays it on the screen. The number must conform to occam syntax and be terminated by `RETURN'. The boolean variable **error** is set to **TRUE** if an invalid number is typed, **FALSE** otherwise.

so.read.echo.real64

```
PROC so.read.echo.real64 (CHAN OF SP fs, ts,
REAL64 n, BOOL error)
```

As **so.read.echo.real32** but for 64 bit real numbers.

1.5.7 Screen output

Procedure	Parameter Specifiers
so.write.char	CHAN OF SP fs, ts,
VAL BYTE char	
so.write.nl	CHAN OF SP fs, ts,
so.write.string	CHAN OF SP fs, ts,
VAL [] BYTE string	
so.write.string.nl	CHAN OF SP fs, ts,
VAL [] BYTE string	
so.write.int	CHAN OF SP fs, ts,
VAL INT n, width	
so.write.int32	CHAN OF SP fs, ts,
VAL INT32 n, VAL	
	INT width
so.write.int64	CHAN OF SP fs, ts,
VAL INT64 n, VAL	
	INT width
so.write.hex.int	CHAN OF SP fs, ts,
VAL INT n, width	
so.write.hex.int32	CHAN OF SP fs, ts,
VAL INT32 n, VAL	
	INT width
so.write.hex.int64	CHAN OF SP fs, ts,
VAL INT64 n, VAL	
	INT width
so.write.real32	CHAN OF SP fs, ts,
VAL REAL32 r, VAL	
	INT Ip, Dp
so.write.real.64	CHAN OF SP fs, ts,
VAL REAL64 r, VAL	
	INT Ip, Dp

Procedure definitions

so.write.char

```
PROC so.write.char (CHAN OF SP fs, ts,
VAL BYTE char)
```

Writes the single byte **char** to the screen.

so.write.nl

```
PROC so.write.nl (CHAN OF SP fs, ts)
```

Writes a newline sequence to the screen.

so.write.string

```
PROC so.write.string (CHAN OF SP fs, ts,
VAL []BYTE string)
```

Writes the string **string** to the screen.

so.write.string.nl

```
PROC so.write.string.nl (CHAN OF SP fs, ts,
VAL []BYTE string)
```

As **so.write.string**, but appends a newline sequence to the end of the string.

so.write.int

```
PROC so.write.int (CHAN OF SP fs, ts,
VAL INT n, width)
```

Writes the value **n** (of type **INT**) to the screen as decimal ASCII digits, padded out with leading spaces and an optional sign to the specified field width, **width**. If the field width is too small for the number it is widened as necessary; a zero value for **width** specifies minimum width. A negative value for **width** is an error.

so.write.int32

```
PROC so.write.int32 (CHAN OF SP fs, ts,
VAL INT32 n, VAL INT width)
```

As **so.write.int** but for 32 bit integers.

so.write.int64

```
PROC so.write.int64 (CHAN OF SP fs, ts,
VAL INT64 n, VAL INT width)
```

As **so.write.int** but for 64 bit integers.

so.write.hex.int

```
PROC so.write.hex.int (CHAN OF SP fs, ts,
VAL INT n, width)
```

Writes the value **n** (of type **INT**) to the screen as hexadecimal ASCII digits, preceded by the **`#**' character. The number of characters printed is **width + 1**. If **width** is larger than the size of the number then the number is padded with leading **`0**'s or **`F**'s as appropriate. If **width** is smaller than the size of the number, the number is truncated, from the left, to **width** digits. A negative value for **width** is an error.

so.write.hex.int32

```
PROC so.write.hex.int64 (CHAN OF SP fs, ts,
VAL INT32 n,
VAL INT width)
```

As **so.write.hex.int** but for 32 bit integers.

so.write.hex.int64

```
PROC so.write.hex.int64 (CHAN OF SP fs, ts,
```

```
VAL INT64 n,
VAL INT width)
```

As **so.write.hex.int** but for 64 bit integers.

so.write.real32

```
PROC so.write.real32 (CHAN OF SP fs, ts,
VAL REAL32 r,
VAL INT Ip, Dp)
```

Writes the value **r** (of type **REAL32**) to the screen as ASCII characters formatted using **Ip** and **Dp** as described under **REAL32TOSTRING** (see section 1.8). **Note**: Due to fixed size internal buffers, this procedure will be invalid if the string representing the real number is longer than 24 characters. If this is a problem, it is suggested you write your own procedure to perform this function. The procedure should include a buffer set to the required size, a call to **REAL32TOSTRING**, followed by a call to **so.write**.

so.write.real64

```
PROC so.write.real64 (CHAN OF SP fs, ts,
VAL REAL64 r,
VAL INT Ip, Dp)
```

As **so.write.real32** but for 64 bit real numbers. The formatting variables **Ip** and **Dp** are described under **REAL32TOSTRING** (see section 1.8). **Note** : Due to fixed size internal buffers, this procedure will be invalid if the string representing the real number is longer than 30 characters. If this is a problem, it is suggested you write your own procedure to perform this function. The procedure should include a buffer set to the required size, a call to **REAL64TOSTRING**, followed by a call to **so.write**.

1.5.8 File output

These routines write characters and strings to a specified stream, usually a file. The result returned can take the values **spr.ok**, **spr.notok** or, very rarely, . **spr.operation.failed**.

```
Procedure                   Parameter Specifiers
so.fwrite.char              CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL BYTE
                            char, BYTE result
so.fwrite.nl                CHAN OF SP fs, ts,
VAL INT32 streamid, BYTE
                            result
so.fwrite.string            CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            []BYTE string, BYTE result
so.fwrite.string.nl         CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            []BYTE string,
BYTE result
so.fwrite.int               CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT
                            n, width, BYTE result
so.fwrite.int32             CHAN OF SP fs, ts,
VAL INT32 streamid, n,
VAL
                            INT width,
```

```
BYTE result
so.fwrite.int64             CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            INT64 n, VAL INT width,
BYTE result
so.fwrite.hex.int           CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT
                            n, width, BYTE result
so.fwrite.hex.int32         CHAN OF SP fs, ts,
VAL INT32 streamid, n
VAL
                            INT width, BYTE result
so.fwrite.hex.int64         CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            INT64 n, VAL INT width,
BYTE result
so.fwrite.real32            CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            REAL32 r, VAL INT Ip, Dp,
BYTE result
so.fwrite.real64            CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL
                            REAL64 r, VAL INT Ip, Dp,
BYTE result
```

Procedure definitions

so.fwrite.char

```
PROC so.fwrite.char (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL BYTE char,
BYTE result)
```

Writes a single character to the specified stream. The result **spr.notok** will be returned if the character is not written.

so.fwrite.nl

```
PROC so.fwrite.nl (CHAN OF SP fs, ts,
VAL INT32 streamid,
BYTE result)
```

Writes a newline sequence to the specified stream. If **result** takes a value . **spr.operation.failed** then this denotes a server returned failure, details of which are documented in in section C.2 of the *Toolset Reference Manual*.

so.fwrite.string

```
PROC so.fwrite.string (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL []BYTE string,
BYTE result)
```

Writes a string to the specified stream. The result **spr.notok** will be returned if not all the characters are written.

so.fwrite.string.nl

```
PROC so.fwrite.string.nl (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL []BYTE string,
BYTE result)
```

As **so.fwrite.string**, but appends a newline sequence to the end of the string. The result returned can take any of the following values:

```
spr.ok                      The operation was successful.
spr.notok                   Not all of the characters were written.
wspr.operation.failed       If     result w spr.operation.failed
                            then this denotes a server returned
                            failure. (See section C.2 in the
                            Toolset Reference Manual.)
```

so.fwrite.int

```
PROC so.fwrite.int (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT n, width,
BYTE result)
```

Writes the value **n** (of type **INT**) to the specified stream as decimal ASCII digits, padded out with leading spaces and an optional sign to the specified field width, **width**. If the field width is too small for the number it is widened as necessary; a zero value for **width** specifies minimum width. A negative value for **width** is an error. The result **spr.notok** will be returned if not all of the digits are written.

so.fwrite.int32

```
PROC so.fwrite.int32 (CHAN OF SP fs, ts,
VAL INT32 streamid, n,
VAL INT width,
BYTE result)
```

As **so.fwrite.int** but for 32 bit integers.

so.fwrite.int64

```
PROC so.fwrite.int64 (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT64 n, VAL INT width,
BYTE result)
```

As **so.fwrite.int** but for 64 bit integers.

so.fwrite.hex.int

```
PROC so.fwrite.hex.int (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT n, width,
BYTE result)
```

Writes the value **n** (of type **INT**) to the specified stream as hexadecimal ASCII digits preceded by the **`#'** character. The number of characters printed is **width + 1**. If **width** is larger than the size of the number then the number is padded with leading **`0'**s or **`F'**s as appropriate. If **width** is smaller than the size of the number, then the number is

truncated, from the left, to **width** digits. A negative value for **width** is an error. The result **spr.notok** will be returned if not all the characters are written.

so.fwrite.hex.int32

```
PROC so.fwrite.hex.int32 (CHAN OF SP fs, ts,
VAL INT32 streamid, n
VAL INT width,
BYTE result)
```

As **so.fwrite.hex.int** but for 32 bit integers.

so.fwrite.hex.int64

```
PROC so.fwrite.hex.int64 (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL INT64 n,
VAL INT width,
BYTE result)
```

As **so.fwrite.hex.int** but for 64 bit integers.

so.fwrite.real32

```
PROC so.fwrite.real32 (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL REAL32 r,
VAL INT Ip, Dp,
BYTE result)
```

Writes the value **r** (of type **REAL32**) to the specified stream as ASCII characters formatted using **Ip** and **Dp** as described under **REAL32TOSTRING** (see section 1.8). The result **spr.notok** will be returned if not all the characters are written. **Note**: Due to fixed size internal buffers, this procedure will be invalid if the string representing the real number is longer than 24 characters. If this is a problem, it is suggested you write your own procedure to perform this function. The procedure should include a buffer set to the required size, a call to **REAL32TOSTRING**, followed by a call to **so.write**.

so.fwrite.real64

```
PROC so.fwrite.real64 (CHAN OF SP fs, ts,
VAL INT32 streamid,
VAL REAL64 r,
VAL INT Ip, Dp,
BYTE result)
```

As **so.fwrite.real32** but for 64 bit real numbers. The formatting variables **Ip** and **Dp** are described under **REAL32TOSTRING** (see section 1.8). **Note** : Due to fixed size internal buffers, this procedure will be invalid if the string representing the real number is longer than 30 characters. If this is a problem, it is suggested you write your own procedure to perform this function. The procedure should include a buffer set to the required size, a call to **REAL64TOSTRING**, followed by a call to **so.write**.

1.5.9 Miscellaneous

This miscellaneous group includes procedures for:

- Time and date processing
- Buffering and multiplexing

Time processing

```
Procedure                     Parameter Specifiers
so.time                       CHAN OF SP fs, ts,
INT32 localtime, UTCtime
so.time.to.date               VAL INT32 input.time,
[so.date.len]INT date
so.date.to.ascii              VAL [so.date.len]INT date,
VAL BOOL
                              long.years,
VAL BOOL
                              days.first,
[so.time.string.len]BYTE string
so.time.to.ascii              VAL INT32 time,
VAL BOOL long.years,
VAL BOOL
                              days.first
[so.time.string.len]BYTE string
so.today.date                 CHAN OF SP fs, ts,
[so.date.len]INT date
so.today.ascii                CHAN OF SP fs, ts,
VAL BOOL long.years,
VAL
                              BOOL days.first,
[so.time.string.len]BYTE string
```

so.time

```
PROC so.time (CHAN OF SP fs, ts,
INT32 localtime, UTCtime)
```

Returns the local time and Coordinated Universal Time. Both times are expressed as the number of seconds that have elapsed since midnight on 1st January, 1970. If UTC time is unavailable then it will have a value of zero. The times are given as unsigned **INT32S**.

so.time.to.date

```
PROC so.time.to.date (VAL INT32 input.time,
[so.date.len]INT date)
```

Converts time (as supplied by **so.time**) to six integers, stored in the **date** array. The elements of the array are as follows:

```
 Element of array    Data
         0           Seconds past the minute
         1           Minutes past the hour
         2           The hour (24 hour clock)
         3           The day of the month
         4           The month (1 to 12)
         5           The year (4 digits)
```

so.date.to.ascii

```
PROC so.date.to.ascii
(VAL [so.date.len]INT date,
VAL BOOL long.years,
VAL BOOL days.first,
[so.time.string.len]BYTE string)
```

Converts an array of six integers containing the date (as supplied by **so.time.to.date**) into an ASCII string of the form:

HH:MM:SS DD/MM/YYYY

If **long.years** is **FALSE** then year is reduced to two characters, and the last two characters of the year field are padded with spaces. If **days.first** is **FALSE** then the ordering of day and month is changed (to the U.S. standard).

so.time.to.ascii

```
PROC so.time.to.ascii
(VAL INT32 time,
VAL BOOL long.years,
VAL BOOL days.first
[so.time.string.len]BYTE string)
```

Converts time (as supplied by **so.time**) into an ASCII string, as described for **so.date.to.ascii**.

so.today.date

```
PROC so.today.date (CHAN OF SP fs, ts,
[so.date.len]INT date)
```

Gives today's date, in local time, as six integers, stored in the array **date**. The format of the array is the same as for **so.time.to.date**. If the date is unavailable all elements in **date** are set to zero.

so.today.ascii

```
PROC so.today.ascii
(CHAN OF SP fs, ts,
VAL BOOL long.years, days.first,
[so.time.string.len]BYTE string)
```

Gives today's date, in local time, as an ASCII string, in the same format as procedure **so.date.to.ascii**. If the date is unavailable **string** is filled with spaces.

Buffers and multiplexors

This group of procedures are designed to assist with buffering and multiplexing data exchange between the program and host.

Procedure	Parameter Specifiers
so.buffer	CHAN OF SP fs, ts,
from.user,	
	to.user,
CHAN OF BOOL stopper	
so.overlapped.buffer	CHAN OF SP fs, ts,
from.user,	
	to.user,
CHAN OF BOOL stopper	
so.multiplexor	CHAN OF SP fs, ts,
[]CHAN OF SP	
	from.user,
to.user,	
CHAN OF BOOL	
	stopper
so.overlapped.multiplexor	CHAN OF SP fs, ts,

```
[]CHAN OF SP
                                        from.user,
to.user,
CHAN OF BOOL
                                        stopper
[]INT queue
so.pri.multiplexor                      CHAN OF SP fs, ts,
[]CHAN OF SP
                                        from.user,
to.user,
CHAN OF BOOL
                                        stopper
so.overlapped.pri.multiplexor           CHAN OF SP fs, ts,
[]CHAN OF SP
                                        from.user,
to.user,
CHAN OF BOOL
                                        stopper
[]INT queue
```

Buffering procedures

so.buffer

```
PROC so.buffer (CHAN OF SP fs, ts,
from.user, to.user,
CHAN OF BOOL stopper)
```

This procedure buffers data between the user and the host. It can be used by processes on a network to pass data to the host across intervening processes. It is terminated by sending either a **TRUE** or **FALSE** value on the channel **stopper**.

so.overlapped.buffer

```
PROC so.overlapped.buffer (CHAN OF SP fs, ts,
from.user,
to.user,
CHAN OF BOOL stopper)
```

Similar to **so.buffer**, but allows many host communications to occur simultaneously through a train of processes. This can improve efficiency if the communications pass through many processes before reaching the server. It is terminated by either a **TRUE** or **FALSE** value on the channel **stopper**.

Multiplexing procedures

Note: when pairs of channels are passed as parameters, they are normally passed as input then output. Hence all **so.** ... routines take the first parameters **fs**, **ts** (i.e. *from server*, *to server*). The multiplexors take the next two parameters as *from user*, *to user* which will normally correspond with *to server*, *from server*. **so.multiplexor**

```
PROC so.multiplexor (CHAN OF SP fs, ts,
[]CHAN OF SP from.user,
to.user,
CHAN OF BOOL stopper)
```

This procedure multiplexes any number of pairs of **SP** protocol channels onto a single pair of **SP** protocol channels, which may go to the file server or another **SP** protocol

multiplexor (or buffer). It is terminated by sending either a **TRUE** or **FALSE** value on the channel **stopper**. For n channels, each channel is guaranteed to be able to pass on a message for every n messages that pass through the multiplexor. This is achieved by cycling the selection priority from the lowest index of **from.user**. However, **stopper** always has highest priority.

so.overlapped.multiplexor

```
PROC so.overlapped.multiplexor
(CHAN OF SP fs, ts,
[]CHAN OF SP from.user, to.user,
CHAN OF BOOL stopper,
[]INT queue)
```

Similar to **so.multiplexor**, but can pipeline server requests. The number of requests than can be pipelined is determined by the size of **queue**, which must provide one word for each request that can be pipelined. If **SIZE queue** is zero then the routine simply waits for input from **stopper**. Pipelining improves efficiency if the server requests have to pass through many processes on the way to and from the server. It is terminated by sending either a **TRUE** or **FALSE** value on the channel stopper. The multiplexing is done in the same cyclic manner as in **so.multiplexor. stopper** has higher priority than any of **from.user**.

so.pri.multiplexor

```
PROC so.pri.multiplexor
(CHAN OF SP fs, ts,
[]CHAN OF SP from.user, to.user,
CHAN OF BOOL stopper)
```

As **so.multiplexor** but the multiplexing is *not* done in a cyclic manner; rather there is a hierarchy of priorities amongst the channels - **from.user**: **from.user[i]** is of higher priority than **from.user[j], for i < j**. Also **stopper** is of lower priority than any of **from.user**.

so.overlapped.pri.multiplexor

```
PROC so.overlapped.pri.multiplexor
(CHAN OF SP fs, ts,
[]CHAN OF SP from.user, to.user,
CHAN OF BOOL stopper,
[]INT queue)
```

As **so.overlapped.multiplexor** but the multiplexing is done in the same prioritized manner as in **so.pri.multiplexor**. **stopper** has higher priority than any of **from.user**.