AdvFS system calls & kernel interfaces

Module 4

Objectives

- List the various entry points to AdvFS
- Describe how an AdvFS system call is processed
- Describe the algorithms for startup and recovery
- Explain the storage management algorithms
- Describe the cloning algorithms
- Define the file migration and deletion algorithms
- Describe the algorithms for threads

VFS switch table

- 13 entry points for file system operations (includes V5 smooth sync)
- An interface defined in:
  - `/usr/include/sys/mount.h`
  - `struct vfsops * m_op;`
- The interface implemented in:
  - `msfs/osf/msfs_vfsops.c`

VFS switch table routine list (1 of 2)

```
/*
 * msfs_vfsops

 *
 * Defines function pointers to AdvFS specific VFS fs
operations.
 */
struct vfsops msfs_vfsops = {
    msfs_mount,
    msfs_start,
    msfs_unmount,
```

VFS switch table routine list (2 of 2)

```
    msfs_root,
    advfs_quotactl,
    msfs_statfs,
    msfs_sync,
    msfs_fhtovp,
    msfs_vptofh,
    msfs_init,
    msfs_mountroot,
    msfs_noop,
    msfs_smoothsync,
};
```

vnode switch table

- **42 entry points for file operations**
- **An interface defined in:**
  - `/usr/include/sys/vnode.h`
  - `struct vnodeops * v_op;`
- **The interface implemented in:**
  - `msfs/osf/msfs_vnops.c`

File (vnode) operations (1 of 3)

```
/*
 * msfs_vnodeops
 * Defines function pointers to AdvFS specific VFS 
vnode operations.
 */
struct vnodeops msfs_vnodeops = {
    msfs_lookup,          /* lookup */
    msfs_create,          /* create */
    msfs_mknod,           /* mknod */
    msfs_open,            /* open */
    msfs_close,           /* close */
    msfs_access,          /* access */
    msfs_getattr,         /* getattr */
    msfs_setattr,         /* setattr */
    msfs_read,            /* read */
    msfs_write,           /* write */
    msfs_ioctl,           /* ioctl */
```

File (vnode) operations (2 of 3)

```
seltrue,            /* select */
msfs_mmap,          /* mmap */
msfs_fsync,         /* fsync */
msfs_seek,          /* seek */
msfs_remove,        /* remove */
msfs_link,          /* link */
msfs_rename,        /* rename */
msfs_mkdir,         /* mkdir */
msfs_rmdir,         /* rmdir */
msfs_symlink,       /* symlink */
msfs_readdir,       /* readdir */
msfs_readlink,      /* readlink */
msfs_abortop,       /* abortop */
msfs_inactive,      /* inactive */
msfs_reclaim,       /* reclaim */
msfs_bmap,          /* bmap */
msfs_strategy,      /* strategy */
```

File (vnode) operations (3 of 3)

```
    msfs_print,           /* print */
    msfs_page_read,       /* page_read */
    msfs_page_write,      /* page_write */
    msfs_swap,            /* swap handler */
    msfs_bread,           /* buffer read */
    msfs_brelse,          /* buffer release */
    msfs_lockctl,         /* file locking */
    msfs_syncdata,        /* fsync byte range */
    msfs_noop,            /* Lock a node */
    msfs_noop,            /* Unlock a node */
    msfs_getproplist,     /* Get extended attributes */
    msfs_setproplist,     /* Set extended attributes */
    msfs_delproplist,     /* Delete extended attributes
*/
    msfs_pathconf,        /* pathconf */
};
```

UBC interface

- **vnode operations used in paging**
- **`msfs_getpage`**
  - to obtain a page from disk
- **`msfs_putpage`**
  - to write a page to disk
- **The implementation is in:**
  - `msfs/osf/msfs_misc.c`

Device driver interface routines

- **In AdvFS `struct buf`**
  - `b_iodone` field contains address of `msfs_iodone()`
    - Or bs_raw_complete() for raw I/O operations
  - represents a buffer of data
  - listhead is `bsBufList`
- **At interrupt**
  - device driver calls `msfs_iodone()`
- **`msfs_iodone()`**
  - temporarily raises system priority level
  - places buffer on `MsfsIodoneBuf` queue (holds completed I/O operations for AdvFS) found within the processor structure.
  - posts `LWC_PRI_MSFS_UBC`
- **The implementation is in: `msfs/osf/msfs_io.c`**

AdvFS Lightweight Context (LWC) interface

- **Priority: `LWC_PRI_MSFS_UBC`**
- **Entry: `msfs_async_iodone_lwc()`**
- **`msfs_async_iodone_lwc()`**
  - removes buffer from `MsfsIodoneBuf`
  - calls `bs_osf_complete()`
- **The implementation is in:**
  - `msfs/osf/msfs_io.c`

AdvFS I/O completion function

- **Checks for many errors**
  - if appropriate, prints error messages
  - if error while writing to log, panic kernel
- **Call `bs_io_complete()` to reach BAS layer**
- **Initiate more I/O if appropriate**
- **Source location: `msfs/bs/bs_qio.c`**

AdvFS I/O descriptors

- **AdvFS `struct ioDesc`: Element of the I/O queue**
  - contains reference to the "standard" `struct buf`
  - AdvFS structure for queueing I/O requests
- **AdvFS `struct ioDescHdr`**
  - header element for an I/O queue of stuct ioDesc
- **Source location is `msfs/msfs/bs_ims.h`**

Bitfile buffer structure

- **AdvFS `struct bsBuf`**
  - associates I/O descriptions with bitfile sets
    - contains transaction information
    - queues “dirty” buffers of a bitfile
  - for "normal" files, contains reference to `struct ioDesc`
  - for Direct I/O, contains reference to `struct buf`
- **Source location is `msfs/msfs/bs_buf.h`**

AdvFS I/O queues in Tru64 UNIX V5

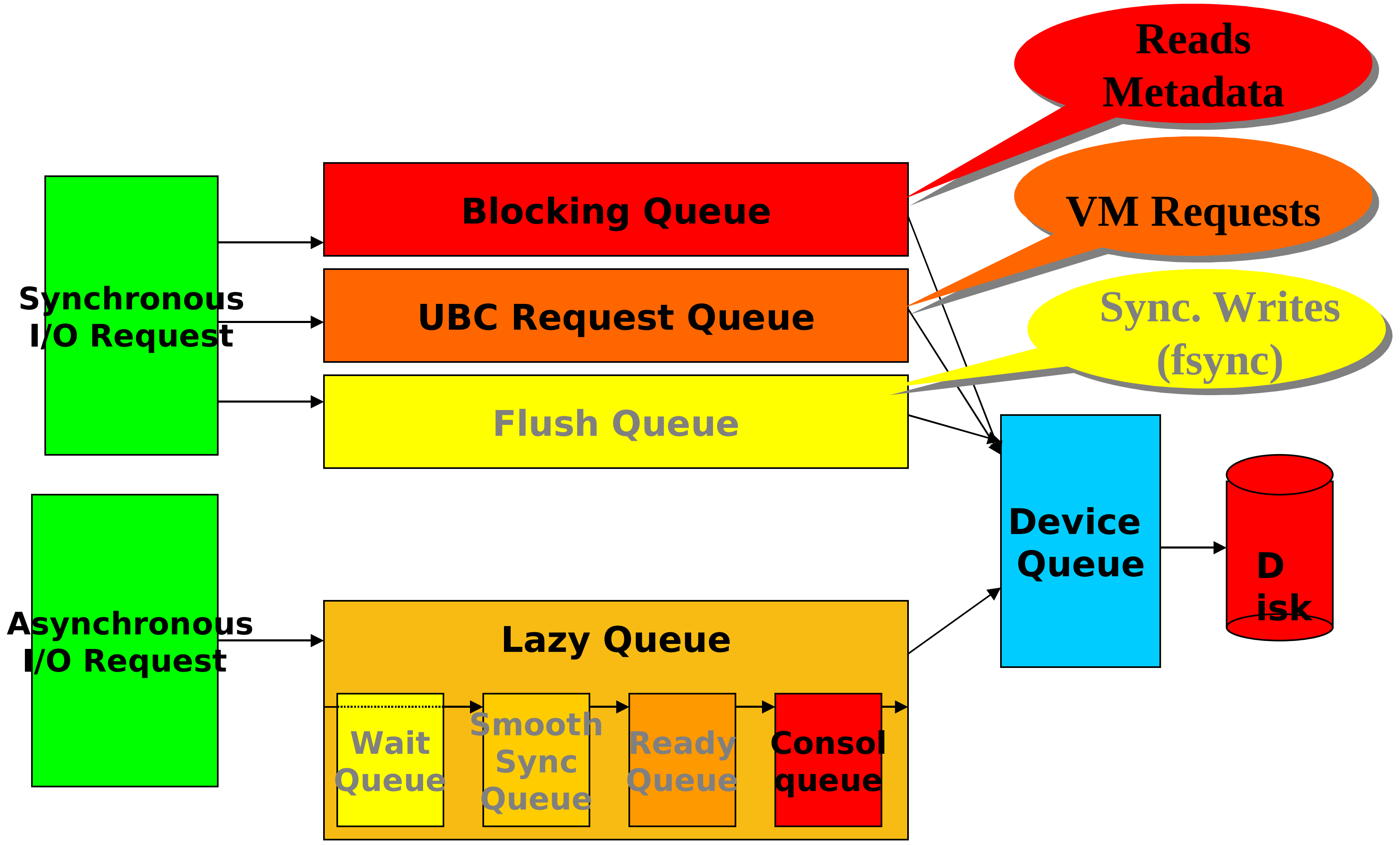

Figure taken from "What's New with AdvFS" by Thomas Sjolshagen.

AdvFS API in Tru64 UNIX V5

Introduced to support user-written backup and restore routines

- `advfs_clonefset`
- `advfs_get_fdmn_list`
- `advfs_get_file_attributes`
- `advfs_get_fset_list`
- `advfs_get_fset_quotas`
- `advfs_rmfset`
- `advfs_set_file_attributes`
- `advfs_set_fset_quotas`

True AdvFS system call

- **`msfs_real_syscall()`**
  - single call - many flavors
  - called through `MsfsSyscallp` (filled in when AdvFS is started) with the lower 32 bits of the KSEG address of `msfs_real_syscall()`
  - `MsfsSyscallp + 0xffffc000000000 = &msfs_real_syscall()`
- **First argument is the operation type**
  - used in a large case statement to determine the action
- **Source location: `msfs/bs/bs_misc.c`**

Prototype for msfs_real_syscall()

```
int
msfs_real_syscall(
    opTypeT     opType,
        /* in - msfs operation to be performed */
    libParamsT  *parmBuf,
        /* in - ptr to op-specific parameters 
buffer;*/
        /*      contents are modified. */
    int         parmBufLen
        /* in - byte length of parmBuf */
    );
```

Types of AdvFS system calls

- **60 types**
- **User interface**
  - library wrappers for system call
    - compiled into `/usr/shlib/libmsfs.so`
    - included from `msfs_syscalls.h`

Operation types within msfs_real_syscall() (1 of 5)

```
typedef enum {
    OP_NONE,
    OP_GET_BF_PARAMS,
    OP_SET_BF_ATTRIBUTES,
    OP_GET_BF_XTNT_MAP,
    OP_ADD_STG,
    OP_ADD_OVER_STG,
    OP_MIGRATE,
    OP_DMN_INIT,
    OP_GET_DMNNAME_PARAMS,
    OP_GET_DMN_PARAMS,
    OP_SET_DMN_PARAMS,
    OP_GET_DMN_VOL_LIST,
    OP_GET_VOL_PARAMS,
    OP_SET_VOL_IOQ_PARAMS,
```

Operation types within msfs_real_syscall() (2 of 5)

```
    OP_DUMP_LOCKS,
    OP_TRACE,
    OP_FSET_CREATE,
    OP_FSET_DELETE,
    OP_FSET_CLONE,
    OP_FSET_GET_INFO,
    OP_FSET_GET_ID,
    OP_GET_BFSET_PARAMS,
    OP_SET_BFSET_PARAMS,
    OP_ADD_VOLUME,
    OP_CRASH,
    OP_MSS_RESV1,
(...)
    OP_MSS_RESV17,
    OP_UNDEL_ATTACH,
```

Operation types within msfs_real_syscall() (3 of 5)

```
OP_UNDEL_DETACH,
OP_UNDEL_GET,
OP_GET_NAME,
OP_REM_STG,
OP_EVENT,
OP_TAG_STAT,
OP_SWITCH_LOG,
OP_GET_BF_IATTRIBUTES,
OP_SET_BF_IATTRIBUTES,
OP_MOVE_BF_METADATA,
OP_GET_VOL_BF_DESCS,
OP_REM_VOLUME,
OP_ADD_REM_VOL_SVC_CLASS,
OP_SWITCH_ROOT_TAGDIR,
OP_SET_BF_NEXT_ALLOC_VOL,
OP_DISK_ERROR,
```

Operation types within msfs_real_syscall() (4 of 5)

```
OP_FTX_PROF,
OP_REWRITE_XTNT_MAP,
OP_RESET_FREE_SPACE_CACHE,
OP_SET_NEXT_TAG,
OP_REM_NAME,
OP_REM_BF,
OP_FSET_RENAME,
OP_GET_LOCK_STATS,
OP_FSET_GET_STATS,
OP_GET_BKUP_XTNT_MAP,
OP_GET_VOL_PARAMS2,
OP_GET_GLOBAL_STATS,
OP_GET_SMSYNC_STATS,
OP_GET_IDX_BF_PARAMS,
OP_ADD_REM_VOL_DONE,
```

Operation types within msfs_real_syscall() (5 of 5)

```
    OP_GET_CLUDIO_XTNT_MAP,
    OP_SET_BFSET_PARAMS_ACTIVATE,
    OP_SS_SET_LICENSE,
    OP_SS_GET_LICENSE,
    OP_SS_DMN_OPS,
    OP_SS_GET_PARAMS,
    OP_SS_SET_PARAMS,
    OP_SS_GET_FRAGLIST,
    OP_SS_GET_HOTLIST
} opIndexT;
```

Domains and volumes

Utilities:

• `msfs_dmn_init()`	`mkfdmn`
• `msfs_add_volume()`	`addvol`
• `advfs_remove_volume()`	`rmvol`
• `msfs_get_dmn_params()`	`showfdmn`
• `msfs_syscall_op_get_dmn_vol_list()`	

Prototype for msfs_dmn_init() (1 of 2)

```
mlStatusT
msfs_dmn_init(
    char* domain,                /* in - bf domain name */
    int maxVols,                 /* in - maximum number of
                                    virtual disks */
    u32T logPgs,                 /* in - number of pages
                                    in log */
    mlServiceClassT logSvc,      /* in - log service
                                    attributes */
    mlServiceClassT tagSvc,      /* in - tag directory
                                    service attributes */
    char *volName,               /* in - block special
                                    device name */
    mlServiceClassT volSvc,      /* in - service class */
    u32T volSize,                /* in - size of the
                                    virtual disk */
```

Prototype for msfs_dmn_init() (2 of 2)

```
u32T bmtXtntPgs,            /* in - number of pages
                               per BMT extent */
u32T bmtPreallocPgs,        /* in - number of pages to
                      be preallocated for the BMT */
u32T domainVersion,         /* in - on-disk version
                               of domain */
mlBfDomainIdT* bfDomainId  /* out - domain id */
);
```

Prototype for msfs_add_volume()

```
mlStatusT
msfs_add_volume(
    char *domain,             /* in - domain name */
    char *volName,            /* in - block special
                                 device name */
    mlServiceClassT *volSvc, /* in/out –
                                 service class */
    u32T volSize,             /* in - size of the
                                 virtual disk */
    u32T bmtXtntPgs,          /* in - number of pages
                                 per BMT extent */
    u32T bmtPreallocPgs,      /* in - number of pages to
                          be preallocated for the BMT */
    mlBfDomainIdT *bfDomainId,/* out - domain id */
    u32T *volIndex            /* out - vol index */
 );
```

Prototype for advfs_remove_volume()

```
mlStatusT
advfs_remove_volume(
    mlBfDomainIdT bfDomainId,   /* in */
    u32T volIndex,  /* in */
    u32T forceFlag  /* in */
    );
```

Prototype for msfs_syscall_op_get_dmn_params()

```
mlStatusT
msfs_syscall_op_get_dmn_params(
   libParamsT *libBufp
);
```

Filesets

- **System call**
- **Utility**

`msfs_fset_create()`	`mkfset`
`msfs_fset_clone()`	`clonefset`
`msfs_fset_delete()`	`rmfset`
`msfs_set_bfset_params()`	`chfsets`

- **And many more**

Prototype for msfs_fset_create()

```
mlStatusT

msfs_fset_create(
    char *domain,                /* in - domain name */
    char *setName,               /* in - set's name */
    mlServiceClassT reqServ, /* in - required service
                                  class */
    mlServiceClassT optServ, /* in - optional service
                                        class */
    u32T userId,                 /* in - user id */
    gid_t quotaId,               /* in - group ID for
                                    quota files */
    mlBfSetIdT *bfSetId          /* out - bitfile set id */
    );
```

Miscellaneous operations

- **`advfs_migrate()`**
  - moves blocks of open file
- **`msfs_syscall_op_set_bf_attributes()`**
  - stripes a file
- **`msfs_undel_attach()`**
  - attaches a trashcan directory
- **`advfs_ss_set_params()`**
  - sets parameters for vFast
- **`advfs_ss_get_hotlist()`**
  - gets list of hot files from vFast

Startup and recovery overview

- **Begins with a `mount(2)` system call**
  - Or `vfs_mountroot()` which does part of the job
- **Invokes `msfs_mount()` found in `msfs_vfsops.c`**
- **Calls `get_domain_disks()`**
  - searches `/etc/fdmns/domain` (for list of virtual disks)
- **Calls `advfs_mountfs()`(found in `msfs_vfsops.c`) to do the real work**

Mounting the file system

- **Obtains names of the fileset**
- **Activates the bitfile-set**
  - with `bs_bfset_activate()`
- **Initializes various in-memory structures**
- **Opens significant bitfiles**
  - tagdir, root, fragment
- **Links file system into mount list**

Activating the bitfile-set

- **`bs_bfset_activate_int()`**
- **Activates or finds a domain structure**
  - with `bs_bfdmn_tbl_activate()`
- **Finds the appropriate bitfile-set**
  - with `bs_bfs_find_set()` (which looks in the root tag directory)

Activating the domain – search for virtual disks

- **`bs_bfdmn_tbl_activate()`**
- **If domain not active:**
  - search virtual disks of domain
  - check for consistencies:
    - virtual disk count on disk
    - number of links in `/etc/fdmns`
  - find the transaction log
  - activate the domain
    - with `bs_bfdmn_activate()`

Activating the domain – full activation

- **`bs_bfdmn_activate()`**
- **Open the transaction log**
  - with `lgr_open()`
- **Open root tag directory**
  - when appropriate
- **Start crash recovery activities**
  - with `ftx_bfdmn_recovery()`
- **Remove delete-pending filesets**

Recovering a domain

- `ftx_bfdmn_recovery()`
- **Three recovery passes**
  - pass 1 -- RBMT file
  - pass 2 -- Other reserved metadata bitfiles
  - pass 3 -- Other metadata bitfiles
- **After the three passes**
  - perform any further recovery actions

Recovery pass (1 of 2)

- **Recovers Domain Consistency**
- **`ftx_recovery_pass()`**
- **Scan the log**
  - read a record
  - put in slot for this FTX ID
    - allocate new one if needed
  - On pass 1
    - buffer continuation and root done record
  - If record matches current pass
    - perform record image redo records
    - perform operation redo record

Recovery pass (2 of 2)

- if level and member are zero, free the FTX slot
- **Loop through remaining FTX slots**
  - if level is not zero:
    - this is part of an uncompleted transaction
    - fail the transaction

```
Execute the undo records
In pass appropriate manner
```

  - if level is zero, better do the root done operations

BAS-level storage allocation

- **Disk free storage list**
  - starting address and size of free storage
  - may not be large enough to hold all free storage locations (especially if disk is very fragmented)
- **BAS-level routines add storage**
  - without much regard to efficiency
  - though they will join adjacent grants into one extent (thus small sequential extents may become one)

FAS-level storage allocation

- **If file is being written sequentially:**
  - data space is preallocated in page sizes of
    - MIN(pg_to_write/4, MAX_PREALLOC_PAGES)

      ```
      pg_to_write is present page number
      MAX_PREALLOC_PAGES is presently 16
      ```
  - if this fails, data space is allocated as needed
  - BAS-level will combine adjacent allocations

Truncating bitfiles

- **When bitfile closes:**
  - AdvFS sees if last page should be allocated in the fragment file
- **If necessary:**
  - a fragment is allocated
  - last page is now unused
- **If there are unused pages at end of file:**
  - unused pages are deallocated
  - this can result in the release of small disk areas

Creating a clone

`fs_fset_clone()`

- Perform various access checks

`bs_bfs_clone ()`

- Create new bitfile-set
- Copy original’s tagfile to clone’s tagfile
- Make appropriate modifications to bitfile-set attributes record

Files open when cloning may not have perfect snapshots

Prototype for fs_fset_clone()

```
/*
 * fs_fset_clone
 *
 * Creates a clone file set of an 'original' file set.
 */
statusT
fs_fset_clone(
    char *domain,         /* in - name of set's domain */
    char *origSetName,    /* in - name of orig set */
    char *cloneSetName, /* in - name of new
                              clone set */
    bfSetIdT *retCloneBfSetId,  /* out - clone
                              set's id */
    long xid              /* in - CFS transaction id */
    )
```

Writing to a cloned original

- **Bitfile pages of original are copy-on-write**
- **On first modification of bitfile**
  - new mcell is allocated for clone bitfile
  - original and clone primary mcells are now different
- **On first modification of bitfile page**
  - new extent is allocated for clone bitfile
  - original data is copied to clone’s extent
  - clone extent map has holes for original data

Reading from a clone

- **See if clone bitfile has requested page**
- **If not:**
  - see if page really is within range of clone bitfile
  - check extent maps of original bitfile for page
- **If a page is written into a hole of the original**
  - clone must be given a ‘permanent hole’ extent

Deleting bitfile from cloned original

- **Must ensure data is available for clone after deletion from original fileset**
- **Original fileset is marked delete with clone**
  - it exists until clone fileset is deleted
- **Not the same as unlinking a file from fileset**
  - FAS-level understands multiple links for one file

Deleting a bitfile

- Set bitfile attributes state to `BSRA_DELETING`
- Delete the bitfile from the tagfile
- Add bitfile to DDL, Deferred-Delete List for disk
  - if system crashes, on recovery DDL is processed
- Wait for bitfile to close to reap the storage

Closing a deleted bitfile

Carefully delete the storage

- **Perform a series of root transactions**
  - pin several pages of SBM
  - update the storage bit map to delete extents
  - update the `delRst` field of bitfile's extent map to point to next extent to delete

Carefully delete the bitfile's mcell chain

- **Perform a series of continued transactions**
  - pin several pages BMT
  - free the mcells on those pages
  - start a continuation transaction which knows next mcell to delete

Migrating a bitfile

- **Allocate new target storage**
  - place target on deferred delete list (if system crashes, it is gone on recovery)
- **Put target storage on copy extent map list**
  - modifications will go to both source and target!
- **Copy blocks -- source to target**
- **Flush blocks**
- **Switch roles on target and source**
  - source will be reclaimed

Deleting a fileset

- **Add bitfile-set to domain’s delete pending list**
- **Iterate through the tags of the bitfile-set**
  - delete each bitfile
- **Remove bitfile-set from bitfile-set delete pending list**
- **Delete tagfile**

AdvFS threads

- **Created by kernel idle thread routine (PID 0)**
- **Receive typed messages on queue**
- **Block with `cond_wait()`**

Fragment bitfile thread

- **One per system**
- **Deallocates frag groups of type 0**
  - when there are too many
  - target is `AdvfsMinFragGrps` (default is 16)
- **Awakened from `frag_group_dalloc()`**
  - with message containing bitfile-set ID

I/O thread

- **For `START_MORE_IO` messages**
  - calls `bs_startio()` for a virtual disk
  - awakened by `bs_osf_complete()` when queue is small
- **For `LF_PB_CONT` messages**
  - check if a log flush continue or a pin block continue is needed
  - awakened by `bs_io_complete()` if `HiFlushLSN` has changed

Bitfile access thread

- **Allocates `bfAccess` structures**
- **Awakened by `bfAccess` allocation routines**
- **For `ALLOC_BFAP_NORMAL` messages**
  - respects `AdvfsAccessMaxPercent` limit
- **For `ALLOC_BFAP_ROOT` messages**
  - gives root 1% more than `AdvfsAccessMaxPercent`
- **For `ALLOC_BFAP_NORMAL` messages**
  - ignores `AdvfsAccessMaxPercent` limit

Extend RMBT thread

- **For `FINSH_DIR_TRUNC` messages**
  - allocates a new page to the RBMT
  - awakened when there are only two free Mcells in the RBMT

AdvFS cleanup thread

- **For `FINSH_DIR_TRUNC` messages**
  - truncates space from directory
  - awakened by routines to insert directory entries
- **For `CLEANUP_CLOSED_LIST` messages**
  - moves bfAccess structures from closed to free list
  - awakened by routines which allocate bfAccess structures
- **For `DEALLOCATE_BFAPS` messages**
  - deallocates bfAccess structures
  - doesn’t seem to be used in V5.1B
- **For `UPDATE_BAD_FRAG_GRP_HDR` messages**
  - marks a fragment group header as bad
  - awakened by routines that allocate fragments

Freeze thread

- Added in Tru64 UNIX V5.1A
  - supports functionality of `freezefs` and `thawfs`
- Maintains a queue of timeouts for frozen domains
- Responsible for initiating a file system thaw at timeout

AdvFS vFast threads

- **Added in Tru64 UNIX V5.1B**
  - Supports vFast
- **Three types of threads**

– **Boss**	**Only one of these**
– **Monitor**	**Only one of these**
– **List**	
– **Worker**	

- Source files:
  - `msfs/bs/vfast.c`
  - `msfs/msfs/vfast.h`

vFast boss threads

- **Creates and manages the thread pools**
- **Terminates and restarts thread pools when appropriate**
- **Adjust the rate at which hot file messages are generate**
- **Executes `ss_boss_thread`**

- **For the most part, follows the orders of the monitor thread**

vFast monitor thread

- Monitors message queues
  - tells the boss thread when to create new threads
  - tells the boss thread when to adjust rate of hot file messages
- Periodically checks I/O load balance
  - to see if any files should be moved to lightly loaded volume
- Checks degree of fragmentation within domain
  - to see if any files should be defragmented
- Executes ss_monitor_thread
- Tells the boss what to do

vFast list thread pool

- Maintains list of “hot files”
  - using information regarding bitfile page references
- Maintains list of fragment files
  - using information provided by monitor

- Executes `ss_list_thd_pool`

vFast worker thread

- Waits for messages on the lists
- Invokes `ss_vd_migrate` to move files
- Only works when system I/O load is low
- Executes ss_work_thd_pool

- Finally, a thread that does some real work invent

Learning check

Lab 4

i n v e n t