The Helios File Server
Manual

PERIHELION SOFTWARE LTD

October 1990

Copyright

This manual is Edition 2.0, October 1990.

Printed in the UK.
PDF generated in AT (Vienna).

Acknowledgements

The Helios File Server was written by N.Garnett, C.Selwyn, A.Ishan, H.Ermen and J. Powell. Helios software is available for multi-processor systems hosted by a wide range of computer types. Information on how to obtain copies of Helios software is available from Distributed Software Limited, The Maltings, Charlton Road, Shepton Mallet, Somerset BA4 5QE, UK. (Telephone: 0749 344345.)

Contents

Chapter 1

Introduction

The `Helios File Server (HFS)` is a powerful tool designed to deal with mass storage random access devices in multi processor networks. You can use the file server to perform tasks which require high data transfer rates, particularly when you combine it with transputer hosted SCSI interfaces, or other disc interfaces. Some typical uses for the file server would be, for example, in the fields of image processing, real time databases and distributed databases. When it is in daily use, the file server will improve productivity and reduce your need to use host processors which are based on old technology. A tape server is also included in the package, giving you access to tapes on the same interface as the discs. This makes it easier for you to back up your directories. For example, you might use the `tar` command, which will be described later in this guide, to archive your material.

1.1 Main Features

`(1)` The `disc organisation` of the Helios file server is based on that of the Berkeley Fast File System of `BSD version 4.2` and later. Extensions have been added to include the Helios protection mechanisms and to improve the overall performance of the file server, which means that the best possible use of disc capacity and bandwidth is now guaranteed.

`(2)` A `cache` gives you access to data out of memory, when it is present. Using `read ahead` and `write behind`, the disc access times will move outside the client's critical path, allowing data to be transferred between the client and the file server at a very high rate. It is possible to reach data transfer rates of over 500Kbytes per second between processors, and 2 to 3 Mbytes per second within a processor, to and from the `cache`.

`(3)` `Multi threading` within the server gives access to many clients at the same time. This also means that high data transfer rates can be reached and maintained at a speed comparable to that of the hardware.

`(4)` The `block size` of the file server is set to 4Kbytes. This makes large disc transfers possible, increasing the band width in the process, by allocating adjacent file blocks to adjacent disc blocks. The disc is divided into several `cylinder groups` of equal size which can be individually controlled for the allocation of blocks. The `block allocation algorithms` make sure that there is a good compromise between the clustering and the spreading of information, so that any wasted empty space at the end of a file is kept to a minimum.

`(5)` The Helios file server differs from the `BSD Fast File System` in that it does not use `hard links`, and that `inodes` are stored in the parent directory. This means that directory searching, access rights calculation and update will operate more quickly. You can still achieve the same effect as `BSD links` by using `Helios symbolic links`, which allow the file server to store links to objects both inside and outside the volume of the disc.

`(6)` The Helios file server is used as a `device independent` Helios server. This means that it can run on any node in a processor network, more than one file server may be present in a system, and all these file servers may be accessed in a `transparent` manner by clients.

`(7)` The Helios file server is `fully parameterised`, which means that you can use it with a wide range of hardware. The `device interface` is only bound to the server at run time, allowing you to run the server on any hardware for which you can write a device.

`(8)` In addition to interfaces for the SCSI and M212 disc devices, an inexpensive alternative is provided in the form of a `raw disc I/O server interface`. This means that the Helios file server can make use of a free partition on the disc of a host such as a PC. Support for access to a tape device using the same I/O driver is also included.

`(9)` The HFS software package comes complete with a number of `utilities` to make it easier to use the file server and the file system consistency checker. There are mechanisms, (based on capabilities), for the protection of files. The file server can also be used within the protection mechanisms of Helios, version 1.2. (This will prevent other people from damaging your files when working in a multi user environment.)

1.2 Tape Support

HFS provides support for access to a tape. This comes in the form of a file which you can use with the normal `I/O file access functions`. To make this option work, you will need to define a second volume in the `devinfo.src` file, with the type `raw`. The name of the tape file will be the same one that appears in the name entry of the volume. Remember that the `raw` volume *must* be the second volume. Note that a tape does not contain the Helios Filing System, so it should be specified as of type ‘raw’ in the devinfo file. The normal way to gain access to the tape system is to use the `tar` command, which is described in Chapter 5 of this guide.

1.3 SCSI and M212 Device Support

The Helios File Server is hardware independent. It uses device drivers to access many of the most popular transputer disc interfaces. Several possible configurations are currently available for the Helios File Server. Please refer to the separate, loose leaf notes entitled: 'The Helios File Server: Installation Information', for an explanation of product specific hardware requirements.

1.4 Software Requirements

You will need to use version 1.1A, or a later version of The Helios Operating System to run the file server. The commands for manipulating the protection mechanism are documented in this guide, but the relevant software is not included in this package because you will find that it is already provided as a part of the standard operating system release.
Note: For a list of the files contained in the Release Disc, please refer to the separate, loose leaf notes entitled: 'The Helios File Server: Installation Information'.

Chapter 2

Installation

These are the steps you should take to install the Helios File Server in your system:

1. Copy the software into your system.
2. Configure the software to run with your hardware.
3. Format and initialise the hardware to work with the file server.
4. Run the file server.

(The rest of this chapter describes in detail how to work through each of the above stages.)

2.1 How to copy the disc

Before you can install the file server, you must first make a backup copy of the release disc. Copy the directories `lib`, `bin`, and `etc` from the floppy disc into the equivalent Helios directories. You will not need to copy the `devs` directory unless you intend to write your own device driver. You will not need to copy the `msdos` directory either, because the device drivers it contains only have to be run once. If you would like to find out more about this, read The Perihelion Software Technical Report No.19: The Helios Interface To Hardware Devices. Users of Helios version 1.2 onwards can use the `Loadpac` installation program to load their software automatically.

2.2 The DevInfo File

Before you run the file server, make an appropriate entry in the `devinfo.src` file. (You will find that the example file contains entries for each of the currently supported devices.) Now, find the `discdevice` which corresponds to the device you want to use, and edit the drive parameters to match your disc. (Please refer to the note at the end of this section.) You must also decide how much `cache memory` the file server should be allowed to use. The practical minimum is `100k`, and any value of over 1 Mbyte will allow most of your working files to be `cached`. When you have edited the `devinfo.src` file, you should compile it again by using the following command:

```
gdi /helios/etc/devinfo.src /helios/etc/devinfo
```

Note 1: Currently, several possible configurations are available for the Helios File Server. For details of hardware specific entries, please refer to the separate, loose leaf notes entitled: 'The Helios File Server: Installation Information'. There is also an example of the `devinfo.src` file in the appendix of these notes, and on page 8 of The Perihelion Technical Report, Number 20: 'Device Configuration'.

2.2.1 Entries for the Raw Disc

The raw disc service is provided by a server within the Helios I/O server. This provides you with a directory called `/rawdisc`, containing a list of partitions which are available for the file server to use. you will need to `initialise` the parameters in the `fileserver` and `discdevice` structures of the `devinfo.src` file, to set up the partitions you intend to use. Run the `makedisc` program and the disc parameters, (such as sectorsize, and the number of sectors, tracks and cylinders), will be displayed on your screen. If you have already run the `makedisc` program once, simply rerun it to obtain this information.

2.2.2 The PC Raw Disc Service

If you are using a `PC`, the `rawdisc` directory will only contain one partition, which you must `define` by making the entry:

```
rawdisc_drive = 'x'
```

in your `host.con` file. (The `'x'` in our example above represents the drive you intend to use. It might be drive D for instance.) You will need to set the **controller** field to `0` in the disc device entry for the file server you want to use, (i.e. The Helios file server), because `rawdisc` only contains a `0` file.

2.3 Formatting and initialisation

Before you can run the file server on a disc or on o partition, it must be formatted and then initialised with the file server data structures. For details of the `fsformat` command, see Chapter 3 of this guide. For hardware specific information, read the separate notes entitled 'The Helios File Server: Installation Information'.

Once formatted, the disc or partition must be initialised before use. This is achieved by starting the file server with the `-m` option. **For example:**

```
/helios/lib/fs -m <name>
```

Note that the name must be the same as the name of the 'fileserver' entry in the `devinfo.src` file. It is also the name you associate with the disc or partition you have just formatted. After initialisation, you can run the fileserver.

2.4 How to run the File Server

The Helios File System can be booted with the following set of options:

```
fs [-m|-f|-b|-n|-s] <device>
```

`-m`	Call the Format routine.
`-f`	Perform Full Checks (default).
`-b`	Perform Basic Checks only.
`-n`	Bypass the checker completely.
`-s`	Create a new superblock.

If you want to run the file server `locally`, you should use the command:

```
/helios/lib/fs <name> &
```

<`name`> is the file server 'entry' name in the `devinfo.src` file.

Caution: If the CDL shell variable is set, the `task force manager` will automatically place the file server on a processor of its own choice.
If you want to run the file server on a specific processor, use the command:

```
remote -d <FSMachine> /helios/lib/fs <name>
```

This command would be particularly useful when you want to run the file server on a processor which is adjacent to the controller hardware. As an alternative, you might run the file server from the `initrc` file, by changing the command to:

```
run -e /helios/lib/fs fs <name>
```

You will now have the file server running, and you can gain access to it as a server by using the volume name which you gave in the `devinfo` file.

Chapter 3

Commands

This chapter consists of an alphabetical list of the commands you can use with the Helios file server. Most of these commands are supplied on the file server release disc, (See the separate, loose leaf notes entitled:'The Helios File Server: Installation Information', Table 4.1), and the remainder are included in version 1.2 onwards of The Helios Operating System.

chmod

Purpose: To alter the access matrix of an object.

Format:

```
chmod {[vxyz][+-=][rwefghvxyzda] <file>}
```

Description:

The `chmod` command is supplied with version 1.2 onwards of The Helios Operating System, and you would use it to alter the access matrices of objects within servers such as the file server and the ram disc server, which can use protection. The command line is made up of a sequence of matrix modifiers and object names. A matrix modifier starts with a category letter, which is followed by an operator and then finally, there is an access mask specification. The category letter will be one of the following letters: `v,x,y,` or `z`. (The letter `v` is the symbol for the rights of the owner, `z` is for public rights, and `x` and `y` are for friends.) As the operator, you would need to tell your system how you want to combine the new access mask with the existing one, for the category you are working on. The example below shows you how to do this:

`+` The new mask is added to the current mask
`-` The new mask is deleted from the current mask
`=` The current mask is replaced by the new mask

To specify the access mask , give the letters which stand for the access rights in any order. The matrix changes you specify will affect all the subsequent objects in the argument list. Some more modifiers may also appear, and the effect they have will be combined with that of previous modifiers. The access matrix of an object will only be changed if the client has alter rights to that object.
For example: Imagine that you have a file called `fred` which you want to make inaccessible to the public, but still available to all other clients. You could use the following command:

```
chmod v-w x-w y-w z-rw fred
```

Now, imagine that you have a directory called `adm`, and you want to make it inaccessible to all clients except those with `x access rights`. Use the command:

```
chmod v= x=rwvxyzda y= z= adm
```

If you want to know more about protection, refer to The Helios Operating System, chapter 13: ‘Protection and Authentication’.

See also: refine

Note: For the Disc Editor, see Chapter 4.

fs

Purpose: The binary object of the Helios File System.

Format:

`/helios/lib/fs -m` <`device`> (To create a new file system.)
`/helios/lib/fs` <`device`> (To boot the file server.)

Description:

This command contains the `binary code` of the file server. If you call it up by using the option `-m`, the `formatting routine` is carried out, so that a new file system is generated on the disc. (The program ends when this step is completed.) If you call up `fs` without using the option `-m`, the file server will be `booted.` (The <`device`> in the format illustrated above corresponds to the fileserver entry in the `devinfo` file. If you want to place the server on a processor of your choice, you can use the `remote` utility. There are two further options: `-b` and `-f`. These options call up the two different checking modes of the file system checker. (Please refer to Chapter 6, section 2 for a detailed description of the checking modes.)

fsformat

Purpose: To format a disc partition.

Format:

```
fsformat [-p<number>] [-i<number>] [-t<number>]
[-c<number>] devname
```

Description:

This command will format a specified disc partition according to the parameters supplied. You would use `devname` as the discdevice entry name of the device you are going to use. The flag `p` introduces the number of the partition which will be formatted, and the default is `0`. There are three other `flags`, as follows:

`-i` introduces the `interleave factor`, (Default 1)
`-t` introduces the `track skew` to be used, (Default 0).
`-c` introduces the `cylinder skew` to be used, (Default 0).

Remember that variations in the format of a disc can make a significant difference to the performance of the file server. The most important thing for you to consider here is the interleave of sectors on a single track. If the interleave is too small, it might make the device wait for an entire revolution between each sector, and if it is too large, it could cause unnecessary latency. The track skew is the difference between the positions of the first sectors on adjacent tracks in the same cylinder, and its purpose is to allow for any head change latency,(although this is usually zero). The `cylinder skew` is the difference between the positions of the first sector on the last track of the cylinder `n`, for example, and the first sector on the first track of the cylinder `n+1`. You can use the cylinder skew to change the position of the first sector of track `0` to the start position on each cylinder. You can also use the cylinder skew to make allowance for seek latency between adjacent cylinders. (The `disc editor`, which will be described in Chapter 4, includes some commands designed to let you experiment with the format of a disc.)

Note: Some disc drivers may not use some of the parameters supplied for the format of a disc. Remember that formatting a disc can be a lengthy process, depending on the size of the disc, speed and driver implementation.

fsync

Purpose: Toggle between partly and fully synchronous modes.

Format: `fsync fileserver -[a/s]`

Description:

If you want to change the `cache` behavior of the file server, you will need to use the `fsync` command. When the file server is operating normally, it will only write blocks of data from the `cache` to disc

`(a)` When it needs to re-use the cache memory.
`(b)` When you use a `sync` command , or
`(c)` Automatically, at intervals of 20 seconds.

This is called `asynchronous` behaviour, and it is the default behaviour for the file server. There is an alternative way for the file server to behave, and this is called `synchronous`, which means that the file server will write the data to disc immediately. You can pre-set this alternative behaviour in the `devinfo` file, but you can also change from the default to the alternative dynamically, by using the `fsync` command. If you use this command, remember that the first argument you use with it must be the server name of a file server volume, and the second argument will be either `a` or `s`, depending on whether you are going to use the `asynchronous` or the `synchronous` mode.

Example: `fsync /raw -s` change fileserver /raw to synchronous mode.
See also: sync

gdi

Purpose: To compile a devinfo file.

Format: `gdi source destination`

Description:

This command is a `compiler` for the `devinfo` file. Its source file should be a description of `devinfo` in the form of text. When the file has been compiled, it will be placed in the destination file. When the file server is searching for the compiled file, it will look in the following files:

```
(a)  /rom/devinfo
(b)  /loader/DevInfo
(c)  /helios/etc/devinfo
```

(The program will tell you if you have made any errors in the syntax.)

To compile a `devinfo.src` file, use a command line like the example illustrated below:

```
gdi devinfo.src devinfo
```

This will create a binary date file `devinfo`, which the file server can use to obtain the disc characteristics.

loadm2

Purpose: To load M212 firmware.

Format: `loadm2 linkno`

Description:

You would use the `loadm2` command to load `firmware` onto an M212 disc controller. The argument to use with this command will be the number of the link attached to the `M212`, on the processor which is running the `loadm2` program. The `M212` should be in `reset state`, and ready to boot from its link. The `loadm2` program will look for the firmware in the file `/helios/lib/b5multi.b2`. This code is supplied with the M212 and it should be copied to the `/helios/lib` directory.

refine

Purpose: To refine and generate a capability.

Format: `refine [+-=][rwefghvxyzda] object...`

Description:

The `refine` command is supplied with The Helios Operating System, version 1.2 onwards. This command will give you the name and the capability, in coded form, for each individual `object` in a list. The capability is encoded in the form of 16 characters. These are `concatenated` with the name of the file, and prefixed with a @ character. The `Helios System Library` will recognise this format for file names, so it will allow you to use such a `string` of 16 characters in situations when it would normally expect to be given the name of a file. In particular, these strings can be placed in a shell variable or in an environment variable, and they can be compiled into programs. The `refine` command also allows you to change the `access mask` which is stored in the `capability`, by taking steps similar to those you would use with the `chmod` command, described earlier in this chapter. (Once again, for further reference you could read The Helios Operating System, Chapter 13: 'Protection and authentication').

See also: chmod

sync

Purpose: To flush the cache of the file server.

Format: `sync <fileserver>`

Description:

When you use this command , the file server will `flush` any blocks of data which are out of date from its `cache` to disc, and this will `synchronise` the cache and the disc. You only need to use one argument, and that will be the `server name` of the file server to be synchronised.

Example: `sync /raw`
See also: **termfs**

Note: For **tar**, see Chapter 5

termfs

Purpose: To stop a file server.

Format: `termfs <fileserver>`

Description:

Use this command to terminate the file server. Once again, you only need to use one argument, which will be the `server name` of the file server to be terminated.

See also: sync

Chapter 4

The Disc Editor

The `disc editor` is a simple program for examining a Helios file server disc or partition. You can call it up by using the command:

```
   de <fsname>
```

(Remember that <`fsname`> must be the same as the name of the `devinfo fileserver entry` of the disc you are going to examine.) The disc editor uses just one `block buffer`. You can place disc blocks in this buffer to read, examine and alter them before finally `writing` them back to disc. (The commands you can use with the disc editor are listed below.)

4.1 A read-write check of the disc

Format: `a`

Caution: (When you use this command, you will wipe out any data that was on the disc.) This command will perform a complete `read -write` check of the disc. The disc editor will then tell you if any of the blocks of data are `bad`. When you have `initialised` the disc by using the `m` command, you can use the list of `bad` blocks with the `k` command to prevent the filing system from using them.

4.2 A read-write test on the disc

Format: `t`

Caution: (You should only use this command on empty discs, because it will destroy any filing system that is present.) When you use this command, a prompt will appear on the screen for you to give the `data size` you are going to use. (The data size is measured in Kbytes, and the default size is 64 kbytes.) The data will then be written to the disc, and read back, starting at `block 0`. (This operation will be timed, and the results will then be reported to you on the screen.) If you use the `m` command to vary the format of this region of the disc, and then use this `t` command to perform a read-write test on the disc, this will help you to find the best format for your needs.

4.3 Mark bad blocks

Format: `k` <n>

This command marks the given block as allocated, so that it will not be used by the file system.

4.4 Set the block size

Format: `b` <n>

You can use this command to set the `block size`. (<n> represents the size you choose.)

4.5 Interpret the current block as a cylinder group block

Format: `c`

This command will give you a summary of the `cylinder group`. It also gives you the `allocation map` and the cylinder group's copy of the `super block`.

4.6 Display the contents of the current block

Format: `d`

When you use this command, the contents of the current block will be displayed in `dump` format.

4.7 Print out a directory entry in the current block

Format: `f` <n>

Use this command to print out a particular directory entry in the current block. (<n> represents the entry you want to print out.) Your printout will also show you which are the `direct data blocks`, and the first `indirect data blocks`, so this command gives you more information than the `l` command, which will be described later in this chapter.

4.8 Print a description of each command

Format: `h`

If you would like to see a brief description of each of the commands you can use with the disc editor, just use this command and it will be printed out for you to study.

4.9 Interpret the current block as a directory block

Format: `l`

Use this command to have the current block interpreted as a `directory block`, and to display a list of the `valid directory entries`.

4.10 Format cylinders on the disc

Format: `m`

Caution: (When you use this command, you will wipe out any data that was on the disc.) With this command, you can format a number of cylinders on the disc. When you have entered the command, a prompt will appear on the screen for the cylinder extent, the interleave and the track and cylinder skews. You would normally use this command to test disc devices, but you can also use it in conjunction with the `t` command to find out the best possible interleave and skew for the disc.

4.11 Quit the disc editor

Format: `q`

Use this command when you want to end your session on the disc editor.

4.12 Read a block

Format: `r <n>`

<n> here represents the `block number` you want to read. If no `argument` is given in the command, the current block number is used. (The current block number is set by any argument you use with an `r` command or with a `w` command.)

4.13 Interpret current block as disc summary block

Format: `s`

The current block will be interpreted as the `disc summary block` when this command is used. This means that you will be able to see the root directory entry and the `allocation summaries` for the whole file server and for each cylinder group. The disc summary block will always be block number `1`.

4.14 Toggle the block valid flag

Format: `v`

If you have altered a block by using the `=` command, the disc editor will not allow a new block to be read until (a): the original block has been `written` back to disc, or (b): you give the `v` command to invalidate the changes you made to the block.

4.15 Write a block

Format: `w` <`n`>

With this command you can write the current block, or another block of your choosing. (The <`n`> represents the number of the block you choose.)

4.16 Toggle the write protect mode

Format: `x`

The default setting for the `write protect mode` is `ON`. This means that the disc editor will not write to the disc. You can use the `x` command to `switch off` the mode.

4.17 Zero a block

Format: `z`

This command zeros the contents of the current block buffer.

4.18 Report size and number

Format: `?`

Use this command to obtain a report of the disc size, the block size and the current block number.

4.19 Set the current block position

Format: <`hex`>

You can set the current block position by using a `hexadecimal number` by itself. The number must start with a digit, so those values which would normally start with the letters `a` to `f` should be prefixed with a `0`.

4.20 Set the byte

Format: `=` <`hex`>

You would use this command to `set the byte` at the current block position. Use the previous command to set the byte position for this command.

Chapter 5

The 'tar' File Archiver

5.1 Background Information

`tar` provides you with a way to store many files into one single `archive`. It is useful for making backup copies or for packaging up a set of files that you want to move to another system. There are several ways for you to store your `archive`:

(a) You could keep it in another Helios file.

(b) You could store it on an I/O device such as tape, floppy, cartridge or hard disc.

(c) You could send it over a network.

(d) You could `pipe` it to another program.

The `tar` file archiver was originally available at the same time as version 7 of Unix, and it has continued basically unchanged. It has been proposed as the standard format for the interchange of files among systems which conform to the IEEE P1003 Portable Operating System (POSIX) standard. The current version of `tar` gives you some of the extensions which were proposed in the draft standards for the P1003, including owner and group names and support for named pipes, fifos, contiguous files, and block and character devices.

5.2 Main Features

5.2.1 tar Options

If you are reading an archive with the current version of `tar`, you will be able to continue even if `tar` finds an error. (In the past, `tar` would always stop if it found `checksum errors`, and you would need to use the `i` option so that you could continue your reading.) You can `specify tar` options in two ways:

(a) You can use normal Unix conventions. (This means that each option you specify must be preceded by a `dash` (like this `-`), and if you use an argument, it must directly follow the option you are specifying. You can combine several options behind one dash (`-`), but only if you are not using any arguments with those options.

(b) For compatibility with the `Unix tar program`, You can also specify the options as `keyletters`. (This means that you put all of the option letters into the first argument to `tar`, without using the dash (`-`). Any argument you use with your options must be placed inside the second or third (etc) argument to `tar`.

`For example:`

Normal: `tar -f arcname -cv file1 file2`

Old: `tar fcv arcname file1 file2`

You must include at least one of the options `-c, -t, -d or -x`, but you can use any of the other options whenever you choose.

5.2.2 A list of the tar options and their effects

Option **Effect**

`-b` *N* This option specifies a `blocking factor` for the archive. The block size will be *N x 512* bytes. Larger blocks typically run faster, and they will allow you to store more data on a tape. The `default blocking factor` is normally 20, and it is set when `tar` is compiled. The maximum block size is limited only by the memory space available, and by the ability of the device containing the archive to `read` and `write` the size you want to use.

`-B` If you are reading an archive, you can use this option to `re block` it as you read it `tar` normally reads each block with a single `read` system call, but this does not work when you are reading from a pipe, because the command read will only give you the data that has arrived `so far`. If you use the `-B` option, `tar` will carry out the `read` command repeatedly to fill out to a record boundary, rather than reporting an error. (The `-B` option is in fact the `default` when you are reading an archive from standard input.

`-c` Use this option to create an archive from a list of files.

`D` When you are creating an archive, this option makes sure that you can `dump` a directory without dumping the files inside it.

`-f` *F* Use this option to specify the `filename` of the archive. If your filename was represented by a dash (-), then the archive would be read from the standard input, or written to the standard output. If you are not using the `-f` option, and the environment variable `TAPE` exists, then you would use its value as the filename. (Otherwise, a default filename which was determined when `tar` was compiled is used. **For example:** `tape`, preceded by a forward slash (/). The default is normally set to the `first tape drive`, or to another transportable `I/O` medium on your system.

`-h` If `tar` comes across a `symbolic link` when you are creating an archive, this option will make sure that you dump the file or directory that the link is pointing to, rather than dumping it as a symbolic link.

`-i` This option tells `tar` to ignore blocks of `zeros` in the archive. (Normally, a block of zeros indicates the end of an archive, but they can sometimes be found in the middle of a damaged archive, or at intervals in an archive which was made up of several others, so in these cases the `-i` option allows `tar` to continue unhindered.) Note that with this option set, `tar` will read right to the end of your file, thus eliminating any possible problems with multi file tapes.

`-k` Use this option to ensure that when you extract a file from an archive, you keep the existing files, rather than overwriting them with the versions from the archive.

`-l` When you are `dumping` the contents of a directory to an archive, the `-l` option allows you to stay inside the local file system of that directory. This option will only affect the files you dump because they are in a `dumped directory`, and not those files which are named on the command line. (The latter are always dumped, and they can be from various file systems.) The `-l` option is useful for making `full dump` archival backups of your file system. Any files which are skipped while you are using this option will be listed on the standard error..

`-m` When you are extracting files from an archive, use this option to set the modified `timestamp` of each file to the current time, rather than extracting the timestamps individually.

`-o` You would rarely need to use this option, but if you are creating an archive, it will write an `old format` archive which does not include any information about directories, pipes, fifos, contiguous files or device files. In most cases you will find that a `new format` archive can be read by an old format tar program without difficulty.

`-p` Use this option when you are extracting files from an archive, to restore them to the same `permissions` that they had inside the archive. (If you do not specify the `-p` option, the current `unmask` will limit the permissions of the files you extract. See `unmask` for further details.

`-R` Each time that `tar` produces a message, this option will print the record number inside the archive where the message occurred. You would find this particularly useful when you are reading damaged archives, because it helps you to pinpoint the damaged sections.

`-s` When you want `tar` to extract a number of files from an archive, or to list their names, you can use the `-s` option to have your list of filenames sorted into the same order as the tape. This means that you can call for very long lists even if you are using a small machine, because the entire list need not be read into memory at once. If you run `tar -t` on your archive and edit its output, this will also create a sorted list.

`-t` If you want to draw up a table of contents for one of your existing archives, use this option. If file names are specified, only the files which match those names will be listed. (The listing appears on the standard output.)

`-T, F` Rather than specifying filenames or regular expressions as arguments to the `tar` command, this option ensures that they are read from the file `F`, at the rate of one per line. If the filename you specify is `-`, the list is read from the standard input. When you use this option in conjunction with the `-s` option, you can process longer lists of files, and you can `pipe` them to `tar`.

`-v` This option causes `tar` to be `verbose` about the files which are being processed or listed. (Archive creation, file extraction and differencing are normally silent, and archive listing simply gives filenames.) The `-v` option produces a listing of the `ls -l` type. The output from this option appears on the standard output, except when creating an archive, in which case it goes to the standard error output. (This is because the new archive might be on the standard output.)

`-x` Use this option to extract files from an existing archive. If filenames are specified, only files which match the specified names will be extracted. (Otherwise, all of the files in the archive will be extracted.)

5.2.3 Archiving files and directories

When you have chosen your options, you must give `tar` the name(s) of the file(s) you want it to work on. (`tar` can read a list of names from a file if you use the `t` option.) If you specify the name of a directory, `tar` will process it and all of the files it contains recursively. If you specify a full path name when you are creating an archive, it will be written to that archive without the initial forward slash (/), so that those files can later be read into a location different from the one where they were dumped, and `tar` will print a warning. If you attempt to extract files from an archive which contains full path names, `tar` will only extract them in relation to the current directory, and once again, a warning message will appear. When you are extracting or listing files, `tar` will treat the filenames as regular expressions, mainly using the same syntax as the shell. The shell itself will separately match each substring between a forward slash (/), but `tar` will match the whole string at once, so do not be surprised if some anomalies occur. (**For example**, a star (*) or a question mark (?) can match a forward slash (/).) If you want to specify a `regular expression` as an argument to `tar`, you must quote it so that the shell will not expand it.

Chapter 6

The Helios File System Checker

6.1 What the file system checker checker can do

This chapter describes the standard integrated version, `(1.1-1)`, of the file system checker. The checker is a maintenance tool which protects data stored on hard disc from damage in the event of a power loss, a hardware malfunction or a head crash, for example. Its purpose is primarily to gain access to a filing system after such a disaster, and secondly to discover and repair any corrupted parts of your filing system. It should be noted at this point that if a comprehensive repair of damaged files and sub directories is required, the `File System Interactive Maintenance Tool` should be used.(See section 6.6 of this guide for further reference.)

6.1.1 How the file system checker starts up

The checker is built into The Helios File System HFS, so it is automatically started each time the file system is booted. It checks and repairs the file system without user interaction. The checker normally runs in `silent` mode. This means that it simply reports to you as it passes through the different phases of the checking process. If it finds any damage on the disc it reads it as an error, and reports it by writing a message in the server window on your screen. When the checking process is completed, the file server will start up, and you will be able to gain access to your filing system in the normal way.
Note: If you attempt to use the file system checker with a damaged disc controller or to check a non formatted disc, for example, you may not be able to start it until the original cause of the damage, such as a hardware malfunction, has been solved.

6.2 The checking modes

There are two different `checking modes`. `BASIC CHECK` is the default mode, which automatically starts when the file server is booted. In this mode, the file system checker tests the basic data structures on the disc, such as those used to manage the allocation of blocks, for example. The second mode is `FULL CHECK`, which scans the complete directory tree and tests each entry, whether it is a file, a symbolic link or a sub directory entry. If the default mode discovers an error on the disc, the `FULL CHECK` mode automatically takes over to search for any further damage.

The Helios File System can be booted with the following extended set of options:

```
fs [-m|-b|-f] <device>
```

1. `-m` Call up the format routine.
2. `-b` Perform only basic checks. (default.)
3. `-f` Perform full checks.

6.3 Understanding HFS data structures on disc

This section provides you with a brief description of the main data structures which are managed by The Helios File System (HFS), and tested and corrected by the Helios File System Checker. This will make it easier for you to understand the basic principles of the checker and to follow the different phases of the checking process.

6.3.1 Disc layout

The Helios File System divides a disc logically into sections of equal size, each containing the same number of blocks. These sections are called `cylinder groups`. This type of disc layout has several advantages. The use of algorithms for block allocation and the creation of new directory entries and files ensures space efficiency on the disc. There is minimal fragmentation of disc space, and the best possible medium access times are achieved because the data is organised in such a regular pattern on the disc.
The information required to manage a cylinder group is stored in one data block, which is called an `info block`. Each cylinder group has its own info block, and it is placed with a fixed `relative rotational offset`. The most important data structure is called a bit map. This is an array of bytes which keeps a record of which blocks are allocated or free within the specific cylinder group, and it forms the major part of each info block. In addition to this, the `allocation summary` of a cylinder group will reveal the number of free blocks and the number of sub directories which are stored in that group. Another data block which has a special purpose is called a `summary block`. This collects the allocation summary from the info block of each cylinder group, and it also collects the allocation summary for the file system as a whole. This information allows the file server to decide where to allocate new blocks on the disc quickly and with space efficiency when you want to create a new file, for example.
The `super block` is where the fixed parameters of the established filing system are stored. These parameters include the number of cylinder groups to be used,

the size of each cylinder group and the rotational offset for the placement of info blocks in each cylinder group. A copy of this super block is stored in each info block. Under normal operating conditions, the contents of the superblock should never be changed once a disc has been formatted. See Figure 1 for a simple overview of the layout of a disc.

```
Block Number
------------


            ------------------------
0:          | boot block           |
            |_____________________ |
1:          | summary block        |
            |_____________________ |____cylinder group 0___
2:          | info block of the    |                     ||
            | first cylinder group |                     ||
            |_____________________ |                    \||/
3:          | first data block     |                     \/
            |_____________________ |
            \                      /
            /_____________________ \____cylinder group 1____
            | info block of the    |
            | second cylinder group|
```

Fig.1: An overview of the layout of a disc.

6.4 Directory entries and files

The information required to describe a file or a sub directory entry is stored in a data structure called an `inode`. This information includes the date of creation, the size (measured in bytes) and a `type flag` to indicate whether the entry describes a sub directory, a symbolic link or an ordinary file. The access matrix and the number of blocks allocated to a particular entry are also included.
References to data blocks are stored in `direct blocks`. To maintain space efficiency, only a limited number of these are created within an inode. When the system has to manage a large file, a `single indirect block` is allocated, which only contains references to data blocks. In the same way, a very large file which takes up hundreds of megabytes is described by using a `double indirect block` . This is noted in the inode. The inode stores references to indirect blocks, which in turn refer to data blocks. This arrangement allows the system to deal with a wide range of file sizes without wasting space on the disc.
There is a special type of data block which is called a `directory block`. This is stored by an inode of the `directory` type. (A sub directory entry.) A directory block only contains inode data structures, and this allows The Helios File System to build hierarchical directory structures.
When the file system checker is in `FULL CHECK` mode, it searches the whole directory tree, starting with the inode of the root directory. It checks each

entry and corrects any errors it finds. If the damage to a particular entry is so great that it is rendered unusable, its corresponding inode is deleted when the checking process is completed.

6.5 The different phases of the checking process

The checking process moves through four distinct phases.

6.5.1 Phase 1

During this phase, the file system checker is in `BASIC CHECK` mode. Firstly, the checker tests the validity of the super block data structure. (It finds this in the info block of the first cylinder group.) Next, the checker compares the super block with the copies which are stored in the info blocks of the other cylinder groups. It then compares the bit map of each cylinder group with the allocation summaries. (You will remember that these are stored in the info blocks and in the summary block.)
Note: Make sure that you run a final `sync` phase before terminating the file server. (During the `sync phase`, all modified data blocks which are stored in the buffer cache are written to disc.) If this is not done, the information in the bit maps and the summary block will not match when you start the file system checker.
Finally, the checker inspects the `root directory inode`, which is part of the summary block. It does this to make sure that you can gain initial access to the root directory of the filing system. If the checker detects no errors on the disc, then the actual checking process is completed at this point. However, if an error is found, the `FULL CHECK` mode starts up automatically, and the following three phases are performed.

6.5.2 Phase 2

During this phase, the file system checker searches the whole directory tree to test all of the single directory entries. (This is the main part of the checking process.) The checker starts at the inode of the root directory, and eventually tests each individual entry. If an entry is so badly damaged that it is unusable, then it will be deleted when the checking process is complete.

6.5.3 Phase 3

At this stage, the file system checker reads the results of the tests it performed in phases 1 and 2. It then corrects any errors in block allocation. There are two possible errors which might occur. The first of these would be a block which is referred to by more than one entry. The checker decides which of the entries is the `legal owner` of such a block. The second error occurs when a block is recorded in a bit map as an allocated block, but it is not referred to by any of the entries which were tested in phase 2. This is called a `lost block`, and the checker will store it in the `lost and found directory` if it contains information about directories. (This will eventually allow you to re integrate any sub directory structures which have been cut off from your directory tree because of damage to the disc.)

6.5.4 Phase 4

In phase 4, the file system checker performs some general tidying operations on the disc, and it also adjusts the allocation summary of the filing system, which is stored in the summary block.

6.6 The File System Interactive Maintenance Tool

The file system checker runs in a non interactive mode, which means that its approach is fixed and systematic. The procedures it follows are designed to achieve the best results possible. However, if you would like more control over the checking process, use the `File System Interactive Maintenance Tool`, which runs in a fully extended interactive mode. This will provide you with a range of options for dealing with the errors on a damaged disc. It offers extended features such as a disc editor which can inspect and modify individual data blocks, interpret disc administration data structures and search byte patterns on disc.

Index