#ifdef \_WIN32

#include "getopt.h"

#ifdef \_\_cplusplus

extern "C" {

#endif

#ifdef REPLACE\_GETOPT

int opterr = 1; /\* if error message should be printed \*/

int optind = 1; /\* index into parent argv vector \*/

int optopt = '?'; /\* character checked for validity \*/

#undef optreset /\* see getopt.h \*/

#define optreset \_\_mingw\_optreset

int optreset; /\* reset getopt \*/

char\* optarg; /\* argument associated with option \*/

#endif

static void

\_vwarnx(const char\* fmt, va\_list ap)

{

(void)fprintf(stderr, "%s: ", \_\_progname);

if (fmt != NULL)

(void)vfprintf(stderr, fmt, ap);

(void)fprintf(stderr, "\n");

}

static void

warnx(const char\* fmt, ...)

{

va\_list ap;

va\_start(ap, fmt);

\_vwarnx(fmt, ap);

va\_end(ap);

}

/\*

\* Compute the greatest common divisor of a and b.

\*/

static int

gcd(int a, int b)

{

int c;

c = a % b;

while (c != 0) {

a = b;

b = c;

c = a % b;

}

return (b);

}

/\*

\* Exchange the block from nonopt\_start to nonopt\_end with the block

\* from nonopt\_end to opt\_end (keeping the same order of arguments

\* in each block).

\*/

static void

permute\_args(int panonopt\_start, int panonopt\_end, int opt\_end,

char\* const\* nargv)

{

int cstart, cyclelen, i, j, ncycle, nnonopts, nopts, pos;

char\* swap;

/\*

\* compute lengths of blocks and number and size of cycles

\*/

nnonopts = panonopt\_end - panonopt\_start;

nopts = opt\_end - panonopt\_end;

ncycle = gcd(nnonopts, nopts);

cyclelen = (opt\_end - panonopt\_start) / ncycle;

for (i = 0; i < ncycle; i++) {

cstart = panonopt\_end + i;

pos = cstart;

for (j = 0; j < cyclelen; j++) {

if (pos >= panonopt\_end)

pos -= nnonopts;

else

pos += nopts;

swap = nargv[pos];

/\* LINTED const cast \*/

((char\*\*)nargv)[pos] = nargv[cstart];

/\* LINTED const cast \*/

((char\*\*)nargv)[cstart] = swap;

}

}

}

#ifdef REPLACE\_GETOPT

/\*

\* getopt --

\* Parse argc/argv argument vector.

\*

\* [eventually this will replace the BSD getopt]

\*/

int getopt(int nargc, char\* const\* nargv, const char\* options)

{

/\*

\* We don't pass FLAG\_PERMUTE to getopt\_internal() since

\* the BSD getopt(3) (unlike GNU) has never done this.

\*

\* Furthermore, since many privileged programs call getopt()

\* before dropping privileges it makes sense to keep things

\* as simple (and bug-free) as possible.

\*/

return (getopt\_internal(nargc, nargv, options, NULL, NULL, 0));

}

#endif /\* REPLACE\_GETOPT \*/

//extern int getopt(int nargc, char \* const \*nargv, const char \*options);

#ifdef \_\_cplusplus

}

#endif

/\*

\* POSIX requires the `getopt' API to be specified in `unistd.h';

\* thus, `unistd.h' includes this header. However, we do not want

\* to expose the `getopt\_long' or `getopt\_long\_only' APIs, when

\* included in this manner. Thus, close the standard \_\_GETOPT\_H\_\_

\* declarations block, and open an additional \_\_GETOPT\_LONG\_H\_\_

\* specific block, only when \*not\* \_\_UNISTD\_H\_SOURCED\_\_, in which

\* to declare the extended API.

\*/

#ifdef \_\_cplusplus

extern "C" {

#endif

struct option /\* specification for a long form option... \*/

{

const char\* name; /\* option name, without leading hyphens \*/

int has\_arg; /\* does it take an argument? \*/

int\* flag; /\* where to save its status, or NULL \*/

int val; /\* its associated status value \*/

};

enum /\* permitted values for its `has\_arg' field... \*/

{

no\_argument = 0, /\* option never takes an argument \*/

required\_argument, /\* option always requires an argument \*/

optional\_argument /\* option may take an argument \*/

};

/\*

\* parse\_long\_options --

\* Parse long options in argc/argv argument vector.

\* Returns -1 if short\_too is set and the option does not match long\_options.

\*/

static int

parse\_long\_options(char\* const\* nargv, const char\* options,

const struct option\* long\_options, int\* idx, int short\_too)

{

char \*current\_argv, \*has\_equal;

size\_t current\_argv\_len;

int i, ambiguous, match;

#define IDENTICAL\_INTERPRETATION(\_x, \_y) \

(long\_options[(\_x)].has\_arg == long\_options[(\_y)].has\_arg && long\_options[(\_x)].flag == long\_options[(\_y)].flag && long\_options[(\_x)].val == long\_options[(\_y)].val)

current\_argv = place;

match = -1;

ambiguous = 0;

optind++;

if ((has\_equal = strchr(current\_argv, '=')) != NULL) {

/\* argument found (--option=arg) \*/

current\_argv\_len = has\_equal - current\_argv;

has\_equal++;

} else

current\_argv\_len = strlen(current\_argv);

for (i = 0; long\_options[i].name; i++) {

/\* find matching long option \*/

if (strncmp(current\_argv, long\_options[i].name,

current\_argv\_len))

continue;

if (strlen(long\_options[i].name) == current\_argv\_len) {

/\* exact match \*/

match = i;

ambiguous = 0;

break;

}

/\*

\* If this is a known short option, don't allow

\* a partial match of a single character.

\*/

if (short\_too && current\_argv\_len == 1)

continue;

if (match == -1) /\* partial match \*/

match = i;

else if (!IDENTICAL\_INTERPRETATION(i, match))

ambiguous = 1;

}

if (ambiguous) {

/\* ambiguous abbreviation \*/

if (PRINT\_ERROR)

warnx(ambig, (int)current\_argv\_len,

current\_argv);

optopt = 0;

return (BADCH);

}

if (match != -1) { /\* option found \*/

if (long\_options[match].has\_arg == no\_argument

&& has\_equal) {

if (PRINT\_ERROR)

warnx(noarg, (int)current\_argv\_len,

current\_argv);

/\*

\* XXX: GNU sets optopt to val regardless of flag

\*/

if (long\_options[match].flag == NULL)

optopt = long\_options[match].val;

else

optopt = 0;

return (BADARG);

}

if (long\_options[match].has\_arg == required\_argument || long\_options[match].has\_arg == optional\_argument) {

if (has\_equal)

optarg = has\_equal;

else if (long\_options[match].has\_arg == required\_argument) {

/\*

\* optional argument doesn't use next nargv

\*/

optarg = nargv[optind++];

}

}

if ((long\_options[match].has\_arg == required\_argument)

&& (optarg == NULL)) {

/\*

\* Missing argument; leading ':' indicates no error

\* should be generated.

\*/

if (PRINT\_ERROR)

warnx(recargstring,

current\_argv);

/\*

\* XXX: GNU sets optopt to val regardless of flag

\*/

if (long\_options[match].flag == NULL)

optopt = long\_options[match].val;

else

optopt = 0;

--optind;

return (BADARG);

}

} else { /\* unknown option \*/

if (short\_too) {

--optind;

return (-1);

}

if (PRINT\_ERROR)

warnx(illoptstring, current\_argv);

optopt = 0;

return (BADCH);

}

if (idx)

\*idx = match;

if (long\_options[match].flag) {

\*long\_options[match].flag = long\_options[match].val;

return (0);

} else

return (long\_options[match].val);

#undef IDENTICAL\_INTERPRETATION

}

/\*

\* getopt\_internal --

\* Parse argc/argv argument vector. Called by user level routines.

\*/

static int

getopt\_internal(int nargc, char\* const\* nargv, const char\* options,

const struct option\* long\_options, int\* idx, int flags)

{

char\* oli; /\* option letter list index \*/

int optchar, short\_too;

static int posixly\_correct = -1;

if (options == NULL)

return (-1);

/\*

\* XXX Some GNU programs (like cvs) set optind to 0 instead of

\* XXX using optreset. Work around this braindamage.

\*/

if (optind == 0)

optind = optreset = 1;

/\*

\* Disable GNU extensions if POSIXLY\_CORRECT is set or options

\* string begins with a '+'.

\*

\* CV, 2009-12-14: Check POSIXLY\_CORRECT anew if optind == 0 or

\* optreset != 0 for GNU compatibility.

\*/

if (posixly\_correct == -1 || optreset != 0)

posixly\_correct = (getenv("POSIXLY\_CORRECT") != NULL);

if (\*options == '-')

flags |= FLAG\_ALLARGS;

else if (posixly\_correct || \*options == '+')

flags &= ~FLAG\_PERMUTE;

if (\*options == '+' || \*options == '-')

options++;

optarg = NULL;

if (optreset)

nonopt\_start = nonopt\_end = -1;

start:

if (optreset || !\*place) { /\* update scanning pointer \*/

optreset = 0;

if (optind >= nargc) { /\* end of argument vector \*/

place = EMSG;

if (nonopt\_end != -1) {

/\* do permutation, if we have to \*/

permute\_args(nonopt\_start, nonopt\_end,

optind, nargv);

optind -= nonopt\_end - nonopt\_start;

} else if (nonopt\_start != -1) {

/\*

\* If we skipped non-options, set optind

\* to the first of them.

\*/

optind = nonopt\_start;

}

nonopt\_start = nonopt\_end = -1;

return (-1);

}

if (\*(place = nargv[optind]) != '-' || (place[1] == '\0' && strchr(options, '-') == NULL)) {

place = EMSG; /\* found non-option \*/

if (flags & FLAG\_ALLARGS) {

/\*

\* GNU extension:

\* return non-option as argument to option 1

\*/

optarg = nargv[optind++];

return (INORDER);

}

if (!(flags & FLAG\_PERMUTE)) {

/\*

\* If no permutation wanted, stop parsing

\* at first non-option.

\*/

return (-1);

}

/\* do permutation \*/

if (nonopt\_start == -1)

nonopt\_start = optind;

else if (nonopt\_end != -1) {

permute\_args(nonopt\_start, nonopt\_end,

optind, nargv);

nonopt\_start = optind - (nonopt\_end - nonopt\_start);

nonopt\_end = -1;

}

optind++;

/\* process next argument \*/

goto start;

}

if (nonopt\_start != -1 && nonopt\_end == -1)

nonopt\_end = optind;

/\*

\* If we have "-" do nothing, if "--" we are done.

\*/

if (place[1] != '\0' && \*++place == '-' && place[1] == '\0') {

optind++;

place = EMSG;

/\*

\* We found an option (--), so if we skipped

\* non-options, we have to permute.

\*/

if (nonopt\_end != -1) {

permute\_args(nonopt\_start, nonopt\_end,

optind, nargv);

optind -= nonopt\_end - nonopt\_start;

}

nonopt\_start = nonopt\_end = -1;

return (-1);

}

}

/\*

\* Check long options if:

\* 1) we were passed some

\* 2) the arg is not just "-"

\* 3) either the arg starts with -- we are getopt\_long\_only()

\*/

if (long\_options != NULL && place != nargv[optind] && (\*place == '-' || (flags & FLAG\_LONGONLY))) {

short\_too = 0;

if (\*place == '-')

place++; /\* --foo long option \*/

else if (\*place != ':' && strchr(options, \*place) != NULL)

short\_too = 1; /\* could be short option too \*/

optchar = parse\_long\_options(nargv, options, long\_options,

idx, short\_too);

if (optchar != -1) {

place = EMSG;

return (optchar);

}

}

if ((optchar = (int)\*place++) == (int)':' || (optchar == (int)'-' && \*place != '\0') || (oli = (char\*)strchr(options, optchar)) == NULL) {

/\*

\* If the user specified "-" and '-' isn't listed in

\* options, return -1 (non-option) as per POSIX.

\* Otherwise, it is an unknown option character (or ':').

\*/

if (optchar == (int)'-' && \*place == '\0')

return (-1);

if (!\*place)

++optind;

if (PRINT\_ERROR)

warnx(illoptchar, optchar);

optopt = optchar;

return (BADCH);

}

if (long\_options != NULL && optchar == 'W' && oli[1] == ';') {

/\* -W long-option \*/

if (\*place) /\* no space \*/

/\* NOTHING \*/;

else if (++optind >= nargc) { /\* no arg \*/

place = EMSG;

if (PRINT\_ERROR)

warnx(recargchar, optchar);

optopt = optchar;

return (BADARG);

} else /\* white space \*/

place = nargv[optind];

optchar = parse\_long\_options(nargv, options, long\_options,

idx, 0);

place = EMSG;

return (optchar);

}

if (\*++oli != ':') { /\* doesn't take argument \*/

if (!\*place)

++optind;

} else { /\* takes (optional) argument \*/

optarg = NULL;

if (\*place) /\* no white space \*/

optarg = place;

else if (oli[1] != ':') { /\* arg not optional \*/

if (++optind >= nargc) { /\* no arg \*/

place = EMSG;

if (PRINT\_ERROR)

warnx(recargchar, optchar);

optopt = optchar;

return (BADARG);

} else

optarg = nargv[optind];

}

place = EMSG;

++optind;

}

/\* dump back option letter \*/

return (optchar);

}

/\*

\* getopt\_long --

\* Parse argc/argv argument vector.

\*/

int getopt\_long(int nargc, char\* const\* nargv, const char\* options,

const struct option\* long\_options, int\* idx)

{

return (getopt\_internal(nargc, nargv, options, long\_options, idx,

FLAG\_PERMUTE));

}

/\*

\* getopt\_long\_only --

\* Parse argc/argv argument vector.

\*/

int getopt\_long\_only(int nargc, char\* const\* nargv, const char\* options,

const struct option\* long\_options, int\* idx)

{

return (getopt\_internal(nargc, nargv, options, long\_options, idx,

FLAG\_PERMUTE | FLAG\_LONGONLY));

}

//extern int getopt\_long(int nargc, char \* const \*nargv, const char \*options,

// const struct option \*long\_options, int \*idx);

//extern int getopt\_long\_only(int nargc, char \* const \*nargv, const char \*options,

// const struct option \*long\_options, int \*idx);

/\*

\* Previous MinGW implementation had...

\*/

#ifdef \_\_cplusplus

}

#endif

#endif