ECE 5984 Virtualization Technologies

Project 1: File System Implementation for HermitCore

1. Introduction

The objective of this project is to implement a file system in HermitCore and provide support for file system calls from within the unikernel. Currently, file system calls issued by the HermitCore application are forwarded to the host operating system for processing. It is expected that supporting these in the unikernel would help eliminate switching costs and related overhead since unikernel system calls are simply common function calls. The Postmark filesystem benchmark is used to evaluate the cost of file system calls as supported currently and when implementing a file system in the unikernel.

2. File System Design

Storage

As suggested in the technical guide, a large buffer of memory (ramdisk) is used as the backing store for the file system. The size of the ramdisk is hardcoded in the TOTAL_RAMDISK_SPACE variable.

File Data Structures

<table>
<tr><th>Superblock</th><th>Inode Block</th><th rowspan="2">Data Blocks</th></tr>
<tr><td>Partition, Total size, Block size, Pwd, Pfd, Type, Free inode spaces map, Free address map</td><td>File/Directory name, Path, Type, Inode index, Address, Max size, Inode size, Parent, Children, Starting address, Current address, Ending address</td></tr>
</table>

The allocated memory is divided into three parts – superblock, inode block, and data blocks. The superblock and inode block contain file system metadata and file metadata. The actual directory/file data are stored in the data blocks.

The **superblock**, which is accessed first, comprises information about the file system as a whole (total size, block size, inode size). This enables computation of the inode block location where file metadata is located. Further, the free inode space map and free data space map provide insights into the occupancy status of the inode and data blocks.

The **inode block** is associated with files and each entry in it has metadata about a file. This includes data such as the file type (file or directory), file path, file size as well as information about where the corresponding data blocks reside (pointers to the beginning and end). Moreover, information about path and directory structure (parent, child) are maintained.

Other data structures required to implement the file system calls include structures that temporally store information about files and directories and store current status information. Additionally, to support large files that might span several data blocks, a file block structure is used. These are populated initially from fields in the super and inode blocks.

The **directory** structure simply comprises the directory name, location, path, previous level parent, and the names of files belonging to that directory. Similarly, the **file** structure includes fields like name, path, size, start and end buffer locations, current position in buffer, and so on. The **file block** structure has information about the previous and next blocks, current read and write pointers, and content buffer.

The complete definitions for these data structures are shown below as some of the fields are referred to in the file system call description. In the project code, these are located in *HermitCore_FS/include/hermit/fs.h*. An upper limit is defined in the same file for TOTAL_INODE_SPACE, TOTAL_BLOCK_SPACE, BLOCK_SIZE_MAX, NUMBER_INODES, NUMBER_DIRS, and NUMBER_FILES.

```
struct fs_File{
    char    file_name[20];
    char    file_path[100];
    size_t  location;
    size_t  size_max;
    size_t  file_size;
    size_t  parent;
    size_t  curr_read_blk;
    size_t  content_buf_head;
    size_t  content_buf_curr;
    size_t  content_buf_tail;
};

struct FileBlock_list_node{
    size_t  location;
    size_t  curr_read_ptr;
    size_t  curr_write_ptr;
    char    content_buf[BLOCK_SIZE_MAX];
    size_t  block_prev;
    size_t  block_next;
};

struct fs_Dentry {
    size_t  location;
    char    dir_name[20];
    char    dir_path[100];
    size_t  dir_parent;
    size_t  dir_child[NUMBER_DIRS];
};

struct fs_Inode {
    char    Inode_name[20];
    char    Inode_path[100];
    char    Inode_type[1];
    size_t  Inode_index;
```

```
    size_t  address;
    size_t  size_max;
    size_t  Inode_size;
    size_t  parent;
    size_t  child[NUMBER_INODES];
    size_t  fcontent_buf_head;
    size_t  fcontent_buf_curr;
    size_t  fcontent_buf_tail;
};

struct fs_Superblk {
    size_t  partition;
    size_t  total_size;
    size_t  block_size;
    size_t  inode_size;
    size_t  pwd;
    size_t  pfd;
    char    type[10];
    size_t free_inode_map[NUMBER_INODES];
    size_t free_fsblock_map[NUMBER_FILES];
};
```

Supported Directory System Calls

1. fs_ls
   This system call lists all the files contained in a directory. The pwd field of the superblock is accessed and information about the corresponding directory is looked up from the inode table. The children of the inode entry constitute the files within the directory.

2. fs_mkdir
   mkdir is used to create or make a directory. To implement this, first, the free inode map located in super is traversed to obtain an unoccupied entry. The corresponding inode address is computed and the various inode values are populated with information such as the directory's name, path, parent directory, child files, etc. The content buffer head and tail are initialized to NULL.

3. fs_cd
   The change directory system call is used to navigate to a different directory. The pwd value of the super is used to set the current directory value. If the value ".." is given, the parent of the pwd is set as the new cwd. Else, if there exists a child node whose name matches with that provided, that child is set as the new pwd.

4. fs_cwd
   This system call returns the path of the current working directory. The pwd field of the superblock is accessed and information about the corresponding directory is looked up from the inode table. The dir_path variable provides the required path.

5. fs_frm
   This system call is used to delete both files and directories. The pwd field is obtained from the superblock and its child nodes are checked to locate the one to be deleted. If the node to be deleted is a directory, each of its child nodes are deleted recursively. Once this is done, the inode entry corresponding to that directory is reset. The corresponding location in the free inode map is set to unoccupied.

Supported File System Calls

1. fs_fopen
   The file open system call opens a file by obtaining its inode entry details if it is already present as a child of the pwd (obtained from the superblock). Otherwise, a new file entry is created. First, the free inode map is traversed to check for an empty location. Once found, the actual location of the inode is computed and the various fields of the inode like name, type, size, etc. are populated according to the file details. The content buffer pointers are all initialized to NULL.

2. fs_fclose
   Closing a file involves clearing the current file and current directory fields of the super and setting the pfd to 0.

3. fs_fread
   The current file is obtained from super and the fields in the file structure are populated by accessing related information from super and the inode corresponding to the file. At the outset, the current block is set to the head of the content buffer and the block contents are read into buffer. Then, if block_next is not NULL, another block size worth of buffer is allocated and read. This is repeated till the entire file contents are read.

4. fs_fwrite
   As in the case of file read, the current file is obtained from super. The fields in the file structure are populated by accessing related information from super and the inode corresponding to the file. To account for the case in which a file only partially occupies a memory block, the current block field is used. If the current block is NULL (there is no remaining space), the free block space map in super is traversed to find an unoccupied block of memory. The actual location of this block is computed and buffer contents are written to the block. The current block is updated with the information of this block. Once the write is completed, the content buffer tail and current locations are updated accordingly. If the current block is not NULL, writing commences from the current block's curr_write_ptr till that block is complete. If the data to written is larger than the remaining space, an extra block is allocated and written to, using the procedure described earlier. Finally, the curr_write_ptr is updated.

5. fs_frm

   As mentioned earlier, this system call is used to delete both files and directories. The pwd field is obtained from the superblock and its child nodes are checked to locate the one to be deleted. If the node to be deleted is a file, the corresponding data blocks are traversed. Each of these is released by resetting memory and updating the flags of the free data block map in the superblock.

A comprehensive list of all the implemented file system calls is shown below.

```
ssize_t fs_ls(char* buf);
ssize_t fs_cwd(char* buf);
ssize_t fs_mkdir(char *name);
ssize_t fs_cd(char *name);
size_t fs_fopen(char *name, char *mode);
size_t fs_fclose(char *name);
size_t fs_fwrite(size_t index, char *buf, char *mode, size_t len);
size_t fs_fread(size_t index, char *buf, char *mode, size_t len);
size_t fs_frm(char *name);
```

3. File System Implementation

Setup

HermitCore and newlib were installed and configured according to the instructions in the technical guide. Code was added in *HermitCore_FS/kernel/main.c* to request ramdisk space and set up the virtual to physical mapping. The file systems data structures were defined in *HermitCore_FS\fs\fs.h*. C language support in the unikernel is provided by newlib, and so the system call function definitions were added as C files in the folder *newlib/newlib/libc/sys/hermit*. The newlib makefiles were updated to include these source files and the configuration script was regenerated. The added system calls simply call the HermitCore implementation, and the actual function implementation is in *HermitCore_FS/kernel/syscall.c*. The various helper functions called in the system calls are located in *HermitCore_FS/fs/fs.c*. A mapping between the system calls, HermitCore implementation, and called helper functions is shown below. The associated prototypes were added in *HermitCore_FS/include/hermit/syscall.h*. Then, the kernel was recompiled and reinstalled. The code for the HermitCore file system implementation can be got from https://github.com/anwaymukherjee/HermitCore_FS.

newlib/newlib/libc/sys/hermit	*HermitCore_FS/kernel/syscall.c*	*HermitCore_FS/fs/fs.c*
fs_frm	sys_fs_frm	fs_removeInode
fs_cd	sys_fs_cd	fs_change_dir
fs_fclose	sys_fs_fclose	fs_fileclose
fs_mkdir	sys_fs_mkdir	fs_makedir
fs_fopen	sys_fs_fopen	fs_fileopen
fs_ls	sys_fs_ls	fs_list
fs_cwd	sys_fs_cwd	fs_list

fs_fread	sys_fs_fread	fs_fileread
fs_fwrite	sys_fs_fwrite	fs_fileread, fs_filewrite

4. Performance Evaluation

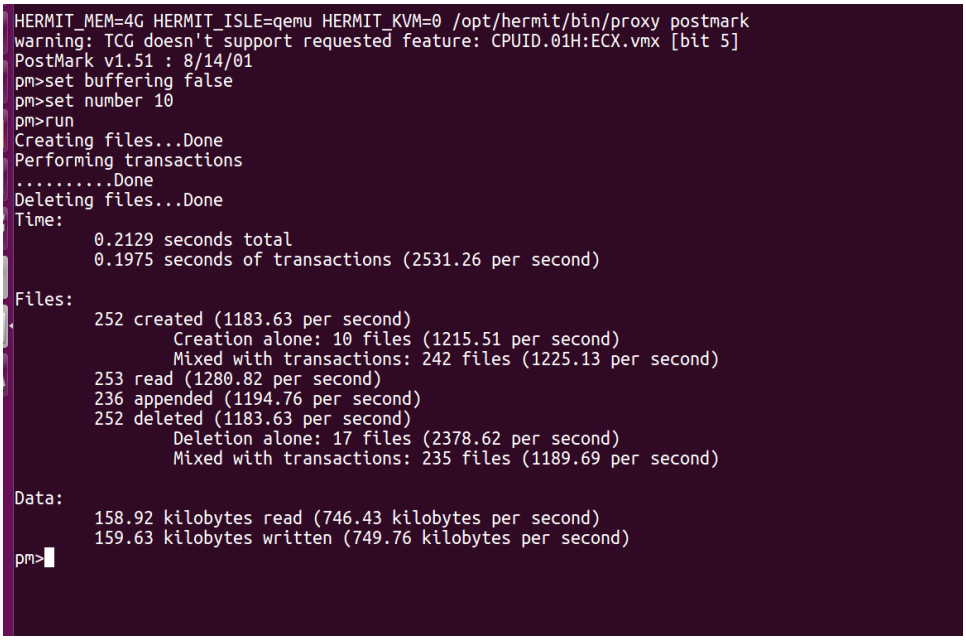
Postmark

Postmark is a benchmark commonly used to test file system implementations. This program creates a pool of files, and executes file creation, deletion, read, and append functions. Before termination, all the files are deleted. Several parameters such as the number of files, transactions, etc. can be specified. Statistics such as the data read and write times are displayed. We used postmark to evaluate the implemented file system and also for comparison with existing HermitCore. A few modifications were made to postmark.c file for evaluating the implemented file system by replacing c system calls with equivalent user-defined system calls. Another improvement was to improve the granularity of the reported time statistics by replacing time_t by the timeval struct. The latter was adapted from https://github.com/Andiry/postmark/blob/master/postmark-1.53.c.

File System Implementation Evaluation

The implemented file system was evaluated for different number of concurrent files. This was defined by using the “set number” command; furthermore, postmark buffering support was turned off. The screenshots for the various scenarios are shown below.

Set number 10



Set number 100

```
pm>set number 100
pm>run
Creating files...Done
Performing transactions
.........Done
Deleting files...Done
Time:
        0.2745 seconds total
        0.2153 seconds of transactions (2322.71 per second)

Files:
        347 created (1264.05 per second)
                Creation alone: 100 files (3520.63 per second)
                Mixed with transactions: 247 files (1147.42 per second)
        256 read (1189.23 per second)
        238 appended (1105.61 per second)
        347 deleted (1264.05 per second)
                Deletion alone: 94 files (3047.50 per second)
                Mixed with transactions: 253 files (1175.29 per second)

Data:
        156.96 kilobytes read (571.78 kilobytes per second)
        212.50 kilobytes written (774.11 kilobytes per second)
pm>
```

Set number 200

```
pm>set number 200
pm>run
Creating files...Done
Performing transactions
.........Done
Deleting files...Done
Time:
        0.3630 seconds total
        0.2329 seconds of transactions (2146.45 per second)

Files:
        465 created (1280.92 per second)
                Creation alone: 200 files (3572.96 per second)
                Mixed with transactions: 265 files (1137.62 per second)
        224 read (961.61 per second)
        274 appended (1176.25 per second)
        465 deleted (1280.92 per second)
                Deletion alone: 230 files (3103.91 per second)
                Mixed with transactions: 235 files (1008.83 per second)

Data:
        130.63 kilobytes read (359.85 kilobytes per second)
        268.30 kilobytes written (739.08 kilobytes per second)
pm>
```

Set number 400

```
pm>set number 400
pm>run
Creating files...Done
Performing transactions
..........Done
Deleting files...Done
Time:
        0.5053 seconds total
        0.2689 seconds of transactions (1859.63 per second)

Files:
        640 created (1266.68 per second)
                Creation alone: 400 files (3373.42 per second)
                Mixed with transactions: 240 files (892.62 per second)
        259 read (963.29 per second)
        241 appended (896.34 per second)
        640 deleted (1266.68 per second)
                Deletion alone: 380 files (3225.42 per second)
                Mixed with transactions: 260 files (967.01 per second)

Data:
        137.76 kilobytes read (272.66 kilobytes per second)
        349.77 kilobytes written (692.26 kilobytes per second)
pm>
```

Set number 600

```
pm>set number 600
pm>run
Creating files...Done
Performing transactions
..........Done
Deleting files...Done
Time:
        0.7218 seconds total
        0.3268 seconds of transactions (1530.22 per second)

Files:
        863 created (1195.63 per second)
                Creation alone: 600 files (2982.66 per second)
                Mixed with transactions: 263 files (804.89 per second)
        260 read (795.71 per second)
        239 appended (731.44 per second)
        863 deleted (1195.63 per second)
                Deletion alone: 626 files (3228.82 per second)
                Mixed with transactions: 237 files (725.32 per second)

Data:
        135.24 kilobytes read (187.36 kilobytes per second)
        456.67 kilobytes written (632.69 kilobytes per second)
pm>
```

Set number 800

```
pm>set number 800
pm>run
Creating files...Done
Performing transactions
.........Done
Deleting files...Done
Time:
        0.9171 seconds total
        0.3786 seconds of transactions (1320.52 per second)

Files:
        1035 created (1128.61 per second)
                Creation alone: 800 files (2672.31 per second)
                Mixed with transactions: 235 files (620.65 per second)
        262 read (691.95 per second)
        238 appended (628.57 per second)
        1035 deleted (1128.61 per second)
                Deletion alone: 770 files (3221.04 per second)
                Mixed with transactions: 265 files (699.88 per second)

Data:
        137.13 kilobytes read (149.54 kilobytes per second)
        549.22 kilobytes written (598.89 kilobytes per second)
pm>
```

Set number 900

```
pm>set number 900
pm>run
Creating files...Done
Performing transactions
.........Done
Deleting files...Done
Time:
        1.0489 seconds total
        0.4022 seconds of transactions (1243.30 per second)

Files:
        1161 created (1106.89 per second)
                Creation alone: 900 files (2478.53 per second)
                Mixed with transactions: 261 files (649.00 per second)
        247 read (614.19 per second)
        250 appended (621.65 per second)
        1161 deleted (1106.89 per second)
                Deletion alone: 922 files (3250.93 per second)
                Mixed with transactions: 239 files (594.30 per second)

Data:
        127.05 kilobytes read (121.13 kilobytes per second)
        611.15 kilobytes written (582.66 kilobytes per second)
pm>
```

Comparison with HermitCore

The same procedure was repeated in the unmodified HermitCore unikernel. Please note that since the rmdir call is not supported, the remove -> unlink -> rmdir failed, and deletion was not successful.

Set number 10

```
Done
Time:
        252 seconds total
        185 seconds of transactions (2 per second)

Files:
        20 created (0 per second)
                Creation alone: 10 files (0 per second)
                Mixed with transactions: 10 files (0 per second)
        266 read (1 per second)
        156 appended (0 per second)
        20 deleted (0 per second)
                Deletion alone: 0 files (0 per second)
                Mixed with transactions: 0 files (0 per second)

Data:
        2.41 megabytes read (9.80 kilobytes per second)
        195.30 kilobytes written (0.78 kilobytes per second)
pm>
```

Set number 100

```
Done
Time:
        1873 seconds total
        1183 seconds of transactions (0 per second)

Files:
        200 created (0 per second)
                Creation alone: 100 files (0 per second)
                Mixed with transactions: 100 files (0 per second)
        247 read (0 per second)
        253 appended (0 per second)
        200 deleted (0 per second)
                Deletion alone: 0 files (0 per second)
                Mixed with transactions: 0 files (0 per second)

Data:
        1.60 megabytes read (0.88 kilobytes per second)
        1.45 megabytes written (0.79 kilobytes per second)
pm>
```

Set number 400

```
Done
Time:
        4952 seconds total
        2341 seconds of transactions (0 per second)

Files:
        672 created (0 per second)
                Creation alone: 400 files (0 per second)
                Mixed with transactions: 272 files (0 per second)
        257 read (0 per second)
        243 appended (0 per second)
        672 deleted (0 per second)
                Deletion alone: 0 files (0 per second)
                Mixed with transactions: 0 files (0 per second)

Data:
        1.42 megabytes read (0.29 kilobytes per second)
        3.86 megabytes written (0.80 kilobytes per second)
pm>
```

Comparison with Linux

Set number 10

```
divya@divya-HP-Spectre-x360-Convertible-13:~/Downloads$ ./postmark
PostMark v1.51 : 8/14/01
pm>set number 10
pm>set transactions 10
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0027 seconds total
        0.0012 seconds of transactions (8271.30 per second)

Files:
        16 created (5878.03 per second)
                Creation alone: 10 files (9199.63 per second)
                Mixed with transactions: 6 files (4962.78 per second)
        5 read (4135.65 per second)
        5 appended (4135.65 per second)
        16 deleted (5878.03 per second)
                Deletion alone: 12 files (28169.01 per second)
                Mixed with transactions: 4 files (3308.52 per second)

Data:
        35.59 kilobytes read (12.77 megabytes per second)
        87.58 kilobytes written (31.42 megabytes per second)
pm>
```

Set number 100

```
PostMark v1.51 : 8/14/01
pm>set number 100
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0249 seconds total
        0.0174 seconds of transactions (28697.70 per second)

Files:
        349 created (14020.00 per second)
                Creation alone: 100 files (15463.12 per second)
                Mixed with transactions: 249 files (14291.45 per second)
        252 read (14463.64 per second)
        248 appended (14234.06 per second)
        349 deleted (14020.00 per second)
                Deletion alone: 98 files (97706.88 per second)
                Mixed with transactions: 251 files (14406.24 per second)

Data:
        1.64 megabytes read (65.87 megabytes per second)
        2.19 megabytes written (87.83 megabytes per second)
pm>
```

Set number 200

```
pm>set number 200
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0301 seconds total
        0.0172 seconds of transactions (29017.47 per second)

Files:
        462 created (15334.06 per second)
                Creation alone: 200 files (20130.85 per second)
                Mixed with transactions: 262 files (15205.15 per second)
        257 read (14914.98 per second)
        243 appended (14102.49 per second)
        462 deleted (15334.06 per second)
                Deletion alone: 224 files (75599.05 per second)
                Mixed with transactions: 238 files (13812.31 per second)

Data:
        1.49 megabytes read (49.59 megabytes per second)
        2.85 megabytes written (94.65 megabytes per second)
pm>
```

Set number 400

```
pm>set number 400
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0488 seconds total
        0.0238 seconds of transactions (20981.07 per second)

Files:
        656 created (13447.58 per second)
                Creation alone: 400 files (20942.41 per second)
                Mixed with transactions: 256 files (10742.31 per second)
        259 read (10868.20 per second)
        241 appended (10112.88 per second)
        656 deleted (13447.58 per second)
                Deletion alone: 412 files (70415.31 per second)
                Mixed with transactions: 244 files (10238.76 per second)

Data:
        1.43 megabytes read (29.33 megabytes per second)
        3.77 megabytes written (77.35 megabytes per second)
pm>
```

Set number 600

```
pm>set number 600
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0479 seconds total
        0.0181 seconds of transactions (27680.89 per second)

Files:
        864 created (18030.43 per second)
                Creation alone: 600 files (26417.75 per second)
                Mixed with transactions: 264 files (14615.51 per second)
        254 read (14061.89 per second)
        246 appended (13619.00 per second)
        864 deleted (18030.43 per second)
                Deletion alone: 628 files (87905.94 per second)
                Mixed with transactions: 236 files (13065.38 per second)

Data:
        1.41 megabytes read (29.34 megabytes per second)
        4.91 megabytes written (102.47 megabytes per second)
pm>
```

Set number 800

```
pm>set number 800
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0518 seconds total
        0.0185 seconds of transactions (27008.05 per second)

Files:
        1055 created (20367.97 per second)
                Creation alone: 800 files (33516.28 per second)
                Mixed with transactions: 255 files (13774.10 per second)
        246 read (13287.96 per second)
        253 appended (13666.07 per second)
        1055 deleted (20367.97 per second)
                Deletion alone: 810 files (86032.93 per second)
                Mixed with transactions: 245 files (13233.94 per second)

Data:
        1.25 megabytes read (24.11 megabytes per second)
        5.80 megabytes written (111.97 megabytes per second)
pm>
```

Set number 900

```
pm>set number 900
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        0.0626 seconds total
        0.0173 seconds of transactions (28896.72 per second)

Files:
        1166 created (18614.60 per second)
                Creation alone: 900 files (26925.95 per second)
                Mixed with transactions: 266 files (15373.06 per second)
        257 read (14852.92 per second)
        243 appended (14043.81 per second)
        1166 deleted (18614.60 per second)
                Deletion alone: 932 files (78247.00 per second)
                Mixed with transactions: 234 files (13523.67 per second)

Data:
        1.26 megabytes read (20.16 megabytes per second)
        6.32 megabytes written (100.89 megabytes per second)
pm>
```

Discussion

The results in terms of the total execution time, transaction execution time, and read and write data rates are summarized in the table shown below.

On Implemented File System							
Set Number	10	100	200	400	600	800	900
Total time taken (s)	0.2129	0.2745	0.363	0.5053	0.7218	0.9171	1.0489
Transaction time (s)	0.1975	0.2153	0.2329	0.2689	0.3268	0.3786	0.4022
Read data rate (kbps)	746.43	571.78	359.85	272.66	187.36	149.54	121.13

Write data rate (kbps)	749.76	774.11	739.08	692.26	632.69	598.89	582.66

On HermitCore			
Set Number	10	100	400
Total time taken (s)	252	1873	4952
Transaction time (s)	185	1183	2341
Read data rate (kbps)	9.8	0.88	0.29
Write data rate (kbps)	0.78	0.79	0.8

On Linux							
Set Number	10	100	200	400	600	800	900
Total time taken (s)	0.0027	0.0249	0.0301	0.0488	0.0479	0.0518	0.0626
Transaction time (s)	0.0012	0.0174	0.0172	0.0238	0.0181	0.0185	0.0173
Read data rate (Mbps)	12.77	65.87	49.59	29.33	29.34	24.11	20.16
Write data rate (Mbps)	31.42	87.83	94.65	77.35	102.47	111.97	100.89

As is observed from the table, the implemented file system is faster than the HermitCore file support by approximately 1000x times. On the other hand, it is approximately 100x slower than the file system support provided natively in Linux.

4. Conclusion and Future Work

Based on the postmark evaluation results, we can conclude that providing support for file system calls from within the unikernel shows an improved performance over the current setup in which file system calls issued by the HermitCore application are forwarded to the host operating system for processing.

Our current file system implementation has a few limitations. The entire file stack is a flat memory allotment. The file structures include arrays and linked list which do offers a significant traversal overhead. Moreover, since we are dealing with a single threaded Unikernel implementation, we neither realized data synchronization policy nor laid down resource access permissions. A stripped down user API (newlib) meant fewer string manipulation functions, hence our file system implementation heavily relies on address pointer manipulation and a few system level restrictions, namely, no absolute path traversal. Thus, for a file/directory deletion, one has to be at the parent directory to execute the functionality; same with other operations. The segments (superblock, inode, data blocks) of our file system are hard coded and inflexible. We have a hardware specific limit on the Ram Disk size beyond which the virtual machine fails to fire up.