

VxWorks 下的文件系统介绍

VxWorks 给用户提供了多种文件系统，大致有如下几种，本文做一个简要的介绍，供参考。

- ❖ VRFS
- ❖ DosFS
- ❖ HRFS
- ❖ TRFS
- ❖ RawFS
- ❖ cdromFS
- ❖ RomFS
- ❖ TSFS
- ❖ TrueFFS
- ❖ NFS

VRFS

组件 INCLUDE_VRFS, virtual root file system。这是一个虚拟的文件系统，仅是提供了一个 POSIX 风格的虚拟根目录"/"。在这个目录里，可以看到其它以"/"开始，且只包含一个"/"的设备

| | |
|---------------------------------|---------------------|
| Unicode Character Set | INCLUDE_UTF |
| Virtual Root File System | INCLUDE_VRFS |
| XBD Block Cache | INCLUDE_XBD_BLK... |

```

-> devs
drv name
0 /null
1 /tyCo/0
1 /tyCo/1
2 /pcConsole/0
2 /pcConsole/1
5 /ram
8 /
10 host:
11 /vio
4 /ata0:1
4 /atal:1
5 /atal:2

-> ls "/"
/null
/ram
/vio
/ata0:1
/atal:1
/atal:2
value = 0 = 0x0
-> cd "/"
value = 0 = 0x0

-> pwd
/
value = 2 = 0x2

-> ls
null
ram
vio
ata0:1
atal:1
atal:2
value = 0 = 0x0
-> cd "ata0:1"
value = 0 = 0x0
-> pwd
/ata0:1
value = 8 = 0x8

```

DosFS

组件 INCLUDE_DOSFS, MS-DOS-compatible file system

文件名最长 255 个字符，不过路径名最长 1024 个字符，而 Windows 默认的最长路径应该只有 260 个字符

| | |
|----------------------------------------------|----------------------|
| DOS filesystem backward-compatibility | INCLUDE_DOSFS |
|----------------------------------------------|----------------------|

DosFS 支持多种本地存储设备，例如硬盘、软盘、优盘等。系统启动时，自动调用 xxxDevCreate() 来创建 XBD(extended block device) 设备。然后用户可以使用 xbdCreatePartition() 分区，并使用 dosfsDiskFormat() 格式化。例如我当前的配置是：在 VxWorks 6.9 里包含 Intel Serial/Parallel ATA 驱动 INCLUDE_DRV_STORAGE_PIIX，并在 X86 的 Target 里添加一个新的 IDE 硬盘

 **Intel PIIX SATA Controller** **INCLUDE_DRV_STORAGE_PIIX**

系统启动后，可以看到多了一个设备

```
-> devs
drv name
 0 /null
 1 /tyCo/0
 1 /tyCo/1
 2 /pcConsole/0
 2 /pcConsole/1
 5 /ram
 8 /
10 host:
11 /vio
 4 /ata0:1
 5 /ata1:0
 6 stdio_pty_0x38f0b50.S
```

使用 xbdCreatePartition() 进行分区

 **XBD Disk Partition Handler** **INCLUDE_XBD_PART_LIB**

```
STATUS xbdCreatePartition
(
  char *pathName, /* 设备名 */
  int nPart,      /* 分区数量 1-4 */
  int size1,      /* 第2分区空间占比 */
  int size2,      /* 第3分区空间占比 */
  int size3,      /* 第4分区空间占比 */
);
```

例如分为两个分区，各占 50%

```
-> xbdCreatePartition "/ata1:0", 2, 50
value = 0 = 0x0
-> devs
drv name
 0 /null
 1 /tyCo/0
 1 /tyCo/1
 5 /ata1:1
 5 /ata1:2
```

使用 dosfsDiskFormat() 将分区格式化为 dosFS

```

-> dosFsDiskFormat "/atal:1"
Formatting /atal:1 for DOSFS
Formatting...Retrieved old volume params with %
Volume Parameters: FAT type: FAT32, sectors per
0 FAT copies, 0 clusters, 0 sectors per FAT
Sectors reserved 0, hidden 0, FAT sectors 0
Root dir entries 0, sysId (null) , serial nu
Label:" " ...
Disk with 4194288 sectors of 512 bytes will be
Volume Parameters: FAT type: FAT32, sectors per
2 FAT copies, 523258 clusters, 4096 sectors p
Sectors reserved 32, hidden 63, FAT sectors 8
Root dir entries 0, sysId VX5DOS32, serial nu
Label:" " ...
OK.
value = 0 = 0x0

```

使用 dosFsShow() 可以看到 DosFS 的卷配置

```

-> dosFsShow "/atal:1"

volume descriptor ptr (pVolDesc): 0x16436b0
XBD device block I/O handle: 0x70001
auto disk check on mount: NOT ENABLED
volume write mode: copyback (DOS_WRITE)
volume options:
max # of simultaneously open files: 26
file descriptors in use: 0
# of different files in use: 0
# of descriptors for deleted files: 0
# of obsolete descriptors: 0

FAT handler information:
-----
- allocation group size: 53 clusters
- free space on volume: 2, 142, 260. 672 bytes
value = 0 = 0x0

```

HRFS

组件 INCLUDE_HRFS, Highly Reliable File System, 针对实时系统而设计的一种事务性文件系统, 具有容错能力, 并兼容 POSIX

| HRFS Format | INCLUDE_HRFS_FORMAT |
|---------------------------------------|-------------------------------------|
| Highly Reliable File System (default) | INCLUDE_HRFS |
| Default Commit Task Priority (HRFS) | HRFS_DEFAULT_COMMIT_TASK_P... int 2 |
| Default Max Buffers (HRFS) | HRFS_DEFAULT_MAX_BUFFER int 10 |
| Default Max Open Files (HRFS) | HRFS_DEFAULT_MAX_FILES int 10 |

使用方式类似于 DosFS, 先用 xbdCreatePartition() 进行分区, 然后使用 hrfsDiskFormat() 格式化

```

STATUS hrfsDiskFormat
(
    const char *pDevName, /* name of the device to initialize */
    int files, /* the maximum number of files to support */
    UINT32 majorVer, /* major version of fs to format */
    UINT32 minorVer, /* minor version of fs to format */
    UINT32 options /* currently unused */
);

```

```

-> hrfsDiskFormat "/atal:2"
Formatting /atal:2 for HRFS v1.2
Formatting...OK.
value = 0 = 0x0

```

使用 hrfsChkDsk() 可以看到文件系统的基本信息

```

-> hrfsChkDsk "/ata1:2", 1
Checking HRFS File System...
Media stats:
---XBD          = 23544680:
---Disk size    = 2147443200:
---Block size   = 512:
---# of blocks  = 4194225:
Superblock parameters:
---File system version: 1.2
---creation time: 482000
---block size: 8192
---inodes: 260082

```

TRFS

组件 INCLUDE_XBD_TRANS, transaction-based reliable file system。为 DosFS 提供一个容错的 IO 层

```

Transaction Block Layer    INCLUDE XBD TRANS
STATUS usrFormatTrans
(
  char *dev,
  int overhead, /* uncommitted workspace, in parts-per-thousand */
                /* 0 for 5% */
  int type      /* FORMAT_REGULAR (0) */
);

```

创建流程：先在 XBD 设备上创建 TRFS，然后再创建 DosFS

```

-> usrFormatTrans "/ata1:1"
value = 0 = 0x0
-> dosfsDiskFormat "/ata1:1"
Formatting /ata1:1 For DosFs

```

在 TRFS 上进行的文件操作，必须提交后，才能永久生效。而事务的提交是以整个分区为单位的，并不是提交某个文件。因此，提交时，整个分区的状态需要一致，不能有正在进行中的文件操作。有两个函数可以用于提交

```

STATUS usrTransCommit
(
  char *volume /* 分区名 */
);

STATUS usrTransCommitFd
(
  int fd /* TRFS上文件的fd */
);

```

注意：尽管 usrTransCommitFd() 的参数是 TRFS 上某个文件的 fd，但提交的是整个分区的文件系统

TRFS 的操作示例如下

```
/* 在TRFS里创建一个文件夹 */
mkdir("/atal:1/aaa");

/* 此时重启Target, 文件夹aaa丢失 */

/* 提交文件系统的操作 */
usrTransCommit ("/atal:1");

/* 此时重启Target, 文件夹aaa不丢失 */
```

RawFS

组件 INCLUDE_RAWFS, raw file system, 将整个分区当作一个文件来处理。新创建的 XBD 设备, 默认就挂载 RawFS, 也不需要格式化。通过基本 IO 函数来操作即可

默认就是 RawFS. -> Instantiating /atal:0 as rawFS, device = 0x50001

```
-> ls "/atal:0"
Can't open "/atal:0".
value = -1 = 0xffffffff
没有文件
-> open "/atal:0", 2
value = 12 = 0xc
直接操作分区
-> write 12, "hello", 5
value = 5 = 0x5
写入数据
-> close 12
value = 0 = 0x0
关闭分区
-> ls "/atal:0"
Can't open "/atal:0".
value = -1 = 0xffffffff
没有文件
-> buf=malloc(10)
New symbol "buf" added to kernel symbol table
buf = 0x1617fe0: value = 23264056 = 0
-> open "/atal:0", 2
value = 12 = 0xc
-> read 12, buf, 5
value = 5 = 0x5
读取数据
-> printf buf
hellovalue = 5 = 0x5
```

cdromFS

组件 INCLUDE_CDROMFS, ISO 9660 standard file system, 支持 CD-ROMs, CD-Rs, CD-RWs

```

v CDRM (ISO9960 High Sierra) filesystem INCLUDE_CDROMFS
  CDRM Common Buffer Size CDRM_COM_BUF_SIZE 3

```

例如有张 CD, 可以在 Windows 里打开



把它放到 VxWorks 的 Target 后, VxWorks 启动时自动加载 cdromFS, 然后可以使用 `open()`、`close()`、`read()`、`ioctl()`、`readdir()`、`stat()` 等函数来访问它。使用 `cdromFsVolConfigShow()` 可以查看 cdrom 的卷配置

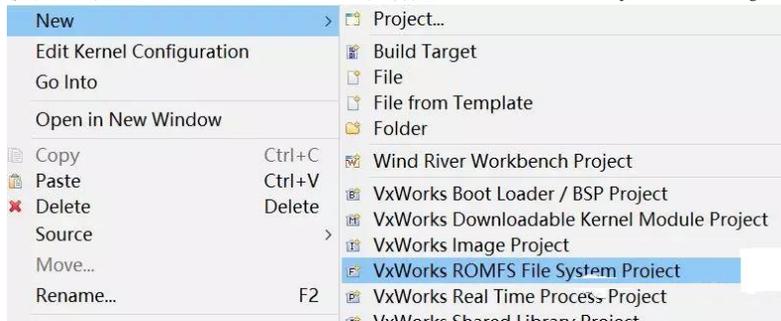
```
-> devs
drv name
 0 /null
 1 /tyCo/0
 1 /tyCo/1
 5 /ata0:1
 4 /cdrom1:0
value = 25 = 0x19
->
-> ls "/cdrom1:0"
/cdrom1:0/.
/cdrom1:0/..
/cdrom1:0/FILE.TXT
/cdrom1:0/FOLDAA
value = 0 = 0x0
-> cdromFsVolConfigShow "/cdrom1:0"

device config structure ptr    0x1665850
device name                    /cdrom1:0
bytes per physical sector      2048
```

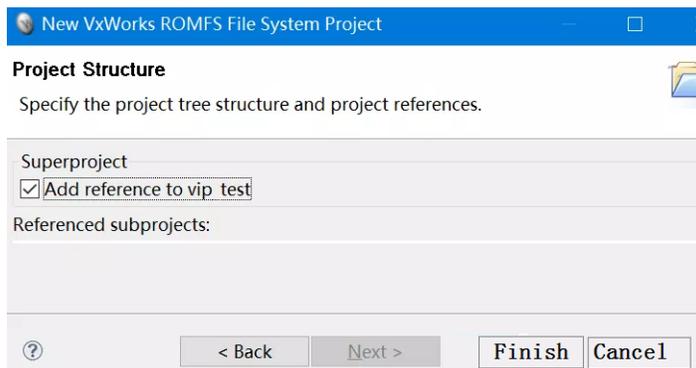
RomFS

组件 INCLUDE_ROMFS, Read-Only Memory File System, 将任意文件存放到操作系统中, 不需要本地存储设备或网络设备

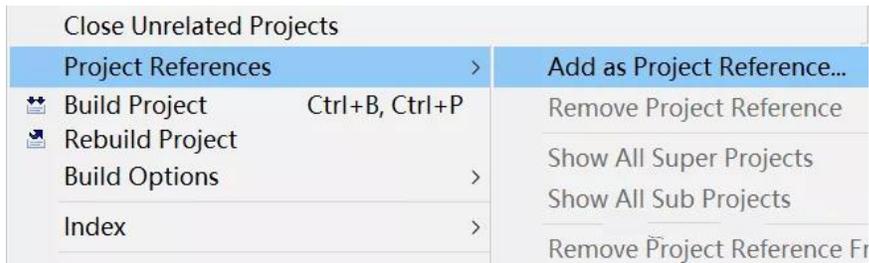
使用流程是: 在 Workbench 中新建 ROMFS File System Project



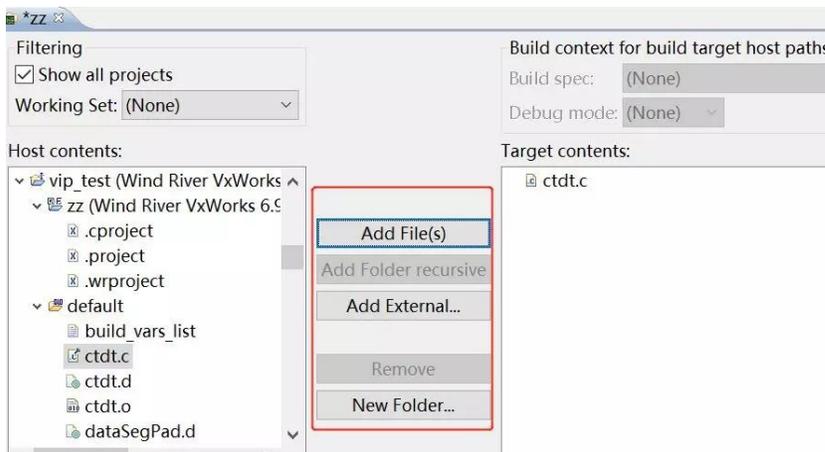
创建时选择 Add reference to VIP project



或者创建之后，在工程上右击选择 Add as Project Reference



在工程的 Contents 里可以添加文件、文件夹



然后直接 build VIP，重启 Target，就可以看到 VxWorks 里多出来的设备/romfs，里面就包含之前添加的文件

```

-> devs
drv name
 0 /null
 1 /tyCo/0
 1 /tyCo/1
 2 /pcConsole/0
 2 /pcConsole/1
 9 /romfs
value = 25 = 0x19
-> ls "/romfs"
/romfs/.
/romfs/..
/romfs/ctdt.c
value = 0 = 0x0
\

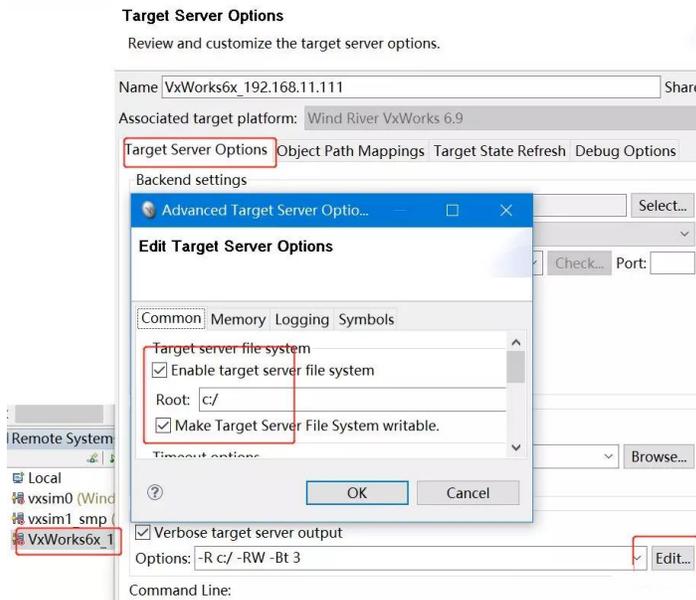
```

TSFS

组件 INCLUDE_WDB_TSFS, Target Server File System, 是的 WDB 的成员之一，在 Target 端创建一个叫做 /tgtsvr 的文件系统，但实际上操作的是 Host 端的文件

WDB target server file system INCLUDE_WDB_TSFS

在 Target Connection 的属性中配置 Target Server File System 的 Root 路径，默认值是当前的 workspace



启动 VxWorks，连接 Target Connection 后，就可以在 VxWorks 中通过 /tgtsvr 访问 Host 的文件了。这个例子里，访问的是 Host 端 Windows 的 C:/ 目录

```

-> devs
drv name
0 /null
1 /tyCo/0
1 /tyCo/1
13 /tgtsvr
7 stdio_pty_0x5964080.S
8 stdio_pty_0x5964080.M
5 /ata0:1
4 /cdrom1:0
value = 25 = 0x19
-> ls "/tgtsvr"
/tgtsvr/$Recycle.Bin
/tgtsvr/Config.Msi
/tgtsvr/Documents

```

TrueFFS

组件 INCLUDE_TFFS, Flash File System。通过 TrueFFS，可以使用 DosFS 或 HRFS 来访问 Flash 设备

NFS

Network File System