CpnetSerialServer
May 15, 2021

Table of Contents

Introduction

CpnetSerialServer is a JAVA program the implements a CP/NET server, with CP/M 3 extensions. It uses a dedicated serial connection and provides a “server dispatch” function. It receives requests on the serial link, performs the dispatch to the appropriate server, and provides CP/M-style access to native file folders on the host. It can serve as a gateway between the serial link and network servers (CpnetSocketServer) using socket connections.

This JAVA program requires an external add-on package to support manipulation of serial ports. The package jSerialComm is required. See https://fazecast.github.io/jSerialComm/. See “contrib/serialserver” script for example syntax to run. This general command syntax should work on Windows, substituting appropriate paths (and path syntax).

This server has a great deal of code in common with CpnetSocketServer, particularly regarding access to native files.

Serial Protocol

Serial protocol is selected by the **cpnet_proto** property. The string contains the property name followed by options. The SNIOS (for all clients) serial protocol must this setting. Protocols available are:

Simple Serial Protocol

The default serial protocol (SIMPLE ASCII CRC) used by this server (and compatible SNIOS implementations) is as follows for each message:

1. Message start sequence "++".
2. Message data (CP/NET request/response) in 6 to 261 pairs of (uppercase) hexadecimal digits.
3. Two pairs of hexadecimal digits representing the CRC16 of the binary message data.
4. Message end sequence "--".

Receipt of the dash character(s) is required to consider the message complete. Once the message CRC has been verified, the message data may be inspected and the CP/NET function performed.

The CRC16 algorithm is represented by the following (JAVA) code. The crc is initialized to the value 0xffff at the start of each message.

```
private int crc;
static final int POLY = 0x8408;
private void crcByte(int data) {
        int i;
        int mask;
        for (i = 0; i < 8; ++i) {
                mask = ((crc ^ data) & 1);
                crc >>= 1;
                data >>= 1;
                if (mask != 0) {
                        crc ^= POLY;
                }
        }
}
```

The SNIOS in src/serial/snios.asm uses this protocol. The exact character transport is defined by the chrio.asm module, for example in src/ft245r/chrio.asm.

Two special message functions are recognized, used to manage the network. Both use CP/NET FMT=0/1 packets, but use FNC codes 255 and 254. FNC=255 is a message used by the client to discover it's node ID, the response will contain the client's node ID in the DID field. FNC=254 is used to notify CpnetSerialServer that the client wishes to shutdown all connections. This message has no response.

The standard CP/NET function requests (and responses) are described in the DRI CP/NET manuals.

No additional debug levels are provided by this protocol.

DRI Serial Protocol

The "DRI" protocol is the one shown in the Digital Research sample SNIOS (reference implementation). It uses ASCII control characters to implement the protocol, and by default sends data in binary. It is summarized here:

1. ENQ, *response:* ACK
2. SOH <FMT> <DID> <SID> <FNC> <SIZ> <checksum>, *response:* ACK
3. STX <data>… ETX <checksum> EOT, *response:* ACK

The checksum byte is a simple 8-bit sum of bytes sent, negated (i.e. the checksum of the message including checksum byte is 00). In the case of the data packet, checksum does not include EOT.

The only protocol option is ASCII, to send/receive data bytes as ASCII hexadecimal digit pairs instead of binary. Note, the SNIOS must also be compiled with this option for it to work.

Debug level 5 will add protocol and timeout error messages for received characters. Level 9 will add logging of all characters received. Note, level 9 may disrupt timing enough to prevent proper operation.

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**". 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.

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

The following properties are recognized:

cpnet_log = *log-file*

Diverts stderr to *log-file*.

cpnet_tty = *tty-dev* [*baud*]
The tty device to use for the connection. For example, “/dev/ttyUSB0”. May be set to “stdio” as well. The tty device name may be followed by the baud rate to use.

cpnet_flow_control = *flow-control*
Specifies the serial port flow control to use. Valid *flow-control* strings are “none”, “rts/cts”, and “xon/xoff”. Note that these require cooperation from both sides, especially for XON/XOFF. Flow control is enabled for both directions, and cannot be limited to the server-to-client direction only.

cpnet_cid = *node-id*
Specifies the CP/NET node ID to assign to the attached client. Value is interpreted as a hexadecimal string, and must be in the range 01-FE.

cpnet_protocol = *protocol*
Specifies the style of serial communications that the SNIOS expects. With no property, the protocol is “SIMPLE ASCII CRC”. Specifying the property clears defaults to BINARY and no-CRC. Protocols supported are DRI and SIMPLE.

cpnet_server*XX* = *server-spec*
Where *XX* is the hexadecimal node ID to use for the server. server-spec may be “HostFileBdos” (for local file folders) or “Socket” to specify a remote CpnetSocketServer instance. These options are discussed below.

cpnet_console = *cons-spec*
Selects the console mode, for systems where the CP/NET communications port is the system console port. The selects how the user console is attached to CpnetSerialServer. The following are supported:

StdioSerial
The stdio of CpnetSerialServer is used as the console. Note that typically this will default to a buffered mode that is not compatible with character-by-character mode of CP/M. It may be necessary to alter the configuration of the stdio device in order for this to work well. For example, on Linux/Unix systems the utility “stty” can be used.

TelnetSerial *host port* [**modem**] [**nodtr**]
Listen for telnet (TCP/IP socket) connections on *host* at *port*.Note that typically telnet (or the equivalent client program) needs to be set to “character” mode in order to satisfy the needs of CP/M console I/O. The status of the telnet socket connection will be reflected in the serial port RTS signal. If the target system does not provide DTR, then the **nodtr** option should be used so that connections are not immediately dropped.

ProgramSerial *program* [*arguments*]
A process is created using *program* [*arguments*] and the console is connected to the stdio of that process.

See Console Mode for more details on console mode operation.

cpnet_console_oob
Enables OOB (Out-Of-Band) mode for console mode. In OOB mode, the SNIOS may send console characters directly between CP/NET message. A CP/NET request/response sequence is never interrupted by console output characters.

cpnet_console_log = *file* [**a**]
Logs all console output to *file*. If the "a" (append) option is added, the file is not truncated at the startup of CpnetSerialServer.

cpnet_debug = *level*
Enables debug mode for *level* > 0. Level 1 dumps messages received and sent (in hex). Each higher level increases debug, but does not turn any lower level debug off. Handling of higher levels is determined by the protocol, as described in the protocol sections. Specifying no level is the same as specifying "1".

DRI Protocol Properties

dri_ack_timeout = *msec*
Use *msec* milliseconds as the timeout when waiting for an ACK. Default is 15mS.

dri_char_timeout = *msec*
Use *msec* milliseconds as the timeout when waiting for a character. Default is 15mS.

dri_retries = *num*
Retry a maximum of *num* times. Default is 10. This normally should match the retries compiled into the SNIOS.

HostFileBdos Server Properties

HostFileBdos properties may be specified in-line with (in the same file as) the CpnetSerialServer properties. In addition, the first argument to the "HostFileBdos" server-spec (may) specify the root dir,

hostfilebdos*XX***_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.

hostfilebdos*XX***_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 "*${HOME}*/**HostFileBdos**". Note: JAVA does not expand "~" or environment variables the way a shell does, and so the exact path must be specified.

hostfilebdos*XX***_nofile** = *num*
Specify the number of open files to allow. Minimum/default is 32. Some CP/M utilities, such as PIP (for 2.2), do not close files that are only read and this can result in “Too many open files” errors, for example during a long PIP transfer from a networked drive.

hostfilebdos*XX***_nosys**
Disable the CP/M SYS attribute, so files will not be hidden on Windows.

hostfilebdos*XX***_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.

hostfilebdos*XX***_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.

Socket Server Properties

Note: host and port may be specified after the “Socket” server-spec, in that order. In that case, there is no need for properties.

cpnetserver*XX***_host** = *host-or-ip*
Specifies the host network address of the remote server.

cpnetserver*XX***_port** = *port-number*
Specifies the port number of the remote server.

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.

Because of the need for jSerialComm, the startup command is more complicated. A bash script exists in **contrib/serialserver** that may work for most non-windows systems, and may provide an example for others.

Typical startup command using ‘serialserver’:

```
./serialserver conf=file
```

Startup using the ‘java’ command directly:

```
java -cp CpnetSerialServer.jar:/path/to/jSerialComm-2.5.3.jar \
CpnetSerialServer conf=file
```

Note that the location, and name, of the jSerialComm JAR may vary, and would require customization of the command or shell script.

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

Console Mode

The console mode is provided for special cases where the CP/M system is actually a diskless machine with only a console port for communications. In this case the connection between the CP/M machine and CpnetSerialServer can be used for both console and CP/NET. The server initially operates in a pass-through mode where all characters are directly passed through between the CP/M system and the attached console device.

Once CP/NET traffic begins, the server (by default) requires all traffic to be CP/NET messages, at which point the CP/M machine must interact with the console through CP/NET (the CON: device must be networked).

The optional OOB mode may be selected, where CP/NET messages represent a temporary cessation of console traffic which will resume at the end of the CP/NET request-response sequence. In this case the client sends console output directly to the serial port, but must perform extra steps to ensure that the start of a CP/NET request will save any console input received at that point.

When the network shutdown message is received (FNC 254), as performed by the NETDOWN.COM utility, the server reverts to pass-through mode.

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 way 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 from the network is supported the same as described for CpnetSocketServers.