NAME

cc, pc, m2 − Minix C, Pascal, and Modula-2 compilers

SYNOPSIS

cc|**pc**|**m2** [**−D** *name*[=*value*]] ... [**−U** *name*] ... [**−I** *directory*] ... [**−.***suffix*] ... [**−c**] [**−E**] [**−P**] [**−S**] [**−c.***suffix*] [**−O**] [**−O***level*] [**−OS**] [**−OT**] [**−g**] [**−n**] [**−a**] [**−R**] [**−A**] [**−s**] [**−fsoft**] [**−fnone**] [**−w**] [**−wo**] [**−ws**] [**−wa**] [**−3**] [**−_**] [**−W***name−option*] ... [**−m***arch*] [**−o** *outfile*] [**−L** *directory*] ... [**−i**] [**−sep**] [**−com**] [**−r**] [**−stack** *size*] *operand* ...

(Minix-86 subset:)

cc|**pc**|**m2** [**−D***name*[=*value*]] ... [**−U***name*] ... [**−I***directory*] ... [**−.o**] ... [**−c**] [**−E**] [**−P**] [**−S**] [**−c.***suffix*] [**−O**] [**−O***level*] [**−n**] [**−a**] [**−R**] [**−A**] [**−s**] [**−f**] [**−w**] [**−wo**] [**−ws**] [**−wa**] [**−3**] [**−_**] ... [**−m**] [**−o** *outfile*] [**−L***directory*] ... [**−i**] [**−sep**] [**−com**] *operand* ...

DESCRIPTION

Cc, **pc**, and **m2** are the call names of the Minix C, Pascal, and Modula-2 compilers from the Amsterdam Compiler Kit (ACK).

All these call names are links to the **acd** driver program. **Acd** uses the driver description file **/usr/lib/descr** that describes the steps necessary to compile a source file. The **acd**(1) manual page describes a few more flags, like **−v**, that may be useful for debugging compiler problems.

Minix-86 uses a C program as the compiler driver. This driver is not as flexible as the one implemented with the **acd** driver, and offers a smaller number of options. The second line of the synopsis above shows the options that the Minix-86 driver supports. The rest of this manual page is geared towards the **acd** driver. People writing software for Minix-86, or that should be portable to all Minix versions should stick to the options listed under the Minix-86 compiler.

OPTIONS

The transformations done by the compiler are modified by the following options. They are a superset of the options required by POSIX, with the Minix or compiler specific ones are marked as such. Options for one specific compiler are ignored for others. Read the OPTIONS section of **acd**(1) for the driver specific options.

−D *name*[=*value*]
: Same as if **#define** *name value* had been given. **1** is assumed if *value* is omitted. This argument, like all the other double arguments, may also be given as a single argument. (I.e. either as **−D** *name* or **−D***name*.) (The Minix-86 driver is not so flexible, the proper form can be seen in the synopsis.)

−U *name*
: Undefine the pre-defined symbol *name*.

−I *directory*
: Extend the include directory path with the given directory. These directories are searched for include files in the given order before the standard places. The standard place for the C compiler is **/usr/include**, and for the Modula-2 compiler it is **/usr/lib/m2**.

−.*suffix*
: Act as if a source file with the given suffix is present on the command line. For each language found on the command line the appropriate libraries are selected. The first language mentioned selects the runtime startoff. The call name of the driver also chooses the language, so **cc** is an implicit **−.c**. The runtime startoff can be omitted by specifying **−.o** for those rare cases where you want to supply your own startoff. (Minix)

−c
: Transform the input files to object files and stop. The **−o** option may be used under Minix to set the name of the object file. **Make**(1) likes this, because **cc −c** *dir/file***.c** puts *file***.o** in the current directory, but **cc −c** *dir/file***.c −o** *dir/file***.o** puts the **.o** file where **make** expects it to be by its builtin **.c.o** rule. (Minix-86 can only use **−o** to name an executable.)

−E
: Run the preprocessor over the input files and send the result to standard output or the file named by **−o**. Standard input is read if an input file is named "**−**".

−P Run the preprocessor over the input files and put the result to files with the suffix **.i**. File and line number information is omitted from the output. Use **−P −E** under Minix to omit this info for **−E** too.

−S Transform the input files to assembly files with suffix **.s**.

−c.*suffix*
Transform the input files to files with the given suffix. This can only succeed if there is a valid transformation from the input file to the given suffix. The same goes for **−c** and other options that are just special cases of this option, except for **−P**, **−c.i** keeps the line number info. The option **−c.a** makes the driver transform the input files to object files and add them to a library. (So you do not need to know how the archiver works.) Note that you need to give object files as arguments if you want to replace old object files. Transformed files are added under a (unique) temporary name. With **−o** you can name the library. (Minix) (Minix-86 can't do **−c.a**.)

−O Optimize code. This option is a no-op, because all the compilers already use the **−O1** optimization level to get code of reasonable quality. Use **−O0** to turn off optimization to speed up compilation at debug time.

−O*level* Compile with the given optimization level. (Minix)

−OS
−OT
Optimize for space or for time. (Minix)

−g Compile the C source with debugging information. (The way **−g**, **−s** and **−O** interact is left unspecified.)

−n Omit the file and line number tracking that is used for runtime error reports from Pascal or Modula-2 programs. The **−n** flag is normally used to compile library modules, but may also be useful to make a program smaller and faster once debugged. (Pascal & Modula-2)

−a Enable assertions, i.e. statements of the form **assert** *test* that cause a descriptive runtime error if the boolean expression *test* evaluates false. (Pascal & Modula-2)

−R Disable runtime checks like overflow checking. (Pascal & Modula-2)

−A Enable array bound checks. (Pascal & Modula-2)

−s Strip the resulting executable of its symbol table.

−fsoft
−f
Use software floating point instead of hardware floating point. This is a loader flag, but in general it is best to specify this flag in all phases of the compilation. (Minix)

−fnone Ignored. Used under Minix-vmd to omit floating point printing/scanning code. The standard Minix compiler figures this out automatically using a special loader trick. (Minix)

−w Do not produce warnings about dubious C language constructs. Normally the compiler is configured to do the maximum amount of checking without being too annoying. (Minix)

−wo Omit warnings about old (K&R) style. (Minix)

−ws Omit strict warnings. (Minix)

−wa Omit all warnings. (Minix)

−3 Only accept 3rd edition Modula-2. (Modula-2)

−_ Allow underscores in Pascal or Modula-2 identifiers, but not at the beginning of an identifier. (Pascal & Modula-2)

−W*name−option*
If *name* is the name of the compiler this driver is working for, then *option* is activated for that compiler. See below for a per-compiler list. Any other **−W** option is ignored. (**−W** is

described by POSIX as an optional flag to send options to the different compiler passes with a totally different (and nicely ignored) syntax as described here.) (Minix-86 ignores any **−W** flag.)

−m Under Minix-86 this option transforms the function declarations (prototypes) to the old K&R form, i.e. the arguments declarations are removed. This saves a lot of memory in the compiler and may allow a large program to be compiled. One must make sure that function arguments are properly type-cast where necessary. (Minix)

−m*arch* Set the target architecture for a cross compiler. Normally the compiler produces code for the same architecture it itself is compiled for. The **ARCH** environment variable may also be used to set the architecture. Architectures names are: **i86** (Intel 8086 and 286), **i386** (Intel 386, 486, ...), **m68000** (Motorola MC68000 & MC68010, 16-bit ints), **m68010** (Motorola MC68000 & MC68010, 32-bit ints), **m68020** (Motorola MC68020, 32-bit ints), **sparc** (Sun SPARC). (Minix) (Ignored under Minix-86.)

−o *outfile*
Set the output file for the **−c**, **−c.a**, and **−E** options, or choose the executable name instead of the default **a.out**. (Minix-86 can only choose the executable name.)

−L *directory*
Extend the library search path with *directory*. These directories are searched for libraries named by **−l** in the given order before the standard places. The standard places are **/lib/***arch*, and **/usr/lib/***arch*. The search for libaries in directories added with **−L** looks in *directory*/*arch* and *directory* itself. (*Arch* is the machine architecture name. This is Minix dependent, compilers on other systems usually only look in *directory*.) (Minix-86 only has **/lib** and **/usr/lib** as the standard places.)

−sep
−com
Create a Separate I&D or a common I&D executable. The text segment of a separate I&D executable is read-only and shareable. For an **i86** binary this also means that the text and data segment can each be 64 kilobytes large instead of just 64 kilobytes together. Separate I&D is the default. Common I&D is probably only useful for the bootstraps. The **−i** option has the same meaning as **−sep**, but should no longer be used. (Minix)

−r Makes the loader produce a relocatable object file, i.e. a file that may be loaded again. The runtime startoff and the default libraries are omitted, only the files mentioned are combined. (Minix)

−stack *size*
Allow the process *size* bytes of heap and stack. *Size* is a C-style decimal, octal, or hexadecimal number, optionally followed by the multipliers **m**, **k**, **w**, and **b** for mega (1024*1024), kilo (1024), "word" (2 or 4), and byte (1). Uppercase letters are accepted too. A size of **32kw** is used by default, translating to 64k for **i86**, and 132k for other architectures. Too large a size is rounded down to keep the data segment within 64 kilobytes for the **i86**. (Minix)

OPERANDS

All leftover operands are treated as files to be compiled, with one exception. The construct **−l** *library* is used to denote a library, usually **lib***library***.a**, that is to be searched in the directories mentioned with **−L** or the standard places. These libraries keep their place among the (transformed) input files when presented to the loader. (It is a common mistake to write **cc −lcurses x.c** instead of **cc x.c −lcurses**.)

IMPLEMENTATION

The Minix compiler implementation uses the ACK compilers adapted for use under Minix as described below. Read **ACK**(7) for more detailed information on the ACK compilers themselves.

Feature test macros

The preprocessors are given these arguments to define feature test macros: **−D__ACK__** tells what compiler is used. **−D__minix** tells that this is Minix. **−D__***arch* tells the architecture. (More macros are defined, but they are only to be used in the include files.)

The symbols above are predefined by the preprocessor so that your program is able to "sense" the environment it is in. It is also possible for your program to do the opposite, to tell what kind of environment it likes to have. By default, **cc** compiles a standard C program. If you want the extensions described in POSIX.1 to become visible, then you have to set **_POSIX_SOURCE** to **1** at the start of your program. To enable UNIX or Minix extensions you need to also set **_MINIX** to **1**. If you don't want to clutter your source files with these symbols then you can use **cc −D_MINIX −D_POSIX_SOURCE** to get the POSIX.1 and the Minix extensions.

Preprocessing

Pascal, Modula-2, EM source (see below), and Assembly source are preprocessed by the C preprocessor if the very first character in the file is a '**#**' character.

Assembly dialects

No two compilers use the same assembly language. To be able to use the same assembly dialect for the low level support routines an assembly converter is provided. The input of this converter can be of type **ack**, **ncc**, or **bas**, and the output can be of type **ack**, **ncc**, or **gnu**. The suffix of the file tells the assembly dialect (see below), or one can use the option **−Was−***dialect* to tell the driver what the dialect of a plain **.s** file is. The assembly converter is not as smart as the assembler, the translation is more or less a text substitution. It leaves a lot of checking to the target assembler. You have to restrict yourself to a subset that is understood by both assemblers. The ACK assembler for instance doesn't care if you use 'ax' or 'eax' for a 32 bit register, it looks at the instruction type. The GNU assembler doesn't like this, so you have to use the proper register name in ACK assembly that is to be translated to GNU assembly. Expressions are converted as is, even if the operator precedence rules of the two assembly languages differ. So use parentheses. The converter does promise one thing: compiler output can be properly translated. (Note that under Minix-86 **−W** is ignored. All assembly should therefore be in the "ncc" dialect.)

FILES

/usr/lib/descr
The compiler description file.

.c Suffix of a C source file.

.mod Modula-2.

.p Pascal.

.i Preprocessed C source.

.k ACK machine independent compact EM code produced by the C, Pascal, or Modula-2 front end (or any other ACK front end.) The ACK compilers are based on the UNCOL idea where several front ends compile to a common intermediate language, and several back ends transform the intermediate language to the target machine language. The ACK intermediate language is named "EM".

.m Peephole optimized EM.

.gk Result of the (optional) EM global optimizer.

.g Result of the second EM peephole optimizer used after the global optimizer.

.e Human readable EM. (Human created or decoded compact EM.)

.s Target machine assembly. (Current compiler dialect.)

.ack.s ACK assembly.

.ncc.s ACK Xenix style assembly. This dialect is used by the 16 bit ACK ANSI C compiler.

.gnu.s GNU assembly.

.bas.s BCC assembly. (Used by the Bruce Evans' BCC compiler, for many years the compiler for Minix-386.)

.o Object code.

.a Object code library.

a.out Default output executable.

SEE ALSO

acd(1), **ACK**(7).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)