Inter-Office Memorandum

To	Distributed Computing	Date updates	November 26, 1979 June 9, 1980
From	Glenn Krasner updated by Tim Diebert (CSL)	Location	Palo Alto
Subject	Tape Server Software	Organization	PARC/SSL

XEROX

Filed on: [IVY]<TapeServer>Docs>TapeServerSoftware.memo, .press

Overview

This document is an explanation of the software written to implement the Tape Server Protocol (TSP) as outlined in [IVY]<TapeServer>Docs>TapeServerProtocol.press. For related files, see [IVY]<TapeServer>Docs>TapeServerSoftwareOverview.press.

The tape server software is a program written in BCPL that implements the TSP for an Alto the has the Alto Tape Controller Hardware. It can support up to four drives simultaneously (currently the limitation is for 2 tape units), each being a 1600 bpi phase-encoded drive running at 45 ips or 125 ips. Currently it has been used on Mohawk and Kennedy drives, but should work on any drive that can be attatched to the Hardware.

The tape software consists of four parts: the tape controller microcode, the tape controller software, the pup package software and the tape server software. Only the latter is described here; for information about the other four, see the overview document.

The tape server software allows multiple TSP (which uses BSP) connections controlling multiple drives. The details of this are outlined below.

Structure

The tape server software source code lives on three files: TSP.decl, TapeServer.bcpl and TSPDOs.bcpl. TSP.decl contains the declarations of constants and structures used by the software. TapeServer.bcpl contains the main-body code, and TSPDOs.bcpl contains the command/message processing routines.

In TSP.decl are the constant and structure declarations. Constants include the size of the shared Read/Write block (rwBlockLength), the size of a typical command record (cmdBlockLength), the maximum number of servers and drives (maxServers and maxDrives), the "well-known" socket number for rendezvous in this protocol (tapeSocket) and the number of the current version of server software (currentVersion). Also included in the constants are the numbers assigned to each message (command) type (spelled cmd***) and for numbers assigned to each No message type. Structures include the structure for a Service block (see below) and a structure for each message in the TSP protocol.

In TapeServer.bcpl are the static variable declarations, the initialization code and the code that is executed by each Server instance. Static variables include the shared Read/Write block (rwBlock), the key to use of that block (rwKey), the key to use of the operator’s console (kbdKey), a vector of potential id numbers (usedidnos), a vector of potential drives (useddrives) the string describing the current version (currentVersionString), a vector of BSP sockets to be used (bspSockets) and a vector of server blocks, one used per Server instance, (serviceBlocks).

In TSPDOs.bcpl are the routines that are run in response to a user message (command). There is one routine per message, plus a few utility routines for common message and message parts. Each message routine parses its own messages. The routines are named "Do<message>" for the given message. (*e.g.* DoOpenDrive handles an Open Drive message) Also in TSPDOs.bcpl are the replying routines ReplyYes, ReplyNo and ReplyStatus (the latter is used when returning device ending status).

How it Works

The tape server system is a multi-process system that consists of a connector process, a disconnector process and zero or more active instances of server processes. The connector process watches out for attempts by users to establish a BSP connection. When one is established, an instance of a server process is activated and the connector goes back to looking. The disconnector process monitors potentially open connections for disconnections. When one is caught, its corresponding server instance is killed. This catches normal as well as abnormal disconnections. Each server process waits for a TSP Version message block. Once a Version message is received, the server sends a Version reply and waits for another message. When a message is received, the server runs the corresponsding Do*** routine to process that message and then waits for the next.

Server instances consist of a BCPL process running the Server() routine and pointing at a data block called a Service block which holds the data unique to this server. Included in the Service block are a unique id number for this server (>>Service.idnumber), a BSP socket and corresponding stream (>>Service.bspSoc and >>Service.bspStr), a pointer to a block to be used by all commands except Reads and Writes (>>Service.cmdblock), a pointer to a tape block (>>Service.tape, see XEOS tape documentation), the drive, if any, that this instance is using (>>Service.drive), the number of retries and the tape speed.

The routine StartServing() initializes the tape server system. It loads the microcode, allocates Service blocks, command blocks, process contexts and BSP structures and opens the low level BSP to listen to the "well-known" socket. It then passes control to the other processes.

Connector() is the connector process routine. When a packet is heard at the rendezvous "well-known" socket, it checks to see if it is a good request for connections. Conditions that cause bad connection requests are broadcast packets, duplicate connection requests, or lack of available servers (current number of servers = maxServers). If none of these conditions holds, then InitializeServer() is called to initialize the Service block for a server, and that server is activated.

DisConnector() looks down the list of used idnumbers to see which have activated servers. For each, the BSP connection is checked to see if it is open. When one is found that is not open, the server's keys are returned, its tape closed if open, its BSP stream closed and its server deactivated.

As mentioned above, Server() is the routine that handles each server instance, waiting for a Version command and then dispatching to the appropriate Do** routine to parse and respond to the rest of the commands.

Miscellaneous

In this section are listed some peculiarities about the server software that may be worth noting. Unfortunately they are in random order.

The Read/Write block is shared by all server instances. This allows the software to statically allocate one 16k-word block to handle the largest expected record. In the future, we may want either dynamic allocation or a smaller non-shared block statically allocated for each server.

Message replies attempt to correspond to the messages. For example ReadRecord will reply No if End of File is seen, but WriteEOF will not. I will not list here who replies how; if you are interrested, either read the code, see me, or force me to include it here.

The only status parameters allowed to be changed are number of retries and drive speed. This means that there is no software write protection or other such feature.

The "well-known" socket number is 44b. This was assigned to Tapes from the keeper of the keys in Webster. Major changes to this assignment should go through there.

Messages are responded to in order. The length field of a message received from a user determines the length of data until a new message block is expected. There is no provision for getting back into synch with a bad user. The only known way is for the user to allow its BSP connection to time out, and to establish a new connection.

Building Your Own

Below is the command line used to link up the TapeServer.run file for the tape server software. To build one’s own mutation of the system, one should edit and compile the corresponding .bcpl file, and have the other .br files around to link and load in. I have not described every .br file in the list, it should be clear from the TapeServerSoftwareOverview document which files do what.

```
bldr/f 470/w TapeServer TSPDOs  TapeIO LoadRam TapeTfsSwat Pup1b Pup1Init Pup1OpenClose PupAl1a PupAlEtha
PupAlEthb PupAlEthInit altobyteblt altoqueue altotimer PupBSPa PupBSPBlock PupBSPOpenClose PupBSPProt
PupBSPStreams context contextinit PupDummyGate interupt intiniterupt PupNameLookup PupRoute PupRTP
PupRTPOpenClose TapeMicrocodeXM
```

Comments

I am willing to make changes to the system at least for the next while, at least until there are some frequent users.

Please send me any comments, questions or problems you have with this document or with the system.