NAME

hostio.lib – INMOS occam toolset host file server library

SYNOPSIS

```
#INCLUDE "hostio.inc"
#USE "hostio.lib"
```

SUMMARY

The occam toolset library hostio.lib provides the main interface between the host and the transputer network.

OVERVIEW

To use the functions in this library do the following:

```
#INCLUDE "hostio.inc"
#USE "hostio.lib"
```

The pair of channels `CHAN OF SP fs, ts` to and from the server are the first two parameters for most `hostio.lib` library procedure.

When a new stream is opened with `so.open`, a `VAL INT32 streamid` is returned. This is used to reference the stream and is used in all procedure calls related to files.

Error returns from procedures are passed out a variable called `result` which can take the following values:

lf(CR) lw(5i). spr.ok The operation was successful spr.notok Too many temporary files opened (`so.open.temp`) spr.bad.name Invalid name parameter spr.bad.type Invalid type parameter (`so.open, so.open.temp`) spr.bad.mode Invalid mode parameter (`so.open`) spr.bad.origin Invalid origin parameter (`so.seek`) spr.bad.packet.size Some data was larger than the built in buffer spr.buffer.overflow Part of the operation exceeded the 256 byte buffer >=spr.operation.failed Server returned a failure

SO Procedures

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

Prompt for user keyboard response from valid replies and return reply.number as index. If `display.possible.replies` is `TRUE` then print permitted replies on the screen. If `echo.reply` is `TRUE` then echo response on screen.

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

This routine provides buffering of the SP protocol passing requests from the user to the system until any boolean is sent along `stopper`.

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

Close an open stream.

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

Return the command line passed by the host in string of `SIZE length` bytes. If `all` is set to `sp.short.commandline` then strip server options or present them if `sp.whole.commandline`.

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

Return peeked memory of root transputer.

```
PROC so.date.to.ascii (VAL [6]INT date, VAL BOOL long.years,
                       VAL BOOL days.first, [19]BYTE string)
```

See `so.time` and `so.time.to.ascii` etc.

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

Return result `spr.ok` if end of file on the given stream.

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

Terminate the server and exit the program with status `sps.success` or `sps.failure`.

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

For the given host `error.no`, put a description of `SIZE length` bytes in message.

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

Flush the output buffer of the given stream.

```
PROC so.fwrite.char (CHAN OF SP fs, ts, VAL INT32 streamid,
                     VAL BYTE char, BYTE result)
PROC so.fwrite.hex.int (CHAN OF SP fs, ts, VAL INT32 streamid,
                        VAL INT n, VAL INT width, BYTE result)
PROC so.fwrite.hex.int32 (CHAN OF SP fs, ts, VAL INT32 streamid,
                          VAL INT32 n, VAL INT width, BYTE result)
PROC so.fwrite.hex.int64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                          VAL INT64 n, VAL INT width, BYTE result)
PROC so.fwrite.int (CHAN OF SP fs, ts, VAL INT32 streamid,
                    VAL INT n, VAL INT width, BYTE result)
PROC so.fwrite.int32 (CHAN OF SP fs, ts, VAL INT32 streamid,
                      VAL INT32 n, VAL INT width, BYTE result)
PROC so.fwrite.int64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                      VAL INT64 n, VAL INT width, BYTE result)
PROC so.fwrite.nl (CHAN OF SP fs, ts, VAL INT32 streamid,
                   BYTE result)
PROC so.fwrite.real32 (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL REAL32 r, VAL INT Ip, VAL INT Dp,
                       BYTE result)
PROC so.fwrite.real64 (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL REAL64 r, VAL INT Ip, VAL INT Dp,
                       BYTE result)
PROC so.fwrite.string (CHAN OF SP fs, ts, VAL INT32 streamid,
                       VAL []BYTE string, BYTE result)
PROC so.fwrite.string.nl (CHAN OF SP fs, ts, VAL INT32 streamid,
                          VAL []BYTE string, BYTE result)
```

Write the given data type to a stream, usually a file, formatting where necessary and possibly padding with spaces to a given field `width` or printing `Ip` integer places and `Dp` decimal places for `REAL32` and `REAL64` types. Hex numbers are written prefixed by a '#'. PROCedures ending in `.nl` write a newline sequence.

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

Return the value of the given host environment variable of `SIZE length` bytes.

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

Wait for keypress.

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

Read up to `SIZE data` bytes from a line of the stream and return the number of bytes read. The newline is not returned.

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

See `so.overlapped.multiplexor`.

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

Attempt to open the file name and return the streamid for file operations if successful. Valid types are `spt.binary` and `spt.text`. Valid modes are `spm.input`, `spm.output`, `spm.append`, `spm.existing.update`, `spm.new.update` and `spm.append.update`.

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

Open a temporary file and return its `filename` and streamid. See `so.open` for valid types.

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

This routine provides buffering of the SP protocol passing requests from the user to the system until any boolean is sent along `stopper`. Communications are overlapped allowing inputs and outputs to happen concurrently.

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

These routines provide multiplexing of the SP protocol. Procedures with `pri` in the name give priority to the lower numbered channels. Procedures with `overlapped` in the name, overlap communications allowing inputs and outputs to happen concurrently.

```
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)
```

Parse the command line for options. See INMOS documentation for details.

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

Read a character from the keyboard (`result` = `spr.ok`) or return immediately if no key is ready (`result` >= `spr.operation.failed`).

```
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 stream.id,
                    BYTE result)
```

Open a file like `so.open` but use the given environment variable as a path to search for it. The mode is always `spm.input` and the full name and length are returned also.

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

See `so.overlapped.multiplexor`.

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

Write the given line to the stream followed by a newline.

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

Read up to `SIZE data` bytes from the given stream and return the number read.

```
PROC so.read.echo.any.int (CHAN OF SP fs, ts, INT n, BOOL error)
PROC so.read.echo.hex.int (CHAN OF SP fs, ts, INT n, BOOL error)
PROC so.read.echo.hex.int32 (CHAN OF SP fs, ts, INT32 n, BOOL error)
PROC so.read.echo.hex.int64 (CHAN OF SP fs, ts, INT64 n, BOOL error)
PROC so.read.echo.int (CHAN OF SP fs, ts, INT n, BOOL error)
PROC so.read.echo.int32 (CHAN OF SP fs, ts, INT32 n, BOOL error)
PROC so.read.echo.int64 (CHAN OF SP fs, ts, INT64 n, BOOL error)
PROC so.read.echo.line (CHAN OF SP fs, ts, INT len, []BYTE line,
                        BYTE result)
```

Read some data from the keyboard terminated by return, echoing output to the screen. Hex numbers can be prefixed by a '#' or '$' or with

```
PROC so.read.echo.real32 (CHAN OF SP fs, ts, REAL32 n, BOOL error)
PROC so.read.echo.real64 (CHAN OF SP fs, ts, REAL64 n, BOOL error)
```

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

Read a line of `SIZE len` bytes from the keyboard with no screen echo. The line terminator is removed. See also `so.read.echo.line`.

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

Attempt to delete the given file.

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

Attempt to rename file oldname to newname.

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

Move the file pointer on the given stream to a new offset. Origin can be `spo.start`, `spo.current` or `spo.end`.

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

Execute the given command on the host server.

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

Return the position of the file pointer for the given stream.

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

Return `TRUE` in `exists` if the file exists.

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

Return the local and UTC time (UNIX format - see time(3)).

```
PROC so.time.to.ascii (VAL INT32 time, VAL BOOL long.years,
                       VAL BOOL days.first, [19]BYTE string)
PROC so.time.to.date (VAL INT32 input.time, [6]INT date)
PROC so.today.ascii (CHAN OF SP fs, ts, VAL BOOL long.years,
                     VAL BOOL days.first, [19]BYTE string)
PROC so.today.date (CHAN OF SP fs, ts, [6]INT date)
```

Various time and date to/from ascii conversions. See also `so.date.to.ascii`, `so.time` and **time**(3). `[6]INT date` is `[seconds, minutes, hour, day, month, year]` Date format is `"HH:MM:SS DD/MM/YYYY"`. If `long.years` is `FALSE` use 2 digit years. If `days.first` is `FALSE` then swap DD and MM (for U.S.)

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

Return version information. See INMOS documentation for values.

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

Write up to `SIZE data` bytes to the given stream and return the `length` written.

```
PROC so.write.char (CHAN OF SP fs, ts, VAL BYTE char)
PROC so.write.hex.int (CHAN OF SP fs, ts, VAL INT n,
                       VAL INT width)
PROC so.write.hex.int32 (CHAN OF SP fs, ts, VAL INT32 n,
                         VAL INT width)
PROC so.write.hex.int64 (CHAN OF SP fs, ts, VAL INT64 n,
                         VAL INT width)
PROC so.write.int (CHAN OF SP fs, ts, VAL INT n, VAL INT width)
PROC so.write.int32 (CHAN OF SP fs, ts, VAL INT32 n, VAL INT width)
PROC so.write.int64 (CHAN OF SP fs, ts, VAL INT64 n, VAL INT width)
PROC so.write.nl (CHAN OF SP fs, ts)
PROC so.write.real32 (CHAN OF SP fs, ts, VAL REAL32 r, VAL INT Ip,
                      VAL INT Dp)
PROC so.write.real64 (CHAN OF SP fs, ts, VAL REAL64 r, VAL INT Ip,
                      VAL INT Dp)
PROC so.write.string (CHAN OF SP fs, ts, VAL []BYTE string)
PROC so.write.string.nl (CHAN OF SP fs, ts, VAL []BYTE string)
```

Write the given data type to the screen, formatting where necessary and possibly padding with spaces to a given field width or printing `Ip` integer places and `Dp` decimal places for `REAL32` and `REAL64` types. Hex numbers are written prefixed by a '#'. PROCedures ending in `.nl` write a newline sequence.

SP Procedures

```
PROC sp.receive.packet (CHAN OF SP fs, INT16 length,
                        []BYTE packet, BOOL error)
PROC sp.send.packet (CHAN OF SP ts, VAL []BYTE packet,
                     BOOL error)
```

See section H.3 of the INMOS manual for descriptions of these procedures.

SEE ALSO

INMOS occam 2 toolset user manual - part 2 (occam libraries and appendices) INMOS document number 72 TDS 276 02.

AUTHOR