ZARC

STran Users' Guide

Serial file transfer for ZARC, but may have other applications.
Written by Merlin Skinner-Oakes, 14/3/2021

Last updated .

Introduction

There are various options for transferring files over a serial line. The monitor's tpadownload command works but is crude and manually entering the file length is prone to error. It also requires quite a bit of typing. Other options are gkermit and PCGET / PCPUT (Xmodem), but none seem to really do what I want them to do. Gkermit does work, but again is quite a bit of typing for each file. Scripts could ease this somewhat, but ideally I would like a single command from one end only that would transfer a file. It would be nice to be able to use wildcards or transfer an entire disk / directory as well.

Architecture

The basic structure is:

1) A Python Linux program ("stran") that would perform the PC side of the operation.
2) A CP/M program (also called stran) that would act as a server.

The Linux side would be controlling the transfers. There are some advantages to having the Linux (rather than the CP/M) side control things:

1) Running the CP/M system with a single serial line and a terminal emulator on the PC is much simpler this way. Start the CP/M server using the terminal emulator, then exit and control things from the PC, optionally transferring data over the same line. Having the CP/M system in control would force the use of two serial lines, or one and a "built-in terminal" (display and keyboard).
2) This keeps the CP/M side simple. This is advantageous, as this will need to be written in assembler.
3) It will be easier to write scripts for operations such as drive backup in Python.

Protocol

Byte Encoding

ZARC serial ports are capable of full 8-bit operation with no special codes. However, some CP/M BIOS implementations remove bit 7, and other systems may use control codes such as XON and

XOFF for special functions. For these reasons, a protocol that avoided use of control codes or bit 7 is preferred. Only codes 0x20 to 0x7f are used, with all but the synchronisation and packet start sequences sent using 6 bits (0x40 to 0x7f). Each three 8-bit bytes are encoded into four as follows:

Encoded Byte Number	**Data**
0	<0, 1> <byte 0 bits 7 to 2>
1	<0, 1> <byte 0 bits 1 to 0> <byte 1 bits 7 to 4>
2	<0, 1> <byte 1 bits 3 to 0> <byte 2 bits 7 to 6>
3	<0, 1> <byte 2 bits 5 to 0>

Zero padding is added as required to complete the four byte group when less than three bytes are to be sent at the end of a packet.

Packet Format

Block length (BLKLEN) is fixed at 512 bytes. This was chosen to be shorter than that known to cause issues with at least some USB RS-232 adaptors (1024 bytes). Note that this length defines the maximum size of the data portion of the packet only. Allowance must also be made for the 3 to 4 bytes encoding scheme overhead.

In normal operation, transfers are in the form of packets with the following general format:

Start Offset	**Data**	**Description**
0	“*” (0x2a)	Start of packet (not encoded)
1	8 bits	Packet type (see below)
2	16 bits	Packet Sequence Number (“PSN”)
4	16 bits	Data length (0 to BLKLEN).
6	Data	Packet data (if present)
	0 to 16 bits	Padding to align to a 3-byte boundary.
	16 bits	CRC16 / XMODEM checksum of all fields except start of packet byte. Initial CRC is zero.
	8 bits	Pad to align to a 3-byte boundary.

The entire packet is encoded using the 3 to 4 bytes encoding scheme with exception of the initial “*”. 16-bit values are sent LS byte first.

When started, the CP/M side awaits a valid command packet. The response will depend on the command.

Packet Type Codes

Packet Type	Name	Source	Data Field Contents	Packet Sequence Number	Description
0x00	ACK	PC or CP/M	No data	Number of corresponding packet.	Positive acknowledgement.
0x01	NAK	PC or CP/M	Reason for NAK (see below)	Number of corresponding packet.	Negative acknowledgement.
0x02	Exit	PC	No data	N/A	Exit CP/M *stran* and return to CP/M.
0x03	Tx	PC	File path *$_1$ or data *$_2$	0 to number of blocks in file.	Transmit file (PC to CP/M).
0x04	Rx	PC (PSN = 0) or CP/M	File path *$_1$ or data *$_2$	0 to number of blocks in file.	Receive file (CP/M to PC).
0x05	End	PC or CP/M	No data	N/A	Normal end of transfer.
0x06	Abort	PC or CP/M	No data	N/A	Abort transfer.
0x07	Dir	PC (PSN = 0) or CP/M	No data *$_3$	N/A	Read directory.

*$_1$ <drive letter><8 character filename><3 character extension>, so 12 bytes in total.

*$_2$ File path if PSN = 0, or file data otherwise.

*$_3$ <drive letter> if PSN = 0, or directory data otherwise. Protocol is otherwise identical to receive file, substituting “Dir” for “Rx” in packet type field.

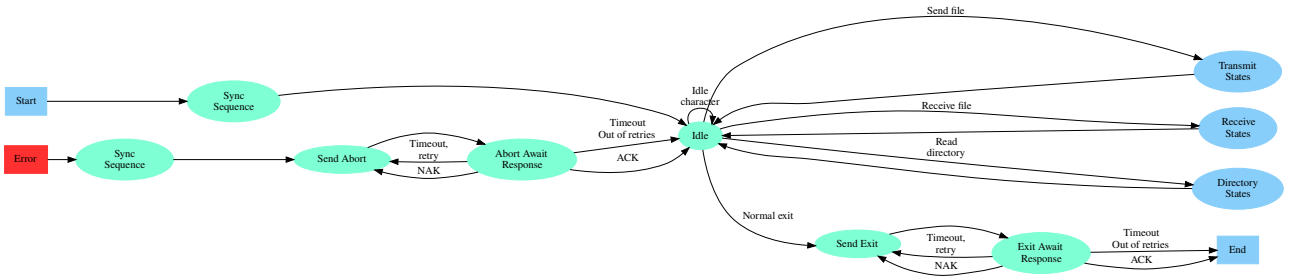
NAK Reason Codes

These are sent as a single byte in the data field, so data length = 1.

Reason Code	Description
0x00	CRC error
0x01	Unexpected PSN
0x02	Unexpected command
0x03	Can’t open file
0x04	BDOS file read / write error
0x05	Abort received when no transfer is underway

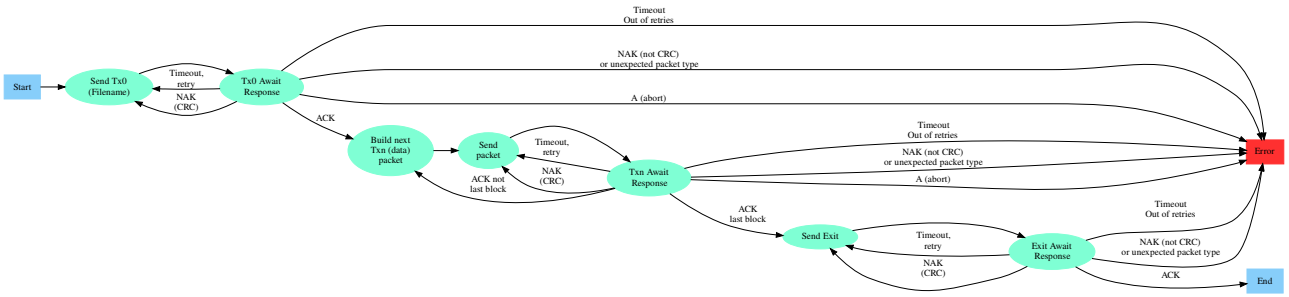
State Diagrams

These diagrams show the state transitions at the Linux end.

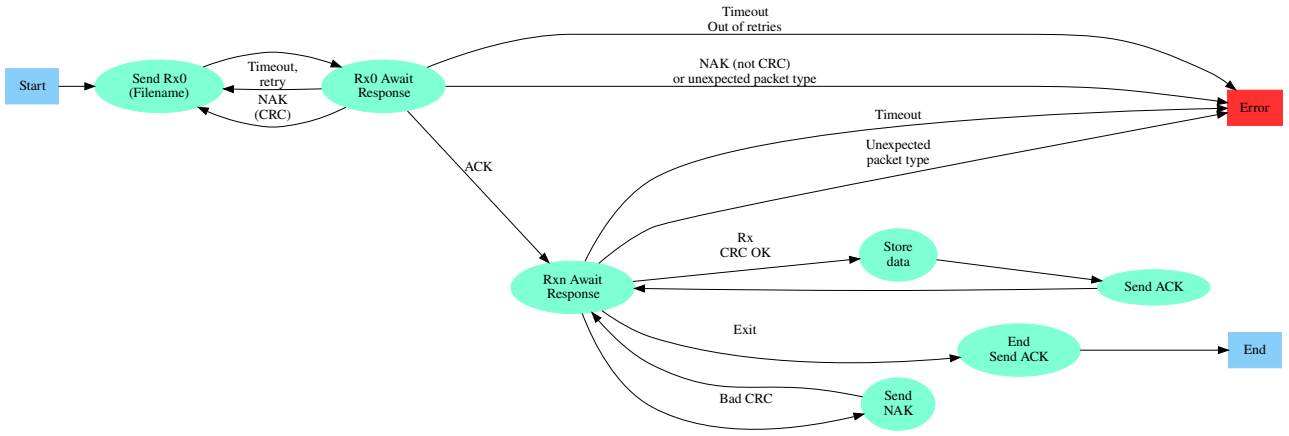


Linux Side Master State Diagram

Blue “Transmit States” and “Receive States” invoke the state sequences shown below. “Directory States” are identical to receive states, but the file data received is the required directory information. Periodic idle characters (spaces) are sent to allow the CP/M side to poll for break characters (^C), since anything but console port reads are blocking when using the BIOS for serial I/O.



Linux Side Transmit File State Diagram



Linux Side Receive File State Diagram

Serial Ports

The CP/M code is designed to be portable to other CP/M systems. For this reason, it uses BIOS functions to access the serial ports. The READER (input) and PUNCH (output) devices are used for serial I/O. These may be redirected to various physical devices using the IOBYTE mechanism. See https://www.seasip.info/Cpm/iobyte.html for possible mappings. The mappings may be changed using the CP/M STAT command, for example:

CP/M Command	Description

STAT PUN:=TTY: STAT RDR:=TTY:	TTY: (serial port 1 for ZARC) is used for data transfer.
STAT PUN:=PTP: STAT RDR:=PTR:	RDR / PUN (serial port 2 for ZARC) are used for data transfer.
STAT DEV:	Shows device mapping