Helios - A Distributed Operating System

Perihelion Software Technical Report No. 2

Tim King

December 1988

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

Copyright (c) 1988,1989 Perihelion Software Ltd.

Permission to copy this technical note without fee is hereby granted, provided that the copyright message and this permission appears in all copies.

You may not:

1. Modify the Materials or use them for any commercial purpose, or any public display, performance, sale or rental;
2. Remove any copyright or other proprietary notices from the Materials;

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Contents

Helios is an operating system specifically designed for the transputer, and was intended right from the start to run on multiple processors. Although appearing similar to Unix at the user level the underlying implementation is entirely different in order to handle this. This technical note describes some of the major features.

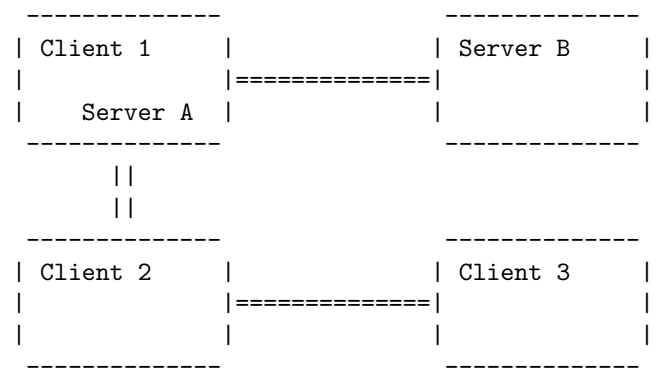
1 Underlying Primitives

Helios is based on the client-server model for operating systems, a technique which is widely used in many current systems. Inter-process communication is handled by message passing although this underlying mechanism is effectively hidden by layers of system software above it. A client process wishing to access a system resource, such as opening a file, sends a message to a server process requesting this action to be performed on its behalf. The server replies with another message indicating success or failure. Subsequently the client can read or write this file by sending further messages to the server.

This mechanism is convenient for a number of reasons. The server is responsible for handling the resource, for example keeping locks on files. This is easy to arrange when the server runs as a distinct process within the machine. The client may either behave in a simple fashion and send a message and then wait for the reply, alternatively the client may engage in asynchronous I/O and perform some other action while waiting for a reply message to arrive from a server. The interface for servers is consistent, thus making it simple for extra servers to be added to the system to handle any added hardware.

The underlying Helios design uses this client-server model, but with the additional feature that the processes which act as clients and servers may reside in different processors. The client always makes the same call to send a message to a server, but the actual delivery mechanism may either pass the message to a process in the same transputer or transmit it through any number of other transputers before reaching the final destination. The actual location of the destination process is unknown to the sender, as is the route by which it is sent.

Unix is a registered trademark of AT&T



In the example above, all the clients communicate with the two servers using identical calls. When Client 1 sends a message to Server A, the message is passed via a memory to memory copy. When Client 3 sends a message to Server B the message goes into the transputer running Client 2, then into that running Client 1 before entering the correct transputer and being routed to the Server B process. The route the message takes is transparent to the clients, which means that the topology of the network is also transparent.

Servers are located within the network of processors by name. A naming hierarchy is maintained by Helios, with a named network at the top level and processor clusters below this. Each processor within a cluster is also named, so that a server may be either uniquely identified by their full name or specified generically by a shorter subset of the name. Thus a specific server may be referenced as /NetA/Cluster5/00/tasks, while a generic name such as /tasks will cause the nearest server to be used. The location of a server is produced as the result of a distributed search, spreading in an ink-blot fashion out of all the links of the processors in the search path.

This scheme also allows an element of fault tolerance to be introduced. All messages sent by Helios have a timeout associated with them. If a message fails to get through within the specified timeout, the system automatically retries a number of times. If these all fail then a new distributed search is performed which attempts to identify a new route to the server.

2 Implementation

Helios is implemented as a true distributed operating system. Each processor node contains a Helios kernel, which handles memory management and message passing. Each node also contains two servers - the processor manager and the loader. The processor manager is responsible for process

creation within that processor and other housekeeping jobs, while the loader handles the loading and unloading of both program modules and resident libraries which are loaded on demand.

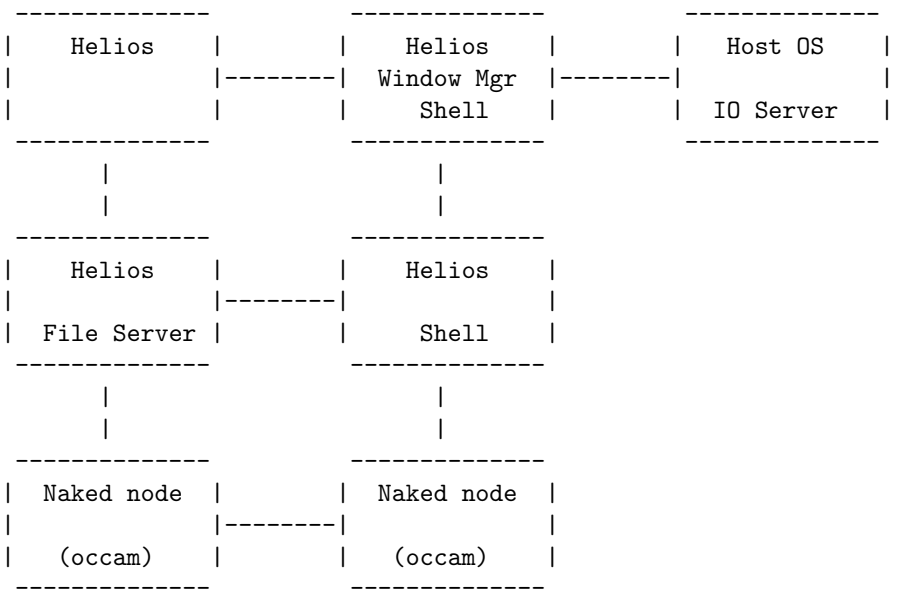
Other servers run on one or several processing nodes within the network. Some servers must run on nodes with particular hardware attached; for example the file system needs the disc device connected while a window manager must run in a processor with video memory attached. Other servers with no particular hardware requirement may be distributed to share the load equitably.

It is also possible to have processing nodes within a Helios network which do not run Helios all the time. One or more processors may be reset and loaded with a program written in a language such as occam. Links connected to these nodes are specified as "dumb" links and a program is written which runs under Helios and supports a private protocol down the dumb links. This provides the connection between the program running on the naked hardware and Helios. It is especially useful where specific hardware configurations must be used at certain times, but not at others. It is also useful when interconnecting T212 or M212 processors which are too small to run the Helios kernel.

Most transputer networks are normally connected to other computer systems which act as hosts. Helios treats these host systems just like a transputer processing node, and achieves this by running a program called the I/O Server within the host system. This program, written in C, causes the host processor to appear to the rest of the network just like another node running Helios. Messages sent to the link adapter connected to the host are replied to in the same way as messages sent to real transputer nodes.

The I/O Server provides support for Helios servers running on the host. As far as the rest of a Helios network is concerned, the host processor is a normal Helios node running servers. These are usually servers for consoles, file systems, serial ports and so on. They are implemented by the I/O Server communicating with the host operating system. In particular, the file server is implemented by mapping Helios requests onto the existing file system. This has the advantage of using the host filing system disc format, and also allowing access to any remote filing system supported over other networks such as Ethernet.

Any Helios network may contain any number of transputer nodes and host nodes. The transputer links may be regarded as a local area network connecting different hosts. A simple example is described below, where there are four Helios nodes, two naked nodes and one host system. One Helios node is running nothing except the base system, while other servers are distributed around the network.



3 User Interface

There are two main user interfaces within Helios. The first is the shell, which provides a command line interface at which commands and parameters may be typed. The other interface is the graphical interface provided by Xwindows, although the shell runs within an Xwindows window in this case as well.

The shell is a intended to be as similar as possible to the Unix csh. It provides pipes as a way of communicating between programs, and redirection of standard input, output and error streams. Jobs may be run in the background and shell scripts executed. The use of scripts is enhanced by the wide range of control structures available, such as while and foreach loops, conditional statements and so on. Values may be assigned to variables, and these expanded at a later date.

Particular attention has been given to the ease of use of the shell. The command names are identical to those in Unix, but the alias facility may be used to change the names that are typed or to provide shortcuts for commonly used commands and option combinations.

All of the commands entered into the shell are stored in a history list. Items may be retrieved from this list in two ways. In the first way, traditional Unix instructions such as !cc retrieve the most recent command in the list

that starts with the character(s) that follow the !, which in this case would be cc. Alterations to previous commands may be also be made using adaptations of this syntax; but most users prefer the second way of retrieving commands, which-is simply by using the cursor keys. Command lines may be edited using the cursor keys and other control key combinations chosen to be compatible with the Helios editor.

A further feature allows both commands and filenames to be completed by the system. At any point while typing a command or a filename argument the user may press ESCAPE. If the name is now uniquely specified the system will automatically complete it. If there are more than one possible completion an error is signaled; the user may then press CTRL-D and a list of the possible completion is printed

The commands supported by Helios include the standard Unix-like file manipulation commands such as ls, mv, rm and so on. These have a more general use than might as first be imagined, as all the servers respond to the same protocol. This means that when ls is used to examine the directory of a file server, the server is sent a request to open and read the file system directory. This is part of the general server protocol supported by all servers. The same request may be passed, for example, to the processor manager. The implementation of the processor manager provides a list of active tasks as the reply to a request to read from the directory, and so the command line

```
ls /tasks
```

will cause a list of the active processes on processor 00 to be listed. In the same way the loader can be examined, so for example

```
ls -l /loader
```

will cause the size of currently loaded modules to be displayed.

Other standard Unix-like commands provided include make with an identical syntax to the Unix command, the Micro Emacs editor and text processing tools such as diff and grep.

Helios is written in C, and a new C compiler conforming to the proposed ANSI standard was written as part of the Helios project. A number of other languages have been provided by third parties including FORTRAN, Pascal, Modula2 and occam. Programming tools under development include support libraries and a source level debugger.

4 Parallel Programming

Helios was designed from the outset to support multiple processors and it provides built-in mechanisms for handling the resource of multiple processors. One of the design aims for Helios was ability to run the same binary code on different transputer installations, independent of the topology or the number of processors.

Each computer running Helios runs a network manager, which is responsible for initially booting the network. This server also monitors the network, detects when a processor has crashed and attempts to reboot it if possible. The network manager is handed a blueprint file which describes the resources within the network. This includes the number of processors and how they are connected, but also other information about each processing node such as the type of processor, amount of external memory, existence of video memory and so on.

The network manager is complemented by the Task Force Manger or TFM. This is responsible for managing the resources indicated by the network manager. It is handed a similar blueprint file whenever a job is to be executed, and attempts to match the resources required with those available. For example, a job may require three processors connected so that the central one has video memory. The TFM decides on a suitable mapping from the requested network to that available in the current system. The decision on the way in which the mapping is done depends on the current load in the system as well as the physical resources available.

The normal interface to Helios is via the shell described earlier. When requested to do so, the shell will execute commands by passing them to the TFM rather than simply spawning a child process in the same processor. In this way a simple shell command of the form:

```
ls | more
```

will result in the ls command running in one processor, the original shell in another and the more command in yet another, assuming sufficient processors are available. Output from ls is sent to more via a pipe between the two processors. Pipes are implemented as direct message passing between the two processors, with efficiency enhanced by removal of redundant copying.

More complicated examples are possible though the use of a special language called the Component Distribution Language or CDL. This is a type of job control language which allows different components of a parallel program to be described. The description includes the components of the program, the way in which they are interconnected and any special resources needed by

each component.

An example of the use of CDL can be shown using the standard technique of dividing up a program into a master task which distributes work, and a number of slaves which handle work in parallel. Two programs, master and slave, are separately compiled. The CDL system is then used to specify the parallelism. The CDL compiler would be given an input line such as

```
master [5]||| slave
```

which will cause Helios to load a copy of the master, the load balancer and five copies of the slave into suitable spare processors. Pipes are provided to link the master to the load balancer, and the balancer to each of the slaves.

The CDL is in general more complicated than this simple example, because of the way in which attributes can be specified for each of the different parallel parts, but this simple case shows how it allows tasks to be distributed among processors. Note that the use of the CDL and pipes is independent of the language used; indeed master and slave could be written in different languages.

occam is a trademark of the Inmos Group of Companies.
X Windows System is a trademark of MIT.
Helios is a trademark of Perihelion Software Ltd.