Dual Boot with NVMe

Introduction

NVMe drives are available for RISC OS and offer quite fast storage. Using the standard 512B sector size (more on that later) the RISC OS partition can be up to 256GB in size. Quite inexpensive drives offer a total capacity of 512GB so what do we do with the unused part of the drive?

Firstly there is no CMOS byte reserved to specify the boot drive if the boot filesystem is set to NVMe so RISC OS cannot boot directly from NVMe, even if the NVMe drivers are included in ROM.

There is also no support in the Pi model 4 EEPROM to talk directly to NVMe so booting directly from NVMe is not supported. However it is easy to use the eMMc storage provided on the CM4 to specify the NVMe drive to be used as the boot drive, both for RISC OS and Linux. The eMMc is thus only used during the boot process.

Partitioning the NVMe drive (1)

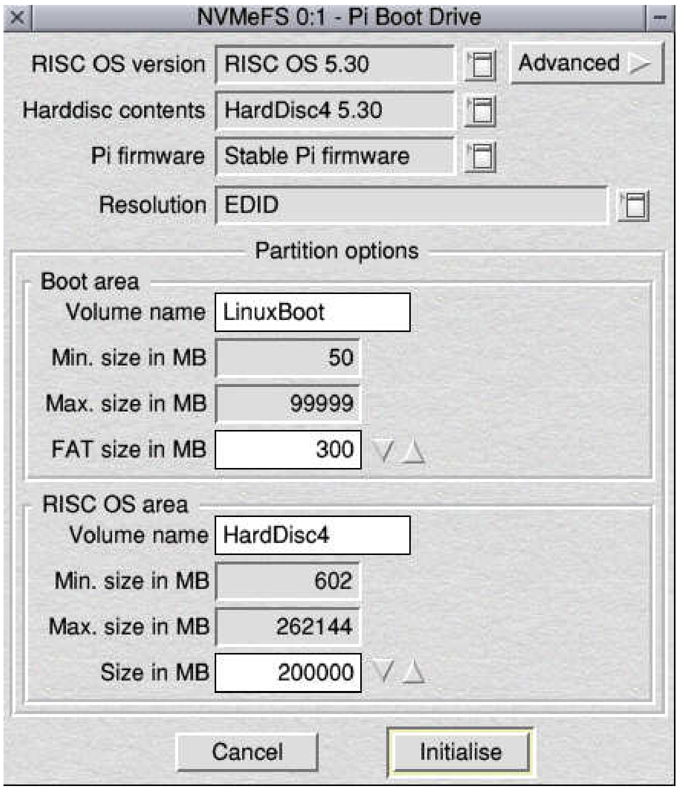
Let us assume that you have a Raspberry Pi with a standard RISC OS distro on SDFS and you plug in a new NVMe drive to the PCI express connector.

The first step is to make the drive available to RISC OS. Using PartMgr is the easiest way to do this. Although a 'Loader' partition on an NVME drive cannot be seen by the Pi 4 EEPROM software, it can serve a useful function as a partition shared between RISC OS and Linux and used by Linux to load its kernel.

!PartMgr 1.03 (23-May-2024) will initialise, partition and format NVMe drives using version 0.03 of the NVMe drivers:

NVMeDriver	0.03 (25 Mar 2024)
NVMeFiler	0.03 (25 Mar 2024)
NVMeFS	0.03 (25 Mar 2024)

Most NVMe drives will use 512B sectors by default. PartMgr can partition these and provide a two-partition disc image where the FAT partition can be read by RISC OS (via the file Boot:Loader). It will also download the standard HardDisc4 and firmware.


NVMeFS 0:1 - Pi Boot Drive

RISC OS version	RISC OS 5.30
Harddisc contents	HardDisc4 5.30
Pi firmware	Stable Pi firmware
Resolution	EDID

Partition options

Boot area

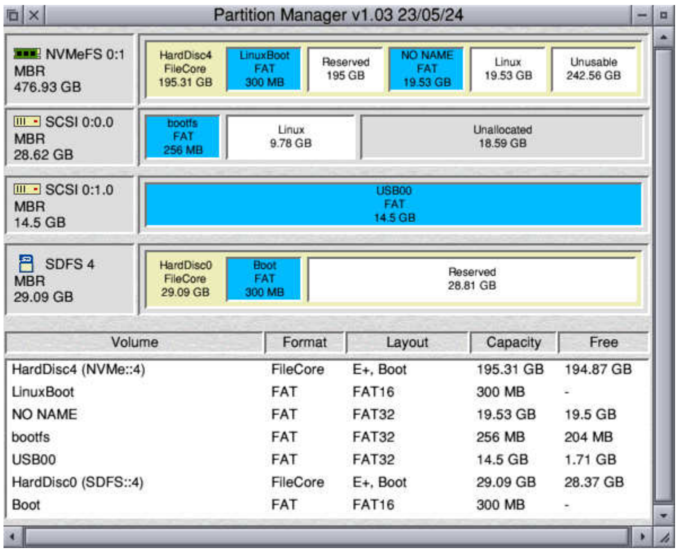
Volume name	LinuxBoot
Min. size in MB	50
Max. size in MB	99999
FAT size in MB	300

RISC OS area

Volume name	HardDisc4
Min. size in MB	602
Max. size in MB	262144
Size in MB	200000

Above: *The options here will generate a 200GB filecore partition (maximum is 250GB with 512B sectors) containing a 300MB FAT partition.*

Below: *the result as seen by PartMgr.*


Partition Manager v1.03 23/05/24

NVMeFS 0:1 MBR 476.93 GB: HardDisc4 FileCore 195.31 GB | LinuxBoot FAT 300 MB | Reserved 195 GB | NO NAME FAT 19.53 GB | Linux 19.53 GB | Unusable 242.56 GB

SCSI 0:0.0 MBR 28.62 GB: bootfs FAT 256 MB | Linux 9.78 GB | Unallocated 18.59 GB

SCSI 0:1.0 MBR 14.5 GB: USB00 FAT 14.5 GB

SDFS 4 MBR 29.09 GB: HardDisc0 FileCore 29.09 GB | Boot FAT 300 MB | Reserved 28.81 GB

Volume	Format	Layout	Capacity	Free
HardDisc4 (NVMe::4)	FileCore	E+, Boot	195.31 GB	194.87 GB
LinuxBoot	FAT	FAT16	300 MB	-
NO NAME	FAT	FAT32	19.53 GB	19.5 GB
bootfs	FAT	FAT32	256 MB	204 MB
USB00	FAT	FAT32	14.5 GB	1.71 GB
HardDisc0 (SDFS::4)	FileCore	E+, Boot	29.09 GB	28.37 GB
Boot	FAT	FAT16	300 MB	-

Booting

On the CM4 an EEPROM has the code previously in BOOTCODE.BIN and loads START4.ELF and FIXUP4.DAT directly from a FAT partition on an SD card or eMMc storage. The files CONFIG.TXT and CMDLINE.TXT then specify how to load the Operating System.

The RISC OS ROM (plus the CMOS settings) are loaded from the same FAT partition. The CMOS settings specify the RISC OS boot filesystem and drive.

When Linux is being started, the file CMDLINE.TXT specifies the UUID of an ext4 partition on any drive which contains the Linux root filesystem. A file in that partition specifies the UUID of a FAT partition that contains the Linux kernel.

At present it is not possible in RISC OS to specify the boot drive number for an NVME drive as there is no CMOS byte reserved for this. However it is possible to place an Obey file !Boot.!Run on SDFS that will load the NVMe drivers (from SDFS) and run boot from the NVMe drive making the NVMe drive the boot drive.

What we have done so far will get RISC OS booting from SDFS and having a 250GB RISC OS partition available on the NVMe drive for storage.

The first objective is to produce a USB or NVMe drive with a Linux partition on it and to update the firmware on the SD card or eMMc storage so that the machine will boot up in either RISC OS or Raspberry Pi OS (aka Raspbian) depending whether a switch (or link) on pins 29 and 30 of the 40-pin header is closed (or present).

Dual boot using USB for Linux

The next step is to download a standard Linux distro image and write it to a USB drive. The standard distro for Raspian is designed to fit onto a range of SD cards so long as they are at least 8.01 GB (8602517504 bytes) in size. If you write this image to a USB drive, it will have the same two-partition structure but default boot will still be from SDFS.

With a RISC OS image on the SD card (or eMMc drive) and the USB drive with the Raspian image plugged in, then start RISC OS. The NVMe drive has already been partitioned as described above so that RISC OS can 'see' it.

The next step, using RISC OS, is to copy all the files from the FAT partition of the Raspian (USB) drive to NVMe::$.!Boot.Loader.*. RISC OS will not use these files but they will be visible to RISC OS once it has started up.

Now we copy the file 'CMDLINE.TXT' from the Raspian partition on the USB drive (SCSI::$.!Boot.Loader.CMDLINE/TXT) to SDFS::$.!Boot.Loader.CMDLINE/TXT on the SD card or eMMc storage and add the text 'disable_gamma' as shown below:

```
disable_gamma console=serial0,115200 console=tty1 root=PARTUUID=e79118ca-02
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet init=/usr/lib/raspi-
config/init_resize.sh splash plymouth.ignore-serial-consoles
```

The contents of CMDLINE/TXT - the location of the root file system is specified by a partition UUID, highlighted above. For a particular SD card image this magic number is known. If you specify the correct partition UUID for an ext4 partition on a USB pen drive then Linux will boot from that pen drive. It looks at the file '/etc/fstab' in its rootfs for the location from which to load the kernel (bootfs).

Whereas RISC OS requires very few, if any, commands in the CMDLINE/TXT file (`disable_gamma` and `disable_mode_changes` for example), Linux requires lots of stuff, which RISC OS can happily ignore.

The partition UUID is a unique identifier for a partition and may refer to a USB drive, SD card, eMMc storage or an NVMe drive.

The effect of all that we have done so far is to make a Loader partition on the NVME drive a bit larger than normal so that it is large enough for the Linux firmware and kernel. The SDFS will still control the boot process and will still boot into RISC OS exactly as before.

Now we make some edits to the file 'CONFIG.TXT' on the SDFS card (or eMMc storage) as shown below:

```
[gpio5=1]
fake_vsync_isr=1
framebuffer_swap=0
gpu_mem=64
init_emmc_clock=100000000
ramfsfile=CMOS
ramfsaddr=0x508000
kernel=RISCOS.IMG
device_tree=
hdmi_drive=2
hdmi_blanking=1
disable_overscan=1
[pi4]
enable_gic=1
[all]
[gpio5=0]
# Additional overlays and parameters are
documented /boot/overlays/README
# Enable audio (loads snd_bcm2835)
dtparam=audio=on
[pi4]
# Enable DRM VC4 V3D driver on top of the
dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2
[all]
#dtoverlay=vc4-fkms-v3d
```

The edited CONFIG/TXT file - the conditional statements are in square brackets with the RISC OS bit highlighted.

With no switch fitted to pins 29 and 30 on the 40-pin header, the status of GPIO 5 will be read by 'CONFIG.TXT' as level 1 and the commands to load RISC OS will be followed.

We now edit the file 'CMDLINE.TXT' on the USB drive to remove the command 'init=/usr/lib/raspi-config/init_resize.sh' so that Linux will not resize itself to occupy the full size of the USB drive. We can now start up Linux so that it loads its kernel and its root file system from the USB drive if the switch between pins 29 and 30 is closed.

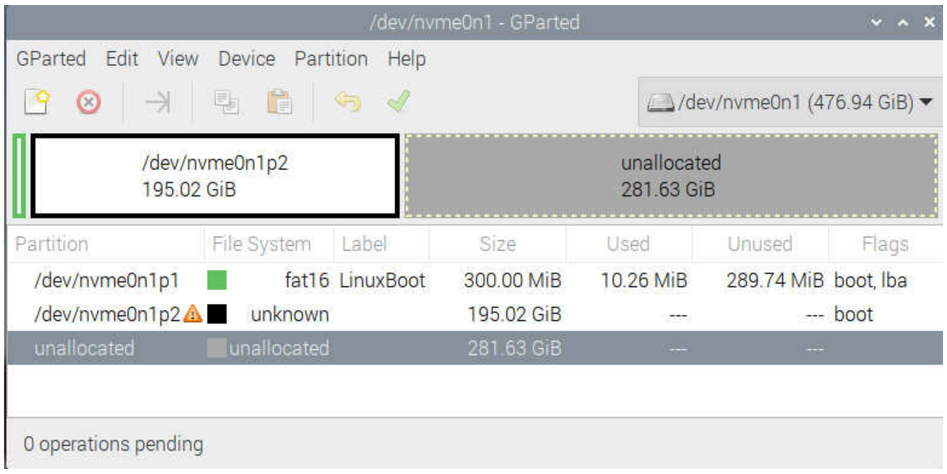
We now have a dual boot facility using the USB drive for Linux and the SDFS storage for RISC OS with a switch selecting the OS: closed for Linux, open for RISC OS.

Partitioning the NVMe drive (2)

If you hold down the switch (or fit a link) between pins 29 and 30 and reboot, then Linux will boot.

The first time that Raspbian starts up it will ask for some parameters (such as Language etc.) to be set and then show the desktop. It may reboot and so you should continue to keep the switch closed until the Linux desktop appears.

The command 'sudo apt-get GPartEd' will download and install the Linux Partition Editor GPartEd onto the USB drive. We can now use this to examine the NVMe drive.

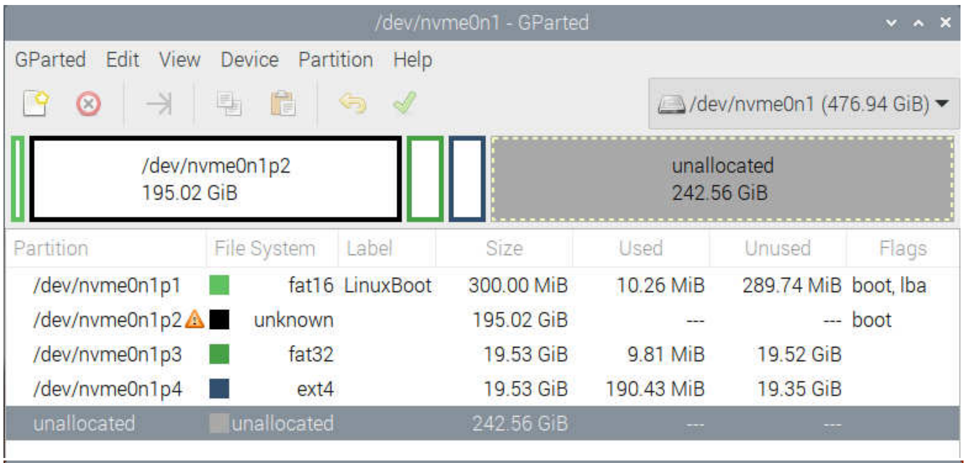


Partition	File System	Label	Size	Used	Unused	Flags
/dev/nvme0n1p1	🟩 fat16	LinuxBoot	300.00 MiB	10.26 MiB	289.74 MiB	boot, lba
/dev/nvme0n1p2 ⚠	■ unknown		195.02 GiB	---	---	boot
unallocated	unallocated		281.63 GiB	---	---	

0 operations pending

This is the standard RISC OS distro written to the NVMe drive by PartMgr it includes the latest RISC OS 5.30 firmware and HardDisc4 contents.

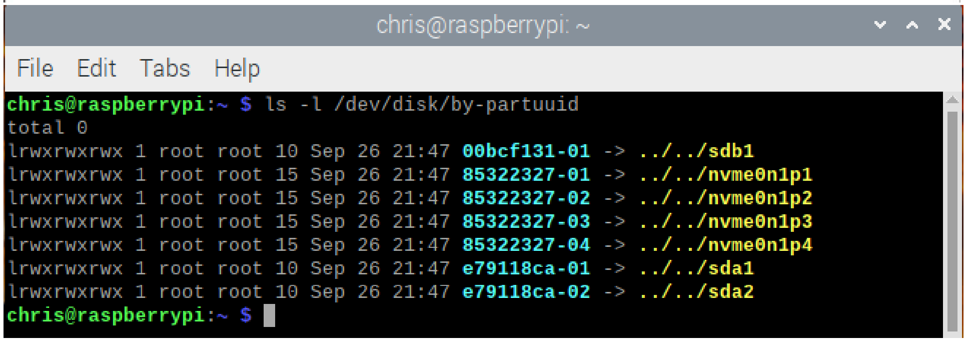
We'll add two further partitions using the GPartEd utility in Linux.



Partition	File System	Label	Size	Used	Unused	Flags
/dev/nvme0n1p1	🟩 fat16	LinuxBoot	300.00 MiB	10.26 MiB	289.74 MiB	boot, lba
/dev/nvme0n1p2 ⚠	■ unknown		195.02 GiB	---	---	boot
/dev/nvme0n1p3	🟩 fat32		19.53 GiB	9.81 MiB	19.52 GiB	
/dev/nvme0n1p4	■ ext4		19.53 GiB	190.43 MiB	19.35 GiB	
unallocated	unallocated		242.56 GiB	---	---	

Now there are two further, blank partitions on the NVMe drive.

Using the 'Partition/New' command we add a 20GB fat32 partition and format it, then add a 20GB ext4 partition and format it.

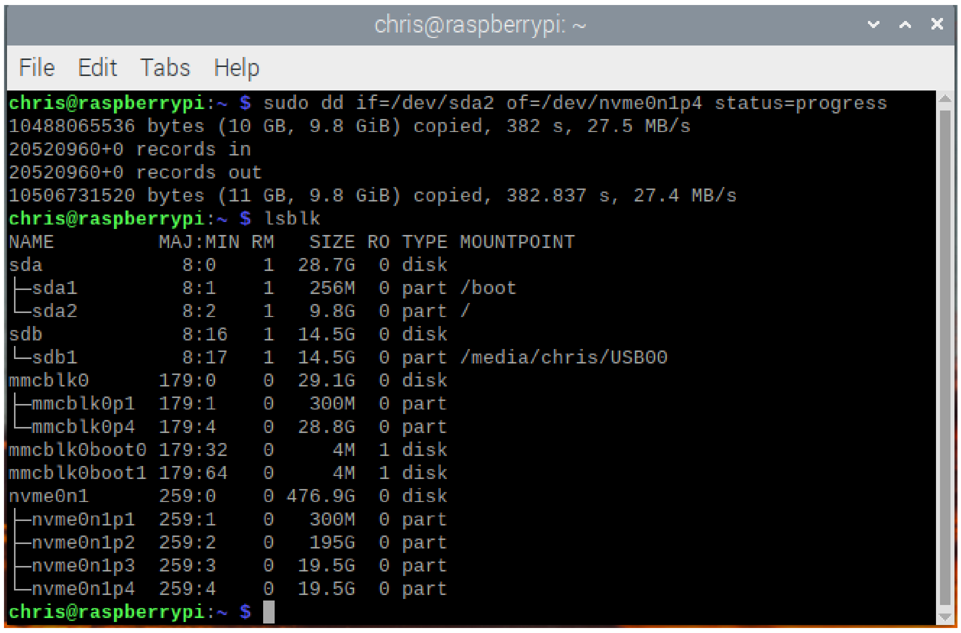


```
chris@raspberrypi:~ $ ls -l /dev/disk/by-partuuid
total 0
lrwxrwxrwx 1 root root 10 Sep 26 21:47 00bcf131-01 -> ../../sdb1
lrwxrwxrwx 1 root root 15 Sep 26 21:47 85322327-01 -> ../../nvme0n1p1
lrwxrwxrwx 1 root root 15 Sep 26 21:47 85322327-02 -> ../../nvme0n1p2
lrwxrwxrwx 1 root root 15 Sep 26 21:47 85322327-03 -> ../../nvme0n1p3
lrwxrwxrwx 1 root root 15 Sep 26 21:47 85322327-04 -> ../../nvme0n1p4
lrwxrwxrwx 1 root root 10 Sep 26 21:47 e79118ca-01 -> ../../sda1
lrwxrwxrwx 1 root root 10 Sep 26 21:47 e79118ca-02 -> ../../sda2
chris@raspberrypi:~ $
```

The Linux distro is mounted as /dev/sda with a UUID of e79118ca and the NVMe drive is mounted as /dev/nvme0n1 with a UUID of 85322327.

Now we will copy the Linux rootfs partition from the USB drive to the NVMe drive.

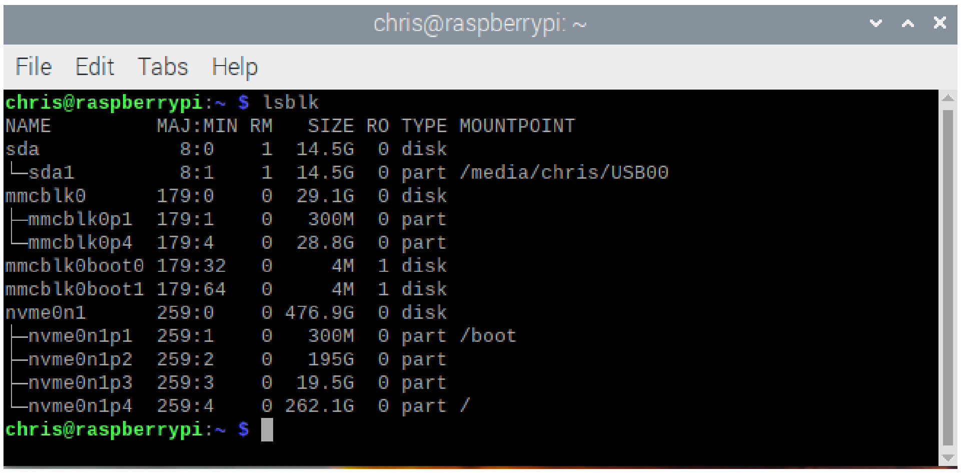
Linux is still booting from the USB drive and both its rootfs and its bootfs partitions are on the USB drive.



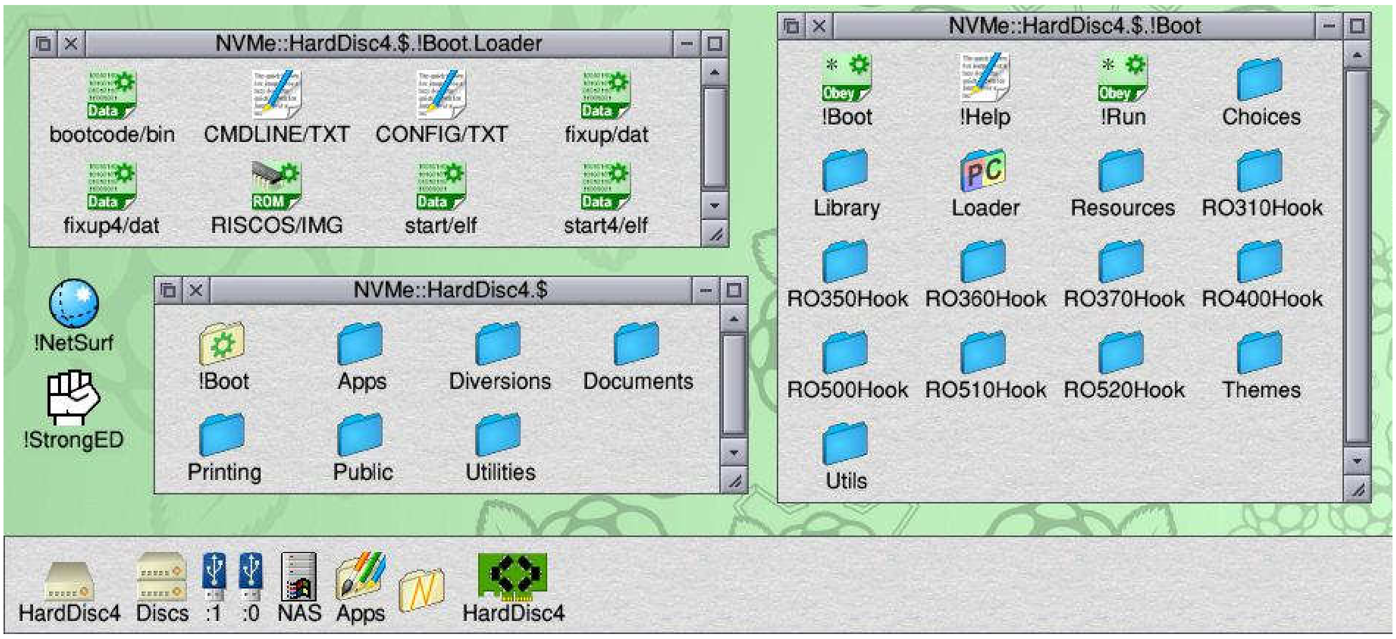
```
chris@raspberrypi:~ $ sudo dd if=/dev/sda2 of=/dev/nvme0n1p4 status=progress
10488065536 bytes (10 GB, 9.8 GiB) copied, 382 s, 27.5 MB/s
20520960+0 records in
20520960+0 records out
10506731520 bytes (11 GB, 9.8 GiB) copied, 382.837 s, 27.4 MB/s
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    1  28.7G  0 disk
├─sda1         8:1    1   256M  0 part /boot
└─sda2         8:2    1   9.8G  0 part /
sdb            8:16   1  14.5G  0 disk
└─sdb1         8:17   1  14.5G  0 part /media/chris/USB00
mmcblk0      179:0    0  29.1G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part
└─mmcblk0p4  179:4    0  28.8G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
nvme0n1      259:0    0 476.9G  0 disk
├─nvme0n1p1  259:1    0   300M  0 part
├─nvme0n1p2  259:2    0   195G  0 part
├─nvme0n1p3  259:3    0  19.5G  0 part
└─nvme0n1p4  259:4    0  19.5G  0 part
chris@raspberrypi:~ $
```

We can now change the file CMDLINE.TXT on the eMMc storage to specify the UUID of the NVMe drive. We will not add back the command to expand the rootfs partition to occupy the unused space on the NVMe drive - we can do this using GPartEd.

When Linux starts up the next time, bootfs is still on the USB drive. The location of bootfs is specified in a file '/etc/fstab' in rootfs. We can edit this file in Linux (using the command 'sudo gedit /etc/fstab') to specify the UUID of the NVMe drive for boots. The bottom window to the left shows that we have unplugged the Raspian USB drive and Linux is now booting from the NVMe drive.



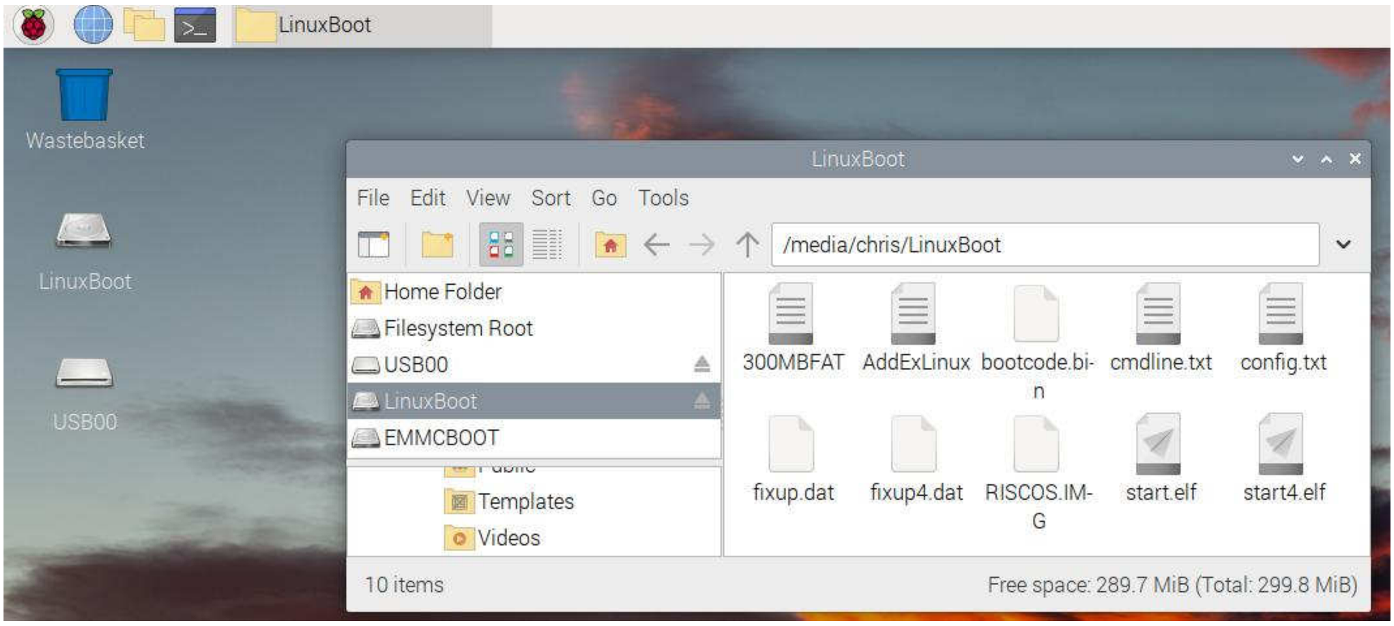
```
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    1  14.5G  0 disk
└─sda1         8:1    1  14.5G  0 part /media/chris/USB00
mmcblk0      179:0    0  29.1G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part
└─mmcblk0p4  179:4    0  28.8G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
nvme0n1      259:0    0 476.9G  0 disk
├─nvme0n1p1  259:1    0   300M  0 part /boot
├─nvme0n1p2  259:2    0   195G  0 part
├─nvme0n1p3  259:3    0  19.5G  0 part
└─nvme0n1p4  259:4    0 262.1G  0 part /
chris@raspberrypi:~ $
```



On a drive with 512B sectors, we get a standard RISC OS filecore partition of 250GB with access to the contents of a 300MB FAT16 partition: at present PartMgr has put the standard firmware into that partition but RISC OS does not use it. It can be used for file sharing between RISC OS and Linux.

***Above:** the view from RISC OS.*

***Below:** the view from Linux after one file has been added from RISC OS and then one from Linux.*

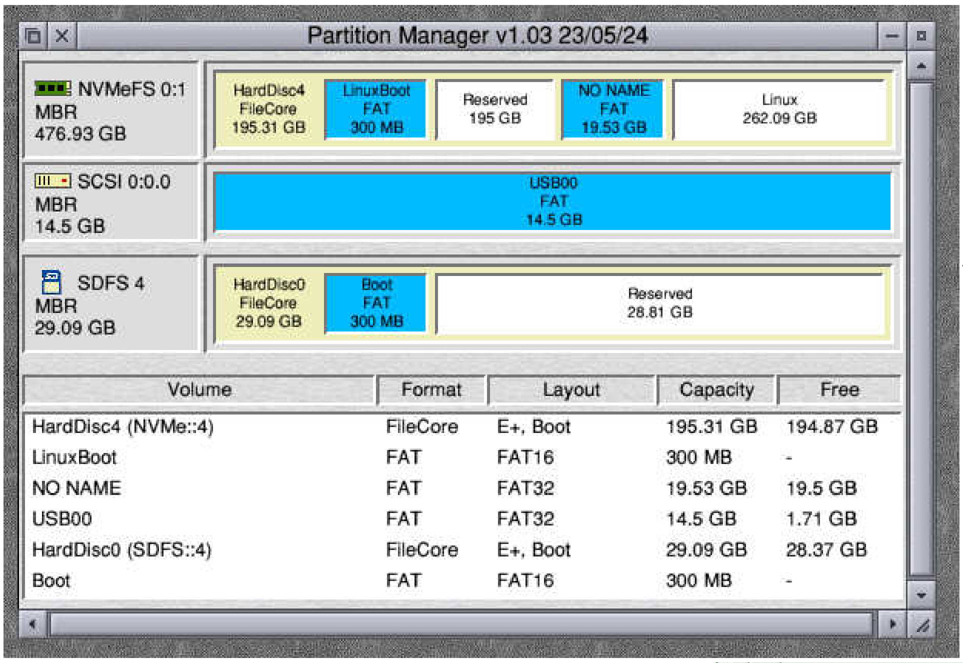


We now have a four partition NVMe drive and a dual boot facility using a switch on the 40-pin header. The NVMe drive has a 200GB RISC OS partition and a 260GB Linux partition as well as two FAT partitions, 300MG and 20GB.

At present it is not possible to specify the boot drive number for an NVME drive as there is no CMOS byte reserved for this. However it is possible to place an Obey file !Boot.!Run on SDFS that will load the NVMe drivers (from SDFS) and then run !Boot from the NVMe drive making the NVMe drive the boot drive so far as RISC OS is concerned.

Why add a FAT partition?

On a 512B sector drive it can be used for file sharing between Linux and RISC OS but that could probaly be done more simply by just plugging in a FAT-


Partition Manager v1.03 23/05/24

NVMeFS 0:1
MBR
476.93 GB

| HardDisc4 FileCore 195.31 GB | LinuxBoot FAT 300 MB | Reserved 195 GB | NO NAME FAT 19.53 GB | Linux 262.09 GB |

SCSI 0:0.0
MBR
14.5 GB

| USB00 FAT 14.5 GB |

SDFS 4
MBR
29.09 GB

| HardDisc0 FileCore 29.09 GB | Boot FAT 300 MB | Reserved 28.81 GB |

Volume	Format	Layout	Capacity	Free
HardDisc4 (NVMe::4)	FileCore	E+, Boot	195.31 GB	194.87 GB
LinuxBoot	FAT	FAT16	300 MB	-
NO NAME	FAT	FAT32	19.53 GB	19.5 GB
USB00	FAT	FAT32	14.5 GB	1.71 GB
HardDisc0 (SDFS::4)	FileCore	E+, Boot	29.09 GB	28.37 GB
Boot	FAT	FAT16	300 MB	-

We have used GPartEd to 'grow' the Linux partition from 20GB to fill the unused space on the drive (Linux can do this without losing the files stored on the partition). We have a fat32fs formatted USB drive to make sharing files between computers easier.

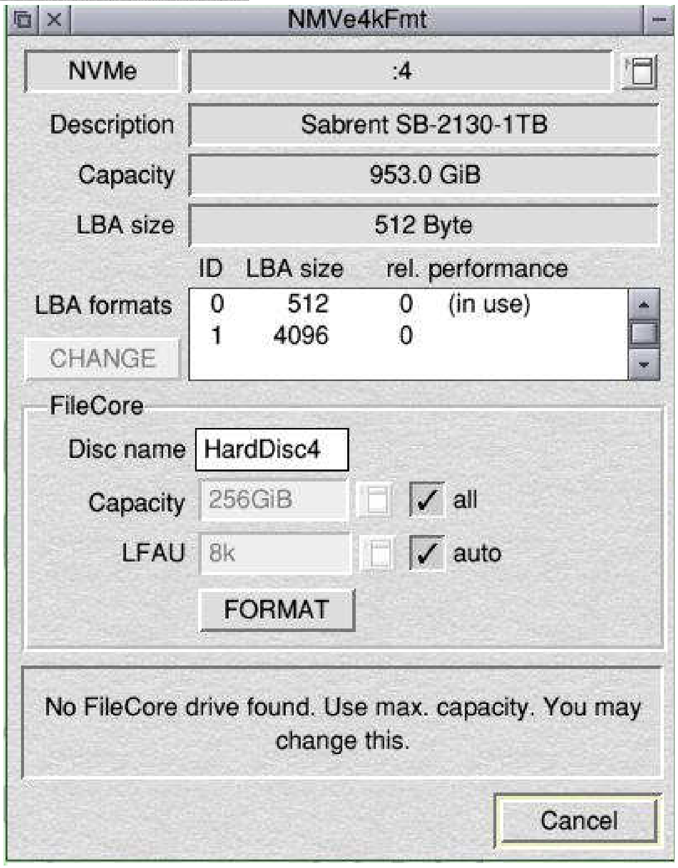
formatted USB drive, which both RISC OS and Linux can see and use. The real reason is to provide a FAT partition for the Linux kernel and its bootfs.

Use of 4k sectors

There are advantages and disadvantages if we convert the NVMe drive to use 4k sectors rather than 512e sectors. Both fat32fs and DOSFS do not support 4k sectors and so 'Loader' partitions cannot be seen from RISC OS and thus are not useful for file sharing. The RISC OS partition can be larger than 250GB (up to 2TB) with a correspondingly larger LFAU. NVMefs also runs considerably faster for random file access. The next step is therefore to convert the NVME drive to use 4k sectors.

Care is needed for this step as changing an NVME disc between 512e and 4k sectors will destroy all data on the drive. Only some NVMe drives can be converted to and from 4k sectors.

NVME4kFmt version 0.05 can show whether the drive will support both formats and, if so, convert from one to the other. We will run it to examine a 1TB drive that is currently using 512e sectors.


NMVe4kFmt

NVMe :4

Description	Sabrent SB-2130-1TB
Capacity	953.0 GiB
LBA size	512 Byte

LBA formats

ID	LBA size	rel. performance
0	512	0 (in use)
1	4096	0

CHANGE

FileCore

Disc name HardDisc4

Capacity 256GiB ✓ all

LFAU 8k ✓ auto

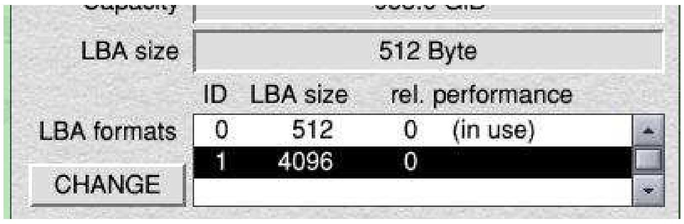
FORMAT

No FileCore drive found. Use max. capacity. You may change this.

Cancel

Above: *it shows the drive supports 4k sectors.*

Below: *so we select 4k sectors and click the option 'CHANGE' which then appears. It will then allow filecore partitions above 250GB to be created.*


LBA size	512 Byte

LBA formats

ID	LBA size	rel. performance
0	512	0 (in use)
1	4096	0

CHANGE

What are 4k discs?

We are used to a standard sector size of 512 bytes. Originally discs were simply a sheet of magnetic material that was spun at 3600 or 7200 rpm. The start of each sector on a track was marked by a special magnetic 'marker' and 256 or 512 (sometimes 1024) bytes were read or written a short time after this passed by the head. There was room for about 16 sectors on a track.

A single-sided disc had room for perhaps 35 tracks and the 'address' of some data was therefore composed of track number (0 to 34) and sector number (either 0 to 15 or 1 to 16). Double sided discs had also the head (0 or 1). Hard discs had a platter of discs with multiple heads and a disc address was thus now composed of a 'CHS' address, cylinder, head and sector.

FileCore (from RISC OS 3.80) used a 19-bit value (idlen=19) to identify a disc object, so there can be no more than 2^{19} objects, each requiring 20 bits in the map, to allow for the extra terminating bit. The granularity of a partition is the minimum separation between each disc object, given by the formula (idlen + 1)*LFAU. There can be no more than 2^{idlen} objects in the partition. This sets the minimum value of LFAU which is one parameter chosen when formatting a partition.

Partition size	Min. LFAU	Granularity (512)	Granularity (4096)	idlen
16GB = 2^{34}	2kB = 2^{11}	40k = 80s	-	19
128GB = 2^{37}	16kB = 2^{14} 4kB = 2^{12}	320k = 640s 88k = 176s	= 80s = 22s	19 21
256GB = 2^{38}	32kB = 2^{15} 8kB = 2^{13}	640k = 1280s 176k = 352s	= 160s = 44s	19 21
512GB = 2^{39}	64kB = 2^{16} 16kB = 2^{14}	-	1280k = 320s 352k = 88s	19 21
1TB = 2^{40}	128kB = 2^{17} 16kB = 2^{14}	-	2560k = 640s 704k = 176s	19 21
2TB = 2^{41}	256kB = 2^{18} 64kB = 2^{16}	-	5120k = 1280s 1408k = 352s	19 21

16GB: LFAU >= $2^{34} \div 2^{idlen} \div$ (idlen+1) i.e. 1.6k, so LFAU can be 2k/4k ...

The granularity of a 256GB partition is 640k (idlen=19) or 176k (idlen=21).

The Large File Allocation Unit (LFAU) is the internal space allocation unit for large files. This can be increased above the default to give improved performance but at the expense of consuming more space per file (wasting on average about half the LFAU for each large file).

Detail you can ignore

Device drivers typically use a 29-bit addressing method to identify a particular sector within a partition (plus 3 bits to identify the disc drive). A partition using 4k sectors can be up to 2TB (=2^{29} x 4k).

RISC OS 5.22 extended the maximum sector size to 4k (maximum partition size now 2TB). RISC OS 5.24 extended idlen to 21 bits so the granularity of a 2TB partition became 1408kB and on a 256GB 4k disc became 176k.

An alternative to the 'CHS' address is the 'LBA' address - this is just a 29-bit value giving the sector number. Conversion between 'CHS' and 'LBA' (Logical Block Addressing) addresses is simple arithmetic:

LBA = ((C * N) + H) * SPT) + S - 1

where N is the number of heads (max. 255) and SPT is the number of sectors per track (max 63). Note that C is a 10-bit quantity (0 to 1023), H is 8-bit (0 to 255) and S is 6-bit (1 to 63).

Where a hard disc is partitioned the 512 bytes at CHS (0,0,1) = LBA 0 identify where each partition starts and how big it is. FileCore ignores this partition table and stores its 512 byte boot block &C00 bytes from the start which is CHS(0,0,7) = LBA 6 (with 512 byte sectors).

A disc object can store more than one file or directory.

Solid state discs

NVMe and SATA discs are pretty much all supplied with geometry that uses 512 byte sectors. Some actually arrange data in 4096 byte units and emulate a disc with 512 byte sectors. Some NVMe drives using 512B sectors can be switched to use 4096B sectors and reformatted.

SATA drives for RISC OS using 4k sectors are nothing new, you've been able to use such drives since 2017 (on Titanium at least, due to updated ADFS and SATADriver modules). Elesar has carried stock of preformatted 2TB SATA drives since April 2018 as a spare part.

There are two advantages in switching to 4k sectors for RISC OS: it increases the maximum partition size from 256GB to 2TB and, in the case of NVMe discs, offers very much improved performance for random reads and writes (two or three times the speed than on the same NVMe drive with 512B sectors - roughly equal speed to SATA drives).

The incentive to use 4k SATA drives on RISC OS has however been muted as a 256GB partition is enough for most users.

So now that we have a strong incentive to use 4k NVMe discs (much faster than with 512B sectors), how easy is it to switch them to 4k?

For this we need Linux, which provides a set of nvme utilities:

Switching to 4k

There are two ways to switch an NVMe drive from 512e to 4k - in RISC OS using some software called NVME4kFmt (still in beta development) or in Linux, as described below.

Switching a drive to or from 4k sector size destroys all the data on the disc. Once the switch has been done, the driver needs to be formatted again.

```
sudo apt install nvme-cli
sudo nvme id-ns -H /dev/nvme0n1
...
LBA format 0 : ... 512 bytes (in use)
LBA format 1 : ... 4096 bytes
sudo nvme format --lbaf=1 /dev/nvme0n1
```

this installs the 'nvme' command and uses it to show whether the drive supports 4k sectors and, if so, to re-format it to use 4k sectors vice 512B sectors. Changing between 512e and 4k will destroy all data on the disc.

The blank drive then needs to be formatted.

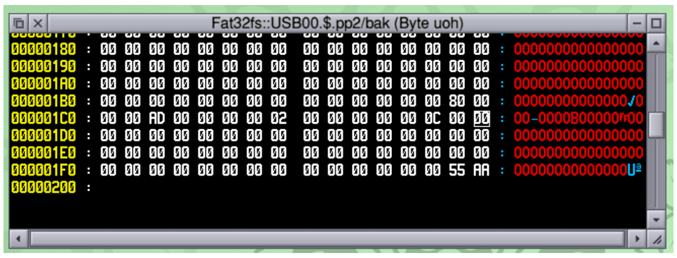
```
RISCOSmark 2.06 (29-Mar-2016) by Richard Spencer 2003
Filing system: NVMe:HardDisc4.$.SpeedTests sector size=512B LFAU=4k partition=110GB
HD Read - Block load 8MB file        165415    42%
HD Write - Block save 8MB file       270415    20%
FS Read - Byte stream file in          1050    21%
FS Write - Byte stream file out        1558    30%

Filing system: NVMe:HardDisc4.$.SpeedTests sector size=512B LFAU=32k partition=110GB
HD Read - Block load 8MB file        194661    49%
HD Write - Block save 8MB file       281098    21%
FS Read - Byte stream file in          2937    60%
FS Write - Byte stream file out        2967    58%

Filing system: NVMe:HardDisc4.$.SpeedTests sector size=4k LFAU=8k partition=240GB
HD Read - Block load 8MB file        192752    48%
HD Write - Block save 8MB file       270415    20%
FS Read - Byte stream file in          4216    86%
FS Write - Byte stream file out        4347    85%

percentages are proportion of RAMfs speed
```

Speed comparisons using the same NVMe drive with different formatting options, as shown.

More complicated

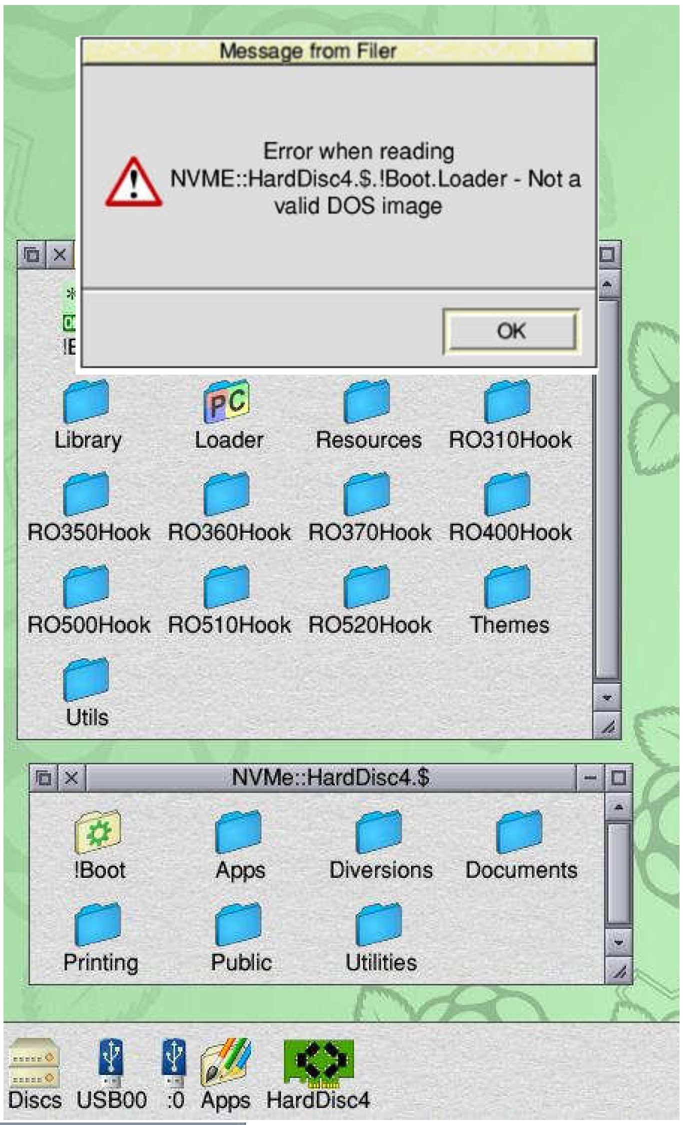
We'll use NVME4kformat to change the LFA from 512e to 4k and to format a 768GB RISC OS partition (with an LFAU of 32k). PartMgr does not, however, create a suitable 'MBR' partition table at disc address &000-&1FF to protect the RISC OS partition. Other operating systems place restrictions on where partitions may start and end (e.g. on a 1MB boundary on Linux with 4k sectors). This depends on the drive design.

So, if RISC OS filecore is not the only show in town, it is a good idea to have a partition table defining the space used and how much of the disc space is still unallocated. We'll create a MBR partition table and write it to the NVMe drive.



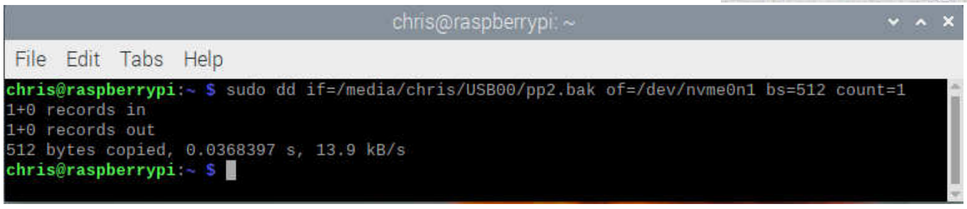
```
Fat32fs::USB00.$.pp2/bak (Byte uoh)
00000180 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 : 0000000000000000
00000190 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 : 0000000000000000
000001A0 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 : 0000000000000000
000001B0 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 80 00 : 00000000000000✓0
000001C0 : 00 00 AD 00 00 00 00 02  00 00 00 00 00 0C 00 00 : 00-0000B00000FF00
000001D0 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 : 0000000000000000
000001E0 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 : 0000000000000000
000001F0 : 00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA : 00000000000000Uª
00000200 :
```

Above: *This is a suitable MBR partition table, 512 bytes all zero apart from those shown.*



Message from Filer

Error when reading NVME::HardDisc4.$.!Boot.Loader - Not a valid DOS image



```
chris@raspberrypi: ~
File  Edit  Tabs  Help
chris@raspberrypi:~ $ sudo dd if=/media/chris/USB00/pp2.bak of=/dev/nvme0n1 bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.0368397 s, 13.9 kB/s
chris@raspberrypi:~ $
```

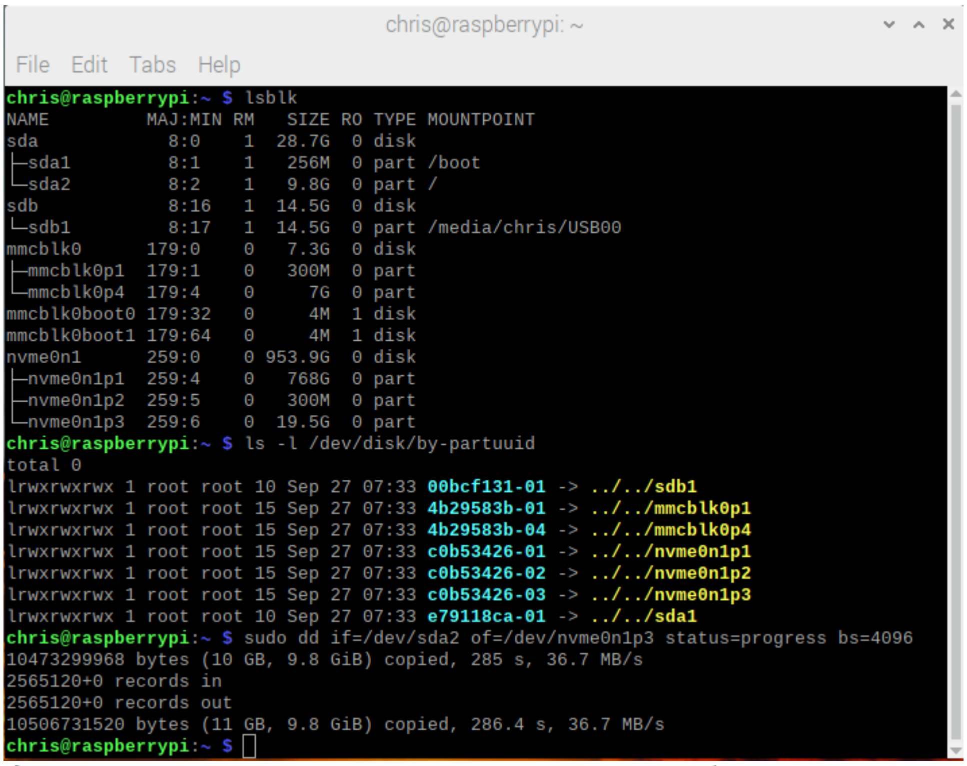
Left: *Writing the partition table to the NVMe drive in Linux.*

Other operating systems use the partition table at CHS(0,0,1) to identify which parts of the disc have been already used by partitions and whether or not it can recognise them and read or write to or from them.

We can then, in Linux, add a FAT partition of 300MB (for Linux's bootfs and kernel) and an ext4 partition using the remaining space on the drive for Linux's rootfs. Neither OS will now trample on each other.

The software (DOSFS) that reads the FAT partition in the RISC OS file Boot:Loader is hard coded to use 512 byte sectors and therefore cannot read or write data to such a partition on a 4k drive. If you do try the 'two-partition' approach then clicking on Boot:Loader to open it will produce the error window shown (above).

We'll repeat the steps in Linux shown on page 4 to add two partitions - one a 300MB FAT partition and one a 20GB ext4 partition. This will give us a three-partition NVMe drive.



chris@raspberrypi: ~

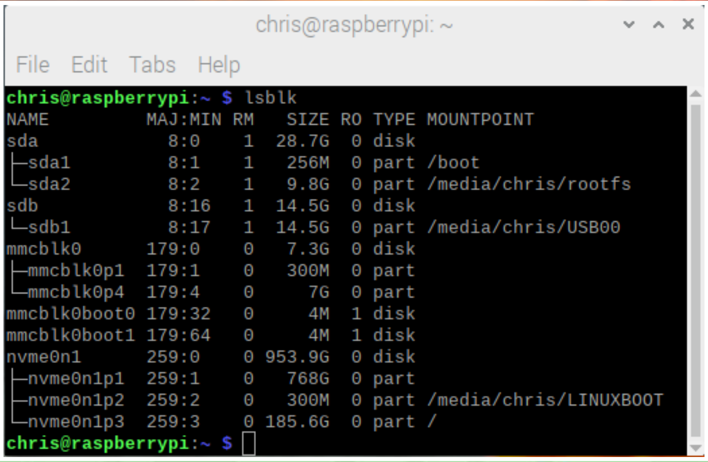
File Edit Tabs Help

```
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    1  28.7G  0 disk
├─sda1         8:1    1   256M  0 part /boot
└─sda2         8:2    1   9.8G  0 part /
sdb            8:16   1  14.5G  0 disk
└─sdb1         8:17   1  14.5G  0 part /media/chris/USB00
mmcblk0      179:0    0   7.3G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part
└─mmcblk0p4  179:4    0     7G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
nvme0n1      259:0    0 953.9G  0 disk
├─nvme0n1p1  259:4    0   768G  0 part
├─nvme0n1p2  259:5    0   300M  0 part
└─nvme0n1p3  259:6    0  19.5G  0 part
chris@raspberrypi:~ $ ls -l /dev/disk/by-partuuid
total 0
lrwxrwxrwx 1 root root 10 Sep 27 07:33 00bcf131-01 -> ../../sdb1
lrwxrwxrwx 1 root root 15 Sep 27 07:33 4b29583b-01 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Sep 27 07:33 4b29583b-04 -> ../../mmcblk0p4
lrwxrwxrwx 1 root root 15 Sep 27 07:33 c0b53426-01 -> ../../nvme0n1p1
lrwxrwxrwx 1 root root 15 Sep 27 07:33 c0b53426-02 -> ../../nvme0n1p2
lrwxrwxrwx 1 root root 15 Sep 27 07:33 c0b53426-03 -> ../../nvme0n1p3
lrwxrwxrwx 1 root root 10 Sep 27 07:33 e79118ca-01 -> ../../sda1
chris@raspberrypi:~ $ sudo dd if=/dev/sda2 of=/dev/nvme0n1p3 status=progress bs=4096
10473299968 bytes (10 GB, 9.8 GiB) copied, 285 s, 36.7 MB/s
2565120+0 records in
2565120+0 records out
10506731520 bytes (11 GB, 9.8 GiB) copied, 286.4 s, 36.7 MB/s
chris@raspberrypi:~ $
```

Now that the RISC OS filecore partition is protected by the MBR partition table, we can add two further partitions to the NVMe drive. Using the 'Partition/New' command we add a 300MB fat16 partition and format it, then add a 20GB ext4 partition and format it.

Now we will copy the Linux rootfs partition from the USB drive to the NVMe drive.

Linux is still booting from the USB drive and both its rootfs and its bootfs partitions are on the USB drive. We now change the file CMDLINE.TXT on the eMMc storage to specify the UUID of the NVMe drive.



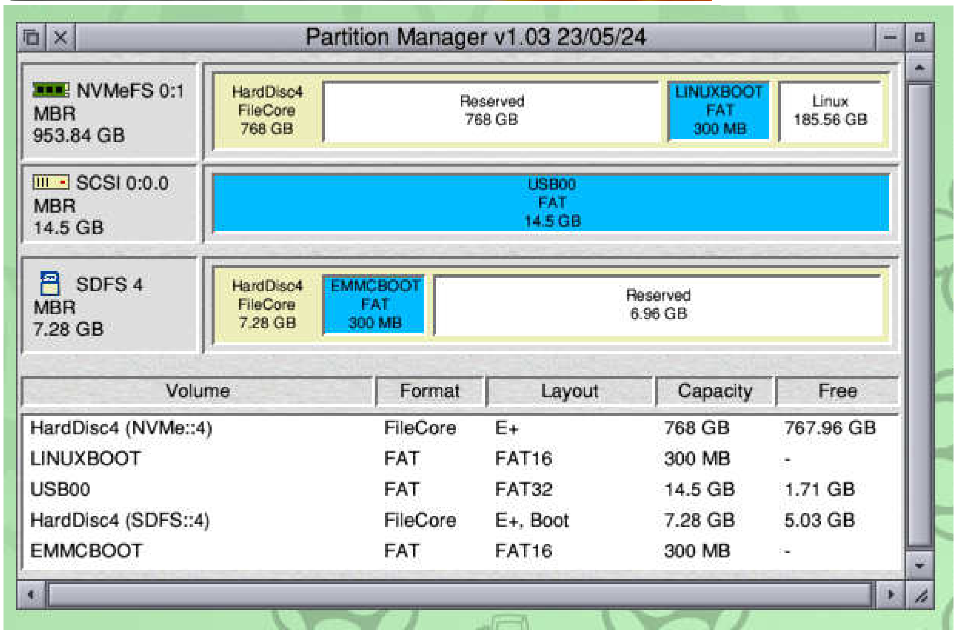
chris@raspberrypi: ~

File Edit Tabs Help

```
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    1  28.7G  0 disk
├─sda1         8:1    1   256M  0 part /boot
└─sda2         8:2    1   9.8G  0 part /media/chris/rootfs
sdb            8:16   1  14.5G  0 disk
└─sdb1         8:17   1  14.5G  0 part /media/chris/USB00
mmcblk0      179:0    0   7.3G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part
└─mmcblk0p4  179:4    0     7G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
nvme0n1      259:0    0 953.9G  0 disk
├─nvme0n1p1  259:1    0   768G  0 part
├─nvme0n1p2  259:2    0   300M  0 part /media/chris/LINUXBOOT
└─nvme0n1p3  259:3    0 185.6G  0 part /
chris@raspberrypi:~ $
```

We will expand the rootfs partition to occupy the unused space on the NVMe drive using GPartEd. We will copy the files from /boot on the USB drive to /media/chris/LINUXBOOT on the NVME drive) using Linux (as RISC OS cannot see both partitions).

When Linux starts up the next time, bootfs is still on the USB drive. We edit the file '/etc/fstab' in Linux (using the command 'sudo gedit /etc/fstab') to specify the UUID of the NVMe drive for bootfs.



Partition Manager v1.03 23/05/24

NVMeFS 0:1
MBR
953.84 GB

HardDisc4 FileCore 768 GB | Reserved 768 GB | LINUXBOOT FAT 300 MB | Linux 185.56 GB

SCSI 0:0.0
MBR
14.5 GB

USB00 FAT 14.5 GB

SDFS 4
MBR
7.28 GB

HardDisc4 FileCore 7.28 GB | EMMCBOOT FAT 300 MB | Reserved 6.96 GB

Volume	Format	Layout	Capacity	Free
HardDisc4 (NVMe::4)	FileCore	E+	768 GB	767.96 GB
LINUXBOOT	FAT	FAT16	300 MB	-
USB00	FAT	FAT32	14.5 GB	1.71 GB
HardDisc4 (SDFS::4)	FileCore	E+, Boot	7.28 GB	5.03 GB
EMMCBOOT	FAT	FAT16	300 MB	-

In the final configuration, the mount point of nvme0n1p2 becomes /boot and the sda drive is unplugged. Linux boots entirely from NVMe and the USB drive is no longer required.

PartMgr now shows this final position - only the filecore partition (768GB) is seen by RISC OS. The FAT and ext4 partitions can only be seen by Linux.

Making the same edits to SDFS::$.!Boot.Loader.CONFIG.TXT and SDFS::$.!Boot.Loader.CMDLINE.TXT will give us a dual boot system with RISC OS having 768GB of storage on the NVMe drive.

Now we have a dual-boot machine with RISC OS and Linux using separate portions of the NVMe drive with plenty of room for both. The FAT partition is for the Linux mount '/boot' and cannot be seen by RISC OS as fat32fs cannot yet read 4k sector partitions.

Chris Hall *chris@svrsig.org*