VirtualCpm
CP/M Emulator
January 20, 2023

Table of Contents

Introduction

VirtualCpm is a JAVA program that emulates a CP/M CCP, BDOS, and BIOS running on a Z80 with 64K RAM, with CP/M 3 extensions. It is normally used to execute a single CP/M command, but is capable of handling “submit” files as well as native (on the host PC) text files filled with commands.

Native Files

Native files must have lower-case only names. Mixed-case filenames will cause unpredictable results. All files created by CP/M programs will be in lower-case.

The file’s write permission is used to reflect the CP/M RO attribute. CP/M programs that change a file’s RO attribute will change the native file’s write permission.

The file’s execute permission is used to reflect the CP/M SYS attribute. Note that Windows will always show files as executable, and thus files on a Windows host will always have the SYS attribute set. Also remember that CP/M normally hides files that have the SYS attribute set. There is a server configuration setting that disables the SYS attribute, to avoid these issues on Windows hosts. Note that some non-Windows systems may set file execute bits incorrectly, such as ZIP archive extractions or files copied from a Windows system. On non-windows systems the execute permissions can be removed from CP/M files that should not be SYS.

The CP/M ARCHIVE attribute is not supported.

Files that are not an even multiple of 128 bytes in size will be padded to a 128-byte multiple, using Ctrl-Z (EOF, 0x1a), when reading. Writing to a file from CP/M always involves a full 128-byte record, so no additional padding is performed. The CP/M 3 feature “Set File Byte Count” will truncate a file to a specific, arbitrary, number of bytes, after which the file may no longer be an even multiple of 128 bytes.

Configuration

The emulator is configured using a “configuration file”, which is plan text formatted as “property = value” lines. The configuration file to be used is based on the following search order. First, it will look in the current directory for “**vcpm.rc**” and second in the user’s home directory for “**.vcpmrc**”. Comments in the file start with ‘#’ as the first non-space character of the line. Property values may use backslash (‘\’) to extend long values to the following line.

The following properties are recognized:

vcpm_dso = *string*
: Specifies the Drive Search Order used to locate commands. Accepts drive letters and the keyword “def”, separated by commas. Only four items are allowed. Default is “def,a”.

silent
: Suppresses informative messages about configuration, etc.

vcpm_root_dir = *path*
: Specifies the top-level (root) directory to be exported to CP/NET. Subdirectories named “a” through “p” are assumed, but not created automatically. Default will be **~/HostFileBdos**. Note that JAVA Properties does not recognize the ‘~’ syntax, so the absolute path must be used.

vcpm_nosys
: Disable the CP/M SYS attribute, so files will not be hidden on Windows.

vcpm_drive_*X* **=** *path*
: Where “*X*” is one of “a” through “p”. Specify the path to use instead of “root_dir/X” for the CP/M drive.

vcpm_trace = *command*
: Enables program (instruction) tracing based on *command*. See section “Tracing”.

Environment Variables

The following environment variables are recognized:

CPMDrives = *path-list*
: The *path-list* is a list of (up to) 16 paths separated by commas. Each path is positional dependent, i.e. the first path is for drive A:, second for drive B:, etc. If a path is empty (“,,”), then the default is used. All non-empty paths are translated into **vcpm_drive_***X* properties.

CPMDrive_*X* = *path*
: Specifies the path to use for drive *X*. This is translated into a **vcpm_drive_***X* properties.

CPMDefault = *usr-drv*
: Specifies the default/initial user number and/or drive. Default is “0A:”.

VCPMCoreDump = *file*
: Triggers a core dump of RAM when exiting the emulation. The dump file contains the raw (binary) contents of RAM. Note that subsequent dumps to the same file will overwrite any previous dump. Default file name is “vcpm.core”.

VCPMTrace = *command*
: Enables program (instruction) tracing based on *command*. This overrides any **vcpm_trace** property in the config file. See section “Tracing”.

Running the Emulator

Typical startup command:

```
java -jar VirtualCpm.jar cpm-command
```

Typically, a wrapper script or batch file will be created, to simplify running commands. On Linux, this might look like:

```
#!/bin/bash
exec java -jar /path/to/VirtualCpm.jar "${@}"
```

If this script were named “vcpm”, and is on the user’s PATH, then one would run CP/M commands using:

```
vcpm cpm-command
```

Note that this emulator does not provide any I/O hardware emulations. Any programs that execute input/output instructions will cause (non-fatal) messages to be printed for each I/O. While these are non-fatal incidents to the emulator, it is likely that the CP/M program will not run correctly.

On Linux/Unix systems (or compatible shells on Windows), a command may be prefixed by environment variables, which is a convenient way to do ad-hoc tracing and coredumps. For example:

```
VCPMCoreDump=mydumpfile vcpm cpm-command
```

Console Input

Translating CP/M console input to JAVA and modern “stdin” is problematic. This emulator is optimized for whole-line input, as in CP/M BDOS function 10. Programs like DDT, ED, PIP, and LINK/L80 work fine, although because of echoing there is an extra blank line after entering input.

Batch Execution (SUBMIT)

Conventional CP/M 3 submit files may be executed as a command. In addition, a local (native) file may be used even though it does not reside on a CP/M drive. The file must be entered as a path, i.e. must contain a slash (‘/’), even if only a “./” prefix.

Each line is a command, or if prefixed with '<' a line of input to the program being run at the time. Argument substitutions are supported, using the standard "$1", "$2", … "$9" notation. Comment lines must begin with '#' or ';'. Empty lines are skipped.

There is no way to disable searching for "command.sub" when only "command" is entered, but entering "command.com" will avoid matching any submit file.

Setup Example

Here is an example set of commands used on Linux to setup vcpm. These assume no vcpm config file is present. In this example, the repository ~/cpm-dist is used as a source for CP/M distribution files. Also, the 'vcpm' script has already been copied and edited and is on the user's PATH.

```
$ mkdir -p ~/HostFileBdos/{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}
$ cp ~/cpm-dist/*.com ~/HostFileBdos/a
$ vcpm dir
Creating HostFileBdos 00 device with root dir /home/<user>/HostFileBdos
A>dir
A: DDT.COM        ED.COM         RMAC.COM       SUM.COM        DUMP.COM
A: GENCPM.COM     GENMOD.COM     Z80.LIB        DIR.COM        MATHLIB.REL
A: AS.COM         LINK.COM       PRINTF.H       PLILIB.IRL     GENHEX.COM
A: HEXCOM.COM     CLIBRARY.REL   STDLIB.REL     LIB.COM        XREF.COM
A: CCONFIG.COM    Z180.LIB       SHOW.COM       STAT.COM       GENSYS.COM
A: FPRINTF.H      M80.COM        PLI.COM        PLI0.OVL       PLI2.OVL
A: ZSID.COM       PIP.COM        MAC.COM        EDIT.COM       GENCOM.COM
A: FLIBRARY.REL   C.COM          PRINT.COM      L80.COM        ASM.COM
A: LOAD.COM
$
```

The exact list of files, and order, above depends on what files were copied into the A: directory.

Note that the "Creating HostFileBdos…" message can be suppressed by setting up a vcpm config file that contains the "silent" property. For example, "echo silent >>vcpm.rc" in the current directory will add the 'silent' property, possibly creating a new "vcpm.rc" properties file.

Makefile Example

Here is an example Makefile that uses vcpm to assemble using RMAC and link the result into an SPR file, using source files "prog1a.asm" and "prog1b.asm". This makefile and the source file reside in the same directory. The default VirtualCpm A: drive must contain RMAC.COM and LINK.COM, and any other necessary files (like Z80.LIB). Note how some of the vcpm commandline arguments are quoted, because otherwise the host shell may try to interpret them or stumble over special characters.

```
export CPMDrive_D = $(PWD)
export CPMDefault = d:

all: prog1.spr

%.rel: %.asm
        vcpm rmac "$?" '$$SZLA'
```

```
prog1.spr: prog1a.rel prog1b.rel
        vcpm link "prog1=prog1a,prog1b[os,nr]"
```

Tracing

Trace commands take the form:

range [*count*] [**oneshot**]

Where *range* has the syntax:

*low***:***high*

A range expression of only ‘:’ matches the entire address range (a missing *low* or *high* equates to the respective end of the full address range). An expression of only *low* (no ‘:’) matches a single address. The *high* address is exclusive, i.e. a match means *low* <= PC < *high*.

The *count* argument sets the number of instructions to be traced after matching the range. Note that this counter starts after “leaving” the range, i.e. all instructions within the range will always be traced.

The **oneshot** keyword causes the tracing to occur only once. Tracing is essentially disabled after being triggered (after “leaving” the range).

The *range* argument is required and must be the first.