III.c. ISIS-II System Interface Library

NAME

Interface - to ISIS-II system

FUNCTION

Programs written in C for operation on the 8080 under ISIS-II are translated into A-Natural according to the following specifications:

external identifiers - may be written in both upper and lower case. The first eight letters must be distinct. Any underscore is left alone. An underscore is prepended to each identifier.

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

literal data - such as strings and switch tables, are normally generated into the .text section.

initialized data - are normally generated into the .data section. External data names are published via public declarations.

uninitialized declarations - result in a public 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 "call _func",

3) popping the arguments off the stack.

Except for returned value, the registers fa, bc, and hl, and the in-core pseudo registers c.r0 and c.r1, are undefined on return from a function call. All other registers are preserved. The returned value is in bc (char sign-extended to integer, integer, pointer to), or in c.r0 (long, float widened to double, double). c.r0 and c.r1 are each eight-byte memory areas that are not preserved on a function call; c.r2, c.r3, and c.r4 are each two-byte memory areas that must be preserved on a function call.

stack frames - are maintained by each C function, using de as a frame pointer. On entry to a function, the call "call c.ents" will stack de and leave it pointing at the stacked de, then stack c.r2, c.r3, and c.r4. Arguments are at 4(de), 6(de), etc. and auto storage may be reserved on the stack at -7(de) on down. To return, the jump "jmp c.rets" will use de to restore de, sp, c.r4, c.r3, and c.r2, then return. The previous sp is ignored, so the stack need not be balanced on exit, and none of the potential return registers are used. If it is not necessary to save or restore c.r2, c.r3, and c.r4, the call "call c.ent" and jump "jmp c.ret" may be used instead.

data representation - integer is the same as short, two bytes stored less significant byte first. Long integers are stored as two short integers, more significant short first. All signed integers are twos complement. 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 - no storage bounds need to be enforced. Two-byte boundaries may be enforced, inside structures and among automatic variables, to ensure data structure compatibility with machines such as the PDP-11, but boundaries stronger than this are not fully supported by the stacking hardware; the compiler may generate incorrect code for passing long or double arguments if a boundary stronger than even is requested.

module name - is not produced.

SEE ALSO

c.ent(IV), c.ents(IV), c.entx(IV), c.r0(IV), c.ret(IV), c.rets(IV)

NAME

Conventions - ISIS-II system subroutines

SYNOPSIS

```
#include <isis.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, <isis.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:

```
FAIL - -1, standard failure return code
IOPEN - 0, ISIS-II system call codes
ICLOSE - 1
IDELETE - 2
IREAD - 3
IWRITE - 4
ISEEK - 5
ILOAD - 6
IRENAME - 7
ICONSOL - 8
IEXIT - 9
IATTRIB - 10
IRESCAN - 11
IERROR - 12
IWHOCON - 13
ISPATH - 14
NFILES - 9, maximum number of open files
WOPEN - 1, WCB flags
WCR - 2
```

SEE ALSO

ISIS-II Manual

NAME

_main - setup for main call

SYNOPSIS

BOOL _main()

FUNCTION

_main is the function called whenever a C program is started. It reads the remainder of the command line from STDIN, 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, which is called as described in the Intro to the C SUBROUTINE MANUAL. The command name, av[0], is taken from _pname.

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, which is then passed to exit.

SEE ALSO

_pname, exit

NAME

_pname - program name

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.

RETURNS

close returns the now useless file descriptor, if successful, or a negative number, which is the ISIS-II error return code, 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 makes a new file of specified name, if it did not previously exist, or truncates the existing file to zero length. If (mode == 0) the file is opened for reading, else if (mode == 1) it is opened for writing, else (mode == 2) of necessity and the file is opened for updating (reading and writing).

If (rsize == 0), carriage returns are deleted on input, and a carriage return is injected before each newline on output. If (rsize != 0) data is transmitted unaltered.

RETURNS

create returns a file descriptor for the created file or a negative number, which is the ISIS-II error return code, negated.

EXAMPLE

```
if ((fd = create("xeq", WRITES, 1)) < 0)
    write(STDERR, "can't create xeq\n", 17);
```

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, then terminates program execution. success is ignored.

RETURNS

exit will never return to the caller.

EXAMPLE

```
if ((fd = open("file", READ)) < 0)
    {
    write(STDERR, "can't open file\n", 16);
    exit(NO);
    }
```

SEE ALSO

onexit

NAME

isis - call ISIS-II

SYNOPSIS

```
struct cb *isis(code, pcb)
    COUNT code;
    struct cb *pcb;
```

FUNCTION

isis is the C callable entry to the ISIS-II system. It loads the less significant byte of code into the c register, and pcb into de, then calls ISIS-II. Each struct cb is typically a function of code.

RETURNS

isis returns pcb.

EXAMPLE

To close a file:

```
    IMPORT WCB _wcb[];
    COUNT stat;
    struct {
        COUNT aftn, *pstat;
        } cb;

    cb.aftn = _wcb[fd].waftn;
    cb.pstat = &stat;
    isis(ICLOSE, &cb);
    return (stat ? -stat : fd);
```

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. ISIS-II uses only the low order 22 bits of the offset; the rest are ignored.

RETURNS

lseek returns zero if successful, or a negative number, which is the ISIS-II error return code, negated.

EXAMPLE

To read a 512-byte block:

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

BUGS

It doesn't check for illegal values of sense, which can cause mayhem when used by ISIS-II.

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

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.)

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

RETURNS

Nothing.

NAME

open - open an existing file

SYNOPSIS

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

FUNCTION

open associates a file descriptor with an existing file, or with a new instance if the file is to be written and does not currently exist. If (mode == 0) the file is opened for reading, else if (mode == 1) it is opened for writing, else (mode == 2) of necessity and the file is opened for updating (reading and writing).

If (rsize == zero), carriage returns are deleted on input, and a carriage return is injected before each newline on output. If (rsize != 0) data is transmitted unaltered.

RETURNS

open returns a file descriptor for the file or a negative number, which is the ISIS-II error return code, negated.

EXAMPLE

```
if ((fd = open("xeq", WRITES, 1)) < 0)
    write(STDERR, "can't open xeq\n", 15);
```

SEE ALSO

close, create, remove, uname

NAME

read - read characters 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 the file was created or opened with (rsize == 0), or if STDIN is not redirected from the console, carriage returns are discarded on input.

RETURNS

If an error occurs, read returns a negative number which is the ISIS-II error code, negated; if end of file is encountered, read returns zero; otherwise the value returned is between 1 and size, inclusive.

EXAMPLE

To copy a file:

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

SEE ALSO

write

NAME

remove - remove a file

SYNOPSIS

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

FUNCTION

remove deletes the file fname from the ISIS-II diskette.

RETURNS

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

EXAMPLE

```
if (remove("temp.c") < 0)
    write(STDERR, "can't remove temp file\n", 23);
```

NAME

sbreak - set system break

SYNOPSIS

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

FUNCTION

sbreak moves the system break, at the top of the data area, algebraically up by size bytes.

RETURNS

If successful, sbreak returns a pointer to the start of the added data area; otherwise the value returned is NULL.

EXAMPLE

```
if (!(p = sbreak(nsyms * sizeof (symbol))))
    {
    write(STDERR, "not enough room!\n", 17);
    exit(NO);
    }
```

BUGS

The stack is assumed to lie above the data area, so sbreak will return a NULL if the new system break lies above the current stack pointer; this may not be desirable behavior on all memory layouts.

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 suffix of up to three letters, so that a family of process-unique 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, 0)) < 0)
    write(STDERR, "can't create sort temp\n", 23);
```

SEE ALSO

close, create, open, remove

NAME

write - write characters 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 the file was created or opened with (rsize == 0), or if STDOUT or STDERR is not redirected from the console, each newline output is preceded by a carriage return.

RETURNS

If an error occurs, write returns a negative number which is the ISIS-II error code, 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

read