CpnetSocketServer
October 24, 2020

Table of Contents

Introduction

CpnetSocketServer is a JAVA program the implements a CP/NET server, with CP/M 3 extensions. It listens for socket connection requests and provides CP/M-style access to native file folders on the host.

Multiple servers may be running on the same host, provided they don't conflict. They must have different root_dir, port, and sid values at a minimum.

Native Files

Native files must have lower-case only names. Mixed-case filenames will cause unpredictable results. All files created by CP/M 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 the disables the SYS attribute, to avoid these issues on Windows hosts.

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 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.

Server Configuration

The server is configured using a “configuration file”, which is plan text formatted as “property = value” lines. The configuration file to be used is specified on the commandline using the parameter “**conf=***file*”. The environment variable **CPNET_CONFIG** may also specify the configuration file. If nothing is specified, the server will look in the current directory for “**cpnetrc**” and then the user’s home directory for “**.cpnetrc**”. Property values may use backslash (‘\’) to extend long values to the following line.

Many properties may be specified on the commandline, using a “parameter=value” format. The parameter names are the property names with the “cpnetserver_” prefix removed.

The following properties are recognized:

cpnetserver_log = *log-file*
: Diverts stderr to *log-file*.

cpnetserver_log_time
: Prepends timestamp to stderr (log file) output.

cpnetserver_host = *host-or-ip*
: Specifies the host network address to listen on for connections.

cpnetserver_port = *port-number*
: Specifies the port number to use for listening.

cpnetserver_blacklist = *host-specs...*
: Whitespace-separated list of hostnames/IP-addresses that are not allowed to connect. A string beginning with a period will match all hostnames on that domain. For example, the string “.censys-scanner.com” will match (reject connections from) all hosts in the domain “censys-scanner.com”. An IP address may include a mask length, such as “45.146.164.179/24”, to match a range of addresses. This property is typically only useful for servers that listen on public interfaces.

cpnetserver_temp = *tmp-drv*
: Specifies the CP/M drive letter to designate as the temporary drive. Default is “P”. The server does not use the temporary drive, but CP/M applications may. For example, MAIL.COM uses the temporary drive as the location for mail message files.

cpnetserver_sid = *node-id*
: Specifies this server’s node ID on CP/NET. All CP/NET requests sent to this server must have their destination node ID (DID field) set to this value. Value is interpreted as a hexadecimal string, and must be in the range 00-FE.

cpnetserver_max = *num-clients*
: Specifies the maximum number of clients that may have active connections at the same time. Values may range from 1 to 16. Clients are not required to login (login and logoff is not used).

However, the first request from a client will “register” itself on the server, provided the maximum has not been exceeded.

cpnetserver_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**”.

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

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

cpnetserver_lst*X* = *lst-spec*
Where “*X*” is a hexadecimal LST: device number, 0-F. Specify the implementation of the LST: device. The supported strings are:
“>*file*” - send all printer out to *file*. Note, this file will contain all output sent during the life of the server.
“**Diablo630Stream** [*options*]” - Use a basic emulation of the Diablo 630 dasiy-wheel printer, producing Postscript output. See section on the Diablo 630 implementation for options.
At this time, no other printer handlers exist.

cpnetserver_debug
Enables debug mode. Currently dumps messages received and sent (in hex).

Starting the Server

Typical setup will include creating a configuration file that specifies all the necessary parameters. Then invoke the JAR file while specifying the configuration file. If stderr has not been redirected (using the log property/parameter) then the terminal will have messages displayed.

Typical startup command:

```
java -jar CpnetSocketServer.jar conf=file
```

Other techniques may be used to start the server in the background or set it up to start automatically whenever the system is booted.

Diablo 630 Implementation

The Diablo 630 printer emulation is provided for convenience in serving printers to CP/NET clients. It is not a complete emulation, but supports the functions used by Magic Wand word processor. Magic Wand uses the printer’s micro-spacing, and does not depend on the printer’s “word processing” functions like print bold, or center a line.

Output will be turned into Postscript and sent to a file. Upon receipt of the CP/NET “end list” character (0xff), the output file will be closed and disposed of as configured (may be deleted depending on the configuration). Configuration properties may reside in the same file used for CpnetSocketServer. Similar to CpnetSocketServer, properties may also be used as commandline parameters by removing the prefix. The commandline in this case is the “[options]” part of the CpnetSocketServer LST: property.

The following properties are recognized:

diablo630_file = *file*

Establishes the name of the output file. This file is the current printer output. The contents of this file will be handled upon receipt of the “end list” character, depending on other configuration parameters. Default is “ps.out” and so must be changed if multiple LST: devices are being used on the same host, running in the same directory.

NOTE: currently, this parameter must be specified on the commandline.

diablo630_nogui

Disables the GUI printer control panel. This is typically required in cases like CpnetSocketServer, although careful invocation of the server may allow for the GUI.

diablo630_jobend = *action*

Determines what will be done to the printer output file when the “end list” character is received. Actions:

discard - erase the previous output and start over with an empty file.
save - save output in a unique filename.
queue *cmd args* – use the command template to process the output. Any “%f” in *args* will be replaced by the output file name.

diablo630_paper = *paper*

Determine the default paper size and orientation. The control panel allows paper to be changed. Recognized paper keywords are LETTER, LEGAL, FORMS, PORTRAIT, LANDSCAPE, and GREENBAR. “FORMS” and “GREENBAR” are experimental values. “FORMS” is for 11x14 inch form-feed paper, “GREENBAR” overlays the familiar computer-center green-bar paper background in the postscript, but that does not print to paper if the postscript is sent to a printer.

diablo630_cpi = *cpi*

Determine the default CPI (characters per inch) setting. If not specified, “10” will be used. The control panel allows cpi to be changed.

diablo630_lpi = *lpi*

Determine the default LPI (lines per inch) setting. If not specified, “6” will be used. The control panel allows cpi to be changed.

diablo630_font = *font*

Determine the default font (typewheel). Font typically needs to be mono-spaced. If not specified, the system provided “Monospaced/PLAIN/12” font will be used. The control panel allows font to be changed.

The value contains three fields, separated by spaces in the property file or commas on the commandline. The values are name, style, and point-size. These must match JAVA font parameters.

NOTE: currently, font style is not parsed. The font will always be “PLAIN”.

Network Boot

Booting off the network is supported. Network boot images are stored in a single directory, speficied by a property in the configuration file. By default, the directory “~/NetBoot” is used. Properties are:

netboot_dir = *path*
Use *path* as the directory to search for network boot images. Default is **~/NetBoot**.

netboot_default = *file*
Use *file* as the default boot image. If not specified, the file **defboot.sys** will be the default.

netboot_org0 = *file*
Use *file* as the special binary file to be executed before starting the booted OS image. This is for systems like the Heathkit H8/H89 where RAM at 0000h needs to be enabled before the OS can start. Default is org0boot.bin. The binary is loaded into 3000h on the target machine, and executed. Bytes 3 and 4 contain the true entry point of the booted OS, so that the ORG0 program can jump to the OS start.

TODO: make this more configurable.

Network Boot Protocol

The protocol used is as follows. Refer to CP/NET documentation for packet fields.

1. Client sends a boot request message, optionally containing a string representing the desired image. FMT=0B0H, FNC=1. String is NUL-terminated.
2. If server refuses (or has an error), response is FMT=0B1H, FNC=0
3. If server accepts, and will send boot code, response is FMT=0B1H, FNC=1, data is a ‘$’-terminated signon string, typically containing a load map (similar to CPNETLDR output).
4. Assuming no error, client sends an ACK message to indicate it is ready for the next packet. FMT=0B0H, FNC=0
5. The server then sends a series of “set DMA” and “load record” messages, each of which must be ACK’ed by the client. Data records are always 128 bytes.

    1. FMT=0B1H, FNC=2, set DMA address to value in DAT (16 bit little-endian).
    2. FMT=0B1H, FNC=3, load 128 bytes of data into DMA, increment DMA by 128.
    3. FMT=0B1H, FNC=4, boot is finished, jump to address in DAT (16 bit little-endian).
6. If the client sends an invalid ACK packet (not FNC=0), the boot will be terminated.

Network Boot Image Format

Files that may be booted by CpnetSocketServer are in an extended CPM3.SYS file format. This also means that an actual CPM3.SYS file may be booted over the network (however, this file has no network support, so the booted OS is operating locally until CP/NET is started). This can, however, be a convenient recovery mechanism if the CP/M 3 partition CPM3.SYS file has been lost or broken.

The general format is as follows.

- The first 128 byte record contains header information:
    - 0: memtop: Top page of memory, “00” means 64K.
    - 1: comlen: Common memory code size, in pages.
    - 2: bnktop: Top page of banked memory, “C0” if bank boundary is 0xC000.
    - 3: bnklen: Banked memory code size, in pages.
    - 4,5: entry: Program start address, in real memory, little-endian.
    - 6,7: cfgtbl: (CP/NET images only) Network Config Table address, in real memory, little-endian.
    - 16: Start of the Copyright message field. If this byte contains ‘C’ then the ORG0 code will be executed.
- The second 128 byte record contains the ‘$’-terminated signon message. If there is no message, the first byte of the record will be ‘$’.
- Subsequent 128 byte records represent the code image. Common memory code is first, followed by banked memory code. Note that these records are in reverse order, they are to be loaded from the top of each memory region downwards.

There is a JAVA tool available that will combine SPR files into a boot image. This is similar to what GENCPM does, however it does not perform any of the special tasks related to configuring CP/M 3.

Note that the boot image need not be an actual OS. Stand-alone programs can be executed off the network without ever booting the machine. This can also provide a mechanism to obtain core dumps of a failing system, provided it is functional enough to boot from the network.