GSoC 2026 Proposal

Btrfs Write Support for Haiku

Implementing full read-write btrfs filesystem support in Haiku OS

Submitted by Anuj Billore | Haiku, Inc. | Google Summer of Code 2026

Personal Information

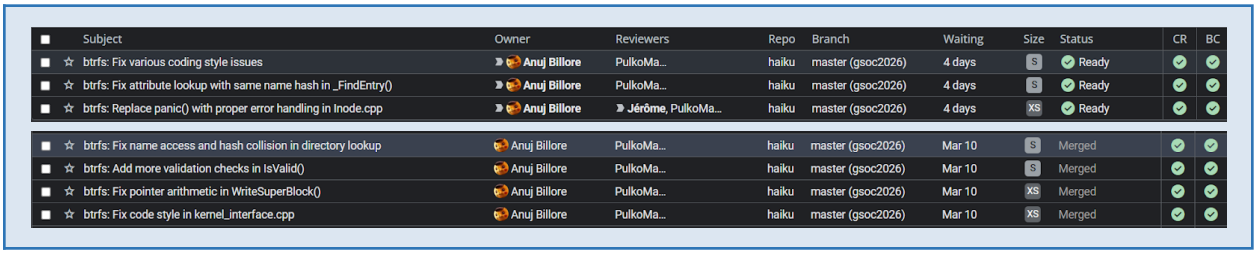
Full Name	Anuj Billore
Timezone	IST (UTC+5:30)
Primary Email	anujbillore2005@gmail.com
Mailing List Email	anujbillore3107@gmail.com
IRC username (oftc.net)	anujbillore
Trac username	anujbillore
Gerrit Changes	#10446, #10447 (CR+2), #10448, #10449 (CR+2), #10450, #10451, #10452 - review.haiku-os.org Gerrit Page : https://review.haiku-os.org/q/owner:anujbillore2005@gmail.com
GSoC Full-time?	Yes
Hours per week	35–40 hours
Obligations	University semester ends 10 May 2026. No conflicts during GSoC period.
University requirement?	No. Personal interest in systems programming and open source
How I found Haiku	Through a senior's GSoC '22 final report on LinkedIn. The filesystem work looked compelling.
Other GSoC orgs?	No. Dedicated entirely to Haiku.
Last day of exams	10th May 2026
First day of autumn classes	28th August 2026

1. Introduction

My name is Anuj Billore, a 2027 undergraduate student at SGSITS, Indore, India studying B.Tech in Computer Science. My primary technical interests are systems-level C++, kernel programming, filesystem internals, and low-level software engineering. I have been writing C++ for three years and am comfortable navigating large, unfamiliar codebases, working with Haiku's Jam-based build system, and submitting production-quality patches through the Gerrit review workflow.

I am not new to Haiku or to the btrfs codebase. Before writing this proposal, I established practical familiarity through deliberate, incremental contribution:

- Built Haiku from source on WSL2 and set up a full Haiku VM with the QEMU image for native testing.
- Studied every source file in `src/add-ons/kernel/file_systems/btrfs/` - not just the headers, but the full implementation of BTree.cpp, Inode.cpp, Journal.cpp, ExtentAllocator.cpp, Volume.cpp, and kernel_interface.cpp.
- Submitted 7 patches to Haiku Gerrit targeting the btrfs driver (changes #10446–#10452). Of these, #10447 and #10449 have received CR+2 from PulkoMandy and are ready to merge. The remaining five are under review.
- Exchanged multiple rounds of feedback with PulkoMandy on both Gerrit and the haiku-development mailing list, establishing an active working relationship before GSoC begins.



Subject	Owner	Reviewers	Repo	Branch	Waiting	Size	Status	CR	BC
btrfs: Fix various coding style issues	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	4 days	S	Ready	✓	✓
btrfs: Fix attribute lookup with same name hash in _FindEntry()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	4 days	S	Ready	✓	✓
btrfs: Replace panic() with proper error handling in Inode.cpp	Anuj Billore	Jérôme, PulkoMa...	haiku	master (gsoc2026)	4 days	XS	Ready	✓	✓
btrfs: Fix name access and hash collision in directory lookup	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	S	Merged	✓	✓
btrfs: Add more validation checks in IsValid()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	S	Merged	✓	✓
btrfs: Fix pointer arithmetic in WriteSuperBlock()	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	XS	Merged	✓	✓
btrfs: Fix code style in kernel_interface.cpp	Anuj Billore	PulkoMa...	haiku	master (gsoc2026)	Mar 10	XS	Merged	✓	✓

The 7 patches cover: fixing code style in kernel_interface.cpp and btrfs_disk_system.cpp and Inode.h, fixing pointer arithmetic in WriteSuperBlock(), adding validation checks in IsValid(), fixing name access and hash collision handling in directory lookup, and clarifying a non-obvious design choice in PreviousLeaf(). These patches are not cosmetic; they fix real bugs and close code-quality gaps that would block write support work downstream.

I chose the btrfs write support project because it sits at the intersection of what I find most technically compelling: kernel VFS integration, B-Tree algorithms, copy-on-write semantics, and transactional correctness. The prior GSoC work by Hy Che (2017) and Bharathi Ramana Joshi (2019) built a solid foundation - the metadata layer, the extent allocator skeleton, the journal/transaction wrappers, and CoW primitives are all present. What remains is a clearly defined, tractable set of missing pieces that I have analyzed in depth. This proposal describes exactly what those pieces are and how I plan to implement them.

2. Project Proposal

2.1 Title

Btrfs Write Support for Haiku - Implementing full read-write filesystem operations including file creation, data writes, directory management, and volume initialization.

2.2 Background and Problem Statement

Haiku currently mounts btrfs volumes in read-only mode. The read-only restriction is not a limitation of the underlying infrastructure - the transaction API, block cache, CoW primitives, and extent allocator skeleton are all present. The restriction is a consequence of specific, well-defined gaps in the implementation that have accumulated since 2019.

This matters for Haiku's practical usability. Btrfs has been the default filesystem on Fedora since 2022 and is increasingly common on Linux systems. A user booting Haiku alongside Linux, or accessing a data drive shared between both systems, cannot write to any btrfs partition from Haiku. This is a concrete interoperability gap that this project will close.

2.3 Current Status - What Works

The following subsystems are functional and have been verified by reading the source:

- BTree CoW (`CopyOnWrite`, `InternalCopy`, `InsertEntries`, `RemoveEntries`)
- Transaction system - `Journal` wraps `cache_start_transaction` / `cache_end_transaction`
- ExtentAllocator - `AllocateTreeBlock` and `AllocateDataBlock` (first-fit strategy)
- Inode CRUD - `Create`, `Insert`, `Remove`, `MakeReference`, `Dereference`
- Directory operations - `create_dir`, `remove_dir`, `unlink`
- WriteSuperBlock - CRC32c checksum calculated correctly
- ReadAt - inline, regular, zlib, and zstd extents all functional

2.4 Current Status - What Is Missing

The following gaps were identified by reading every source file in the btrfs driver. They are ordered by implementation priority:

Gap 1: B-Tree Node Split / Merge (Critical)

In `BTree::MakeEntries()`, when a leaf node overflows (`B_DEVICE_FULL`), execution halts at a TODO comment. Similarly, `BTree::RemoveEntries()` detects underflows (`B_DIRECTORY_NOT_EMPTY`) but also halts. Without node splitting, any write that fills a leaf node will permanently fail - making write support effectively non-functional for any non-trivial workload.

```
680         status = path->InternalCopy(transaction, 1);
681         if (status != B_OK)
682                 return status;
683
684         status = path->CopyOnWrite(transaction, 0, slot, num, length);
685         if (status == B_DEVICE_FULL) {
686                 // TODO: push data or split node
687                 return status;
688         }
689
690         if (status != B_OK)
691                 return status;
692         return slot;
```

```
748         if (status != B_OK)
749                 return status;
750
751         status = path->CopyOnWrite(transaction, 0, slot, num, length);
752         if (status == B_DIRECTORY_NOT_EMPTY) {
753                 // TODO: merge node or push data
754         }
755
756         return status;
757 }
758
```

Gap 2: WriteAt() Not Implemented

The `Inode::WriteAt()` function is completely absent from Inode.cpp. The VFS read path is functional but the write path has no implementation at all - not even a stub that returns `B_NOT_SUPPORTED`. This is the core data write function that must be implemented.

Gap 3: fs_create is NULL in VFS ops

In `gBtrfsVnodeOps` in kernel_interface.cpp, `fs_create` is set to NULL. File creation is completely disconnected from the VFS layer.

```
anuj@LAPTOP-KCT8U8VA:~$ sed -n '1154,1200p' ~/haiku-dev/haiku/src/add-ons/kernel/file_systems/btrfs/kernel_interface.cpp
fs_vnode_ops gBtrfsVnodeOps = {
        /* vnode operations */
        &btrfs_lookup,
        NULL, // btrfs_get_vnode_name - optional, and we can't do better than the
              // fallback implementation, so leave as NULL.
        &btrfs_put_vnode,
        NULL,   // btrfs_remove_vnode,

        /* VM file access */
        &btrfs_can_page,
        &btrfs_read_pages,
        NULL,   // btrfs_write_pages,                    <-- write_pages = NULL

        NULL,   // io()
        NULL,   // cancel_io()

        &btrfs_get_file_map,

        &btrfs_ioctl,
        NULL,
        NULL,   // fs_select
        NULL,   // fs_deselect
        NULL,   // fs_fsync,

        &btrfs_read_link,
        NULL,   // fs_create_symlink,

        NULL,   // fs_link,
        &btrfs_unlink,
        NULL,   // fs_rename,

        &btrfs_access,
        &btrfs_read_stat,
        &btrfs_write_stat,
        NULL,   // fs_preallocate

        /* file operations */
        NULL,   // fs_create,                            <-- fs_create = NULL
        &btrfs_open,
        &btrfs_close,
        &btrfs_free_cookie,
        &btrfs_read,
        &btrfs_write,                                    <-- btrfs_write present
                                                             but returns B_NOT_SUPPORTED

        /* directory operations */
        &btrfs_create_dir,
        &btrfs_remove_dir,
```

```
status_t
btrfs_write(fs_volume* _volume, fs_vnode* _node, void* _cookie, off_t pos,
                const void* buffer, size_t* _length)
{
        Volume* volume = (Volume*)_volume->private_volume;
        Inode* inode = (Inode*)_node->private_node;

        if (volume->IsReadOnly())
                return B_READ_ONLY_DEVICE;

        if (pos < 0)
                return B_BAD_VALUE;

        if (!inode->IsFile())
                return B_BAD_VALUE;

        return B_NOT_SUPPORTED;
}
```

Gap 4: btrfs_write_stat - File Size Changes Rejected

The `btrfs_write_stat()` function currently returns `B_NOT_SUPPORTED` for any `st_size` change. Truncation, append, and file creation all require stat updates - without this, none of them can work end-to-end.

Gap 5: Volume::Initialize() Incomplete

Three TODOs remain in `Volume::Initialize()`: the extent tree, chunk tree, and fs tree are not initialized, and the superblock is not written at transaction commit. This blocks `mkfs`-style volume creation from Haiku.

```
Volume.cpp  –  Volume::Initialize()
385                 return B_ERROR;
386         off_t numBlocks = deviceSize / sectorSize;
387         // create valid superblock
388         fSuperBlock.Initialize(label, numBlocks, blockSize, sectorSize);
389         fBlockSize = fSuperBlock.BlockSize();
390         fSectorSize = fSuperBlock.SectorSize();
391         // TODO(lesderid):      Initialize remaining core structures      → TODO 1: Initialize extent/chunk/fs trees
392         //                      (extent tree, chunk tree, fs tree, etc.)
393         status_t status = WriteSuperBlock();
394         if (status < B_OK)
395                 return status;
396         fBlockCache = opener.InitCache(fSuperBlock.TotalSize() / fBlockSize,
397                 fBlockSize);
398         if (fBlockCache == NULL)
399                 return B_ERROR;
400         fJournal = new(std::nothrow) Journal(this);
401         if (fJournal == NULL)
402                 RETURN_ERROR(B_ERROR);
403         // TODO(lesderid):      Perform secondary initialization (in transactions)   → TODO 2: Secondary init in transactions
404         //                      (add block groups to extent tree, create root dir, etc.)
405         Transaction transaction(this);
406         // TODO(lesderid): Write all superblocks when transactions are committed   → TODO 3: Write superblocks on transaction commit
407         status = transaction.Done();
408         if (status < B_OK)
409                 return status;
410         opener.RemoveCache(true);
```

Gap 6: Read-Only Flag Hardcoded

In `Volume::Mount()`, `B_MOUNT_READ_ONLY` is hardcoded. `B_DISK_SYSTEM_SUPPORTS_WRITING` is commented out in `btrfs_disk_system.cpp`. These two lines must be changed as a prerequisite to all write operations.

Gap 7: Journal Sync Commented Out

In `Journal::_TransactionDone()`, the `cache_sync_transaction()` call is commented out. Without synchronous transaction commits, write operations can appear to succeed but data can be lost on power failure or crash. This must be restored and carefully tested.

```
Journal.cpp  -  Journal::_TransactionDone()
43  status_t
44  Journal::_TransactionDone(bool success)
45  {
46          if (!success) {
47                  cache_abort_transaction(fVolume->BlockCache(), fTransactionID);
48                  return B_OK;
49          }
50          cache_end_transaction(fVolume->BlockCache(), fTransactionID,
51                  &_TransactionWritten, this);
52          // cache_sync_transaction(fVolume->BlockCache(), fTransactionID);
53          return B_OK;
54  }
```

Sync disabled!
Data loss on crash/
power failure possible

Gap 8: ExtentAllocator - No FreeDataBlock, No DUP/RAID

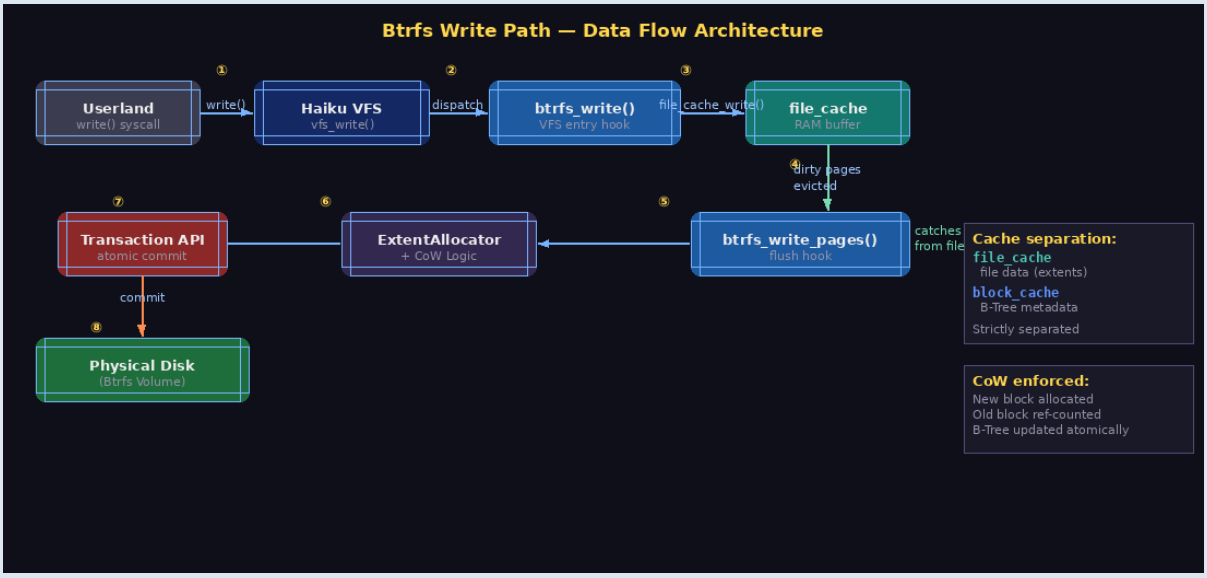
The `ExtentAllocator` can allocate blocks but has no `FreeDataBlock()` API. Old blocks are never freed, causing rapid ENOSPC degradation under CoW. Additionally, both `AllocateTreeBlock` and `AllocateDataBlock` contain TODO comments for DUP and RAID profiles - these are deferred to stretch goals.

Gap 9: Miscellaneous Bugs

- Operator precedence bug in `btrfs_write_fs_info()`: `mask & ~FS_WRITE_FSINFO_NAME != 0` should use parentheses - `(mask & ~FS_WRITE_FSINFO_NAME) != 0`
- All attribute write operations (`write_attr`, `create_attr`, `remove_attr`) return `EROFS` unconditionally
- `free_blocks` is hardcoded to 0 in `btrfs_read_fs_info()`
- `FindBlock()` returns `B_BAD_DATA` on inline extents - crashes any read path that encounters them after a write

2.5 Architecture - How Write Support Fits Together

The write path must respect Haiku's strict separation between block_cache (for B-Tree metadata) and file_cache (for file data). The diagram below shows the full data flow from a userland write() call to physical disk:



2.6 Implementation Plan

The implementation is divided into six sequential phases, each building on the previous:

Phase 1: Unlock the Volume (Community Bonding Week 1) - May 1 to May 11

Prerequisites before any write can reach disk:

- Remove `B_MOUNT_READ_ONLY` from `Volume::Mount()`
- Re-enable `B_DISK_SYSTEM_SUPPORTS_WRITING` in `btrfs_disk_system.cpp`
- Refactor `ExtentAllocator::Initialize()` to handle modern mixed block groups (DATA + METADATA combined), which currently causes initialization to panic on images created by recent btrfs-progs
- Fix operator precedence bug in `btrfs_write_fs_info()`
- Milestone: **touch** on a btrfs volume in Haiku VM without kernel panic

Phase 2: VFS Namespace Hooks (Pre-Midterm Weeks 1-3) - May 25 to June 14

Connect file/directory operations to the VFS layer:

- Implement `btrfs_create` - wire into `gBtrfsVnodeOps`; allocate inode, insert into fs tree, update directory
- Implement `btrfs_mkdir` / `btrfs_rmdir` - create and remove directory entries in the B-Tree
- Implement `btrfs_unlink` - remove file entry, decrement inode reference count
- Implement `btrfs_write_stat` - update inode size, mtime, and permissions after writes
- Fix CRC32c hash collision TODO in `DirectoryIterator::Lookup()` - add string-matching fallback for hash collisions
- Milestone: Create, stat, and delete files and directories on a live btrfs volume

Phase 3: Data Write Path-btrfs_write (Pre-Midterm Weeks 4–5) -June 15 to June 28

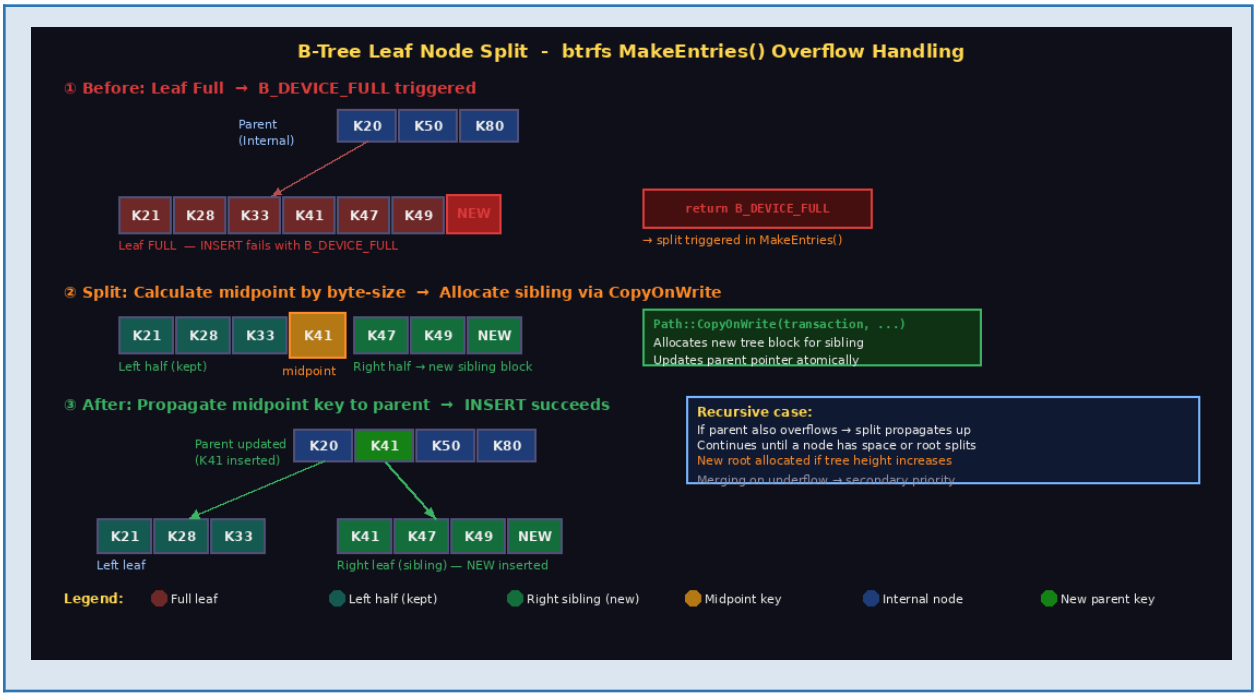
Connect userland write() calls to the VFS cache:

- Implement `Inode::WriteAt()` - the core write function, routing user buffers into Haiku's `file_cache_write()`
- Wire `btrfs_write` in `gBtrfsVnodeOps` to call `WriteAt()`
- Handle write validation: read-only check, negative offset, non-file inode, buffer bounds
- Milestone: Append to and overwrite existing files on a btrfs volume in the Haiku VM

Phase 4: B-Tree Node Splitting (PreMidterm Week 6 + PostMidterm Week1) |Midterm Deadline : July 10

The hardest and most critical piece. When a B-Tree leaf fills up during `MakeEntries()`, the node must be split:

- Allocate a new sibling leaf block via `Path::CopyOnWrite`
- Calculate the midpoint by payload byte-size (not item count) per btrfs spec
- Move the upper half of items to the new sibling
- Propagate the new bounding key up to the parent node recursively
- Handle the case where the parent also overflows - recursive splitting up the tree height
- Implement proactive merging on underflow (`RemoveEntries()`) as secondary priority
- Milestone: Write enough files to force multiple leaf splits - verify tree integrity with btrfsck on Linux



Phase 5: Regular Extent Flush Hook (PostMidterm Weeks 2-3) - July 17 to July 30

Catch dirty pages evicted from file_cache and commit them to disk:

- Implement `btrfs_write_pages` - the flush hook that catches dirty pages from Haiku's file_cache
- Implement `Volume::GetNewDataBlock()` - logical-to-physical address translation for file data blocks
- Implement `ExtentAllocator::FreeDataBlock()` - decrement reference counts and free blocks when count reaches zero
- Generate `BTRFS_EXTENT_DATA_REGULAR` items and insert into the fs tree
- Milestone: Write a file larger than the file_cache page size and verify data integrity after unmount

Phase 6: Atomic Transactions and Checksums (Post MidTerm Weeks 4–5) July 31 to August 16

Wrap all writes in atomic transactions and maintain checksum tree integrity:

- Wrap all write operations in `Transaction::Start()` / `Transaction::Done()`
- Restore `cache_sync_transaction()` in `Journal::_TransactionDone()` - guarantee synchronous commits
- Calculate CRC32c checksums for newly written file data and update `fChecksumTree` atomically alongside extent insertions
- Milestone: Mount Haiku-written btrfs volume in Linux and run `btrfs check` - zero errors

3. Testing Strategy

3.1 btrfs_shell (Host-side, Rapid Iteration)

The `btrfs_shell` binary (built from `src/tests/add-ons/kernel/file_systems/btrfs/btrfs_shell/`) runs on Linux without a VM. I have verified it builds successfully (with uuid-dev installed) and can mount a btrfs image created by mkfs.btrfs.

3.2 Native Haiku VM Testing

The compiled btrfs binary is deployed directly to `/boot/system/non-packaged/add-ons/kernel/file_systems/` by modifying the `sBtrfsFileSystem` struct name in `kernel_interface.cpp` to avoid cache conflicts. This reduces the test cycle to seconds. Each milestone will be verified by running actual filesystem operations in the Haiku VM and checking the results.

3.3 Cross-OS Integrity Verification

After each major milestone, the modified btrfs image will be mounted in Linux and verified with `btrfs check`. This ensures the on-disk format is correct by an independent implementation. Any structural corruption will be detected before it is buried under subsequent writes.

3.4 Stress Testing

In the final phase, `bonnie++` will stress-test the extent allocator and flush hooks under heavy I/O. The existing `btrfs_test.sh` framework will be extended with write tests modeled on the `src/tests/add-ons/kernel/file_systems/ext2/extents_test.sh` pattern (fsx + fstorture), deferred until write support is stable.

4. Timeline

Period	Goals & Milestones
Community Bonding Period (May 1 to May 24)	• Remove B_MOUNT_READ_ONLY from Volume::Mount() • Fix ExtentAllocator::Initialize() for mixed block groups • Fix operator precedence bug in btrfs_write_fs_info() • Milestone: touch on btrfs without kernel panic
Phase 2: VFS Namespace Hooks (May 25 to June 14)	• Implement btrfs_create, btrfs_mkdir, btrfs_rmdir, btrfs_unlink • Implement btrfs_write_stat • Fix CRC32c hash collision in DirectoryIterator::Lookup() • Milestone: Create, stat, delete files and directories
Phase 3: Data Write Path (June 15 to June 28)	• Implement Inode::WriteAt() routing to file_cache_write() • Wire btrfs_write in gBtrfsVnodeOps • Handle write validation and error cases • Milestone: Append and overwrite files in Haiku VM
Phase 4: B-Tree Node Splitting (June 29 to July 16) + Midterm July 10	• Implement recursive leaf node splitting in MakeEntries() • Midpoint calculation by byte-size, sibling allocation, parent propagation • Proactive merge on underflow (secondary priority) • Milestone: Force multiple leaf splits - btrfs check clean on Linux
Phase 5: Regular Extent Flush Hook (July 17 to July 30)	• Implement btrfs_write_pages flush hook • Implement Volume::GetNewDataBlock(), ExtentAllocator::FreeDataBlock() • Generate BTRFS_EXTENT_DATA_REGULAR items • Milestone: Large file writes survive unmount with data integrity
Phase 6: Transactions and Checksums (July 31 to August 16)	• Wrap all writes in Transaction::Start() / Done() • Restore cache_sync_transaction() in Journal • Calculate CRC32c checksums, update fChecksumTree atomically • Run bonnie++ stress test • Milestone: btrfs check clean after heavy I/O
Final Submission Week (August 17 to August 24)	• Cross-OS integrity: mount Haiku-written images in Linux • Run targeted xfstests write subset on Haiku • Update source-tree documentation for all new APIs • Submit final evaluation

5. Risk Management

B-Tree node splitting is the highest-risk component. If it takes longer than anticipated, node merging on underflow will be deferred to post-GSoC - a filesystem can operate correctly without merging (it just becomes gradually less space-efficient). This is the same deferral strategy used by the 2019 contributor.

If CRC32c checksum tree updates prove more tightly coupled to chunk mapping than expected, they can be implemented after the core write path is stable - btrfs mounts without checksum errors if the checksum tree simply lacks entries for new blocks, though Linux's `btrfs check` will report warnings. Full checksum coverage will be completed before the final evaluation.

The following features are out of scope for GSoC 2026 and are explicitly deferred to avoid scope creep:

- Compressed extent writes (zlib/zstd)
- Snapshot and subvolume creation
- Advanced allocation profiles (RAID/DUP) - TODO markers exist but will not be addressed
- Volume::Initialize() / mkfs-style volume creation from Haiku

6. Stretch Goals

If all six phases complete ahead of schedule:

- Extended attribute write support (`write_attr`, `create_attr`, `remove_attr`)
- Implement `NumFreeBlocks()` and wire it into `btrfs_read_fs_info()` to fix the hardcoded `free_blocks = 0`
- Begin `Volume::Initialize()` to allow Haiku to format blank partitions as btrfs

7. After Google Summer of Code

The work this project completes is a foundation, not a finish line. Btrfs write support opens problems that did not exist before , once files can be written, the correctness of truncation, sparse files, and concurrent access become testable for the first time. I intend to stay with those problems after GSoC ends.

8. Expectations from Mentors

There are a handful of architectural decision points in the btrfs write path - transaction ordering, CoW behaviour during node splits, superblock update timing - where getting it wrong costs weeks. Before committing to any of these, I will write up my reasoning and ask for a direct review. Thirty minutes of alignment early is worth two weeks of debugging later.

I will submit patches to Gerrit one subsystem at a time and expect substantive feedback on correctness, not just style. Every patch will be small and reviewable before I submit it.

When I hit a blocker I will raise it within two days, with a specific bounded question rather than a vague one.

At each phase boundary I will bring a ranked list of what to pursue next and ask for a direct call on what to defer. I trust mentor judgment on scope more than my own enthusiasm.

I will post a public forum update after each milestone and report progress honestly, including when something took longer than planned.

9. References

- Haiku btrfs source: src/add-ons/kernel/file_systems/btrfs/
- Hy Che, GSoC 2017 - btrfs write support (extent allocator, CoW primitives, transaction wrappers)
- Bharathi Ramana Joshi, GSoC 2019 - btrfs write support (inode CRUD, directory ops, tree traversal)
- Haiku Gerrit: review.haiku-os.org - changes #10446–#10452 by anujbillore-0-0
- btrfs on-disk format specification: btrfs.wiki.kernel.org
- Haiku filesystem API: src/system/kernel/fs/