

SSCAN

XX
XX
XX
XX
797 - laser1 Sat Feb 9 19:35:53 EST 1985

ARGS: [-C -d -fp -oorig=c++ -otime=476843632]
NAME: [c++]
PASSCOMM: [sfbc!c++:D.Swartout:lpbin=m13:home=bc]
copies: [1]
ctime: [1]
files: [/usr/spool/lp/request/laser/d0-9680]
job: [laser-9004]
lpbin: []
mach: [sfbc (laser-9004)]
mail: []
originator: [c++]
printer: [laser1]
printtime: [476843714]
request: [laser-9680]
route: [nsc]
sequence: [797]
subtime: [476843632]
title: []
transit: [1]
version: [08-02-84]

%Z% %M% %I% %H% %T%
CC=CC
CFLAGS=+f

all: sscan CCcrt0.o CCmcert0.o

sscan: sscan.c ldfcn.h
\$(CC) \$(CFLAGS) -o sscan sscan.c -lld

CCmcert0.o: CCmcert0.s
cc -c CCmcert0.s

CCcrt0.o: CCcrt0.s
cc -c CCcrt0.s

clean:
rm -f *.i *..c

clobber: clean
rm -f sscan *.o

```
/* %Z% %M% %I% %H% %T% */
/*      @(#)ldfcn.h      2.2 2/28/83      */

/*
 *      The following two declarations appear in the IH versions of
 *      "stdio.h" but do not appear in the normal 1.2 versions.
 */

#ifndef LDFILE
struct ldfile {
    int      _fnum_;          /* so each instance of an LDFILE is unique */
    FILE     *ioptr;          /* system I/O pointer value */
    long     offset;          /* absolute offset to the start of the file */
    FILHDR   header;          /* the file header of the opened file */
    unsigned short type;      /* indicator of the type of the file */
};

/*
 *      provide a structure "type" definition, and the associated
 *      "attributes"
 */

#define LDFILE      struct ldfile
#define IOPTR(x)    x->ioptr
#define OFFSET(x)   x->offset
#define TYPE(x)     x->type
#define HEADER(x)   x->header
#define LDFSZ       sizeof(LDFILE)

/*
 *      define various values of TYPE(ldptr)
 */

#define ARTYPE      0177545

/*
 *      define symbolic positioning information for FSEEK (and fseek)
 */

#define BEGINNING   0
#define CURRENT     1
#define END         2

/*
 *      define a structure "type" for an archive header
 */

typedef struct
{
    char ar_name[16];
    long ar_date;
    int ar_uid;
    int ar_gid;
    long ar_mode;
    long ar_size;
```

```

} archdr;

#define ARCHDR archdr
#define ARCHSZ sizeof(ARCHDR)

/*
    define some useful symbolic constants
*/

#define SYMTBL 0      /* section nnumber and/or section name of the Symbol Table

#define SUCCESS 1
#define CLOSED 1
#define FAILURE 0
#define NOCLOSE 0
#define BADINDEX -1L

#define OKFSEEK 0

/*
    define macros to permit the direct use of LDFILE pointers with the
    standard I/O library procedures
*/

LDFILE *ldopen(char*, LDFILE*);
LDFILE *ldaopen(char*, LDFILE*);

#define GETC(ldptr)      getc(IOPTR(ldptr))
#define GETW(ldptr)      getw(IOPTR(ldptr))
#define FEOF(ldptr)      feof(IOPTR(ldptr))
#define FERROR(ldptr)    ferror(IOPTR(ldptr))
#define FGETC(ldptr)     fgetc(IOPTR(ldptr))
#define FGETS(s,n,ldptr) fgets(s,n,IOPTR(ldptr))
#define FILENO(ldptr)    fileno(IOPTR(ldptr))
#define FREAD(p,s,n,ldptr) fread(p,s,n,IOPTR(ldptr))
#define FSEEK(ldptr,o,p) fseek(IOPTR(ldptr),(p==BEGINNING)?(OFFSET(ldptr)+o):o,p)
#define FTELL(ldptr)     ftell(IOPTR(ldptr))
#define FWRITE(p,s,n,ldptr) fwrite(p,s,n,IOPTR(ldptr))
#define REWIND(ldptr)    rewind(IOPTR(ldptr))
#define SETBUF(ldptr,b) setbuf(IOPTR(ldptr),b)
#define UNGETC(c,ldptr)  ungetc(c,IOPTR(ldptr))
#define STROFFSET(ldptr) (HEADER(ldptr).f_symptr + HEADER(ldptr).f_nsyms * 18) /*
18
#endif

```

```

/* %Z% %M% %I% %H% %T% */
#include <stdio.h>
#include <filehdr.h>
#include "ldfcn.h"
#include <syms.h>

char* ldgetname(LDFILE*,SYMENT*) ;

typedef char* Strpstr;
Strpstr copy( Strpstr ) ;

int err = 0 ;

class Strings {
    Strpstr* argv;
    int first ;
    int bound ;
    void check(int);

public:
    int len ;
    Strpstr& operator[] (int x) { check(x) ; return argv[first+x] ; } ;
    void suffix_copy(Strpstr s) {
        check(first+len+1) ; argv[first+(len++)] = copy(s) ; } ;
    Strings( int x = 32 ) {
        argv = new Strpstr[x+1] ; first = x/2 ; len = 0 ; bound = x ;
    } ;
} ;

void Strings.check(int want) {
    if ( want <= 0 || want >= bound ) {
        int new_bound = 3*(len+1) ;
        int new_first = new_bound/3 ;
        Strpstr* new_argv = new Strpstr[new_bound+1] ;
        for ( int x = 0 ; x < len ; ++x ) {
            new_argv[new_first+x] = argv[first+x] ;
        }
        delete argv ;
        first = new_first ;
        bound = new_bound ;
        argv = new_argv ;
    }
}

Strpstr copy ( Strpstr old ) {
    Strpstr new_s = new char[strlen(old)+1] ;
    strcpy(new_s,old) ;
    return new_s ;
}

Strings* cons ;

Strings* dest ;

void dofile( char* n ) {
    LDFILE* f = ldopen(n,0) ;

```

```

if ( f == 0 ) {
    char buffer[BUFSIZ] ;
    sprintf(buffer,"sscan(%s)", n) ;
    perror(buffer) ;
    err = 4 ;
    return ;
}
while ( f != 0 ) {
    SYMENT sym;
    ldtbseek( f ) ;
    for ( int x_sym = 0 ; ldtbread(f,x_sym,&sym) == SUCCESS ; ++x_sym ) {
        char* str = ldgetname(f,&sym);
        if ( strncmp(str,"_STI",4) == 0 ) {
            cons->suffix_copy(str) ;
        }
        else if ( strncmp(str,"_STD",4) == 0 ) {
            dest->suffix_copy(str) ;
        }
        x_sym += sym.n_numaux ; /* skip auxentries */
    }
    if ( ldclose(f) != FAILURE ) f = 0 ;
}
}

main(int argc, char** argv) {
    int monitor = 0 ;
    cons = new Strings ;
    dest = new Strings ;

    for ( int x_arg = 1 ; x_arg < argc ; ++ x_arg ) {
        int len = strlen(argv[x_arg]) ;
        if ( len>=3 && argv[x_arg][0] != '-'
            && argv[x_arg][len-2] == '.'
            && strchr("oa",argv[x_arg][len-1]) != 0 ) {
            dofile( argv[x_arg] ) ;
        }
        else if ( strcmp(argv[x_arg],"-p") == 0 ) monitor=1 ;
    }
    printf("void _STC_all(e) int e ; {\n" ) ;
    for ( x_arg = 0 ; x_arg < cons->len ; ++x_arg ) {
        printf("%s();\n", (*cons)[x_arg] ) ;
    }
    printf("}\n");
    printf("void exit(err) {\n" ) ;
    for ( x_arg = 0 ; x_arg < dest->len ; ++x_arg ) {
        printf("%s();\n", (*dest)[x_arg] ) ;
    }
    if ( monitor ) printf("monitor();\n");
    printf("_cleanup();\n");
    printf("_exit(err);}\n");
    return err ;
}

```

```

.file "crt0.s"
# %Z% %M% %I% %H% %T%
# C runtime startup - call main; call exit when done.
# new startup procedure uses ost "50" to write
# the EPSW to trap on overflow and invalid operation
# this ties in to the 5.0 ucode delivery (I hope).

# modified by Jerry Schwarz to call constructors and
# destructors of CC

.set EXIT, 1
.set EPSWOST,50 # new ost for fp trapping modes
.set WREPSW,10 # 10 says write
.set DIVZERO,1 # trap on divide by zero bit
.set UNFL,2 # trap on underflow bit
.set OVFL,4 # trap on overflow bit
.set INVOP,8 # trap on invalid op bit
.set INEXACT,0x10 # trap on inexact result bit
.set TRAPBITS,DIVZERO+UNFL+OVFL+INVOP+INEXACT
.set STZERO,0x100 # divide by zero sticky bit
.set STUFLOW,0x200 # underflow sticky bit
.set STOFLOW,0x400 # overflow sticky bit
.set STINVOP,0x800 # invalid op sticky bit
.set STINEX,0x1000 # inexact result sticky bit
.set STBITS,STZERO+STUFLOW+STOFLOW+STINVOP+STINEX
.set EMASK,TRAPBITS+STBITS # initially clear all via EMASK
.set NEWEPSW,DIVZERO+OVFL+INVOP # default startup condition
.globl main
.globl _start
.globl _mcount
.globl environ
.globl _STC_all
_start:
    ost &EPSWOST
    call &0,_STC_all
    movaw .ostargs,%ap # sleazy, but it should work
    movw -4(%sp), environ
    call &3, main
    pushw %r0
    call &1, exit
    pushw %r0
    ost &EXIT
# good-bye

_mcount: # dummy in case an object module has been
    rsb # compiled with cc -p but not the load module

.data
.align 4
.ostargs:
    .word WREPSW
    .word EMASK
    .word NEWEPSW
environ:
    .word 0

```



```

.file      "mcr0.s"
# %Z% %M% %I% %H% %T%
# C runtime startup and exit with profiling.
# see crt0.s for explanation of EPSW stuff

.set      CBUFS, 600
.set      EXIT, 1
.set      WORDSIZE, 4
.set      EPSWOST, 50
.set      WREPSW, 10
.set      DIVZERO, 1
.set      UNFL, 2
.set      OVFL, 4
.set      INVOP, 8
.set      INEXACT, 0x10
.set      TRAPBITS, DIVZERO+UNFL+OVFL+INVOP+INEXACT
.set      STZERO, 0x100
.set      STUFLO, 0x200
.set      STOFLO, 0x400
.set      STINVOP, 0x800
.set      STINEX, 0x1000
.set      STBITS, STZERO+STUFLO+STOFLO+STINVOP+STINEX
.set      EMASK, TRAPBITS+STBITS
.set      NEWEPSW, DIVZERO+OVFL+INVOP
.globl    _cleanup
.globl    _start
.globl    environ
.globl    etext
.globl    exit
.globl    main
.globl    monitor
.globl    write
.globl    __Argv

_start:
movaw     .ostargs,%ap
ost        &EPSWOST
movw      -4(%sp), environ
movw      -8(%sp), __Argv
subw3     &.eprol, &etext, %r8
andw2     &-WORDSIZE, %r8

addw2     &8*CBUFS+12+WORDSIZE-1, %r8
andw2     &-WORDSIZE, %r8
addw3     &4,%r8,%r7
pushw     %r7
call      &1, sbrk
cmpw      &-1, %r0
je        .nospace
pushaw    .eprol
pushaw    etext
pushw     %r0
lrsw2     &1, %r8
pushw     %r8
pushw     &CBUFS
call      &5, monitor
call      &3, main

# get prog name for profiling
# get text size in bytes
# range/#buckets ~= scalefactor
# fails if #buckets rounded up
# add in entry counts and header and
# round to word boundary

# get space

# start profiling

# monitor wants # of shorts in buff

# start user program

```

```
        pushw    %r0
        call     &1, exit
.nospace:
        pushw    &2                                # write error message and exit
        pushaw   .emesg
        pushaw   MESSL
        call     &3, write
        pushw    &-1
        ost      &EXIT
.eprol:
        .data
        .align   4
.ostargs:
        .word     WREPSW
        .word     EMASK
        .word     NEWEPSW
environ:
        .word     0
.emesg:
#       "No space for monitor buffer\n"
        .byte     78,111,32,115,112,97,99,101,32,102,111,114
        .byte     32,109,111,110,105,116,111,114
        .byte     32,98,117,102,102,101,114,10
        .set      MESSL, .-.emesg
        .byte     0
```