Chapter 1

Literature Review

This chapter will begin with an overview of concept of persistence in general, and in particular, orthogonal persistence in section 1. Then, in section 2, it will describe how networking in persistent systems is done currently, and what are the requirements for a networking stack in such a system. The last section is dedicated to give a description of Phantom OS, Genode OS framework and current state of porting former to latter.

1.1 Persistence

1.1.1 Persistence definitions

Every program operates some kind of data. This data can be different in nature, and thus has a different lifetimes. This lifetime is a time extent over which data can be used and is commonly called *persistence* [1].

Atkinson et al. [1], [2] give a classification of application data objects by their persistence, which is presented in table I. Usually, levels 1-4 are supported by programming languages themselves, and support of levels 5-8 relies on some external component, such as database or file system.

Table I: Classification of data objects based on their lifetime

1	Transient results in expression evaluation
2	Local variables
3	Global variables and heap items
4	Data that lasts a whole execution of a program
5	Data that lasts for several executions of several programs
6	Data that lasts for as long as a program is being used
7	Data that outlives a succession of versions of such a program
8	Data that outlives versions of the persistent support system

1.1.2 Orthogonal persistence

Different ways of data interactions, namely using builtin programming language constructs and external tools available via some interfaces, bring an extra layer of complexity to programs. To answer this problem, [2] proposes to use persistent support systems, which act as a mediator between a persistent application and transient environment. The authors also summarize requirements for such a system, saying that such a systems should have use of data not dependent of its persistence. They define following principles for such a system:

1. The Principle of Persistence Independence. The form of a program is independent of the longevity of the data which it manipulates. Programs look the same whether they manipulate short-term or long-term data.

2. The Principle of Data Type Orthogonality. All data objects should be

allowed the full range of persistence, irrespective of their type. There are no special cases where objects are not allowed to be long-lived or are not allowed to be transient.

3. The Principle of Persistence Identification. The choice of how to identify and provide persistent objects is orthogonal to the universe of discourse of the system. The mechanism for identifying persistent objects is not related to the type system.

Two popular approaches exist to build such a system.

One is to integrate interactions with databases or file systems into an existing programming languages, which is what most ORM libraries do [3], [4]. However, this approach is not really transparent for a programmer, and may cause errors.

Another approach is to build so-called *persistent worlds* [2]. These worlds are usually done in form of operating systems, which provide necessary mechanisms to make state of userspace processes managed by it persistent across systems restarts. Examples of such systems are KeyKOS [5], Grasshopper OS [6], and Phantom OS.

A question may arise at this point, isn't it possible to take an existing popular operating system, and run it with nonvolatile memory as a main memory? The answer is no, because this raises two problems: problem of dealing with changed peripheral device states during restarts [7], and logical problems with code execution [8]. That is why the question of building a persistent support system that will satisfy orthogonal properties is not trivial.

1.2 Networking in persistent systems

. . . This part is not done yet, however an outline was done for its contents

1. Definition of problems with currently supported network stack.

   Example of problems:

   - Lost connections and strategies to restore.
   - How stateful protocols can be changed?
   - Can we use some intermediary message broker to manage communication?

2. Define requirements for protocol that are suitable for PSs

3. Say what will be focus of work

1.3 Phantom OS and Genode OS framework

1.3.1 Phantom OS

Phantom OS is a general-purpose operating system, developed in 2009-2011. This is an orthogonally persistent operating system, in which persistence is brought using snapshotting of virtual memory. The operating system consists of kernel and virtual machine, which executes all userspace programs written in Phantom bytecode. Managed code execution is persistent, whereas kernel state is transient, which means that it is not kept between reboots. It is assumed that it can be restored. Phantom is an experimental system, in a sense that it is developed as a proof-of-concept and is not ready for production use.

Phantom OS currently supports only i386 ISA, and it has all drivers builtin inside a kernel. It is problematic, because to support wide range of hardware OS needs drivers, which is often delivered by third-party developers. But in case when drivers are embedded in kernel error in driver can crash whole system.

These are two reasons why it is desirable to replace Phantom kernel with a microkernel, which is now in progress by my colleagues. My colleagues from Innopolis University are now working on porting Phantom OS to Genode OS framework. I am going to use this port as soon as it will be ready.

1.3.2 Genode OS framework

. . . To be done

Bibliography cited

[1] M. P. Atkinson, P. J. Bailey, K. J. Chisholm, W. P. Cockshott, and R. Morrison, “Ps-algol: A language for persistent programming,” in *Proc. 10th Australian National Computer Conference, Melbourne, Australia*, 1983, pp. 70–79.

[2] M. Atkinson and R. Morrison, “Orthogonally persistent object systems,” *The VLDB Journal*, vol. 4, no. 3, pp. 319–401, 1995.

[3] П. А. Аннин, “Краткий обзор room persistence library,” in *Актуальные направления научных исследований: перспективы развития*, 2018, pp. 146–147.

[4] M. Bayer, “Sqlalchemy documentation,” *URL: https://www. sql. alchemy. org.(last*, 2010.

[5] A. C. Bomberger, N. Hardy, A. Peri, F. Charles, R. Landau, W. S. Frantz, J. S. Shapiro, and A. C. Hardy, “The keykos nanokernel architecture,” in *In Proc. of the USENIX Workshop on Micro-kernels and Other Kernel Architectures*, 1992.

[6] A. Dearle, R. Di Bona, J. Farrow, F. Henskens, A. Lindström, J. Rosenberg, F. Vaughan, *et al.*, “Grasshopper: An orthogonally persistent operating system,” *Computing Systems*, vol. 7, no. 3, pp. 289–312, 1994.

[7] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Peripheral state persistence and interrupt management for transiently powered systems," in *NVMW 2018-9th Annual Non-Volatile Memories Workshop*, 2018, pp. 1–2.

[8] B. Ransford and B. Lucia, "Nonvolatile memory is a broken time machine," in *Proceedings of the workshop on Memory Systems Performance and Correctness*, 2014, pp. 1–3.