III.b. VERSAdos System Interface Library

NAME

Interface - to VERSAdos system

FUNCTION

Programs written in C for operation on the MC68000 under VERSAdos are translated according to the following specifications:

external identifiers - may be written in both upper and lowercase, but only one case is significant. The first seven letters must be distinct. A '.' is prepended to each identifier.

function text - is normally generated into section 0 and is not to be altered or read as data. External function names are published via xdef declarations.

literal data - such as strings, double constants, and switch tables, are normally generated into section 0.

initialized data - is normally generated into section 1. External data names are published via xdef declarations.

uninitialized declarations - result in an xref reference, one instance per program file.

function calls - are performed by

1) moving arguments onto the stack, right to left. Character and short data are widened to long integer, float is converted to double.

2) calling via "jsr .func".

3) popping the arguments off the stack.

Except for returned value, the registers d0, d1, d2, d6, d7, a0, a1, a2, and the condition codes are undefined on return from a function call. All other registers are preserved. The returned value is in d7 (char or short widened to long integer, long integer, and pointer to), or in d6-d7 (float widened to double, and double).

stack frames - are maintained by each C function, using a6 as a frame pointer. On entry to a function, the instruction "link a6,#" will stack a6 leaving a6 pointing to the stacked a6, and will reserve # bytes for automatics. If more than 32,768 bytes of autos are specified, additional code is generated to reserve space on the stack. Any non-volatile registers used (d3, d4, d5, a3, a4, a5) are then stacked and, if the top of stack scratch cell is used, an additional four bytes are reserved (by either stacking d0 along with other registers or by modifying the stack pointer). Arguments are now at 8(a6), 12(a6), etc. and auto storage is on the stack at -1(a6) on down. To return, the nonvolatile registers are restored from the stack, a6 and a7 are retrieved with an "unlk a6" and control is returned via "rts". Note that the stack must be balanced on exit if the nonvolatile registers are to be properly restored.

data representation - short integers are stored as two bytes, more significant byte first. Long integers are stored as four bytes, in descending order of significance. Floating numbers are represented as for the proposed IEEE Floating Point Standard, four bytes for float, eight for double, and are stored in descending order of byte significance. The IEEE representations are: most significant bit is one for negative numbers, else zero; next eleven bits (eight for float) are the characteristic, biased such that the binary exponent of the number is the characteristic minus 1022 (126 for float); remaining bits are the fraction, starting with the 1/4 weighted bit. If the characteristic is zero, the entire number is taken as zero and should be all zero to avoid confusing some routines that take shortcuts. Otherwise there is an assumed 1/2 added to all fractions to put them in the interval [0.5, 1.0). The value of the number is the fraction, times -1 if the sign bit is set, times 2 raised to the exponent.

storage bounds - even byte storage boundaries must be enforced for multibyte data. The compiler may generate incorrect code for passing long or double arguments if a boundary stronger than even is requested.

module name - is taken as the first defined external encountered, unless an explicit module name is given to the code generator. The module name is published via an idnt declaration.

NAME

Conventions - VERSAdos system subroutines

SYNOPSIS

```
#include <vdos.h>
```

FUNCTION

All standard system library functions callable from C follow a set of uniform conventions, many of which are supported at compile time by including a system header file, <vdos.h>, at the top of each program. Note that this header is used in addition to the standard header <std.h>. The system header defines various system parameters and data structures for use with VERSAdos.

Herewith the principal definitions:

CTRLZ - 032 (ctl-Z), end-of-file from a terminal.
FHS - the VERSAdos file services block and oft used values.
IOS - the VERSAdos I/O service block along with oft used bit values.
MAXCMD - the size of the longest command line to be passed to C.
RCB - struct rcb, the Whitesmiths, Ltd. file control block.
VEOF - 0xC2 VERSAdos end-of-file from a file.

NAME

c - compiling C programs

SYNOPSIS

=CHAIN C sfile

FUNCTION

c is an indirect command file that causes a C source file to be compiled and assembled. It does so by invoking the compiler passes and the assembler in the proper order, then deleting the intermediate files.

The C source file is at sfile.c, where sfile is the name typed in the submit line. The relocatable image from the assembler is put at sfile.ro.

EXAMPLE

To compile test.c:

=CHAIN C TEST

SEE ALSO

clink, pc

FILES

sfile.t?

BUGS

There should be some way to pass the -m flag to p1, or some of the myriad flags acceptable to pp, other than by modifying the command file proper.

NAME

pc - compiling Pascal programs

SYNOPSIS

=CHAIN PC sfile

FUNCTION

pc is an indirect command file that causes a Pascal source file to be compiled and assembled. It does so by invoking the compiler passes and the assembler in the proper order, then deleting the intermediate files.

The Pascal source file is at sfile.p, where sfile is the name typed in the submit line. The relocatable image from the assembler is put at sfile.ro.

EXAMPLE

To compile test.p:

=CHAIN PC TEST

SEE ALSO

c, clink

FILES

sfile.t?

BUGS

There should be some way to pass some of the myriad flags acceptable to ptc, other than by modifying the command file proper.

NAME

clink - link C programs

SYNOPSIS

=LINK CHDR/<files>,NAME,NAME;L=CLIB

FUNCTION

Programs written in C must be linked with certain object modules that implement the standard C runtime environment. The normal VERSAdos LINK program is used, as shown in the example above, which produces an executable file "name.lo", and map file "name.ll" from one or more object files in the list <files> that presumably were produced by the C compiler. Filenames in this list are separated by slashes '/'.

The top of memory space allocated by the linker (section 15) is used for the stack plus heap area, which is grown at task startup time by chdr. The size of this area in bytes is given by the value of _hsize which is an external reference. Its default value is 24K bytes. To reduce or increase this value, define the variable _hsize in one of the <files> and insure that it is statically initialized to the desired value.

chdr.ro is the module that gets control when name.lo is run. It calls the function _main(), described in the VERSAdos library, which redirects the standard input and output as necessary, calls the user provided main(ac, av) with the command arguments, and passes the value returned by main on to exit(). All of this malarkey can be circumvented if <files> contains a defining instance of _main().

<files> can be one or more object modules produced by the C compiler, or produced from Motorola Structured Assembler source that satisfies the interface requirements of C, as described in the C Compiler Runtime Library Manual. The important thing is that the function main(), or _main(), be defined somewhere among these files, along with any other modules needed and not provided by the standard C library.

clib is a library containing all of the portable C library functions, plus those required for the VERSAdos interface. It should be scanned last.

EXAMPLE

To compile and run the program echo.c:

```
=CHAIN C ECHO.C
=LINK CHDR/ECHO,ECHO,ECHO;L=CLIB
=ECHO hello world!
hello world!
```

SEE ALSO

_hsize, c

NAME

chdr - C startup code

SYNOPSIS

=LINK CHDR/<files>;L=CLIB

FUNCTION

chdr is the code that gets control whenever a C program is started. It first sets the total stack and heap size to the value in the long integer _hsize (by growing section 15 to _hsize bytes), sets the stack to the top of memory, translates the command line to lowercase, and calls _main, passing its return value on to exit. It also assures that neither text nor data begin at absolute location zero.

If the heap cannot be grown, the program aborts itself with a code of -1.

SEE ALSO

_hsize, _main

NAME

_hsize - size of the stack plus heap area

SYNOPSIS

```
BYTES _hsize {0x6000};
```

FUNCTION

_hsize specifies the size in bytes of the stack and heap area for the program. This area is allocated in section 15 at task startup time.

The default value provided in clib is 24K bytes. To reduce or increase this value, define the variable _hsize in the C program proper and insure that it is statically initialized to the desired value.

SEE ALSO

clink

NAME

_main - setup for main call

SYNOPSIS

```
BOOL _main()
```

FUNCTION

_main is the function called whenever a C program is started. It obtains the command line from parameters placed in _regs[] at task startup, parses it into argument strings, redirects STDIN and STDOUT as specified in the command line, then calls main.

The command line is interpreted as a series of strings separated by spaces. If a string begins with a '<', the remainder of the string is taken as the name of a file to be opened for reading text and used as the standard input, STDIN. If a string begins with a '>', the remainder of the string is taken as the name of a file to be created for writing text and used as the standard output, STDOUT. All other strings are taken as argument strings to be passed to main. The command name, av[0], is the program name as passed to it by VERSAdos.

EXAMPLE

To avoid the loading of _main and all the the file I/O code it calls on, one can provide a substitute _main instead:

```
BOOL _main()
    {
    <program body>
    }
```

RETURNS

_main returns the boolean value obtained from the main call.

SEE ALSO

exit

NAME

_pname - program name

SYNOPSIS

```
TEXT *_pname;
```

FUNCTION

_pname is the name by which the program is invoked. It is set from the command line at startup time and used for error printouts. Some portable programs provide a default program name by defining _pname, for those systems that are incapable of determining the actual invocation name; but not all programs are obliged to provide a defining instance of _pname. Hence this library module is provided, containing the default string "error".

NAME

_regs - register values on task startup

SYNOPSIS

```
ULONG _regs[15] {d0, d1, d2, d3, d4, d5, d6, d7,
    a0, a1, a2, a3, a4, a5, a6};
```

FUNCTION

_regs is an array which contains the values passed to the task at startup by VERSAdos. The offset values into this array are given in vdos.h. It is written by chdr on task startup.

SEE ALSO

chdr

NAME

close - close a file

SYNOPSIS

```
FILE close(fd)
    FILE fd;
```

FUNCTION

close closes the file associated with the file descriptor fd, making the fd available for future open or create calls. The I/O buffer is released for later use.

RETURNS

close returns the now useless file descriptor, if successful, or a negative number, which is -1 or the VERSAdos error return value, negated.

EXAMPLE

To copy an arbitrary number of files:

```
while (0 <= (fd = getfiles(&ac, &av, STDIN, -1)))
    {
    while (0 < (n = read(fd, buf, BUFSIZE)))
        write(STDOUT, buf, n);
    close(fd);
    }
```

SEE ALSO

create, open, remove, uname

NAME

create - open an empty instance of a file

SYNOPSIS

```
FILE create(name, mode, rsize)
    TEXT *name;
    COUNT mode;
    BYTES rsize;
```

FUNCTION

create enters a new file of specified name into the filesystem; if a file of that name already exists, it is first deleted.

If rsize is equal to 0, carriage returns and NULs are deleted on input, and a carriage return is inserted before each linefeed (newline) on output; otherwise, characters are input and output unchanged. mode is ignored.

RETURNS

create returns a file descriptor for the created file or a negative number, which is -1 or the VERSAdos error return negated.

EXAMPLE

```
if ((fd = create("xeq", WRITE, 1)) < 0)
    putstr(STDERR, "can't create xeq\n", NULL);
```

SEE ALSO

close, open, remove, uname

NAME

exit - terminate program execution

SYNOPSIS

```
VOID exit(success)
    BOOL success;
```

FUNCTION

exit calls all functions registered with onexit, closes all files, and terminates program execution. If (success == NO) the VERSAdos abort trap is called with a code of 0x8000, otherwise there is normal (quiet) termination.

RETURNS

exit will never return to the caller.

EXAMPLE

```
if ((fd = open("file", READ)) < 0)
    {
    putstr(STDERR, "can't open file\n", NULL);
    exit(NO);
    }
```

SEE ALSO

onexit

NAME

lseek - set file read/write pointer

SYNOPSIS

```
COUNT lseek(fd, offset, sense)
    FILE fd;
    LONG offset;
    COUNT sense;
```

FUNCTION

lseek uses the long offset provided to modify the read/write pointer for the file fd, under control of sense. If (sense == 0) the pointer is set to offset, which should be positive; if (sense == 1) the offset is algebraically added to the current pointer; otherwise (sense == 2) of necessity and the offset is algebraically added to the length of the file in bytes to obtain the new pointer.

The call lseek(fd, 0L, 1) is guaranteed to leave the file pointer unmodified and, more important, to succeed only if lseek calls are both acceptable and meaningful for the fd specified, i.e., the file is not a teletype.

RETURNS

lseek returns the file descriptor if successful, or a negative number, which is -1 or the VERSAdos return value, negated.

EXAMPLE

To read a 512-byte block:

```
BOOL getblock(buf, blkno)
    TEXT *buf;
    BYTES blkno;
    {
    lseek(STDIN, (LONG)blkno << 9, 0);
    return (fread(STDIN, buf, 512) != 512);
    }
```

BUGS

It doesn't check for illegal values of offset. If an existing file is opened, the length of the file is taken as the full space allocated for the file, no matter how little has actually been used.

NAME

onexit - call function on program exit

SYNOPSIS

```
VOID (*onexit())(pfn)
    VOID (*(*pfn)())();
```

FUNCTION

onexit registers the function pointed at by pfn, to be called on program exit. The function at pfn is obliged to return the pointer returned by the onexit call, so that any previously registered functions can also be called.

RETURNS

onexit returns a pointer to another function; it is guaranteed not to be NULL.

EXAMPLE

```
IMPORT VOID (*(*nextguy)())(), (*thisguy())();

if (!nextguy)
    nextguy = onexit(&thisguy);
```

SEE ALSO

exit

BUGS

The type declarations defy description, and are still wrong.

NAME

open - open a file

SYNOPSIS

```
FILE open(name, mode, rsize)
    TEXT *name;
    COUNT mode;
    BYTES rsize;
```

FUNCTION

open opens a file of specified name and assigns a file descriptor to it. If (rsize == 0) carriage returns and NULs are deleted on input, and a carriage return is inserted before each linefeed (newline) on output; if (rsize & 01) the record length is set to 256; else the record length is rsize. mode is ignored.

RETURNS

open returns a file descriptor for the opened file or a negative number, which is -1 or the VERSAdos return value, negated.

EXAMPLE

```
if ((fd = open("xeq", WRITE, 1)) < 0)
    putstr(STDERR, "can't open xeq\n", NULL);
```

SEE ALSO

close, create

NAME

read - read from a file

SYNOPSIS

```
COUNT read(fd, buf, size)
    FILE fd;
    TEXT *buf;
    BYTES size;
```

FUNCTION

read reads up to size characters from the file specified by fd into the buffer starting at buf. If STDIN is read, and it has not been redirected from the console, characters are read up to and including the first carriage return or newline; carriage return is changed to newline. Otherwise if the file has been opened with (rsize == 0) carriage returns and NULs are deleted; otherwise characters are input unchanged.

RETURNS

If an error occurs, read returns a negative number which is -1 or the VERSAdos return value, negated. If end-of-file is encountered, read returns zero; otherwise the value returned is between 1 and size, inclusive. When reading from a disk file, size bytes are read whenever possible.

EXAMPLE

To copy a file:

```
while (0 < (n = read(STDIN, buf, BUFSIZE)))
    write(STDOUT, buf, n);
```

SEE ALSO

open, write

NAME

remove - remove a file

SYNOPSIS

```
FILE remove(fname)
    TEXT *fname;
```

FUNCTION

remove deletes the file fname from the filesystem.

RETURNS

remove returns zero, if successful, or a negative number, which is -1 or the VERSAdos error return code, negated.

EXAMPLE

```
if (remove(uname()) < 0)
    putstr(STDERR, "can't remove temp file\n", NULL);
```

NAME

sbreak - set system break

SYNOPSIS

```
TEXT *sbreak(size)
    BYTES size;
```

FUNCTION

sbreak adjusts the top of the user area, algebraically up by size bytes, rounded up if size is odd.

RETURNS

If successful, sbreak returns a pointer to the start of the added data area, rounded up to a word boundary if necessary; otherwise the value returned is NULL.

EXAMPLE

```
if (!(p = sbreak(nsyms * sizeof (symbol))))
    {
    putstr(STDERR, "not enough room!\n", NULL);
    exit(NO);
    }
```

NAME

trap1 - make a task management system call

SYNOPSIS

```
UTINY _trap1(a0arg, d0arg)
    LONG a0arg, d0arg;
```

FUNCTION

_trap1 performs a VERSAdos system call, to perform some task management function. The arguments vary with the actual VERSAdos function desired; a0arg is the value placed in a0 before the call, and d0arg is placed in d0. See chapter 3 of the VERSAdos Preliminary Description for details on these features.

RETURNS

trap1 returns the status code byte returned by the trap. In general, a value of 0 signals success; other codes are function dependent.

SEE ALSO

trap2, trap3

NAME

trap2 - make an input/output system call

SYNOPSIS

```
UTINY _trap2(pios)
    IOS *pios;
```

FUNCTION

_trap2 performs a VERSAdos system call, to perform some input/output function. pios points at a VERSAdos I/O service block, as declared in vdos.h; its contents specify the operation to be performed. See chapter 3 of the VERSAdos Preliminary Description for details on these features.

RETURNS

trap2 returns the status code byte returned by the trap. In general, a value of 0 signals success; other codes are function dependent.

SEE ALSO

trap1, trap3

NAME

trap3 - make a file service system call

SYNOPSIS

```
UTINY _trap3(pfhs)
    FHS *pfhs;
```

FUNCTION

_trap3 performs a VERSAdos system call, to perform some file service function. pfhs points at a VERSAdos file services block, as declared in vdos.h; its contents specify the operation to be performed. See chapter 3 of the VERSAdos Preliminary Description for details on these features.

RETURNS

trap3 returns the status code byte returned by the trap. In general, a value of 0 signals success; other codes are function dependent.

SEE ALSO

trap1, trap2

NAME

uname - create a unique filename

SYNOPSIS

TEXT *uname()

FUNCTION

uname returns a pointer to the start of a NUL terminated name which is likely not to conflict with normal user filenames. The name may be modified by a one letter suffix. The name may be used as the first argument to a create, or subsequent open, call, so long as any such files created are removed before program termination. It is considered bad manners to leave scratch files lying about.

RETURNS

uname returns the same pointer on every call, which is currently the string "ctempc.x". The pointer will never be NULL.

EXAMPLE

```
if ((fd = create(uname(), WRITE, 1)) < 0)
    putstr(STDERR, "can't create sort temp\n", NULL);
```

SEE ALSO

create, open, remove

NAME

write - write to a file

SYNOPSIS

```
COUNT write(fd, buf, size)
    FILE fd;
    TEXT *buf;
    COUNT size;
```

FUNCTION

write writes size characters starting at buf to the file specified by fd. If STDOUT or STDERR is written, and it has not been redirected from the console, or if the file has been created or opened with (rsize == 0) each newline is preceded by a carriage return. Otherwise characters are output unchanged.

RETURNS

If an error occurs, write returns a negative number which is -1 or the VERSAdos return value, negated. Otherwise the value returned should be size.

EXAMPLE

To copy a file:

```
while (0 < (n = read(STDIN, buf, size)))
    write(STDOUT, buf, n);
```

SEE ALSO

create, open, read