Virginia Tech ❖ Bradley Department of Electrical and Computer Engineering

ECE 5984 Virtualization Technologies

Project 1: Unikernel Filesystem

1 Introduction

In a unikernel, a single application is statically compiled with the necessary libraries and a thin operating system layer into a single binary that is run as a virtualized guest on top of a hypervisor. Due to the fact that (1) one unikernel runs only a single application and (2) the hypervisor takes care of isolating the unikernel from the rest of the system, the unikernel application and kernel run in a single address space without any memory protection mechanism. This has the interesting result that **in unikernels, system calls are common function calls**. It makes that they have the potential to be faster than in a regular kernel because there is no world switch between user and kernel space.

In order to measure if there is any benefit from such fast system calls, one would need to run a system-intensive benchmark and compare results in a unikernel versus native execution in Linux. System-intensive macro-benchmarks generally stress network of filesystem services.

In this project we will consider the HermitCore[1] unikernel, due to its simplicity (around 10 000 lines of code). In this kernel the network stack is relatively slow so we will disregard networking to focus on filesystem operations.

The filesystem operations in HermitCore are forwarded to the host operating system (see the left picture on Figure 1. I.e., any filesystem operation (`open`, `read`, `write`, etc.) will access the host filesystem. This is practical as HermitCore designers can avoid implementing a filesystem. However, it makes that filesystem operations cannot benefit from the 'fast system calls' features of unikernels as:

- The forwarding process introduces some overhead and potentially some data copy;
- Once forwarded to the host, host filesystem operations are performed, i.e. Linux filesystem related system calls are made along with the costly world switch unikernels want to avoid in the first place.

2 Project description

The goal of the project is to implement a simple filesystem within the HermitCore kernel, thus avoiding the need to redirect the filesystem operations to the host and the corresponding overheads. This task involves:

- Designing a simple filesystem: setting up the rules that define how file data and file/directories metadata (names, directory hierarchy) is stored and retrieved from the backing store;
- Implementing in the HermitCore kernel the semantics of filesystem related system calls according to the chosen filesystem design so that these system calls can be served from the kernel without any host involvement. Currently HermitCore supports the following system calls (forwarded to the host): `read`, `write`, `open`, `close`, `lseek`. Additional system calls will need to be implemented to support a wider range of applications (for example `creat`, `rename`, `mkdir`, etc.). However, a full implementation of the Linux filesystem API is not asked, only the necessary system calls needed to run standard filesystem benchmarks, for example *Postmark*;
- Testing and evaluating the performance of your solution.

[1] `https://github.com/RWTH-OS/HermitCore`

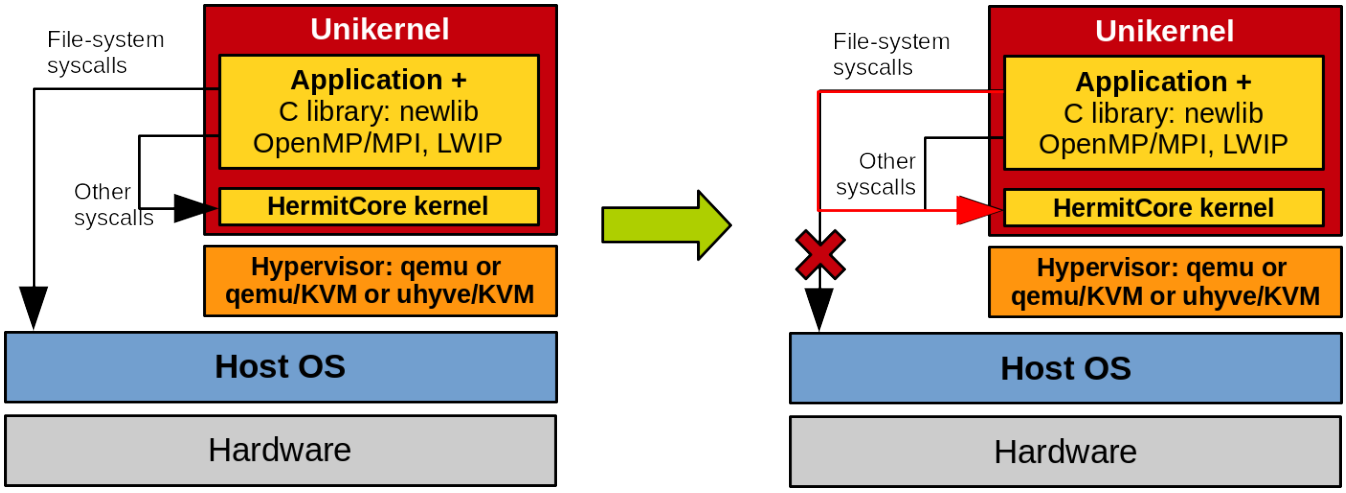

Figure 1: The project goal is to transform the regular HermitCore filesystem operations, that are forwarded to the host (left), into filesystem operations that are internally processed by the HermitCore kernel (right).

For the sake of simplicity, it is strongly advised to use the guest RAM as the backing store (where filesystem data is stored). Thus, you can avoid having to use a real disk, which would mean writing drivers or some sort of communication channel between the guest and the host.

3 Project steps

It is advised to work on the project on subsequent steps of increasing difficulty:

1. **Simple programs**:
   - That step involves writing the core part of the filesystem and a few system calls in order to run simple programs: for example, a unikernel that opens a file with the `O_CREAT` flag, writes a bunch of text in that file, reads that text back in a buffer, then closes the file.
2. **Directory support**:
   - Enhance the filesystem with folders support, and be able to run a test unikernel manipulating directories: for example, creating a directory and multiple files inside, and being able to list the files present in that folder.
3. **Scale-up**:
   - Make sure that the filesystem supports unikernel creating and accessing thousands of files and folders
4. **Performance evaluation**:
   - Compare your filesystem performance to regular HermitCore as well as native Linux
   - Some ad-hoc micro-benchmarks can be written, for example to compare the system call latency between your solution and native Linux/regular HermitCore.
   - A macro-benchmark should also be used. It is asked to use the Postmark[2] macro-benchmark, a relatively simple program written in C exercising the filesystem. Note that a few additional system calls will need to be added to HermitCore to support running Postmark.

[2] `https://raw.githubusercontent.com/mayurva/Distributed-FS/master/postmark.c`

4 Project report

The report should be at least 7 pages and should contain:

- A small introductory part describing the context of the project;
- A detailed description of the filesystem design, including a list of the system calls supported, and what happens under the hood when each one is executed;
- A description of the filesystem implementation;
- A description of the set of experiments run for the performance evaluation, and a justification why each experiment was chosen;
- The performance evaluation results, along with result analysis and interpretation (i.e. how the number look, and why they look like that);
- Any other points that you feel interesting to include in the report.

5 Group work

This project can be accomplished in groups of maximum 2 students. Once your group is constituted, please register it on canvas:

`https://canvas.vt.edu/courses/63785/groups#tab-8760`.

6 Design discussion session: 01/29/2017

We will discuss the filesystem design ideas during the lecture session on 01/29/2017. For that session, each group should come prepared with a design draft.

For that session, it is also asked that each group reproduces all items present in the technical guide (see Section 8).

7 Results to be handed - Deadline: 2018-02-20 11:59PM

The following is expected to be handed by 2018-02-20, 11:59PM:

1. Sources of the project code & README: include everything needed such as unikernel, C library, and test programs sources, etc.. Add a README file describing which specific files have been added/modified, and how to compile and execute tests programs. More generally, add in this README anything that you think will help in evaluating your work;
2. The project report in PDF format.

All of this should be contained in a compressed archive. One archive per group should be submitted.

8 Technical guide

A technical guide is made available to you as a way to get started with HermitCore development, and to get some tips related to the project. It is accessible here:

`https://canvas.vt.edu/files/6115850/download?download_frd=1`