III.d. RT-11 System Interface Library

NAME

Interface - to RT-11 system

FUNCTION

Programs written in C for operation on the PDP-11 under RT-11 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 six letters must be distinct. Any underscore is changed to a '.'. The identifer "_" must be avoided.

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

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

initialized data - is normally generated into the c$data section. External data names are published via .globl declarations.

uninitialized declarations - result in a .globl reference, one instance per program file.

function calls - are performed by

1) moving arguments on the stack, right to left. Character data is sign-extended to integer, float is zero-padded to double.

2) calling via 'jsr pc,func'.

3) popping the arguments off the stack.

Except for returned value, the registers r0, r1, fr0, fr1, and the condition code are undefined on return from a function call. All other registers are preserved. The returned value is in r0 (char sign-extended to integer, integer, pointer to), or in r0-r1 (long), or in fr0 (float widened to double, double). If the floating point processor is not to be used, the double storage location c$fac is the analog of fr0, c$fac+010 is the analog of fr1. If the FPP is used, its mode must be double/integer on entry and exit of a C function.

stack frames - are maintained by each C function, using r5 as a frame pointer. On entry to a function, the call 'jsr r5,c$sav' will stack r5, r4, r3, and r2 and leave r5 pointing at the stacked r5. Arguments are now at 4(r5), 6(r5), etc. and auto storage may be reserved on the stack at -7(r5) on down. To return, the jump 'jmp c$ret' will use r5 to restore r2, r3, r4, r5 and sp then return via 'rts pc'. The previous sp is ignored, so the stack need not be balanced on exit, and none of the potential return registers are used. The alternate jump 'jmp c$rets' will return without restoring r2, r3, r4.

data representation - integer is the same as short, two bytes stored less significant byte first. Long integers are stored as two two-byte integers, more significant integer first. Floating numbers are represented as for the PDP-11 Floating Point Processor, four bytes for float, eight for double, and are stored as two or four short integers, in descending order of significance.

storage bounds - Even byte storage boundaries must be enforced for multi-byte 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 a .title declaration.

SEE ALSO

c~ret(IV), c~rets(IV), c~sav(IV), c~savs(IV)

NAME

Conventions - RT-11 system subroutines

SYNOPSIS

#include <rt11.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, <rt11.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 a useful macro or two.

Herewith the principal definitions:

CTRLZ - 032 (control-z), EOF from teletype.
EMTERR - char at 052, the emt error code.
END - int at 050, the user high water mark.
FACRE - 1, create flag.
FADEL - 2, delete flag.
FAOPN - 0, open flag.
JSW - int at 044, the job status word.
MAXCMD - 183, maximum number of chars passed via .CHAIN.
NFILES - 10, maximum number of open files.
RCB - struct rcb, the Whitesmiths, Ltd. file control block.
SYSCLOS - 03000, emt CLOSE code.
SYSDEL - 00000, emt DELETE code.
SYSDSTAT - 0342, emt DSTATUS code.
SYSENT - 01000, emt ENTER code.
SYSOPEN - 00400, emt OPEN code.
SYSREAD - 04000, emt READ code.
SYSWRIT - 04400, emt WRITE code.
STARTCMD - 0510, start of .CHAIN information.
WCR - 4, status flags for RCB
WDIRT - 16
WOPEN - 1
WTTY - 2

SEE ALSO

RT-11 Advanced Programmer's Guide.

NAME

c - compiler driver for Pascal and C

SYNOPSIS

RUN C
> -[c i* o* p] <files>

FUNCTION

c is a program that causes one or more Pascal or C source files to be compiled and assembled. It does so by producing a command script that invokes the compiler passes and macro assembler in the proper order for each named source file, then exits to that script. Since this form of exit terminates any pending command scripts, c can be instructed only to produce the script for later invocation. Provision is also made for naming the command script, so that multiple users in the same directory can avoid collisions.

The flags are:

-c don't chain to command script after writing it.

-i* use the file * instead of "cproto.com" or "pproto.com" as the input prototype.

-o* output the command file to the file *. Default is "ctempc.com".

-p specifies a Pascal compile, which uses pproto.com as the prototype, unless otherwise directed by a -i* flag.

<files> is a list of one or more source file names, separated by spaces, that are to be compiled. If no extension is given in a <files> filename, it is taken as ".c" for C source or ".p" for Pascal; object files are typically written to the same filename with the extension ".obj". Other extensions are used by the compile scripts for intermediate files, to wit: ".mac", ".tm0", ".tm1", and ".tm2".

The command file is modelled after the cproto.com or pproto.com file, or similar script specified by the user via the -i option, where D stands for the device (defaults to ""), N stands for the program name (character string preceding the period), and E is the extension.

Since a prototype file is a text file, it is easy to vary filenames or flags for local usage.

EXAMPLE

To compile "junk.tst" and "test.c":

```
.RUN C
>junk.tst test
```

Or, from within a command script:

```
run c
-c -o test1.com junk.tst test
@test1
```

SEE ALSO

clink

FILES

cproto.com, pproto.com

NAME

clink - link C programs

SYNOPSIS

.LINK/BOT:3000/STA:3000/EXEC:name CHDR,<files>,CLIB

FUNCTION

Programs written in C must be linked with certain object modules that implement the standard C runtime environment. The normal RT-11 LINK program is used, but with several important constraints. These are shown in the example above, which produces an executable file "name.sav" from one or more object files in the list <files> that presumably were produced by the C compiler.

Since C is a stack intensive language, it is almost always necessary to provide a larger stack than the 0300 or less bytes provided by default. The BOT and STA options work together to provide a stack that starts at 03000 and grows downward to the command line written at 0510 on up; this seems to be sufficient space for most C programs. It may have to be changed, however, to suit individual programs.

chdr.obj is the module that gets control when name.sav is run. It calls the function _main(), described in the RT-11 library, which puts a ">" prompt, reads in a command line, 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 MACRO-11 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 RT-11 interface. It should be scanned last.

EXAMPLE

To compile and run the program echo.c:

```
.RUN C
>echo
.LINK/BOT:3000/STA:3000/EXEC:ECHO CHDR,ECHO,CLIB
.RUN ECHO
>Hello world!
Hello world!
.DEL/NOQU ECHO.OBJ
```

SEE ALSO

c

NAME

_main - setup for main call

SYNOPSIS

BOOL _main()

FUNCTION

_main is the function called whenever a C program is started. It obtains a command line, if possible, parses it into argument strings, redirects STDIN and STDOUT as specified in the command line, then calls main.

If the program is started via a .CHAIN system call, then the command line is assumed to be written as a NUL terminated string starting at 0510 in the system communication area. Otherwise a .GTLIN call is performed, using a ">" prompt, to read a command into this area.

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, which is called as described in the Intro to the C SUBROUTINE MANUAL. The command name, av[0], is taken from _pname.

Bit 14 of location 044 (RT-11 Job Status Word) is set to allow input of lowercase characters.

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:

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

RETURNS

_main returns the boolean value obtained from the main call, which is then used to set the success code SUCCS in 053, if non-zero, or to set the severe error code SEVER, if zero.

SEE ALSO

_pname, exit

NAME

_pname - program name

SYNOPSIS

```
TEXT *pname;
```

FUNCTION

_pname is the (NUL terminated) name by which the program was invoked, at least as anticipated at compile time. If the user program provides no definition for _pname, a library routine supplies the name "error", since it is used primarily for labelling diagnostic printouts.

Argument zero of the command line is set equal to _pname.

SEE ALSO

main

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 RT-11 error return byte, ORed with -256.

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. File size is the standard system default.

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 RT-11 error return byte, ORed with -0400.

EXAMPLE

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

SEE ALSO

close, open, rad50, remove, uname

BUGS

There should be some way to specify file size when necessary.

NAME

emt - make a system call

SYNOPSIS

```
COUNT emt(code, r0, ...)
    COUNT code;
    TEXT *r0;
    ...
```

FUNCTION

emt can be used to perform any RT-11 system call. It builds an emt instruction whose low order byte is the low order byte of code, copies the argument r0 into r0, then performs the emt with any remaining arguments at the top of the stack. If the particular emt call expects stacked arguments, it is important that all be present, else data in the calling program may be corrupted.

RETURNS

If the system call succeeds (carry bit cleared), emt returns the contents of r0 after the emt, ANDed with 077777 to ensure that it is positive even if it contains garbage; otherwise emt returns the RT-11 error byte, ORed with -0400 to ensure that it is negative.

EXAMPLE

To read a character from the console:

```
while ((c = emt(0340, 0)) < 0)
        ;
```

To close a file:

```
if (emt(0374, SYSCLOS + chan) < 0)
    error("can't close\n");
```

SEE ALSO

emt375, RT-11 Advanced Programmer's Guide.

BUGS

Forcing r0 positive on a successful call may not always be wise, e.g., in the case of .SETTOP.

NAME

emt375 - make a system call 0375

SYNOPSIS

```
COUNT emt375(code, args, ...)
    COUNT code;
    ...
```

FUNCTION

emt375 is designed to ease making a broad class of RT-11 system calls that perform an emt 0375 with r0 pointing at a list of arguments. Typically, such a list begins with a code word, often with a channel number in the low byte, and contains one or more additional arguments immediately following. The arguments to emt375 are used as this argument list, with no interpretation.

RETURNS

If the system call succeeds (carry bit cleared), emt375 returns the contents of r0 after the emt, ANDed with 077777 to ensure that it is positive even if it contains garbage; otherwise emt375 returns the RT-11 error byte, ORed with -0400 to ensure that it is negative.

EXAMPLE

To write a block to a file:

```
emt375((11<<8) + chan, blk, buf, wcnt, 0);
```

SEE ALSO

emt, RT-11 Advanced Programmer's Guide.

BUGS

Forcing r0 positive on a successful call may not always be wise.

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. Bit 0 of byte 53 is set to indicate success, while bit 3 is used to signal failure, i.e., (success == 0).

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

fcall - call a Fortran program

SYNOPSIS

```
TYPE fcall(fn, nargs, arg1, ...)
    TYPE (*fn)();
    COUNT nargs;
    ...
```

FUNCTION

fcall is a generic interface function for calling Fortran programs. fn is the Fortran function or subroutine to call, nargs is the number of arguments, and arg1, ... are the arguments to be passed to fn on the call. TYPE is whatever type of value fn returns.

Note that Fortran expects arguments to be addresses, not values.

If a C function is called from Fortran, the arguments can be picked up by the following ruse:

```
cfn(p)
    struct {
        int nargs;
        int *arg[1];
        } *p;
/* (&p)[-2]->nargs gives the number of arguments
 * (&p)[-2]->arg[n] gives pointer to arg n
 */
```

Integer values may be returned in the normal manner.

RETURNS

fcall returns whatever fn returns.

EXAMPLE

```
fcall(&func, 1, &x);
```

BUGS

REALs are returned as C longs, and INTEGER * 4, REAL * 8, or COMPLEX cannot be returned properly. Fortran does not maintain FPP status required by C.

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. RT-11 uses only the low order 25 bits of the offset; the rest are ignored.

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 RT-11 error return byte, ORed with -0400.

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. If that length is greater than 32,767 blocks, the length is curdled by emt375.

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

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

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

SEE ALSO

exit

BUGS

The type declarations defy description, and are still wrong.

NAME

onintr - capture interrupts

SYNOPSIS

```
VOID onintr(pfn)
    VOID (*pfn)();
```

FUNCTION

onintr is supposed to ensure that the function at pfn is called on the occurrence of an interrupt generated from the keyboard of a controlling terminal. (Typing a delete DEL, or sometimes a ctl-C ETX, performs this service on many systems.)

On this system, onintr is currently a dummy, so pfn is never called.

RETURNS

Nothing.

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; otherwise characters are input and output unchanged. mode is ignored.

RETURNS

open returns a file descriptor for the created file or a negative number, which is -1 or the RT-11 error return byte, ORed with -0400.

EXAMPLE

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

SEE ALSO

close, create

NAME

rad50 - convert ASCII to rad50

SYNOPSIS

```
COUNT rad50(s, n)
    TEXT *s;
    COUNT n;
```

FUNCTION

rad50 converts up to three characters of the string starting at s into a packed word containing three "radix 50" characters. If n is less than three, the string is effectively padded on the right by spaces; a zero or negative count n yields a word of all spaces, i.e., zero.

The valid rad50 characters consist of the space ' ', period '.', dollar sign '$', digits '0' through '9', and letters in either case, 'a' through 'z' or 'A' through 'Z'. All other characters are mapped into the illegal rad50 character.

EXAMPLE

To convert an RT-11 style filename:

```
n = scnstr(fn, ':');
if (fn[n])
    {
    dblk[0] = rad50(fn, n);
    fn =+ n + 1;
    }
else
    dblk[0] = 0;
n = scnstr(fn, '.');
dblk[1] = rad50(fn, n);
dblk[2] = rad50(fn + 3, n - 3);
dblk[3] = (fn[n] == '.') ? rad50(fn + n + 1, lenstr(fn + n + 1)) : 0;
```

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 RT-11 error return byte, ORed with -0400; 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

emt, emt375, open, write

NAME

remove - remove a file

SYNOPSIS

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

FUNCTION

remove deletes the file fname from the file system.

RETURNS

remove returns zero, if successful, or a negative number, which is -1 or the RT-11 error return byte, ORed with -0400.

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);
    }
```

BUGS

A kludge version of sbreak must be used for RSTS, since that system doesn't seem to emulate .SETTOP properly.

NAME

uname - create a unique file name

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 letter suffix, so that a family of files may be dealt with. 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.". 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

close, 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 RT-11 error return byte, ORed with -0400. 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, emt, open, read