CpnetSerialServer
May 19, 2020

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 RXTX is required. See http://rxtx.qbang.org/wiki/index.php/Download. Tested with rxtx-2.2pre2-bins.zip.

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

Serial Protocol

The default serial protocol (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.

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**".

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_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_proto = *protocol*
: Specifies the style of serial communications that the SNIOS expects. With no property, the protocol is ASCII with CRC. Specifying the property clears defaults to BINARY and no-CRC.

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.

NOTE: Specifying multiple HostFileBdos servers is currently not supported, and will have unpredictable results.

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 “**~/HostFileBdos**”.

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.

Typical startup command:

```
java -jar CpnetSerialServer.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 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”.