This is the Project Title

Andreas Ramsøy

**MInf Project (Part 2) Report**

Master of Informatics

School of Informatics

University of Edinburgh

2022

Abstract

This skeleton demonstrates how to use the infthesis style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file skeleton.tex generates this document and can be used as a starting point for your thesis. The abstract should summarise your report and fit in the space on the first page

Acknowledgements

Acknowledgements go here.

Table of Contents

[Introduction 1](#_Toc85538296)

[Background 2](#_Toc85538297)

[2.1 Kernels 2](#_Toc85538298)

[2.1.1 Monolithic Operating Systems 2](#_Toc85538299)

[2.1.2 Micro-Kernel Operating Systems 3](#_Toc85538300)

[2.1.3 Multi-Kernel Operating Systems 3](#_Toc85538301)

[2.2 Messaging Layers 4](#_Toc85538302)

[2.2.1 Mach 4](#_Toc85538303)

[2.2.2 Neutrino QNX 4](#_Toc85538304)

[2.2.3 Barrelfish 5](#_Toc85538305)

[2.3 The Popcorn Operating System 5](#_Toc85538306)

[2.3.1 Migration of Threads Between Processors 5](#_Toc85538307)

[2.3.2 Popcorn Messaging Layer 6](#_Toc85538308)

[2.3.3 Problems with the Existing Popcorn Messaging Layer 7](#_Toc85538309)

[Conclusion 10](#_Toc85538310)

[Bibliography 11](#_Toc85538311)

Chapter 1

# Introduction

This is the introduction

The aims of the project

Subsection of previous project summary, better in background so can explain concepts?

Short summary of last year’s work

The motivation of the research/why it’s important

Outline of how the paper is structured

Chapter 2

# Background

## 2.1 Operating Systems

An operating system kernel implements the essential functions of an operating system this can include scheduling, memory management and inter-process communication [1]. There are several different types of implementation of kernel including: monolithic, micro-kernel and multi-kernel. These different types include different services within the kernel. Operating systems (OSes) must be fast as to not introduce overheads to the programs that they run, free from errors (ideally verified to prove that errors should not occur) and be simple for programmers of applications to interface with. Current general-purpose operating systems are built on top of monolithic kernels in order to maximise performance [1].

The kernel is split into two parts or modes of operation: user-space (or user mode) and kernel space (kernel/supervisor mode) [2]. A context-switch is required in order to change from user-mode to kernel-mode this can be triggered through an interrupt or exception.

### 2.X.X Distributed Operating Systems

“The users of a true distributed system should not know (or care) on which machine (or machines) their programs are running, where their files are stored, and so on. ” [3]

There are three broad categories of distributed operating systems referred to as the “minicomputer”, “workstation”, and “processor pool” models [3]. The minicomputer model is where each device on the network has multiple users a user logs into a machine. The logged in user uses that machine but also has remote access to the others on the network [3]. The workstation model is where the user has a powerful machine that does the bulk of the work but with some distributed services like a shared file system. Finally, the processor pool model is where a group of processors are available to each user, where users are able to a flexible number of processors (between them and the other users on the system [3].

A problem with distributed systems is maintaining naming consistency across different machines. E.g., two user may have the same user ID (UID) on different devices, they should not however be considered the same user, or have the same privileges when accessing resources on the other machine. As a result distributed systems either login through a centralised system or use the name of the computer that they are logged into as part of their UID (it can be thought of as user@device\_name). Another solutions are to create a user of minimum privileges (e.g., demo user), however this means users not logged on to that machine have fewer privileges than ideal. It is considered that a true distributed system should have a unique UID for each user that is used across the system [4].

The OSI has significant overheads so message passing is generally favoured for distributed operating systems [4]. Using message passing it must be decided whether to use blocking vs non-blocking, and reliable vs unreliable messages. This refers to code that is required to wait until a message is received vs being allowed to other things in the meantime, and confirming if the message was delivered vs not confirming, respectively [4].

“n the rush to personal workstations, though, some of their weaknesses were overlooked. First, the operating system they run, UNIX, is itself an old timesharing system and has had trouble adapting to ideas born after it. Graphics and networking were added to UNIX well into its lifetime and remain poorly integrated and difficult to administer. More important, the early focus on having private machines made it difficult for networks of machines to serve as seamlessly as the old monolithic timesharing systems. Timesharing centralized the management and amortization of costs and resources; personal computing fractured, democratized, and ultimately amplified administrative problems. The choice of an old timesharing operating system to run those personal machines made it difficult to bind things together smoothly.”[7]

### 2.X.X Capabilities

A capability is a descriptor that contains a unique ID that represents a server process. This ID is cryptographically linked to the process it describes in such a way that only the server can understand which process it is for [4]. This provides protection to the process as only a user or process with the correct capability can access the protected resource (another process, memory, etc.). The rights that are granted can be specific privileges to that resource (e.g. write, read-only, execute) [5].

For a system using capabilities to remain secure it must be ensured that capabilities cannot be modified, only authorised interfaces can create them, and they are only given to processes that are authorised [6].

Capabilities can provide a secure method of granting privileges. however, they are difficult to revoke after creation. For this reason, many capability-based systems either use an access control or an intermediate system (referred to as a reference monitor in EROS) to stop revoked capabilities being used [6].

### 2.X.X Amoeba

The motivation behind Amoeba was to build a system where all resources are automatically managed by a distributed operating system [7]. As a consequence, users do not know which processor their programs run on, or how and where their files are stored in the system [7].

Amoeba provides a combination of the processor pool and workstation model where users are able to login to a particular machine but also run large jobs on a pool of processors [3].

In Amoeba users log into a terminal computer. This device does all the low latency computation whereas the group of computers known as the “processor pool” does all the larger computations [8].

Amoeba has had several iterations of file systems however, the simpler “bullet service” that stores immutable files as contiguous bytes [7].

Amoeba makes use of heterogeneity by using different machines for specialised purposes e.g., devices with large storage disks are used for file storage [7].

Amoeba’s design allows for great scalability, fault tolerance, and for processes to temporarily acquire large processing power [7].

To increase reliability new objects in a directory are replicated a set number of times and distributed across different nodes [7]. Capabilities are then stored in the directory along with the files [7].

To maintain security capabilities are used. By using a sufficiently large address space and having capabilities cryptographically linked to the resource that they are protecting, this protects the capability and therefore the resource it is protecting [7]. The use of cryptography allows capabilities to be safely used within user-space, thus following the principle of minimality and simplifying the kernel [7].

A hash of the port number that the message was sent on along with a shared secret means that an adversary cannot gain access to this [7]. This can be implemented in software or hardware but there should be no way of bypassing this function [7].

F-Box

Capabilities are distributed meaning that transfers cannot be detected, this means that to allow mandatory access control a system within the kernel needs to be implemented [6].

Capabilities are not used for individual pages or memory mapped structures but instead larger structures [6].

For directories, encryption key and a random value are XORed together. The result is stored in the directory itself and the capability given to the user that owns the file. When the file is requested by the user they provide their capability along with the random value. This means only the owner is able to decrypt the directory [7].

New capabilities that have a subset of rights of the other can be created by the owner [8].

Amoeba relies on the security of the capabilities and there any messages carrying them need to be encrypted to maintain the security of the system [8]. Two different systems can provide security to the network: one is a Kerberos like authentication server, and the other uses hashes. The authentication server results in a slower system however, the second system assumes a secure network and kernel [8].

### 2.X.X CAP

The virtual address space is mapped to a physical address space used across all devices [5].

CAP can have a hierarchical structure to represent processes. The position in the hierarchy is used to regulate the resources that the process has access to [6].

Capabilities can be used to restrict memory for example, each capability lists a maximum and minimum contiguous memory range that the permissions are for. A child capability can be generated from this that gives a maximum and minimum memory value relative to its parent. This way only memory in range of the parent can be used by the child. Any number of levels parent-child capabilities can be made [6].

CAP restricts users from procedures using a capability. to restrict who can run it, this is opposed to the conventional way operating systems determine each of the things done are allowed (and throws error when this happens) [6].

### 2.X.X Plan 9

Plan 9 allows for heterogeneity; different processor architectures are able to join the network running Plan 9 [7]. Messages are transferred between nodes in a high-level way, e.g. text, when possible as this simplifies the kernel when dealing with different processor architectures (however, binary can still be used for large transfers of data) [7].

Plan 9 interacts with services as if they are files and uses file operations as such. This means one simple, well understood protocol can be used to access almost all services [7].

Authentication between nodes is conducted by solving encrypting/decrypting challenge messages with some key [7]. Following both of the nodes success on the challenge messages one of the nodes sends a message to the authentication server. The authentication server then sends a message to both nodes containing a conversation key. The message that the authentication server is encrypted such that only that node can get the conversation key. The conversation key is then used to encrypt the conversation from then on [7].

This method does not rely on a synchronized clock, unlike Kerberos [7].

In Plan 9 the use of the same secured protocol to represent services rather than the use of firewalls means security is implemented in any Plan 9 service from the start [7].

There is no superuser, each individual server must ensure security (physical access to the server does give special permissions) [7].

### 2.1.1 Monolithic Operating Systems

MS-DOS, UNIX and the early Mac OS are monolithic kernels [2]. Monolithic kernels implement all of the basic operating system services within kernel space. These include memory and process management, input/output (I/O) communication and the file system [2]. The benefits of a monolithic kernel are that since most of the core functionality is implemented within the kernel context-switches are minimised. This generally means that it is a faster operating system. Monolithic kernels also mean that the operating system becomes large, difficult to maintain and add new features to [1] [2]. The kernel will require recompilation after every change, however minor, which may take several hours to complete and a considerable amount of processing power [2]. Debugging is difficult in a monolithic kernel due to its size and interdependency of modules [1]. It has been proposed to use specialist programming languages or hardware in order to provide safer and more secure monolithic kernels [1]. This is due to the fact that the large size of monolithic kernels makes them hard to debug and formally verify [1].

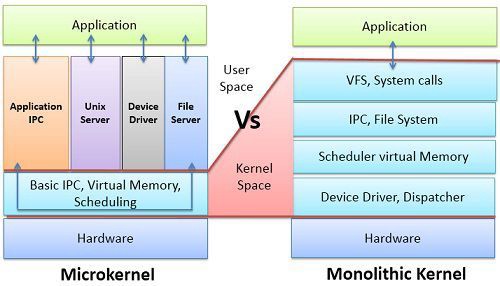


Figure Services included within the kernel for micro and monolithic kernels [9]

### 2.1.2 Micro-Kernel Operating Systems

The micro-kernel was designed to overcome the problems of the monolithic kernel. It runs the minimum required services within kernel mode and all other services run within user-mode [2]. This means that micro-kernels are easier to maintain and debug, faster to compile and easier to add new features [2]. The drawback with the micro-kernel is more context-switches are required which can slow down the system [2].

Hybrid kernels have been implemented in order to gain the benefits of minimality from micro-kernels with the fewer context-switches from monolithic kernels. However, many of these implementations yield difficult to maintain code with more context switches so pure micro or monolithic kernels give better results [2].

It is easier to prove correctness (free from bugs) the smaller the kernel is which is a motivation for micro-kernels [2]. Maintaining as small a kernel as reasonably possible is called the minimality principle [2]. Due to the size and interdependency of the monolithic kernel an error in one process within the kernel can cause the entire system to fail. Because micro-kernels minimise the number of services that are in the kernel this risk is reduced [2].

### 2.1.3 Multi-Kernel Operating Systems

A multi-kernel acts as a network of independent cores where no sharing takes place at the lowest level [10]. A distributed network of system services that communicate using message passing provide the OS functionality [10]. Multi-kernels provide more scalability to support future hardware than current operating system methods and currently yield usable performance on current hardware [10]. Multi-kernels allow for diverse hardware and better optimisation to this hardware [10]. General purpose OSes must provide greater performance on a more diverse range of hardware, multi-kernels provide this over the current operating system implementations [10].

## 2.2 Messaging Layers

Different kernels implement inter-process communication (IPC) in different ways. IPC can be implemented using signals (widely used in UNIX) [2] which send a numerical value to convey a command. These numerical values are fixed as each process has its own signal handler and so all of these signal handlers would have to be updated if the numerical values are changed [2]. These signals are very fast [2].

Sockets can also be used to communicate. A process (client) is able to “bind” itself to a socket (server), this allows it to listen to the any incoming messages from other processes that are sent to this socket [2]. This allows for arbitrary messages to be passed, making it easier to change in future and more extensible [2]. Message queues are a way of IPC that uses a first-in-first-out (FIFO) queue to store the incoming messages and is able to sort them based on their priority [2]. There can be multiple queues per process [2].

Monolithic kernels use signals and sockets whereas micro-kernels tend to use message queues [2].

### 2.2.1 Mach

Mach uses ports for IPC, these ports are protected so that only the tasks that are allowed to access a port can [11] [12]. The resources, facilities and services of the operating system are represented with ports and use the message passing system to send information to them [11] [12]. Ports are protected kernel objects; they act as queues of finite length for messages [13]. There can be multiple senders but only one receiver [13]. All messages are passed using the IPC facility, this has an authentication mechanism to protect this information [11]. The Mach message passing can be transparently extended over a network [12].

A message consists of a fixed length header and a variable length collection of data object [13] [14]. Messages can be synchronous or asynchronous where interrupts are used for asynchronous messages outside the normal flow of execution [13].

### 2.2.2 Neutrino QNX

QNX (Quick-UNIX) is an example of a micro-kernel and uses a messaging bus for IPC. Message passing is used in the Neutrino micro-kernel [15]. Neutrino makes each service modular to promote scalability [15]. Modules communicate through messages so that each are independent of each other [15].

### 2.2.3 Barrelfish

The Barrelfish operating system is a multi-kernel OS that runs a kernel instance on each core. Barrelfish uses message passing to maintain coherency. Messages are implemented with notification drivers [16]. Barrelfish is not heterogeneous although versions have been proposed [17]. Messages can be batched in order to reduce the number of notifications required [16]. Messages are closely tied to the hardware architecture in order to make them as fast as possible [16].

## 2.3 The Popcorn Operating System

Popcorn is a multiple-kernel operating system designed for researching operating systems [18]. A Popcorn kernel runs on a single processor core or a group of cores that have shared physical cache-coherent memory [18]. The kernel appears to be a single system image (SSI) so that the fact it operates as separate kernel instances is hidden to the user or application programmer [18], these are referred to as nodes in this paper. These processors (or groups of processors) do not need to be of the same instruction set architecture (ISA). This is called a heterogeneous setup [18]. When referring to kernels from now on in the paper it will refer to the Popcorn kernel instance that runs on each processor or group of processors. Allowing for different processors within the same operating system means that applications can be migrated to processors that are better tailored to that particular task (in terms of power efficiency, local peripherals, etc.) [18].

### 2.3.1 Migration of Threads Between Processors

The Popcorn operating system allows for threads to be migrated to and from the processor that it was created on (the origin node) to a remote node. For the thread to be able to execute data must be copied to the remote node. When a page of data is read-only then it can be transferred from the origin node to the remote node, shown in Figure 2. If the page required is writable then the page is copied to the remote node and the page is blocked from being executed without first being transferred back from the origin node as shown in Figure 3. If the origin node requires the writable page, then it is transferred back from the remote mode so that this page can be kept consistent from any changes.

Host 1

Host 2

Figure : Shows the transfer of a read-only page from one host to another, the page is copied. Green blocks represent read-only pages, white represents unmapped pages of memory.

Host 1

Host 2

Figure : A writable page being transferred to another host, the page on the original host is blocked from being used until it is moved back. Green represents read-only pages; white represents unmapped pages of memory and red is a writable page.

A Popcorn OS is specific to the architecture that it is deployed on in order to support the different memory architectures [18]. Pages can be shared between the different kernels while still providing coherency [18]. Popcorn does this by giving ownership of pages to a particular kernel [18]. When memory is not shared then pages are replicated for each kernel that needs it [18].

### 2.3.2 Popcorn Messaging Layer

The Popcorn prototype simulates the messaging layer by using shared memory, buffering and inter-process interrupts [18]. Inter-process interrupts introduce latency for this reason Popcorn also uses polling to reduce the number of interrupts [18]. Messaging must be used when there is no shared memory in order for kernels to be able to communicate [18]. The Popcorn messaging layer provides only point-to-point messaging as opposed to being able to broadcast messages to multiple nodes [18].

### 2.3.3 Problems with the Existing Popcorn Messaging Layer

Chapter 8

# Conclusion

This project initially had three different goals: allow for multiple communication protocols at the same time, allow modifications of Popcorn nodes connected from user-space without rebooting and to provide authentication between nodes.

The updated structure of the Popcorn module means that when sending a message, it seeks the appropriate function from the transport structure specified for that node in the node list. This allows for an arbitrary number of communication protocols to be used between different nodes at the same time. TCP socket was used for this project, but a framework was created and outlined in this project for other protocols to be implemented around.

The adding of the nodes within the Popcorn network from user space using a proc file as the input to the kernel module. This allowed for connections to be made and was then further extended to propagate messages to other nodes within the network. This meant that once a connection had been established between two nodes then all other nodes on the network would connect also. This shows that the method is working correctly in future this will be expanded to allow for removing nodes and possibly other commands to the nodes.

Progress was made on providing authentication and security for the Popcorn system. Research into the necessary security and encryption was conducted. Security and authentication was central to many of the design decisions so far. Work also began on implementing encryption for Popcorn but this has been left to part 2 of the project.

In part 2 of this project, I intend to finish providing security and authentication between nodes, add flow control to the Popcorn command messages, add functionality to remove nodes, create a command utility that allows for connection of nodes in a more user-friendly way and perform a more in-depth evaluation of Popcorn including the use of real hardware.

# Bibliography

|  |  |
| --- | --- |
| [1] | E. Novikov and I. Zakharov, “Verification of Operating System Monolithic Kernels Without Extensions,” 30 October 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-03427-6\_19. [Accessed 6 November 2020]. |
| [2] | B. Roch, “Monolithic kernel vs. Microkernel,” 2004. [Online]. Available: http://web.cs.wpi.edu/~cs3013/c12/Papers/Roch\_Microkernels.pdf. [Accessed 2 November 2020]. |
| [3] | A. S. Tanenbaum and R. V. Renesse, “Distributed Operating Systems,” December 1985. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/6041.6074. [Accessed 17 June 2021]. |
| [4] | J. S. Shapiro, “The EROS System Structure,” 11 January 2007. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.5173&rep=rep1&type=pdf. [Accessed 15 October 2021]. |
| [5] | R. Achermann, D. Cock, R. Haecki, N. Hossle, L. Humbel, T. Roscoe and D. Schwyn, “mmapx: Uniform memory protection in a heterogeneous world,” 1 June 2021. [Online]. Available: https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s08-achermann.pdf. [Accessed 23 September 2021]. |
| [6] | J. S. Shapiro, J. M. Smith and D. J. Farber, “EROS: a fast capability system,” 12 December 1999. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/319151.319163. [Accessed 14 October 2021]. |
| [7] | S. Mullender, G. v. Rossum, A. Tananbaum, R. v. Renesse and H. v. Staveren, “Amoeba: a distributed operating system for the 1990s,” May 1990. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/53354. [Accessed 28 May 2021]. |
| [8] | A. S. Tanenbaum, R. v. Renesse, H. v. Staveren, G. J. Sharp and S. J. Mullender, “Experiences with the Amoeba distributed operating system,” 1 December 1990. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/96267.96281. [Accessed 10 June 2021]. |
| [9] | Shubham, “What is kernel - monolithic and microkernel,” 29 January 2018. [Online]. Available: https://medium.com/@shrimantshubham/what-is-kernel-microkernel-and-monolithic-kernel-66c6de358b43. [Accessed 18 April 2021]. |
| [10] | A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach and A. Singhania, “The Multikernel: A New OS Architecture for Scalable Multicore Systems,” October 2009. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1629575.1629579?casa\_token=I7\_hNx4wHdsAAAAA:0SVwWy0PBIxp-ZjoK3g9NLYR0uT1tJUHc29C2HBgPjo\_VysRDtqGmfp1-3Swdqh6lng4qYOkTf3vKg. [Accessed 8 November 2020]. |
| [11] | R.F.Rashid and H.Tokuda, “Mach: A system software kernel,” 15 June 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0956052190900045. [Accessed 20 October 2020]. |
| [12] | D. L. Black, D. B. Golub, D. P. Julin, R. F. Rashid, R. P. Draves, R. W. Dean, A. Forin, J. Barrera, H. Tokuda, G. Malan and D. Bohman, “Microkernel Operating System Architecture and Mach,” 30 April 1992. [Online]. Available: https://courses.cs.washington.edu/courses/cse451/15wi/lectures/extra/Black92.pdf. [Accessed 8 November 2020]. |
| [13] | M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, “Mach: A New Kernel Foundation For UNIX Development,” 1986. [Online]. Available: http://cseweb.ucsd.edu/classes/wi11/cse221/papers/accetta86.pdf. [Accessed 8 November 2020]. |
| [14] | “Openmach Git Repository,” [Online]. Available: https://github.com/openmach/openmach/blob/master/include/mach/message.h. [Accessed 8 November 2020]. |
| [15] | R. Krten, “Getting Started with QNX Neutrino: A Guide for Realtime Programmers,” 2008. [Online]. Available: http://jedrzej.ulasiewicz.staff.iiar.pwr.wroc.pl/Neutrino/getting\_started.pdf. [Accessed 8 November 2020]. |
| [16] | S. Peter, A. Schüpbach, D. Menzi and T. Roscoe, “Early experience with the Barrelfish OS and the Single-Chip Cloud Computer,” 2011. [Online]. Available: https://people.inf.ethz.ch/troscoe/pubs/marc11-barrelfish.pdf. [Accessed 8 November 2020]. |
| [17] | A. Barbalace, A. Murray, R. Lyerly and B. Ravindran, “Towards Operating System Support for Heterogeneous-ISA Platforms,” April 2014. [Online]. Available: http://www.popcornlinux.org/images/publications/sfma14.pdf. [Accessed 21 October 2020]. |
| [18] | M. Sadini, A. Barbalace, B. Ravindran and F. Quaglia, “A Page Coherency Protocol for Popcorn Replicated-kernel Operating System,” 2013. [Online]. Available: http://www.popcornlinux.org/images/publications/marc2013\_camera\_ready\_fixed.pdf. [Accessed 21 October 2020]. |
| [19] | J. Corbet, “Popcorn Linux pops up on linux-kernel,” 5 May 2020. [Online]. Available: https://lwn.net/Articles/819237/. [Accessed 25 October 2020]. |
| [20] | A. Barbalace, B. Ravindran and D. Katz, “Popcorn: a replicated-kernel OS based on Linux,” 2014. [Online]. Available: https://www.linuxsecrets.com/kdocs/ols/2014/ols2014-barbalace.pdf. [Accessed 10 April 2021]. |
| [21] | “Structure of monolithic kernel, microkernel and hybrid kernel-based operating systems,” 17 July 2008. [Online]. Available: https://en.wikipedia.org/wiki/Monolithic\_kernel#/media/File:OS-structure2.svg. [Accessed 8 November 2020]. |
| [22] | A. Barbalace, B. Ravindran and D. Katz, “Popcorn: a replicated-kernel OS based on Linux,” 2014. [Online]. Available: https://www.linuxsecrets.com/kdocs/ols/2014/ols2014-barbalace.pdf. [Accessed 19 April 2021]. |
| [23] | B. Ravindran, “Replicated-kernel Linux,” [Online]. Available: http://popcornlinux.org/index.php/replicated-kernel-linux. [Accessed 19 April 2021]. |