The MiniServer and ServerTask

Perihelion Software Technical Report No. 12

Bart Veer

May 1989

Perihelion Software Limited
The Maltings
Charlton Road
Shepton Mallet
Somerset
BA4 5QE
England
Telephone +44 749 4203
Fax. +44 749 4977

Contents

1 The Bootstrap Process 5

2 The Various Protocols 7

3 The ServerTask and Networking 12

The standard Helios hardware configuration has one or more I/O processors connected to a transputer network. Every I/O processor runs a program, called the I/O Server, which provides a number of servers for the various devices which are provided by the host. The I/O server also communicates with the transputer network in accordance with Helios protocols, so that the host machine looks and behaves like any other Helios node, with the sole exception that you cannot use it to execute transputer programs. The type of processor contained in the host machine is unimportant.

The main advantage in having an intelligent I/O server is that input and output operations make use of the host's resources, and not those of the transputer network. This means that I/O operations do not use up valuable processing time within the transputer network, and they do not use up the transputers' memory. If the resources are directly connected to a transputer, then the transputer's CPU time and memory are needed, and the host's resources are wasted.

Unfortunately, the I/O. Server is a large program which needs a minimum of about 512Kbytes of memory. In addition to this, it is often difficult to port the full Server to some machines, particularly those which are still in prototype form, or are using experimental software. To cope with these problems a separate environment, the MiniServer/ServerTask combination has been developed.

In outline the MiniServer is a simple program that runs on the host, booting up the root transputer and providing input/output facilities. The system image booted into the root transputer contains an extra program, the ServerTask, which contains Helios servers for various devices. The ServerTask communicates with the MiniServer down a dumb link using its own protocol, rather than the usual Helios protocols. Hence the bulk of the processing is now done on the root transputer, and only the low-level I/O still takes place in the I/O processor.

There are major difficulties with this system. First the ServerTask uses up between 100K and 200K of transputer memory, depending on load, as well as some of the CPU time. Secondly this program is difficult to write, debug, and maintain, because it is part of the system image and therefore cannot use any of the posix or C library routines. It is difficult to get any text debugging out until the ServerTask is fully initialised, which takes quite a long time, so when things go wrong it is hard to find out why. Many of these problems are caused by subtle interactions between the ServerTask and the system and server libraries, and unless you have access to the sources of these libraries it is difficult to determine what is happening.

It is particularly difficult to increase the number of facilities which are available, because it involves modifying two programs, that is, both the MiniServer and the ServerTask. At present the ServerTask only provides a simple console device and file I/O, and not the multiple windows, communication ports, X-Windows support, and other facilities available in the I/O Server. It is also difficult to port the MiniServer/ServerTask to work with different host machines, as this involves modifying the ServerTask on the transputer side, as well as the MiniServer on the host side. This compares unfavourably with the standard Helios system where all the machine-dependency is isolated in the I/O Server, and exactly the same version of Helios runs on all the transputers.

On the plus side, the MiniServer/ServerTask does provide a number of facilities not available in standard Helios. During periods of inactivity it is possible to arrange for the two programs to disconnect, and reconnect at some later stage without rebooting the transputer network. The MiniServer is sufficiently small that, given an interrupt-driven link interface, it could run in the background or as a Terminate-Stay-Resident program. This would allow the transputer network to be used as an accelerator for the host, rather than as the main machine. It is possible for the host to send commands to the ServerTask to be executed in the transputer network, using a special command protocol, and it is possible for transputer programs to communicate with the host using Helios messages.

1 The Bootstrap Process

Booting Helios into the root transputer is tedious rather than difficult, as there are a large number of steps.

1. Reset the root transputer.
2. Read the file, nboot.i, which contains a simple transputer bootstrap program. The first 8 bytes should be ignored, the next 4 bytes form a single 32-bit little-endian word, which is the size of the bootstrap program. This program starts at the 13th byte.
3. Send a single byte, the size of the bootstrap program.
4. Send the bootstrap program itself, starting at the 13th byte of nboot.i. The transputer now starts to run this bootstrap program.
5. Clear parity memory, if any, by sending a byte, 5, followed by the memory size as a 32-bit word. Wait for a single byte reply from the transputer, which can be discarded.
6. Send a single byte, 4. This instructs the bootstrap program to read in a system image.

7. Send all of the file, linknuc, which is a system image containing the kernel, system library, server library, utility library, processor manager, loader, and ServerTask. The first word of the file contains the size of the whole file.

8. The bootstrap program now transfers control to the kernel, which requires some configuration information. The MiniServer should send the size of the configuration as a single word. The configuration below may be used if desired, it is exactly 0x48 bytes long.

9. Send the configuration information. You may either construct your own, using the data structure defined in the Helios header file config.h, or you can use the following sequence of hex bytes

   ```
   00 04 00 00 01 00 00 00
   00 10 00 80 yy yy yy yy
   xx xx xx xx 06 00 00 00
   00 00 00 00 00 00 00 00
   00 00 00 00 1c 00 00 00
   1c 00 00 00 04 00 00 00
   70 01 02 00 00 02 06 01
   00 02 06 02 00 02 06 03
   2f 30 30 00 2f 49 4f 00
   ```

   Two of the fields in the configuration need patching at run-time. yyyyyyyy should be the size of the system image, i.e. the first four bytes of the file linknuc. xxxxxxxx should be the current time expressed as a unix time stamp, i.e. the number of seconds since the first of January 1970.

10. The kernel can now initialise itself and start up the processor manager, which in turn starts up the other programs in the system image. In this case the programs are the loader and the ServerTask. The ServerTask needs some additional configuration information which must be provided by the MiniServer.

11. The ServerTask and the MiniServer need to synchronise with each other. To do this the ServerTask will send a single byte 0xF1 down the link, and the MiniServer should respond with a byte 0xF2.

12. The MiniServer should now send a single byte specifying the size of the information below.

13. The MiniServer should send the following configuration

   ```
   byte 1    : identifies host, 1 = PC, 2 = DP2, etc.
   byte 2    : flags byte, currently unused
   ```

```
byte 3     : the number of disk drives connected to
             the host which must accessible from Helios
bytes 4-7 : a message limit, indicating the maximum
             number of bytes which should be transferred
             between the host and the root in one message.
             This should be at least 1K, but preferably
             no larger than about 32K to avoid excessively
             large buffers on the root transputer
bytes 8-   : the name of the Helios directory, for example,
             c:\helios, terminated by a byte '\0'
remainder : the names of the other drives, all null
             terminated
```

Note that the MiniServer uses local filenames throughout, and it is upto the ServerTask to do all the name conversion. Once the ServerTask has received its configuration information it can finish initialising itself and fork off Helios servers for all the devices. The ServerTask then runs /helios/lib/init, which reads the file, /helios/etc/initrc, as per the usual bootstrap.

2 The Various Protocols

There are various protocols used between the ServerTask and the MiniServer. All these protocols are introduced by a special header byte, as follows:

```
0 : synchronisation protocol
1 : IOServer protocol
2 : command protocol
3 : message protocol
4 : polling protocol
5 : exit protocol
6 : sleep protocol
7 : wakeup protocol
```

The polling protocol and the wakeup protocol only go from the MiniServer to the ServerTask. The exit protocol and the sleep protocol only go from the ServerTask to the MiniServer. The other four protocols are bi-directional.

The purpose of the synchronisation protocol is to inform both sides that the other side is still alive. Both sides should send a single byte, 0, down the link at regular intervals. At present these bytes are ignored by the ServerTask, but the PC version of the MiniServer will exit with a suitable message if it does not receive a synchronisation byte once every 10 seconds, on the assumption that the transputer side has crashed. It may be desirable to disable this protocol when using the transputer network as an accelerator, because it causes link activity even when the transputer network is not currently in use.

The exit protocol consists of a single byte, 5, sent by the ServerTask. This will happen if the user logs out of the login shell, which is interpreted at present as a request to return to the host system. On receiving this byte the MiniServer should just exit.

The sleep protocol consists of a single byte, 6, sent by the ServerTask to the MiniServer to disconnect the two programs for a period of time. There is a sleep.c program which makes the ServerTask send this protocol byte. On receiving the sleep byte the MiniServer can exit. At some later stage the MiniServer may be run again, and instead of rebooting the root transputer it can send a single wakeup byte, 7, to the ServerTask. The sleep protocol should be used only when the transputer is not running any software that needs to perform I/O.

The remaining protocols are more complicated and will be described below.

The IOServer Protocol

This is the protocol used between the ServerTask and the MiniServer to perform I/O of some sort, and always consists of a single message from the ServerTask followed by a single reply from the MiniServer. The ServerTask will not send any other requests until the reply has beta received, and the MiniServer should not send any other messages.

The protocol byte from the ServerTask is followed by the following four bytes

```
  1 : a function code
  2 : an extra byte for additional information
  3 : a high byte and
  4 : a low byte.
```

This package may be followed by a data vector of upto 64K of additional information, the amount being controlled by the high and low bytes. The reply sent by the MiniServer consists of the same package, with the function code replaced by a reply code. At present there are only two reply codes, 0x80 (128) for success and 0x81 (129) for failure. The following function codes are defined.

OpenFile = 1
The extra byte specifies the mode; 1 for read-only, 2 for write-only, with an implicit truncate to 0 length; 3 for read-write. The data

vector contains the name of the file to be opened as a null-terminated string, in local format rather than Helios format, and as an absolute name rather than relative to the current directory. On success the MiniServer should return a two-byte integer in the data vector, which is interpreted as a stream identifier and used for subsequent stream requests.

CloseFile = 2
The data vector contains the stream identifier returned by OpenFile.

ReadFile = 3
The first two bytes in the data vector contain the stream identifier returned by OpenFile, and the next two bytes specify the amount of data to be read. The MiniServer should return the amount of data requested, and may return less if, and only if, the end of file is reached.

WriteFile = 4
The data vector contains the stream identifier, followed by the data to be written.

SeekInFile = 5
The extra field contains 1 for seek from start, 2 for seek from end. The data vector contains the 32-bit offset in the file, 0 or positive if seeking from the start, 0 or negative if seeking from the end. This offset is followed by the stream identifier. On success the MiniServer should return the new 32-bit offset in the data vector.

CreateFile = 6
The data vector contains the null-terminated name of the file to be created or truncated to zero length.

DeleteFile = 7
The data vector contains the name of the file to be deleted.

Rename = 8
The data vector contains the current filename, followed by the new filename. Renaming of directories is not supported.

Locate = 9
The data vector contains the name of the file or directory to be located. On success the extra field in the reply packet should contain 1 if the object is a file, or 2 if it is a directory. Note that this function should be implemented as efficiently as possible, because it is invoked before most of the other routines.

ReadDir = 10
The data vector contains the name of the directory to be read. If

successful, the first two bytes of the data vector of the reply should be the number of entries in the directory. This should be followed by the actual entries, each entry consisting of a byte 1 for a file or a byte 2 for a subdirectory, followed by a null-terminated name.

For example, if a directory contains three entries aa, bb and cc where aa and bb are files and cc is a directory, the data vector should contain the following: 00 03 01 41 41 00 01 42 42 00 02 43 43 00

CreateDir = 11
The data vector contains the name of the directory to be created.

RemoveDir = 12
The data vector contains the name of the directory to be deleted.

FileInfo = 13
The data vector contains the name of the file to be examined. On success the MiniServer should return a 32-bit file size, followed by a 32-bit unix time stamp for the file specified. This time stamp should correspond to the time last modified.

DiskUsage = 14
The data vector contains the name of a file or directory. The MiniServer should use this name to determine some statistics for the corresponding disk drive. The first word in the reply should be the size of the disk drive in kilobytes, and the second word should be the space left on the disk in kilobytes.

WriteToScreen = 15
The data vector contains some text to be written to the screen. The ServerTask contains an ANSI screen emulator, which must be modified for different hardware.

ChangeDate = 16
The data vector contains a file name. The MiniServer should set the date-last-modified of that file to the current time. This is used by the touch command.

The Polling Protocol

The MiniServer is responsible for polling devices such as the keyboard, and sending any data received to the ServerTask using the polling protocol. The packet format is similar to that used by the IOServer protocol. The function code specifies the source of the data, the keyboard being source 1, and no other sources being defined at present. The extra field contains the first byte of data. If there is more than one byte of data, this follows in the data vector. The ServerTask should not acknowledge receipt of data, i.e. this protocol is used only from the MiniServer to the ServerTask.

The Message Protocol

The ServerTask contains a device called /dp2mess, which may be opened by Helios programs. Once these programs have an open stream they can send messages to the host via this stream, and the host can use the message port in the first message to send data to the transputer. There is no facility for the host to send the first message. It is up to the applications on both sides to use message passing correctly, i.e. correct use of full and empty reply ports and the MsgHdr_Flags_preserve flag. An example program, dp2mess.c, illustrates such message passing.

When the ServerTask receives a message sent to /dp2mess, it transmits this down the link as follows: the Message protocol byte; the message header's flag field; the control vector size; the data vector size; the destination port; the reply port; the function code; the control vector if any; and the data vector if any. Messages can be sent from the host to the transputer using the same format.

The Command Protocol

The ServerTask contains code allowing it to run programs under the control of the host. It can also run compiled CDL scripts if a Task Force Manager is running. The ServerTask contains its own interpreter rather than using a shell, so it is not possible to execute shell scripts and uncompiled CDL scripts in this way.

The command protocol works as follows. The ServerTask receives a protocol byte followed by the usual four byte packet. The function code is ignored. The extra byte contains an integer identifier for the program, which will be returned by the ServerTask in its replies. The data vector contains the command to be executed, in a format described below; it may consist of upto 512 bytes. The ServerTask will send back replies at various stages, using the same packet format. The following reply codes are defined:

```
0x01 : the program is ruining
0x02 : the program has finished
0x81 : the ServerTask failed to parse the command provided
0x82 : failed to locate the current directory
0x83 : failed to open stdin
0x84 : failed to open stdout
0x85 : failed to open stderr
```

```
0x86 : failed to find command
0x87 : failed to load command
0x88 : failed to start command running
0x89 : failed to send environment to command
0x8a : failed to set up signal handling for command
```

All except the first two reply codes indicate a fatal error and will prevent the program from running. A typical command would be:

```
ls @/helios/etc { /null, /helios/output, /Logger } -l
```

This means that the command to be executed is ls. This command should be in the /helios/bin directory, although an absolute filename may be given instead. The program should run using /helios/etc as the current directory, the default being /helios. The standard streams are /cull for stdin (always returns end-of-file straightaway), the file /helios/output for stdout, and the error logger for stderr (data written to stderr will go to the screen). The default standard streams are /null, /logger and /logger. There is just one argument, "-1", although any number may be given subject to the 512-byte upper limit for a command. The parser is fairly flexible, so the following are also valid commands,

```
ls { , /helios/output } -l @/helios/etc
/helios/bin/ls -l @/helios/etc
```

3 The ServerTask and Networking

Using a special nucleus for the root transputer creates a small problem when booting up a network of transputers, as it is no longer possible to use the existing nucleus to boot up the neighbour, as that neighbour would then run another ServerTask which installs its own servers and expects to talk down a dumb link 0 to a MiniServer. To get around this problem there is a special version of rboot, ms_rboot, which should replace the rboot normally shipped. ms_rboot will load a system image off disk when required and use that to boot a neighbour, rather than always use the current one. In addition to this, the resource map which is used to boot the transputer network, should not contain a reference to the I/O processor, as this is not an intelligent Helios node.