NAME

README.mint - Perl version 5 on Atari MiNT

DESCRIPTION

There is a binary version of perl available from the FreeMiNT project http://freemint.de/ You may wish to use this instead of trying to compile yourself.

The following advice is from perl 5.004_02 and is probably rather out of date.

If you want to build perl yourself on MiNT (or maybe on an Atari without MiNT) you may want to accept some advice from somebody who already did it...

There was a perl port for Atari ST done by ++jrb bammi@cadence.com. This port tried very hard to build on non-MiNT-systems. For the sake of efficiency I've left this way. Yet, I haven't removed bammi's patches but left them intact. Unfortunately some of the files that bammi contributed to the perl distribution seem to have vanished?

So, how can you distinguish my patches from bammi's patches? All of bammi's stuff is embedded in "#ifdef atarist" preprocessor macros. My MiNT port uses "#ifdef __MINT__" instead (and unconditionally undefines "atarist". If you want to continue on bammi's port, all you have to do is to swap the "-D" and "-U" switches for "__MINT__" and "atarist" in the variable ccflags.

However, I think that my version will still run on non-MiNT-systems provided that the user has a Eunuchs-like environment (i.e. the standard envariables like $PATH, $HOME, ... are set, there is a POSIX compliant shell in /bin/sh, and...)

Known problems with Perl on MiNT

The problems you may encounter when building perl on your machine are most probably due to deficiencies in MiNT resp. the Atari platform in general.

First of all, if you have less than 8 MB of RAM you shouldn't even try to build Perl yourself. Better grab a binary pre-compiled version somewhere. Even if you have more memory you should take some care. Try to run in a fresh environment (without memory fragmented too much) with as few daemons, accessories, xcontrol modules etc. as possible. If you run some AES you should consider to start a console based environment instead.

A problem has been reported with sed. Sed is used to create some configuration files based on the answers you have given to the Configure script. Unfortunately the Perl Configure script shows sed on MiNT its limits. I have sed 2.05 with a stacksize of 64k and I have encountered no problems. If sed crashes during your configuration process you should first try to augment sed's stacksize:

```
 fixstk 64k /usr/bin/sed
```

(or similar). If it still doesn't help you may have a look which other versions of sed are installed on your system. If you have a KGMD 1.0 installation you will find three in /usr/bin. Have a look there.

Perl has some "mammut" C files. If gcc reports "internal compiler error: program cc1 got fatal signal 10" this is very likely due to a stack overflow in program cc1. Find cc1 and fix its stack. I have made good experiences with

```
 fixstk 2 cc1
```

This doesn't establish a stack of 2 Bytes only as you might think. It really reserves one half of the available memory for cc1's stack. A setting of 1 would reserve the entire memory for cc1, 3 would reserve three fourths. You will have to find out the value that suits to your system yourself.

To find out the location of the program "cc1" simply type `gcc --print-prog-name cc1' at your shell prompt.

Now run make (maybe "make -k"). If you get a fatal signal 10 increase cc1's stacksize, if you run out of memory you should either decrease the stacksize or follow some more hints:

Perl's building process is very handy on machines with a lot of virtual memory but may result in a disaster if you are short of memory. If gcc fails to compile many source files you should reduce the optimization. Grep for "optimize" in the file config.sh and change the flags.

If only several huge files cause problems (actually it is not a matter of the file size resp. the amount of code but depends on the size of the individual functions) it is useful to bypass the make program and compile these files directly from the command line. For example if you got something like the following from make:

```
 CCCMD = gcc -DPERL_CORE ....
 ...
 ...: virtual memory exhausted
```

you should hack into the shell:

```
 gcc -DPERL_CORE ... toke.c
```

Please note that you have to add the name of the source file (here toke.c) at the end.

If none of this helps, you're helpless. Wait for a binary release. If you have succeeded you may encounter another problem at the linking process. If gcc complains that it can't find some libraries within the perl distribution you probably have an old linker. If it complains for example about "file not found for xxx.olb" you should cd into the directory in question and

```
 ln -s libxxx.a xxx.olb
```

This will fix the problem.

This version (5.00402) of perl has passed most of the tests on my system:

```
 Failed Test  Status Wstat Total Fail  Failed  List of failed

---------------------------------------------------------------------------
 io/pipe.t                    10    2  20.00%  7, 9
 io/tell.t                    13    1   7.69%  12
 lib/complex.t               762   13   1.71%  84-85, 248-251, 257,
272-273,
                                               371, 380, 419-420
 lib/io_pipe.t                10    1  10.00%  9
 lib/io_tell.t                13    1   7.69%  12
 op/magic.t                   30    2   6.67%  29-30
 Failed 6/152 test scripts, 96.05% okay. 20/4359 subtests failed, 99.54%
okay.
```

Pipes always cause problems with MiNT, it's actually a surprise that most of the tests did work. I've got no idea why the "tell" test failed, this shouldn't mean too big a problem however.

Most of the failures of lib/complex seem to be harmless, actually errors far right to the decimal point... Two failures seem to be serious: The sign of the results is reversed. I would say that this is due to minor bugs in the portable math lib that I compiled perl with.

I haven't bothered very much to find the reason for the failures with op/magic.t and op/stat.t. Maybe you'll find it out.

##

Another possible problem may arise from the implementation of the "pwd" command. It happened to add a carriage return and newline to its output no matter what the setting of $UNIXMODE is. This is quite annoying since many library modules for perl take the output of pwd, chop off the trailing newline character and then expect to see a valid path in that. But the carriage return (last but second character!) isn't chopped off. You can either try to patch all library modules (at the price of performance for the extra transformation) or you can use my version of pwd that doesn't suffer from this deficiency.

The fixed implementation is in the mint subdirectory. Running "Configure" will attempt to build and install it if necessary (hints/mint.sh will do this work) but you can build and install it explicitly by:

```
cd mint
make install
```

This is the fastest solution.

Just in case you want to go the hard way: perl won't even build with a broken pwd! You will have to fix the library modules (ext/POSIX/POSIX.pm, lib/Cwd.pm, lib/pwd.pl) at last after building miniperl.

A major nuisance of current MiNTLib versions is the implementation of system() which is far from being POSIX compliant. A real system() should fork and then exec /bin/sh with its argument as a command line to the shell. The MiNTLib system() however doesn't expect that every user has a POSIX shell in /bin/sh. It tries to work around the problem by forking and exec'ing the first token in its argument string. To get a little bit of compliance to POSIX system() it tries to handle at least redirection ("<" or ">") on its own behalf.

This isn't a good idea since many programs expect that they can pass a command line to system() that exploits all features of a POSIX shell. If you use the MiNTLib version of system() with perl the Perl function system() will suffer from the same deficiencies.

You will find a fixed version of system() in the mint subdirectory. You can easily insert this version into your system libc:

```
cd mint
make system.o
ar r /usr/lib/libc.a
ranlib /usr/lib/libc.a
```

If you are suspicious you should either back up your libc before or extract the original system.o from your libc with "ar x /usr/lib/libc.a system.o". You can then backup the system.o module somewhere before you succeed.

Anything missing? Yep, I've almost forgotten... No file in this distribution without a fine saying. Take this one:

```
"From a thief you should learn: (1) to work at night;
(2) if one cannot gain what one wants in one night to
try again the next night; (3) to love one's coworkers
just as thieves love each other; (4) to be willing to
risk one's life even for a little thing; (5) not to
attach too much value to things even though one has
risked one's life for them - just as a thief will resell
a stolen article for a fraction of its real value;
(6) to withstand all kinds of beatings and tortures
but to remain what you are; and (7) to believe your
work is worthwhile and not be willing to change it."
```

```
    -- Rabbi Dov Baer, Maggid of Mezeritch
```

OK, this was my motto while working on Perl for MiNT, especially rule (1)...

Have fun with Perl!

AUTHOR

Guido Flohr

```
 mailto:guido@FreeMiNT.de
```