

III.b. CP/M System Interface Library

III.b - 1	Interface	to CP/M system
III.b - 3	Conventions	CP/M system subroutines
III.b - 5	c	compiling C programs
III.b - 6	pc	compiling Pascal programs
III.b - 7	ld	linking a C program
III.b - 8	chdr	C runtime entry
III.b - 9	_main	setup for main call
III.b - 10	_pname	program name
III.b - 11	close	close a file
III.b - 12	cpm	call CP/M or CDOS system
III.b - 13	create	open an empty instance of a file
III.b - 14	exit	terminate program execution
III.b - 15	lseek	set file read/write pointer
III.b - 15a	onexit	call function on program exit
III.b - 16	onintr	capture interrupts
III.b - 17	open	open an existing file
III.b - 18	read	read characters from a file
III.b - 19	remove	remove a file
III.b - 20	sbreak	set system break
III.b - 21	uname	create a unique file name
III.b - 22	write	write characters to a file

NAME

Interface - to CP/M system

FUNCTION

Programs written in C for operation on the 8080 under CP/M 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 - np 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 - CP/M system subroutines

SYNOPSIS

#include <cpm.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, <cpm.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, ctl-Z for text end of file
EOF - 1, end of file from CP/M read
FAIL - -1, standard failure return code
MCREATE - 0, fopen modes
MOPEN - 1
MREMOVE - 2
MWRITE - 4
SYSBUF - 0x80, location of CP/M buffer
CRESET - 0, CP/M system call codes
CRDCON - 1
CWRCON - 2
CRDRDR - 3
CWRPUN - 4
CWRLST - 5
CDCIO - 6
CGIOST - 7
CSIOST - 8
CPRBUF - 9
CRDBUF - 10
CICRDY - 11
CLFTHD - 12
CINIT - 13
CLOGIN - 14
COPEN - 15
CCLOSE - 16
CSRCH - 17
CSRCHN - 18
CDEL - 19
CREAD - 20
CWRITE - 21
CMAKE - 22
CRENAME - 23
CILOGIN - 24
CIDRNO - 25
CSETAD - 26
CIALLOC - 27
CWPROT - 28
CGETVEC - 29
CSETFA - 30

Conventions

- 2 -

Conventions

CGETPAR - 31
CGSUSER - 32
CRREAD - 33
CRWRITE - 34
CFSIZE - 35
CSETRR - 36
CDRESET - 37
CRZWRT - 40
WOPEN - 1, the WCB flags
WDIRT - 2
WSTD - 4
WCR - 010
WUSED - 020
LST - -4, the device codes
PUN - -3
RDR - -2
CON - -1

SEE ALSO

CP/M or CDOS Manual

c

III.b. CP/M System Interface Library

c

NAME

c - compiling C programs

SYNOPSIS

A:submit 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.o.

EXAMPLE

To compile test.c:

A:submit c test

SEE ALSO

ld, pc

FILES

sfile.tm?

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

A:submit 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.o.

EXAMPLE

To compile test.p:

A:submit pc test

SEE ALSO

c, ld

FILES

sfile.tm?

BUGS

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

NAME

ld - linking a C program

SYNOPSIS

A:submit ld ofile

FUNCTION

Programs written in C must be linked with certain object modules that implement the standard C runtime environment. The ld command script invokes the link program, including the standard object header file, the object file ofile, and any libraries in the correct order, and produces the executable image xeq.com.

The standard object header gets control when xeq.com is run. It calls the function `_main()`, described in the system library, which 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 file contains a defining instance of `_main()`.

ofile must be a standard object file produced by the assembler, or by an earlier (partial binding) invocation of link. The assembler accepts either the output of the C code generator p2 or assembler source that satisfies the interface requirements of C, as described earlier under Interface. 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.

EXAMPLE

To compile and run the program echo.c:

```
A:submit c echo
A:submit ld echo
A:xeq hello world!
hello world!
```

SEE ALSO

c, pc

chdr

III.b. CP/M System Interface Library

chdr

NAME

chdr - C runtime entry

SYNOPSIS

link -htr -tb0x100 a:chdr.o <files>

FUNCTION

All CP/M programs begin execution at location 0x100; chdr.o is the startup routine, linked at this address, that sets the stack to the address stored at location 6, then calls `_main()`.

Included in the chdr module is the C callable interface function `cpm()`, plus several internal functions that isolate 8080 machine dependencies.

SEE ALSO

`_main`, `cpm`

_main

III.b. CP/M System Interface Library

_main

NAME

_main - setup for main call

SYNOPSIS

BOOL _main()

FUNCTION

_main is the function called whenever a C program is started. It parses the command line at 0x80 (in the program base) 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 taken from _pname.

Note that the argument strings remain in the default record buffer beginning at 0x80 in the program base, which is never used by other C interface routines.

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, cpm, exit

_pname

III.b. CP/M System Interface Library

_pname

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

close

III.b. CP/M System Interface Library

close

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. If the file was written to or created, close ensures that the last record is written out and the directory entry properly closed.

RETURNS

close returns the now useless file descriptor, if successful, or -1.

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

cpm - call CP/M or CDOS system

SYNOPSIS

```
COUNT cpm(bc, de, hl)
COUNT bc;
TEXT *de, *hl;
```

FUNCTION

cpm is the C callable function that permits arbitrary calls to be made on CP/M or CDOS. It loads its arguments into registers bc, de, and hl, then performs a call to absolute location 5. The function to be performed is specified by the less significant byte of bc, i.e., the c register; typically de contains a word integer or a pointer. hl is used only by CDOS, and then only on rare occasions.

RETURNS

cpm returns the a register returned by the CP/M call, sign extended to a word integer.

EXAMPLE

To read a line from the console:

```
buf[0] = 125;
cpm(CRDBUF, buf, 0);
cpm(CWRCON, '\n', 0);
```

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). This mode information is largely ignored, however.

If (rsize is zero), carriage returns and NULs are deleted on input, a ctl-Z is treated as an end of file, a carriage return is injected before each newline on output, and a ctl-Z is appended to the data in a partially filled last record. If (rsize != 0) data is transmitted unaltered.

Filenames take the general form x:name.typ, where x is the disk designator, name is the eight-character filename, and typ its three-character type. If x: is omitted, it is taken as the disk logged in on the first call to create, open, or remove; a missing name or typ is taken as all blanks. Letters are forced uppercase.

The four physical devices con:, rdr:, pun:, and lst: are also accepted as filenames. Reading from pun: or lst: gives an instant end of file, while writing to rdr: sends all bytes to hell with no complaint.

RETURNS

create returns a file descriptor for the created file or -1.

EXAMPLE

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

SEE ALSO

close, open, remove, uname

exit

III.b. CP/M System Interface Library

exit

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

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 system uses only the low order 31 bits of the offset; the sign is ignored.

RETURNS

lseek returns zero if successful, or -1.

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

The length of the file is taken as 128 times the total number of records, even if a text file is terminated early with a ctl-Z.

onexit

III.b. CP/M System Interface Library

onexit

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 ensures that the function at pfn is called on a keyboard interrupt, usually caused by a DEL key typed during terminal input or output. (Under DOS on the 8086, a CTL-BREAK also serves this function.) Any earlier call to onintr is overridden.

The function is called with one integer argument, whose value is always zero, and must not return; if it does, an immediate error exit is taken.

If (pfn == NULL) then these interrupts are disabled (turned off).

RETURNS

Nothing.

EXAMPLE

A common use of onintr is to ensure a graceful exit on early termination:

```
onexit(&rmtemp);
onintr(&exit);
...
VOID rmtemp()
{
    remove(uname());
}
```

Still another use is to provide a way of terminating long printouts, as in an interactive editor:

```
while (!enter(docmd, NULL))
    putstr(STDOUT, "?\n", NULL);
...
VOID docmd()
{
    onintr(&leave);
}
```

SEE ALSO

onexit

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. 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). This mode information is largely ignored, however.

If (rsize == zero), carriage returns and NULs are deleted on input, a ctl-Z is treated as an end of file, a carriage return is injected before each newline on output, and a ctl-Z is appended to the data in a partially filled last record. If (rsize != 0) data is transmitted unaltered.

Filenames take the general form x:name.typ, where x is the disk designator, name is the eight-character filename, and typ its three-character type. If x: is omitted, it is taken as the disk logged in on the first call to create, open, or remove; a missing name or typ is taken as all blanks. Letters are forced uppercase.

The four physical devices con:, rdr:, pun:, and lst: are also accepted as filenames. Reading from pun: or lst: gives an instant end of file, while writing to rdr: sends all bytes to hell with no complaint.

RETURNS

open returns a file descriptor for the file or -1.

EXAMPLE

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

SEE ALSO

close, create, remove, uname

read

III.b. CP/M System Interface Library

read

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) carriage returns and NULs are discarded on input.

Reading from pun: or lst: always gives a count of zero.

If the console is read, DEL at the start of a line causes an interrupt, to be processed as specified by onintr.

RETURNS

If an error occurs, read returns -1; 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

onintr, write

remove

III.b. CP/M System Interface Library

remove

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 -1.

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 letter suffix, 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

write

III.b. CP/M System Interface Library

write

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), each newline output is preceded by a carriage return. Moreover, a ctl-Z is appended to a file that does not end on a record boundary.

RETURNS

If an error occurs, write returns -1; otherwise the value returned should be size. Writing to rdr: does nothing, but returns size as if it did everything.

Characters typed at the console are inspected during write calls; typing a DEL causes an interrupt, to be processed as specified by onintr.

EXAMPLE

To copy a file:

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

SEE ALSO

onintr, read