Master Thesis
Software Engineering
Thesis no: MSE-2004:16
June 2004

Experiences from Simulating TSP Clusters in the Simics Full System Simulator

Emil Erlandsson and Olle Eriksson

School of Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Author(s):
Emil Erlandsson
Address: Källundav. 155, 291 92, Kristianstad, Sweden
E-mail: emil@buglix.org

Olle Eriksson
Address: Lindblomsv. 98, 372 33, Ronneby, Sweden
E-mail: mail@olle-eriksson.com

University advisor(s):
Håkan Grahn
School of Engineering

Simon Kågström
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : +46 457 271 25

ABSTRACT

TSP (or Telecommunication Server Platform) is a scalable, high availability cluster operating system developed by Ericsson for use in the telecommunications industry. This thesis describes an attempt to simulate a TSP cluster in the full system simulator Simics, and talks about some of the possibilities offered by such a setup and full system simulation in general. This attempt to simulate TSP was unsuccessful in completely booting the cluster in Simics, but some of the experiences and problems encountered are described. A proposed development environment for working with TSP in Simics is also presented, along with scripts that were created during this thesis to alleviate the working process.

Keywords: TSP, TelORB, Full system simulation, Simics

Contents

Chapter 1

Introduction

Telecommunication platforms today must meet ever increasing demands of high availability, high performance, and scalability. That is especially important in this world where communication services are becoming more and more common. To achieve some of these requirements, distributed computing, or parallel processing by means of computer clusters play a very important role.

Ericsson, a world leading telecommunications company, has developed a distributed operating system for large-scale, embedded, real-time applications, called TSP (Telecommunication Server Platform). TSP is designed for systems with soft real-time constraints and provides an OS kernel, database, and a development environment for writing application code. [23]

However, development and debugging of cluster operating systems, and parallel applications running on clusters, is associated with many difficulties. For example, running the operating system on physical hardware is expensive, and debugging can be cumbersome. To alleviate this, TSP comes with an emulator/simulator called Vega that allows one or more cluster nodes to run on a real computer without requiring TSP to be installed on the physical machines themselves. Vega only simulates the operating system which TSP is interacting with, and nothing else.

A different type of simulator, a so-called full system simulator, simulates processors and entire systems at the instruction-set level and provides a whole other set of possibilities. Simics is one such simulator, developed by the company Virtutech. Simics supports a number of different processor architectures, and can be used to simulate single processors, symmetric multi-processors (SMPs), or several computers connected together via a network in a cluster. With a full system simulator it becomes possible to simulate full computer clusters and perform debugging operations otherwise not possible.

The purpose of this thesis has been to find out if it is possible to simulate a full computer cluster running TSP in the simulator Simics. It was of particular interest to see what the possible problems would be, and if they could be solved. The intention was also to gain some experience in the process, and to evaluate the practicality of using Simics together with TSP in a production environment.

In addition to this, scripts and experiences were to be compiled into something that would function as a development environment for people who work with TSP. The idea was that this would simplify the process of setting up a simulated cluster running TSP. Other potential features of such an environment would also be discussed and suggested.

The report begins with an introduction to parallel computing in chapter 2, and to the operating system TSP in chapter 3. Chapter 4 continues to describe full system simulation and lists a few available simulators and their characteristics. In chapter 5, the details of the attempt to simulate TSP in Simics is described together with the overall results. A more complete list of the problems and experiences encountered are detailed in chapter 6, and in chapter 7 the implementation of the development environment and how it can be extended is described. Chapter 8 contains a discussion of future work with TSP in Simics.

Chapter 2

Parallel Computer Systems

In the world of computers, the need for speed is ever increasing and sometimes the law of Moore [2] is not enough. Large scientific projects, and heavy industrial equipment requires fast and reliable hardware. Installing newer, faster processors is a common way of increasing the speed of computers, but it leaves some problems unsolved. Pfister [29] writes about three ways of doing things faster:

- Work harder.
- Work smarter.
- Get help.

The first is quite obvious, simply add more power to accomplish the goals. While this works in some situations, it is not a universal solution. Another way is to work more efficient, which means performing more work in the same amount time, with the same means of power. As an example, Pfister [29] mentions Henry Ford's automobile assembly line, which revolutionised the world of industrialised automobile creation. Regardless of the the power or efficiency, it is always possible to get help from others to speed things up.

In computer terms, Pfister's three points translates to higher processor speed, smarter algorithms and parallel computing (processing). This chapter is about the latter.

2.1 Parallel Computing

In the computer world, "getting help" means that more than one computing unit (or processor), work together towards the same goal. This is usually referred to as *parallel processing* or *parallel computing*. The most common way of achieving parallel processing is either to equip a computer with more than one processor (SMP), or connect several whole computers to work together (computer clusters). There is also another alternative called NUMA, which is described further down. [29]

2.1.1 Symmetric Multiprocessor Architecture

Symmetric Multiprocessor (SMP) is a technique where several processors (CPUs) work together in one computer. In an SMP there are always multiple processors, but one thing of everything else, such as IO systems and memory. Each processor has exactly the same abilities as any other, can access the same range of memory, control IO devices in the same way etc. The word symmetric comes from the fact that the rest of the system looks the same from the point of view of any of the processors. [29]

While large SMPs containing many processors can certainly be built, there is a practical limit to how large an SMP can be. Since all the processors of an SMP machine share the same memory, this becomes a bottleneck, and also security constraints for memory access. The use of cache memory helps. but the problem still remains. [29]

2.1.2 NUMA

NUMA stands for Non-uniform Memory Access and is an architecture designed to surpass the scalability limits of SMPs. In a NUMA system, the memory buses of several computers are connected together through an intermediary network. To each processor, there is only one large memory, although it may exist on another memory bus. As a result, depending on where in the memory a piece of information exist, it may take longer to access for some of the processor. On the other hand, NUMA systems are more scalable than SMPs. [29]

2.1.3 Computer Clusters

Computer clusters are quite different in that they are not at all symmetric like the processors in the SMP. The computers in a cluster may very well have very varying set of hardware.

Clusters are usually divided into four groups depending on how they are interconnected. There are basically two places in the computers where they can be connected, through the IO subsystem (usually the network adapter or disk drive) and the memory subsystem (memory bus). This is called clusters are either IO attached, or memory attached. The communication can either be message based or storage based. In a message based cluster, the computers communicate by sending messages, and in a storage based cluster, they communicate by reading and writing in the shared memory. [29]

2.2 High availability

Lately, there has also been an increasing demand for highly available systems that continue to run even if a small part breaks. In telecommunication systems in particular, this has always been important. Any break in the operation of a telecommunication system can have dramatic impact on business and the service that customers take for granted.

SMPs and NUMA systems are not highly available by definition because there is no way to configure them to have no single point of failure. If a processor, memory or IO subsystem breaks down, the whole computer will eventually fail. Clusters, on the other hand, are highly available. This is accomplished by avoiding any single point of failure in the hardware. The basic idea is that if one piece of equipment breaks down, the system fails over to a secondary component, and continues to run like nothing happened. [29]

Depending on the requirements for high availability, different step are taken to achieve this goal. In simple scenarios, backing up business data may be enough, while in other scenarios every piece of hardware may be duplicated to ensure that the system will continue to run.

2.3 Software and Applications

One of the biggest problems with parallel processing is the software. The operating system must be very aware of the type of hardware it is running on, whether it is an SMP or a NUMA, or a cluster. Also, ordinary applications written for a uni-processor system will not be able to take advantage of the parallel hardware of an SMP, NUMA or cluster. Writing applications that do is difficult and present more challenges than people might expect.

Chapter 3

TSP

TSP (or Telecommunication Server Platform) is a cluster operating system developed by Ericsson for use in the telecommunications industry to provide an environment for applications that control telecommunication traffic. It is designed to keep the applications running continuously without interruption and it focuses on soft real-time response, high throughput, scalability, and minimal downtime. This is accomplished by running the operating system and applications on nodes in a computer cluster.

The nodes in TSP cluster are connected to each other through an ordinary ethernet network. An application that runs on the cluster does not exist on one single node, actually its process is distributed across all the available nodes. Where exactly each application process exists is completely transparent to the applications.

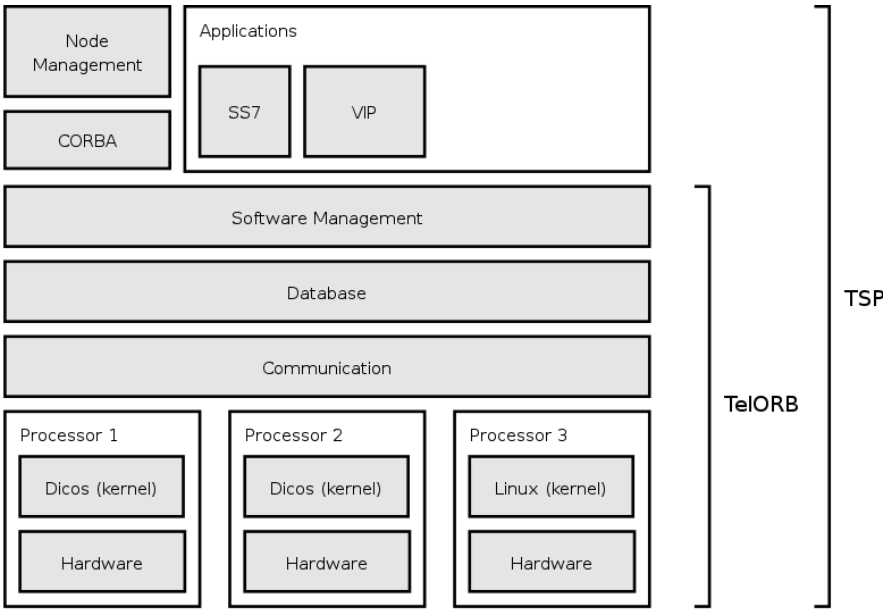

Figure 3.1: Overview of TSP.

Each node in the cluster runs an OS kernel that provides applications with processes, threads, memory management, scheduling etc. The kernel normally used is called Dicos and is developed by Ericsson for this very reason. Work is currently in progress to make TSP run on the Linux kernel as well. On top of the OS kernel is the TelORB middle-ware layer that handles the network communication, provides a persistent database in primary memory, configures executing software to run efficiently on the available nodes, and an object request broker to communicate information

throughout the system. All of this together is called TelORB (which stands for Telecommunications Object Request Broker).

The TSP cluster nodes are managed by a number of node management tools and services. The cluster is also connected to Ericsson's important telecommunications bearer network: the signalling network (SS7) [19]. There are also so called VIP nodes that provide access for administration of the cluster. All these products together are sold under the name TSP.

3.1 Clustering Characteristics

The main reason for TSP is to have a system that will keep running with very little downtime (high availability), that can be extended by adding new hardware (scalability), and that also provides real-time characteristics.

Obtain high availability can be achieved by fault-tolerant hardware, but this is expensive and not 100% reliable. TelORB instead relies on software to obtains the high availability. The benefit of this strategy compared to obtaining it through the hardware is mainly cost. Ordinary off-the shelf computer components can be used, which is both cost effective, and allows operators to take advantage of latest in hardware.

By designing the system to consist of loosely coupled nodes, the system does not depend on all the nodes being available 100% of the time. If one node goes down, the system is intelligent enough to keep on running on the remaining nodes. Applications and data however, could be lost when nodes fail. To prevent this, TSP uses distribution. All data and every piece of equipment in the system is duplicated to avoid failure in case something breaks. The software processes and their data, as well as the database, are stored on more than one node. Not only the nodes themselves, but also the ethernet switches and power units can be duplicated. [18]

Scalability means that processes in the TSP cluster can be added and removed without interrupting the operation of the system. Upgrading the cluster like this is a very flexible way of giving the system more processing and storage capacity, while keeping the operating system running. Actually the whole system can be moved to new hardware without interrupting the operation if that is needed. [23] [18]

Because TSP was designed for use in the telecommunication industry, soft real-time responsiveness was a requirement. It means that applications that run on it should display a statistically deterministic behavior depending on the load of the system. TSP relies on an interruptible OS kernel to provide this. It also has mechanisms to reject certain types of operations to be able to keep the real-time constraints. [18, 23]

3.2 Technical details

Ericsson usually provides all the hardware to run TSP but it is not a requirement by the operating system. TSP can run on most processor architectures (x86, Sparc etc) and an ordinary ethernet network is used to interconnect the nodes. That is basically all the hardware needed to get TSP up and running. [23]

There are two different types of machines in TSP, *IO machines* (short for Input/Output) and *TelORB machines*. There are usually one or two IO machines, and a larger number of TelORB machines. Both types of machines are connected to the same network but only the TelORB machines are part of the TSP cluster. The IO machines run an operating system such as Unix/Linux or Windows and hold the files needed for the initial loading of TSP, handle file dumps, and are used for operation & maintenance of the cluster. The TelORB machines on the other hand are the actual nodes of the cluster, and all the TSP applications run on them. The TelORB machines run the TelORB kernel Dicos, which is loaded from one of the IO machines during start-up. [18]

Everything in TSP is built using object orientation, both the operating system itself, and applications running on it. The system consists of a set of managed objects which contain both the data and the methods/functions to manipulate the data. Each process in the system contains one or more objects of different types [23]:

Dialogue objects Dialogue objects for communication between objects in different processes using the IPC (Inter Process Communication) protocol.

LPC objects Ordinary (LPC) objects with data and methods that communicate through the LPC mechanism (Linked Procedure Call).

CORBA objects CORBA objects (described below) that can communicate with other CORBA objects over an IP network.

One thing that contributes to the openness of TSP is the fact that CORBA is used as an object request broker for communication over IP networks. CORBA stands for Common Object Request Broker Architecture and is produced by the international standardisation body OMG (Object Management Group) [3] to be a common architecture for all object request brokers. Because of object request broker in TSP, applications in TSP can exchange information with non-TSP applications on the other side of the world, provided the ORB on the other side also follow the CORBA specification. In this sense, TSP is a very open system since applications are not limit to the computers running TSP. [18] [23]

While data inside a process is lost if that process is killed or lost, data in the database is not. The database still exists in the primary memory of the nodes (which is volatile storage), but since it is duplicated over multiple cluster nodes, it will remain available provided not too many nodes crash at the same time. Objects in the database are a special kind of LPC object called persistent objects because they are just that, persistent across node crashes. [18] [23]

Special notifications are sent to notify the operator of critical events such as high load or low memory on the nodes. Alarms are also used to inform of node and hardware failures. [23]

TSP also provides other services to applications through different APIs and specifications [23]. These services include timers that execute call-back functions after a specified period of time, a cluster-wide synchronised real-time clock and (through NTP [8]), and calendar functions for converting dates between standardised calendar formats.

3.3 Development Environment

TSP includes a development environment for writing applications that will run on TSP. This development environment includes a software structure model, build tools, and configuration tools. Applications are written in either C/C++ or Java and use the services provided by the TSP API. [23]

All processes that run on TSP are specified in a language called Delos. It contains constructs that ordinary languages lack to describe process characteristics and database objects. The Delos code is compiled using a compiler that produces C++ or Java code. This code is then used as a base when writing applications for TSP. By adding application-specific code to the source code base and then compiling it with a C++ and Java compiler the executable files are created. These can then be loaded by the TSP operating system and distributed across the cluster nodes. [23]

The software structure consists of a number of different so called managed items, for example: software interface and object units (source code), load modules (compiled code), and internal delivery packages (which group load modules together). The configuration and build tools provided by TSP are used to generate these managed items from each other and combine them with application specific code, as well as generating boot-loader information to boot the system. [23]

3.3.1 Debugging

The main tool for working with TSP is a simulator/emulator called Vega which is used to start TSP. It runs as an ordinary software process on a UNIX/Linux machine in which the operating system is started, and it creates a “hardware” adaptation layer that tricks TSP into thinking it is running on real hardware. Several Vega instances can be started on the same machine or on other connected machines to run multiple TelORB nodes in a simulated cluster. Consequently, using Vega has the benefit of a much simpler process of starting a TSP system, and it allows applications to be started without access to a real computer cluster.

Debugging of the operating system and applications running on it can be performed with ordinary Unix/Linux tools such as gdb (GNU debugger) or sysview. There are also inspection tools to allow the content of the database to be examined.

Chapter 4

Simulation

Ever since the early 1950's a need for detailed simulation of hardware with computers have existed. The reason is simply that simulation is a vital tool for designing new computers due to its low costs and simplicity [27].

Today simulation is not only a tool for designing new computer systems, but also a great aid for software development, computer system migration and education in computer science. This chapter aims to provide an overview of computer simulation with software and also introduce some simulation models that are currently used both in academia and industry.

4.1 What Is Simulation?

Simulation is an important concept, not only in the computer world. For instance global weather simulation is used to create more accurate weather forecasts as well as predicting natural disasters. In the software industry simulation is used for instance to develop software for unavailable hardware or to debug complex programs like operating systems.

A simple explanation of simulation is that it is a replacement for something (it is a replacement for the system it simulates). For instance, since it is expensive to crash a real car in a crash test, the car is replaced with a simulation model that acts as the car. The simulation model incorporates the properties and abilities of the car at different accuracy levels. It is not necessary to simulate every little property of the car. For instance in a crash test that measures side impact, it is not very usable to know what the car radio was playing.

Simulation (an especially full system simulation, explained later) has not been feasible until recently, when the hardware price of ordinary PCs have decreased while the performance of the same have increased. [10]

4.2 Advantages of Computer Simulation

As J. Engblom says [17] simulation is "just software" thus it gives a large amount of opportunities compared to using the real thing:

Configurability A simulated computer can be configured to have any feasible hardware setup, regardless of the physical availability of that hardware.

Controllability A simulation is a rigorously controlled environment. The simulation itself can be started, stopped or paused and the environment can be altered in any thinkable way.

Determinism Depending on how the simulation model is implemented, the simulation can be completely deterministic.

Globally synchronous If a simulation session contains several simulated computers interconnected through a virtual network, the simulation of all computers can be stopped at the exact same time.

Checkpointing The current state of a simulated computer can be saved to disk so that the simulation can continue from the saved state later on.

Availability It is easy to set up new computers, it is just a matter of configuration.

Inspectability During execution of the simulation model, everything in the model can be inspected without interfering with the execution.

Sandboxing The simulated environment is a perfect sandbox (a sandbox is an controlled environment where potentially insecure applications can be tested at no risk) for executing tests.

Simulation is a word that are sometimes used as a substitute for emulation, but even if they generally mean the same thing, there is a significant difference. A simulator aims to imitate the functions of a device (or the interface of the device), but not the internal design. An emulator, on the other hand, aims to imitate the internal design of a device, thus is simulation easier to implement since so called "dummy" devices can be created. [20]

4.3 Properties of Simulation

As discussed above, the simulation model can incorporate different accuracy levels of the simulated target. Accuracy is one of three important properties of simulation models. The other two are performance and flexibility [13].

The performance of a simulation model is often measured as a factor of slowdown compared to the "real thing" whilst flexibility is a measure of how easy it is to change the model to fit other conditions.

All computer simulators that exist incorporate these three properties more or less. Often a simulator focuses on one or two of these properties, since it is hard to satisfy all three and still keep the original target group of customers. For instance, simulators which targets end users that need to run another desktop operating system focuses on high performance whilst simulators that are used for research in computer science often focuses on accuracy or flexibility.

4.4 Principle of Computer Simulation

As mentioned before, a simulator which implements a simulation model is a piece of software that runs on a computer (the host) in an operating system like any other program. The simulator provides virtual hardware (target system) on which another operating system can be run, or the simulator provides a simulation of the operating system as well. This is illustrated in Figure 1 where one host computer is running three simulated computers at the same time.

The key element is that parts of the instruction-set of the CPU is simulated more or less completely. The rest of the computer can also be simulated in more or less accurate, depending on the purpose of the simulation. Less accuracy generally leads to higher performance and vice versa.

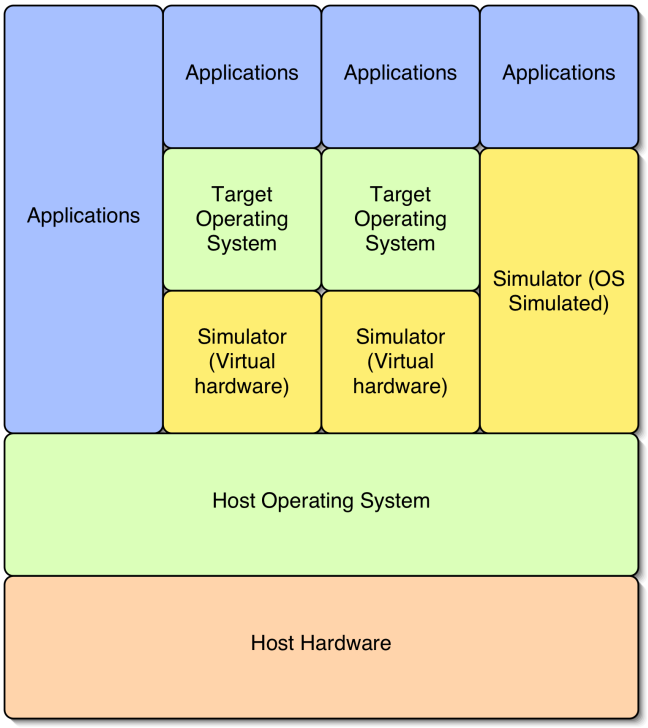

Figure 4.1: A model of a computer running three computers in simulation. *The figure is a based on a figure from [14]*.

4.5 Full System Simulation

Until recently, most simulation tools have only modeled parts of a computer's hardware, which has it drawbacks in terms of what can be achieved with the simulation, and with what level of confidence. Full system simulations (or complete machine simulation) model all the hardware typically found in a computer system [24]. This not only means a subset of digital components found in one computer, but rather all parts of a system of computers, perhaps even connected together over a network [27].

Computer system behavior can be investigated in other ways than through simulation. However, these alternatives have their drawbacks. Analytic models or mathematical approximations can be used to analyse specific issues of a computer system, for instance a disk drive's queuing behavior. However, this always requires some kind of approximation or simplification to be realistic, and it is often difficult to do right in such complex environments as computers are. Building hardware prototypes with the ability to collect data about their behavior is another alternative, but an expensive and time-consuming one. Software simulation is a much easier and more flexible solution. [27].

Using full system simulations, it becomes possible to observe the behavior of the system and not just the individual components. Since full system simulations boot and run unmodified operating systems, it is possible to run a wide range of applications and services and see how they interact with each other and with the hardware. It is especially interesting to see how a system responds when the applications running on it are put under high load. It also allows designers of operating systems to observe how the operating system utilizes the hardware and what types of optimizations can be used to improve the performance. The flexibility of software simulations and the extreme visibility into the behavior of the hardware makes it very easy to try new ideas

and see how the system responds to those changes. [27]

This report covers some simulators for full system simulation, but there are of course many other simulators available, such as VirtualPC [21], QEMU CPU [5], Plex86 [4], SimpleScalar [13], binary rewriters that allow a programs for a non-native instruction set to be executed (for example Digital FX!32 [25]) and hardware-based virtualisation systems such as IBM system/360 [12], but they are outside the scope of this thesis.

4.6 Full System Simulators

To pinpoint which simulator to use for simulating TSP, the characteristics of different simulators have been investigated and compared. In this section, five quite different simulators are described and compared.

4.6.1 Simics

Simics from Virtutech is a full system simulator, meaning that it runs its own BIOS and unmodified operating systems out of the box. The purpose of Simics is to provide a simulator that balances between a high accuracy level and performance. The target users of Simics are both industry and academia, since it is a very flexible framework. [27] [11]

Simics provides a model of the simulated machine at an instruction-set level (including full supervisor state), which in turn provides an abstraction level between hardware and software. An instruction-set simulator simulates every instruction of the target machine, one at a time. Instruction-set simulators are very flexible due to their ability to simulate almost any computer and gather any form of requested statistics. They are suited for usage in architectural design where designers can see the real effect of design trade-offs. [27] [28]

Supported Platforms

The currently supported hardware platforms that Simics is able to simulate are: UltraSparc, Alpha, x86, x86-64 (Hammer), PowerPC, IPF (Itanium), MIPS, and ARM. Since Simics is "just software" it runs on multiple host operating systems which includes: Linux (x86, Alpha and PowerPC), Solaris/UltraSparc, Tru64/Alpha and Windows 2000/x86. [27]

Network Support

Simics have an extensive support for virtual networks managed by a tool called *Simics Central*. Simics central is used to interconnect simulated computers with each other, as well as with the "real world". Simics central is like a router, which is started separately from the other simulations. After Simics central is started, the different machines are connected to the central with virtual network cables.

Checkpoints

Another feature of Simics is checkpoints (also referred as "snapshots") which most current simulators incorporate. It lets the user save the complete state of the simulated machine to a file on the host machine file system. The state of the simulated machine can be started at a later time.

As discussed above, with Simics, Virtutech aims to strike a balance between performance and a high accuracy level of the simulated hardware. It is at an abstraction level which provides good performance (for the implementations it is intended for) and at the same time sufficient timing and functionality accuracy to run commercial workloads. [27]

Scripting Interface and API

Simics incorporates an integrated scripting environment and a powerful API. The scripting language used is Python, which is a widespread, interpreted, interactive, object-oriented scripting language which aims to be easy to use with a clear syntax [16]. This scripting environment gives the. This scripting environment gives the user full control over the simulation in the form of scripts. For instance, the simulation can stop after *n* number of CPU-instructions and print the values of different registers. It is also possible to inject errors like corrupt network packages or hardware failures. [27] [11]

4.6.2 SimOS

SimOS [28] is another full system simulator, or as the creators at Stanford University call it, "a complete machine simulation environment". It can simulate both uniprocessor and multiprocessor systems. Simics and SimOS share many goals, and are thus quite like each other. The main difference is how the simulators pursue the performance properties. Both simulators are designed in a hybrid fashion, i.e. it is possible to change the detail and accuracy of different stages in the simulation. SimOS uses three simulators: one that quickly boots the operating system, one that warms the caches and the last modelling CPU pipelines etc. Simics is designed so that it can model the cache hierarchies quickly, thus give a more detailed processor model. SimOS does not. This is the big difference between them [28].

Supported Platforms

SimOS does not offer as many simulated architectures as Simics does, and that can be explained by the target user group which is almost only academia for SimOS and both commercial and academia for Simics. Currently SimOS can simulate the 64-bit MIPS-IV architecture running SGI IRIX 6.4 and the DEC Alpha platform running the Digital UNIX (support for Linux on the Alpha platform is underway) operating system. SimOS itself runs on the following platforms (and operating systems): SGI MIPS (IRIX 5.*/6.*), DEC Alpha (Digital Unix), Sun SPARC (Solaris) and Intel Pentium/Pro/II (Linux). [30]

4.6.3 VMWare Workstation

In contrast to Simics and SimOS which focus on academic users and system development, VMWare is a machine simulator that is primarily focused on desktop usage. It simulates only the x86 architecture and runs on x86 hosts with one of the following operating systems: Windows NT 4.0, Windows 2000, Windows XP or Linux. The target operating systems that can be simulated are different versions of MS Windows, FreeBSD, Novell Netware and Linux. [14] [22]

Virtualisation

The big difference between simulators like Simics/SimOS and VMWare is that the latter does not simulate *every* single CPU-instruction. Since the target users of VMWare are desktop users who wants to run two operating systems simultaneously on one physical machine, the creators of VMWare emphasized performance when they developed it and took advantage of a technique called *virtualisation*. It means that instead of simulating all x86 instructions, most of them (except privileged instructions) execute at the native hardware speed on the host processor. This increases the performance, but limits the host platform to x86 architectures. [26]

Development and Migration

VMWare provides a platform for software development, testing and deployment. Development of multi-platform applications that are supposed to run on both different versions MS Windows, FreeBSD and Linux is quite simple, since the developers need only one physicalq machine each. The application can be tested in different simulated environments simultaneously. After the application is deployed, it is easy to maintain support on different platforms at the same time. [14] [22] Besides the development features of VMWare, it can be a handy tool for migrating from one operating system to another. During a migration, there are almost always some applications that don't run on the new operating system for some reason. In such scenario VMWare provides the possibility of running both systems side by side, until every needed application is running on the new operating system.

4.6.4 Bochs

Bochs is an open source project that simulates a complete Intel x86 machine (either a 386, 486, Pentium or Pentium Pro). Similar to Simics and SimOS, Bochs simulates every single CPU-instruction from the power-up to the reboot of the machine. Since it doesn't use virtualisation (see 4.6.3 on page 13), it is possible to run it on many different host platforms including: x86, PPC, Alpha, Sun, and MIPS. No matter which the host platform is, Bochs always simulates x86 hardware. Bochs runs a variety of target operating systems including Windows 95/98/NT, BeOS, Linux flavors and BSD flavors. [26]

Networking

Bochs uses a different approach to network handling than Simics (which is using the Simics central). Instead of using a virtual switch, Bochs is sending the network packages directly through the network card of the host system to the "real world". Since different operating systems handles network traffic in different manners, it is complicated to provide a unified way of simulating network in Bochs. Not all the operating systems Bochs run on, support network simulation. For instance, BeOS can't run it, since the code between Bochs and BeOS has not been written. [26]

4.7 Why Simics?

Simics was chosen to be the simulation environment for TSP over the other full system simulators mentioned in this chapter for several reasons. First of all, it provides a scripting environment, which introduces the possibility to create advanced scripts and development environments around the simulated environment. It also strikes a balance between speed and accuracy, which makes it versatile.

SimOS is also a rather dynamic environment, which provides many interesting features that could have been useful when simulating TSP, but unfortunately the hardware support is limited. As mentioned earlier the only architectures SimOS can simulate today are 64-bit MIPS-IV architecture running SGI IRIX 6.4 and the DEC Alpha platform running the Digital UNIX. Thus it is ruled out as simulation host, since the Dicos kernel does not run on those architectures.

Furthermore VMWare is targeted mainly for desktop usage, and Bochs is too slow, thus leaving Simics as the simulator, best capable of handling this experiment. Simics is also known to have dedicated developers at Virtutech (the company who owns and develops it) that provide almost instant support and bug fixes.

The simulator Vega that was mentioned in the beginning of this report is not a full system simulator, thus it is not mentioned in this section. Vega does not simulate the whole computer as full system simulators do. Instead it simulates only the parts of Dicos that are needed for TSP to

run. Furthermore, it does not include features such as scripting, checkpoints, and the deterministic nature of full system simulators.

Chapter 5

Simulating TSP in Simics

The many benefits of using full system simulation have already been mentioned. Designers of operating systems can develop their system in an environment that provides the same interface as their target platform and take advantage of the possibilities offered by full system simulation. The simulator offers high visibility into the system and allows developers to see what effects their design decisions have. It also simplifies the development and debugging process by providing a single tool that simulates the entire computer system. [27]

The reason for this thesis has been to investigate whether or not it is be possible to run a TSP cluster in a simulated Simics environment. Before TSP could be started in Simics, it was decided that TSP be installed in a real cluster environment, separate from Simics. This allowed us to focus on one system at the time, particularly important since we had no previous experience of neither TSP nor Simics.

5.1 Setting Up TSP in a Real Cluster Environment

The first thing to do was to build a reference cluster environment that could be used to run the TSP operating system. It consisted of four computers with Pentium III processors running at 1 GHz (acting as TelORB-machines), and one machine with a Pentium processor running at 233 MHz (acting as the IO machine). To complete the TSP cluster, the IO-machine was installed and configured, and an application was compiled and prepared that would be used to test the cluster.

5.1.1 The IO-machine

Due to incompatibilities between TSP and version 6.2 and 7.0 of *RedHat Linux*, it was concluded that RedHat version 7.1 was required to run on the IO-machine for the cluster to operate properly.

TSP clusters can be built of different types of computers running different operating systems, which introduces a great advantage, but also the drawback of complex configuration. During the configuration of TSP, the MAC-addresses of the TelORB machines were collected and written to the configuration of the IO machine so that it would know which computers the cluster consisted of.

Furthermore different software daemons had to be installed on the IO machine so that TSP could operate properly. For instance the date-daemon *ntp* and the file-sharing daemon *nfs* had to be installed and configured separately from TSP.

5.1.2 Reference Application

Since just running the Dicos kernel on the TelORB machines does not provide much information, nor any possibilities to run benchmarks on the cluster, a *reference application* had to be installed

in the cluster. Ericsson provides such an application bundled together with TSP, under the identity IDP/DEM101. This application was built using the build tools *epbuild* and *epct* that comes with TSP.

The application (IDP/DEM101) had to be aware of which computers the cluster consisted of, at least those it was supposed to run on. Therefore, the same MAC-addresses which was used during the configuration of the IO machines had to be added to the configuration of the application before it was built.

5.2 The Simics Environment

Setting up Simics went without any major problems. Simics uses configuration files written in the language Python to describe the target machines to be simulated. Example configuration files, provided with the installation, were modified for the purpose of this experiment. A base configuration, common to both the IO machines and TelORB machines, was written. This configuration was then extended with support for two network cards for the IO machines, which is a requirement from TSP. Besides this simple procedure, nothing more was needed to prepare Simics for beginning the experiment.

5.3 Merging TSP and Simics

5.3.1 Simics Central

Simics Central is the component of Simics that is used to simulate a virtual ethernet network to which the simulated machines can connect as if it was a real network. This virtual network was used to connect all the TSP cluster nodes, and had to be configured before TSP was configured. The configuration states what IP-addresses and MAC-addresses belong together of all the computers connected to the virtual network, and thus has to be decided and entered into the configuration files before the simulated cluster can be started.

To make it easier to start a cluster simulation, a script that we created (see Section 7.1.1 on page 27) was written to dynamically assign a MAC-address to each new cluster node. This requires that Simics Central has already been configured to include these MAC-addresses, but that is something that only needs to be done once.

5.3.2 Disk Images

One of the issues that had to be dealt with during the experiment with TSP and Simics concerned disk images. To make it easier to get a working TSP cluster running in Simics it was decided that it would be a good idea to create an image of the hard disk of the IO machine. Doing this means you get a ready-to-run IO machine in Simics without the need to install TSP inside Simics, which probably takes an unrealistically long time.

The Unix tool `dd` was used to create an image of the disk partition of the IO machine, and transferred by FTP to the computer running Simics. This image was then mounted as a loopback device in Linux, and all the needed preparations of the image were made, before it was used as the hard disk of the simulated IO machine in Simics.

Two approaches were used when making Simics recognize the images in the simulated target machine. The first was to create a separate partition table in an image file and use that, together with the partition image in the Simics target configuration. The second approach was to mount the partition image as a loopback device, create an empty disk partition containing a partition table and an empty partition, and then copy all the files from the original partition image to the empty partition of the newly created disk image.

The first approach did not always work because Simics complained about invalid disk geometry on some of the images created with *dd*. This resulted in forced file-system checks when the images were mounted in Simics and some other problems. With the second approach these problems were avoided, but it was slightly more time-consuming when creating the image.

Preparations of the Disk Images

These are the steps that were taken to prepare the images before they were used to boot the Simics target machines:

1. Remove partitions mentioned in `/etc/fstab` that do not exist in the simulated machine.
2. Update `/etc/modules.conf` to use *tulip* as the network device driver (as this is the device that both TSP and Simics supports).
3. Update the network settings to use the IP-addresses specified in Simics Central.
4. Update the TSP configuration (particularly `telorb.conf`) with regards to the IP-addresses in Simics Central (see next section).

5.3.3 Reconfiguring TSP

The only thing that had to be done to the TSP installation (on the disk image) before booting it in Simics was to update the configuration file (`telorb.conf`) to reflect any new IP-addresses being used in Simics, and to reconfigure TSP. When TSP is being reconfigured, it uses this configuration file to automatically create another set of files that are required to start TSP.

However, if Simics Central is configured to create a virtual network with the same addresses as those that are used in the real TSP setup, this step becomes unnecessary. One thing to keep in mind though, is that Simics Central itself will always occupy the first IP-address of the virtual network (x.x.x.1). That means that if `telorb.conf` already uses this address, TSP must be reconfigured either in Simics or on the real hardware, to not use this address. Doing that on the real hardware before creating the disk image will be much quicker, but not necessarily possible, depending on how critical that installation is.

5.4 Results

Once TSP had been installed correctly on the real hardware, and images of the disk in the IO machine had been created and prepared for Simics, the IO machine was booted inside Simics. This worked without any big problems, probably as a result of the fact that the IO machine was running Linux, which is well tested in Simics by Virtutech and their customers. The biggest problems however, emerged when the TelORB machines were going to boot in Simics.

5.4.1 Booting Dicos in Simics

Several problems emerged when the TelORB machines were going to boot in Simics. This, unfortunately, has meant that the attempts to completely boot a TelORB machine running the Dicos kernel have been unsuccessful. However, some important obstacles have been identified and fixed though, which should make further attempts more likely to succeed.

One of the biggest problems was the network device module in Simics. Simics currently supports a few network cards, of which the only one Dicos also supports is a tulip device, *DEC21143*. Very early on, a problem with the packet size of a TFTP [31] request prevented the machine from

booting. Normally, the TelORB machines connect to the IO machine to download the Dicos kernel via TFTP, but this didn't work in Simics. In the serial console of the TelORB machines, the following message was displayed: `ld->index 1 ld->noOfeth0, weird rx pktLen = 42`. The problem was identified as being related to the device module in Simics, and was fixed by the developers at Virtutech.

Other minor problems also appeared during the experiment. Simics Central suddenly started crashing, seemingly at random, after an upgrade of Simics, in an attempt to circumvent another issue. This problem was quickly fixed by Virtutech though.

There were also some minor problems with the login prompt in the IO machine in Simics, which had simple solutions, but took a little time to sort out. The time in the simulation was running so fast that it was impossible to login before the login prompt timed out. Another annoying problem was with the disk images of the IO machine. It seemed impossible, for a while, to create another disk image of the IO machine without experiencing file-system errors and other peculiarities when booting from it in Simics. This later turned out to be the result of low disk space on the IO machine where the disk image was stored before being transferred to the computer running Simics.

The last problem, that it the end turned out to prevent TSP from booting completely in Simics in this experiment, seemed to be related to the *local Advanced Programmable Interrupt Controller* (APIC) in Simics. The local APIC is a set of integrated circuits in the computer hardware that makes up an interrupt controller, to handle interrupts to and from I/O devices [15]. APICs have been part of the Intel processors since the first Pentium processor.

Simics crashes when it is loading the Dicos kernel, and according to Virtutech, it seems to be because something called the *Profiler* in the Dicos kernel uses the APIC in a way that is not supported by Simics. Unfortunately the APIC can not be disabled in Simics for any of the simulated processor of the Pentium family, which is what Dicos is compiled for.

The quick fix to the latest problem, at this point, was to recompile Dicos without the *Profiler*. This was done with the help of Ericsson, but when the new version was tried, it resulted in the exact same behaviour. Virtutech claim they might be able to sort it out if they were given access to the TSP installation, but because of time limits and legal issues that has not been possible.

As a result of all this, the answer to the question of this experiment, is that Dicos (or TSP) does *not* currently boot in Simics. However, while the primary question may not have a positive answer, many experiences have been gained during the experiment that are described in this report.

5.4.2 Performance

Working in Simics is slower than working on a real computer, be it to edit configuration files or compile applications. To give an idea of how long it takes to move TSP from a real hardware setup into Simics, this list of required steps and their approximate duration in time may be useful. Note that in theory, these steps should only need to be done once.

1. Create image of IO machine with dd (10 minutes)
2. Transfer image to Simics computer via FTP (20-25 minutes)
3. Prepare image/edit configuration files (a few minutes)
4. Boot IO machine in Simics (6 minutes)
5. Configure TSP in Simics (2 hours)

Then there is the time it takes for every TelORB machine in the cluster to boot. What we know is that it takes at least more than 10 minutes, most likely about 30 minutes or more to boot

every such machine. Once these machines have been booted it should be possible (not tested) to create a checkpoint of each such machine to start from in the future.

As a side note, the partition image created was about 3.2 Gb large (the Linux installation required approximately 800 Mb and TSP another 2.4 Gb). The image was transferred through a 100 Mbit switched network.

As a comparison, configuring the IO machine on a real computer takes about 6 minutes, whereas it takes about 2 hours to complete in Simics. This comparison is crude, however, because the execution was performed on two rather different computers. The computer acting as the IO machine was a 233 MHz Pentium, and the computer running Simics was a 2.4 GHz Pentium 4 with 512 Mb RAM.

Chapter 6

Problems and Experiences

Installing and configuring TSP and Simics and setting them up to work together is not completely without problems. While it may not take much time for someone who is familiar with both products, it may require more time than is available for someone who simply wants to test an idea for TSP in the simulator. During this experiment with TSP and Simics, a number of problems were encountered that took some time to solve. If Simics is to be used together with TSP in a real situation these problems will need likely need to be addressed.

6.1 Simulation Related Problems and Experiences

Following is a summary of problems concerning full system simulation and some of the experiences gained.

6.1.1 Slow Simulation

Simics being as slow as it is probably presents the biggest problem when it comes to actually using it together with TSP in an efficient manner. The whole process of moving from a real cluster environment to one in Simics must not take too much time, or chances are it will not be used in the end. It is also important that it is easy to set up the working environment and to get started without the need to read a lengthy manual or be forced to fall back on trial-and-error in despair to get things to work.

The scripts that were created (see section 7.1.1) was one way to try to speed up certain operations that were done many times and took a lot of time.

6.1.2 Starting the Simulated Cluster

One of the things that was realized during the experiment was that starting the TSP cluster manually in Simics over and over again took a lot of time. To alleviate this, the script `tsp.py` described on page 27 was created. This proved to be time-saving in the long run, especially since the script could be extended with more functionality as it was needed along the way.

6.1.3 Working With Disk Images

Much of the time during the experiment was spent on preparing disk images. The fact that they were very large (>3Gb) meant that it took a lot of waiting from time to time before any more work could be done. It also meant that much of what was being done had to be thought through carefully, so that valuable time was not lost on something that was later to be found unimportant.

A lesson that was learned during all of this, is to make sure TSP works on the real cluster before starting to create disk images. Then, as soon as the disk images have been used to boot Simics, create a checkpoint and use that from then on. It saves a lot of time in the end.

6.1.4 Installing Each System Separately First

It turned out to be very useful to have a working TSP installation on a real cluster before trying to move the installation into Simics. There were a not so few problems during the TSP installation/configuration process. Had these been encountered while booting TSP in Simics, it would have been almost impossible to sort out which problems were associated with TSP and which were associated with Simics.

There is also the benefit of not having to create and prepare disk images more than one time, which is a major time-saver. Unfortunately this happened quite a few times in this experiment when the TSP installation was believed to be working, only to be proven incorrect later, at which a new disk image was created.

6.2 General Problems and Experiences

The following is a list of more general problems and experiences.

6.2.1 Knowledge

A certain degree of knowledge in hardware and low level programming is recommended for any task similar to this one. This has been particularly important in this case since TSP and Dicos (the name of the kernel running the operating system) has not been tested on Simics, and as it turned out, many of the encountered problems were results of hardware specific issues with Simics. In these situations, the developers at Virtutech were quick to provide updates to Simics that allowed the experiment to continue.

Knowledge in Linux, the operating system on which the simulation has been run, has also been important. Much of the knowledge that comes from working with Linux, such as dealing with partitions and file-systems in more detail, sniffing network packets etc has also been useful. In a situation where new applications were to be developed and tested in a simulation of TSP, knowledge in C++ or Java (the languages used in TSP) would also be required.

6.2.2 Documentation

The importance of documentation, particularly installation manuals and user documentation, has become overly clear during this thesis. The lack of documentation of TSP in particular has been a big problem, and has slowed down the work considerably. A lot of time was spent on installing TSP, which gave us little time to focus on the important problems of moving TSP into Simics. This may however not be a problem for the developers of TSP themselves.

6.3 Contact With Developers

Since this experiment was basically about combining two products, from two different companies, there was a need for communication between these two companies and ourselves. We had to rely on them to fix problems that were encountered, and they had to rely on us to provide accurate and informative information to allow the source of the problems to be located.

Talking to the developers at both companies was a kind of substitute for the lack of documentation, something that was proven again and again to be very useful. One of the mistakes

was to wait too long before contacting the companies when an issue came up that could not be solved. So, in retrospect, being able to contact the developers of the systems to get help is of great importance, and one of the lessons learned.

In fact, the work of this thesis probably could have been done in less time if someone from Ericsson had been directly responsible for it. TSP was after all what took the most time to become familiar with, and the problems that were encountered may have been easier to fix for Virtutech with direct help from someone with better knowledge of TSP.

Chapter 7

Potential Use of TSP in Simics

Besides the main experiment, making TSP work in Simics, another side project has also been conducted. The initial ideas of what TSP and Simics could be used for, together with the experiences drawn when bringing them together have been compiled into something that could be described as a prototype for a development environment. The focus of this, so called, development environment is to make it easier to work with TSP in Simics.

Since this development environment is a prototype only, all functionality is not implemented. This chapter is devoted to the development environment, both in terms of the functionality which is and is not implemented.

7.1 Implemented Features

The script `tsp.py` is the core of the created development environment. This script (which you can find a full listing of in appendix B.1) is a wrapper for Simics, which automates the tasks of bringing up a simulated cluster environment. The script is written in the programming language Python [16] to create a seamless integration with the scripting interface in Simics, which also is Python-based.

The main script, `tsp.py`, is holding all key features which was implemented during the experiment. The whole development environment comprises two other scripts as well, `mkdisk.sh` and `mkparttbl.sh`. They are used to work with the disk images Simics uses as hard drives in the simulated environment.

7.1.1 The Development Environment

Following is a list of scripts that were created during the experiment which together forms the so called development environment.

`mkdisk.sh` Script that creates a disk image including a partition table ready to be used in Simics. It uses the contents of a directory in the file system to populate the disk image.

`mkparttbl.sh` Script that uses an already existing partition image and creates a partition table that is required to boot the partition image in Simics.

`tsp.py` The most important script, used to start the TSP cluster in Simics. It accepts UNIX-style flags as arguments to control the behavior of the script and the TSP cluster. The script allows the user to create a new Simics Central instance if no one is running, sets the number of IO machines and TelORB machines to start, whether or not to boot from Simics checkpoints etc. It also allows the simulated machines to start on separate computers on a real network via SSH to distribute the work load.

The following image shows a TSP cluster started in Simics with the script `tsp.py` using the arguments `tsp.py -c -i1 -t1 --ioconf=snapshot-io`, which indicates that it should start a new Simics Central instance (`-c`), *one* IO machine (`-i1`), *one* TelORB machine (`-t1`), and that the IO machine should be started from a checkpoint (`--ioconf=snapshot-io`).

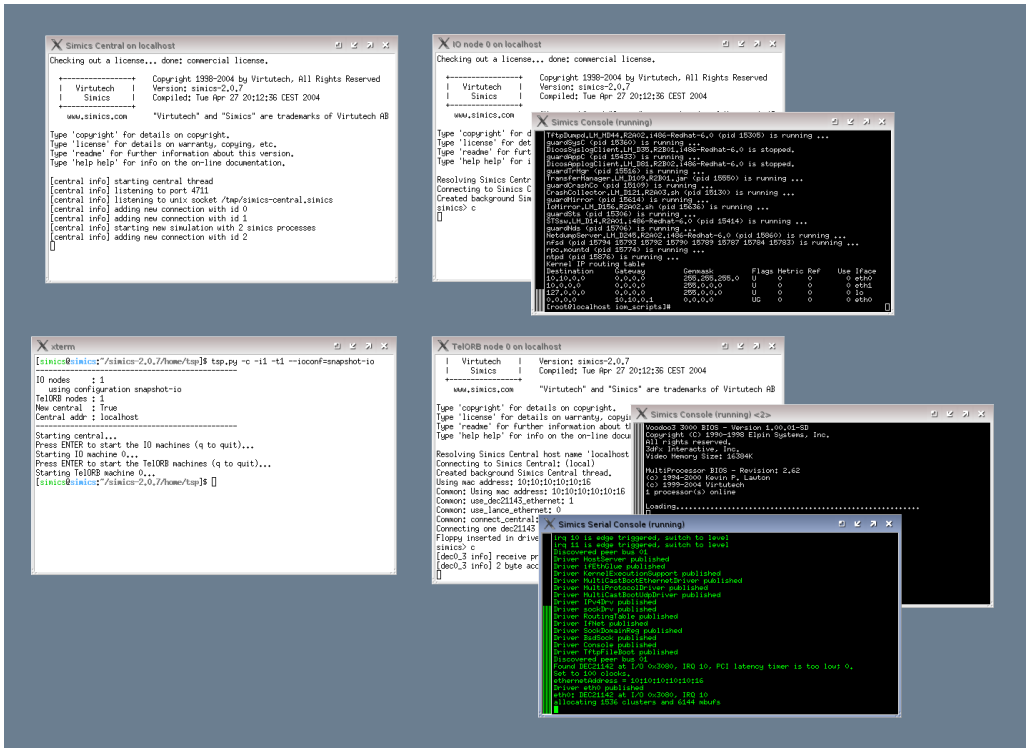

Figure 7.1: TSP cluster in Simics, consists of one IO machine (upper right window), one TelORB machine with its serial console also visible (lower right window), and a Simics Central instance (upper left window). The cluster has been started from the lower left console window.

The features implemented in `tsp.py` are the ones that have been most useful during the experiment. Since the most important thing, from the experiments point of view, is to be able to configure the cluster and start it in a easy way, those features have been higher prioritized in the development environment. The currently implemented features of the development environment prototype is:

Size configuration The size of the cluster in terms of number of IO-machines and number of TelORB-machines can be configured. To achieve this, the two flags `--io-nodes` *n* and `--tp-nodes` *n* are used together with the number (*n*) of each sort of machine and passed to `tsp.py`.

Start from snapshot The IO-machines and TelORB-machines can be configured to start from a previous snapshot, or checkpoint instead of doing the whole boot-process again. The two flags `--ioconf [name]` and `--tpconf [name]` sets the configuration ([name]) to start from.

Simics Central Simics Central, which is used to interconnect the simulated machines though a simulated Ethernet connection, can be started with the script if passing the flag `--new-central`

to `tsp.py`. The script holds static configuration of the MAC-addresses of both the IO- and the TelORB-machines.

Pretend configuration It is possible to "pretend" to start the cluster with the flag `--no-act`. It does everything except start the Simics-instances. This is useful for debugging the development environment itself.

Skeleton for distribution Even though it was never tested, a skeleton for distributing the Simics execution over different hosts with SSH was developed. `tsp.py` holds static information about the hosts to connect to and this feature can be invoked by passing the flag `--distribute` to the script. This is described in more detail in Section 7.2.4.

Cluster execution When `tsp.py` has interpreted all the flags, and configured all options, it starts the Simics instances for the IO-machines and waits for the user to signal. When the user gives the signal (by pressing any key but 'q' in the console where the script was invoked) the Simics instances of the TelORB-machines are started.

7.2 Potential Features to Implement

During the experiment, the development environment have been lower prioritized compared with the main objective, to get TSP to work in Simics. Thus, all the intended features have not been implemented in the development environment. At a meeting [9] with the TSP development section at Ericsson, some ideas of what to implement in the development environment were gathered. Those ideas together with some more features and ideas are described in this section.

7.2.1 Auto Start of TelORB Machines

The script `tsp.py` handles the execution of the IO-machines and TelORB-machines in the cluster as described earlier. As it is currently implemented, `tsp.py` starts the IO-machines first, and waits for user interaction before the TelORB-machines are started. The IO-machines has to be booted and running before the TelORB-machines are booted, otherwise it will not work.

To automate this, so called *magic instructions* [32] could be used. A magic instruction is simply a special instruction encoding which is included in a piece of software on the target host. When this special instruction is carried out, an event in Simics is triggered which scripts can interact upon. The magic instruction for the x86 architecture (the instructions varies between architectures) is `xchg %bx, %bx`. It can be used in any programming language that supports inline assembly.

To have magic instructions help in the automation of starting TelORB-machines, a small application that carries out the magic instruction can be executed on the IO-machines when it finishes the booting process. Thus letting the script `tsp.py` start the TelORB-machines.

Another way of doing this could be to use events triggered by text output from the IO-machines. The magic instruction approach is a more dynamic way of handling the automation though.

7.2.2 Debugging with the GNU Debugger

It is generally difficult to debug parallel programs because they may exhibit non-deterministic behaviour, where the order of execution can change between executions. This can be a problem in certain cases where there are order-dependant calculation. However, in a simulated environment such as Simics, where the behaviour is completely deterministic, debugging becomes easier and more useful.

Through an extension module in Simics (gdb-remote) it is possible to use GNU Debugger [1] (henceforth gdb) running on the host machine to debug applications running in the simulated environment. For this feature to work, gdb has to be compiled with support for the architecture of the simulated target machine.

To use gdb, the target machine has to be executed to the point where debugging is needed. Then the module gdb-remote is loaded into Simics which makes it possible to connect with gdb over TCP/IP to the simulated machine from the host machine.

This feature of Simics could be used to debug both the Dicos kernel, and the applications running on the TSP-cluster. When gdb is connected to the cluster, it gets full control over the execution. Together with an IDE that supports gdb, it is then possible to look at the source code and see where the execution currently is.

7.2.3 Work Outside Simics

In order to decrease the time required to move an application from a working real TSP cluster into Simics (described in greater level in chapter 5), one way is to do as much as possible on the real, physical computers, or via mounted disk images on real computers, before Simics is started. For instance, compiling all TSP applications on a real computer with the TSP development environment installed, and then moving them into Simics, dramatically decreases the time to test and debug programs which have to be re-built before they are executed in the simulated environment.

Since TSP uses its own compiler and libraries to build applications, and not the ones in the operating system, it should be possible to build applications outside of the simulated environment and inject them, either via the disc-image, or the *host* [32] filesystem in Simics.

7.2.4 Load Balancing via Distribution

Full system simulations are inherently slow. One way to make the simulation execute faster is to distribute the work load Simics generates on multiple computers. Simics makes no difference if one of the simulated machines runs on a separate computer, as long as it is connected to the same network as the rest and is told where to find the running instance of Simics Central. Simulations of clusters are particularly suitable to distribution because each simulated machine is a separately executing computing unit, that can be simulated independently of any other machine.

In the development environment created during the experiment, we built the python script `tsp.py` to include support for starting each Simics simulation on a separate computer using SSH as the protocol for communication. If the IP-addresses of available computers are specified in the script it can be instructed to start the simulations on separate computers simply by providing a parameter flag when the script is started.

To make sure the participating computers in this Simics cluster all share the same disc images, settings and scripts, a master server holding the files necessary could be used. The files could then be accessed from nodes in the Simics cluster via the NFS [7] protocol. Another prerequisite of this type of setup, is that all the nodes in the cluster must have their own local copies of Simics together with a valid license. It is not possible to share the Simics executable via NFS, due to licensing restrictions.

GNU Queue

Another option is to use a load balancing tool such as GNU Queue [6] to distribute the Simics load over a range of interconnected computers. This approach introduces a higher level of transparency between the user (who runs Simics) and the distribution of the load, i.e. the user does not know about the distribution in the same way as if it was distributed via SSH. The scripts in the

development environment must be aware of the fact that a load balancing system is used though. This approach is however not implemented in the development environment prototype created during the main experiment (master thesis project phase).

Chapter 8

Discussion and Future Work

Since the main experiment of booting TSP in Simics was not successful, there is obviously more work needed here. The last thing that was attempted was too boot the TelORB machine with a version of Dicos where the *profiler* was disabled. Unfortunately the results were the same as before. However, judging from what the developers at Ericsson and Virtutech says, there should not be much in the way of booting Dicos in Simics.

A few weeks before this thesis was presented, a discussion between Blekinge Institute of Technology, Ericsson and Virtutech took place. The reason was to make the TSP and Simics installation that were used in this experiment available to the developers at Virtutech, for further debugging. Unfortunately, because of legal issues this could not be done quickly enough to be included in this report, and it is uncertain if any more attempts are being done by their part now.

In case Dicos is finally booted in Simics and the whole TSP cluster is started, there are a few things that would be worthwhile investigating.

8.1 Performance evaluation

It would be very interesting to perform a performance experiment to measure the slowdown of running TSP in Simics compared to on the real hardware. That would give an indication as to how useful Simics may be for the TSP developers, and if further actions need to be taken to bypass time-consuming steps involved when using Simics.

To perform such an experiment, a TSP application could be built and executed in the different environments: on real hardware, in the Simics simulator, and in the existing Vega emulator. The time it takes to execute the application in the three environments would then be measured.

One problem with such an experiment is to create an environment where the variables of the experiment can be controlled. It may not be possible to create simulated target computers in Simics for example, that completely resemble the ordinary TSP computers. However, since full system simulators always experience considerable slowdowns (>25 times slower than real hardware [28]), the small differences may be neglected.

8.2 Checkpoints of TelORB machines

Since a TelORB machine was not successfully booted in Simics, it has not been determined if it is possible to create a checkpoint of each such machine once they have been booted in Simics. There are no obvious problems that should prevent this that can be seen at this point. If it is possible, it would definitely save a lot of time when starting the TSP cluster in Simics.

Chapter 9

Conclusion

This thesis has described the operating system TSP and some of its characteristics. It has also shown what full system simulation is, and the possibilities it introduces for operating system developers. The full system simulator Simics, together with a few other similar products have been described, and it has been shown why Simics in particular is suitable for simulating TSP clusters.

The attempt to simulate a TSP cluster in Simics was unsuccessful because Dicos, the operating system kernel used by TSP, was unable to boot inside Simics. Several problems have been encountered during the way that held up the testing for periods at the time. These problems were solved quickly by the developers of Simics at Virtutech.

All the problems that were encountered during the experiment have been described, as well as the factors that played a role in the outcome. These factors include things such as the consequence of full system simulators being so slow, what effect the size of the disk images have, importance of installing a working cluster separately, importance of knowledge and contact with developers, and having system documentation available.

A prototype of a development environment has also been created, particularly a script to simplify the process of starting up a large cluster of simulated computers. Possible extensions of this environment have been discussed, including automating the cluster startup with something called magic instructions in Simics, support for preparing TSP applications outside of Simics, and load balancing the workload of Simics over the network.

At this point, there is nothing that tells us TSP can not be booted in Simics eventually. The latest problems, which seem to be APIC (Advanced Programmable Interrupt Controller) related should be possible to bypass with a little work, at least according to the people at Virtutech. Whatever problems might be encountered after that is impossible to speculate about, although the developers at Virtutech and Ericsson are optimistic.

Acknowledgements

We would like to thank everyone who has contributed to the work of this master thesis.

Special thanks to our supervisors at Blekinge Institute of Technology: Håkan Grahn, for help and encouragement with the thesis, and to Simon Kågström, for endless hours of support in the project room.

We would also like to express thanks to the people at Ericsson, most notably Hans Nordebäck, for help and support with TSP, and for inviting us to talk about this work.

Finally, special thanks to the people at Virtutech, most notably Jakob Engblom, for help and support with Simics.

Bibliography

[1] GDB: The GNU Project Debugger. `http://www.gnu.org/software/gdb/` (accessed: 2004-05-19).

[2] Moore's Law. `http://www.intel.com/research/silicon/mooreslaw.htm` (accessed: 2004-05-19).

[3] Object Management Group. `http://www.omg.org/` (accessed: 2004-04-12).

[4] Plex86 x86 Virtual Machine Project. `http://plex86.sourceforge.net` (accessed: 2004-05-22).

[5] QEMU CPU Emulator. `http://fabrice.bellard.free.fr/qemu/` (accessed: 2004-05-22).

[6] Queue - GNU project. `http://www.gnu.org/software/queue/` (accessed: 2004-05-17).

[7] RFC 1094 - NFS: Network File System Protocol Specification. `http://www.faqs.org/rfcs/rfc1094.html` (accessed: 2004-05-17).

[8] RFC 958 - NTP: Network Time Protocol specification. `http://www.faqs.org/rfcs/rfc958.html` (accessed: 2004-05-18).

[9] Meeting between Ericsson, Virtutech, BTH, and Olle Eriksson and Emil Erlandsson. private communication, February 2004.

[10] A. R. Alameldeen, M. M. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood, and D. J. Sorin. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, pages 50–57, February 2003.

[11] L. Albertsson and P. S. Magnusson. Simulation-based Temporal Debugging of Linux. In *Proceedings of the Second Real-Time Linux Workshop*, November 2000.

[12] G. M. Amdahl, G. A. Blaauw, and B. J. F. P. Architecture of the IBM System/360. *IBM Journal of Research & Development*, 2000.

[13] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2):59–68, February 2002.

[14] F. Chu. VMWare Expands Virtual Horizons. *eWeek*, 20(18):48, May 2003.

[15] I. Corporation. *IA-32 Intel Architecture Software Developers Manual*, volume 1–3. Intel Corporation, 2002.

[16] F. L. Drake. Python language reference. `http://www.byte.com/documents/s=8880/byt1062182129207/` (accessed: 2004-06-07), May 2004.

[17] J. Engblom. Full-System Simulation Technology. *extended abstract appearing in proceedings of the European Summer School on Embedded Systems*, 2003.

[18] Ericsson. *TelORB Documentation*. Bundled with the TSP distribution.

[19] Ericsson. Understanding Telecommunications. `http://www.ericsson.com/support/telecom/` (accessed: 2004-04-12).

[20] M. Fayzullin. How To Write a Computer Emulator. `http://people.ac.upc.es/vmoya/docs/HowToMarat.html` (accessed: 2004-04-08).

[21] T. Franklin. Run Windows on your Mac. *Macworld*, 1999.

[22] M. Gibbs. More VMWare Intricacies. *NetworkWorld*, page 32, November 2003.

[23] L. Hennert and A. Larruy. TelORB - The Distributed Communications Operating System. *Ericsson Review*, 1(3):156–167, 1999.

[24] S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behaivor*. PhD thesis, Dpt. of computer science, Stanford University, February 1998.

[25] R. J. Hookway and M. A. Herdeg. DIGITAL FX!32: Combining Emulation and Binary Translation. *Digital Technical Journal*, 9(1), 1997.

[26] K. Lawton, B. Denney, D. Guarneri, V. Ruppert, C. Bothamy, and M. Calabrese. *Bochs User Manual*.

[27] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberga, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, pages 50–58, February 2002.

[28] P. S. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström, and B. Werner. SimICS/sun4m: A Virtual Workstation. In *Proceedings of Usenix Annual Technical Conference*, June 1998.

[29] G. F. Pfister. *In Search of Clusters*. Prentice Hall PTR, second edition, 1997.

[30] M. Rosenblum, S. A. Herrod, E. Witvhel, and A. Gupta. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel & Distributed Technology*, pages 34–43, Winter 1995.

[31] K. Sollins. The TFTP Protocol. RFC 1350, Internet Engineering Task Force, July 1992. `http://www.ietf.org/rfc/rfc1350.txt` (accessed 2004-05-30).

[32] Virtutech. *Simics User Guide*. Bundled together with the Simics distribution.

Appendix A

Abbreviations

Abbreviation	Description
APIC	Advanced Programmable Interrupt Controller
CPU	Central Processing Unit
FTP	File Transfer Protocol
IO	Input/Output
IP	Internet Protocol
MAC	Media Access Control
NUMA	Non-uniform Memory Access
OS	Operating System
SMP	Symmetric Multiprocessor
SSH	Secure SHell
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol, a simple form of the File Transfer Protocol (FTP).
TSP	Telecom Server Platform

Appendix B

Scripts from the Development Environment

This appendix contains Simics files and scripts which have been mentioned in the thesis. Most of these files and scripts were creating during the experiment phase.

B.1 tsp.py

```
#!/usr/bin/env python

import os
from optparse import OptionParser

# Options
parser = OptionParser()

parser.add_option("-i", "--io-nodes", type="int", dest="io_nodes",
                  help="Number of IO nodes to start")
parser.add_option("-t", "--tp-nodes", type="int", dest="telorb_nodes",
                  help="Number of TelORB nodes to start")
parser.add_option("--ioconf", type="string", dest="io_conf",
                  help="Start the io machines on this snapshot configuration.")
parser.add_option("--tpconf", type="string", dest="tp_conf",
                  help="Start the telorb machines on this snapshot configuration.")
parser.add_option("-c", "--new-central", action="store_true",
                  dest="new_central", help="Start a new central instance")
parser.add_option("-a", "--central-addr", type="string",
                  dest="central_addr",
                  help="Address to the simics central, eg localhost or an ip")
parser.add_option("-s", "--no-act", action="store_true", dest="no_act",
                  default=False,
                  help="Do everything except start the Simics instances")
parser.add_option("-d", "--distribute", action="store_true",
                  dest="distribute", default=False,
                  help="Distribute the Simics instances over hosts by ssh")

options, args = parser.parse_args()


# Default options
if options.io_nodes == None:            options.io_nodes = 1
if options.telorb_nodes == None:        options.telorb_nodes = 1
if not options.new_central:             options.new_central = False
if not options.central_addr:            options.central_addr = "localhost"
if options.io_conf == "-":              options.io_conf = "snapshot-io"
if options.tp_conf == "-":              options.tp_conf = "snapshot-tp"


#                   MAC address           Run at ip..
```

```
io_machine     = [["10:10:10:10:10:12",  "localhost"],  # io1
                  ["10:10:10:10:10:14",  "localhost"]]  # io2

telorb_machine = [["10:10:10:10:10:16",  "localhost"],  # telorb1
                   ["10:10:10:10:10:18",  "localhost"],  # telorb2
                   ["10:10:10:10:10:20",  "localhost"],  # ...
                   ["10:10:10:10:10:22",  "localhost"],
                   ["10:10:10:10:10:24",  "localhost"],
                   ["10:10:10:10:10:26",  "localhost"],
                   ["10:10:10:10:10:28",  "localhost"],
                   ["10:10:10:10:10:30",  "localhost"]]

run_central_on = "localhost"
path_to_simics_files = '/home/simics/simics/home/tsp'


def main():
    print_options()
    start_nodes()


def print_options():
    print "-----------------------------------------------"
    print "IO nodes     : " + str(options.io_nodes)
    if options.io_conf: print "   using configuration " + str(options.io_conf)
    print "TelORB nodes : " + str(options.telorb_nodes)
    if options.tp_conf: print "   using configuration " + str(options.tp_conf)
    print "New central  : " + str(options.new_central)
    print "Central addr : " + options.central_addr
    print "-----------------------------------------------"


def start_central():
    extra1  = ""
    extra2  = ""
    if options.distribute:
        extra1 = "ssh -X -t " + run_central_on + " '" + \
                 "cd " + path_to_simics_files + " ; "
        extra2 = "'"

    title = "Simics Central on " + run_central_on

    os.system(' xterm -T "' + title + '" -e /bin/bash -c "' + extra1 +
              'source simics-central -x tsp-central.simics' + extra2 + '" & ')


def start_io_node(io_node):
    extra1  = ""
    extra2  = ""
    simics_io_command = "-x tsp-io.simics"

    if options.distribute:
        extra1 = "ssh -X -t " + io_machine[io_node][1] + " '" + \
                 "cd " + path_to_simics_files + " ; " + \
                 "export ethernet='" + io_machine[io_node][0] + "' ; "
        extra2 = "'"

    if options.io_conf:
        simics_io_command = "-c " + str(options.io_conf)

    title = "IO node " + str(io_node) + " on " + io_machine[io_node][1]

    os.system(' xterm -T "' + title + '" -e /bin/sh -c "' + extra1 +
              ' source simics -central ' + options.central_addr +
              ' ' + simics_io_command + extra2 + '" & ')


def start_telorb_node(telorb_node):
    extra1 = ""
```

```
    extra2 = ""
    simics_tp_command = "-x tsp-tp.simics"

    if options.distribute:
        extra1 = "ssh -X -t " + telorb_machine[telorb_node][1] + " '" + \
                 "cd " + path_to_simics_files + " ; " + \
                 "export ethernet='" + telorb_machine[telorb_node][0] + "' ; "
        extra2 = "'"

    if options.tp_conf:
        simics_tp_command = "-c " + str(options.tp_conf)

    title = "TelORB node " + str(telorb_node) + " on " + telorb_machine[telorb_node][1]

    os.system(' xterm -T "' + title + '" -e /bin/sh -c "' + extra1 +
              ' source simics -central ' + options.central_addr +
              ' ' + simics_tp_command + extra2 + '" & ')



def start_nodes():
    # Check for MAC address declarations
    if len(io_machine) < options.io_nodes:
        print "Too few MAC addresses for IO machines specified!"
        return
    if len(telorb_machine) < options.telorb_nodes:
        print "Too few MAC addresses for TelORB machines specified!"
        return

    # Starting central
    if options.new_central:
        print "Starting central..."
        if not options.no_act:
            start_central()

    if not options.no_act:
        todo = raw_input("Press ENTER to start the IO machines (q to quit)... ")
        if todo == "q":
            return

    # Starting io nodes
    node = 0
    while node < options.io_nodes:
        os.putenv('ethernet', io_machine[node][0])
        print "Starting IO machine " + str(node) + "..."
        if not options.no_act:
            start_io_node(node)
        node = node + 1

    if not options.no_act:
        todo = raw_input("Press ENTER to start the TelORB machines (q to quit)... ")
        if todo == "q":
            return

    # Starting telorb nodes
    node = 0
    while node < options.telorb_nodes:
        os.putenv('ethernet', telorb_machine[node][0])
        print "Starting TelORB machine " + str(node) + "..."
        if not options.no_act:
            start_telorb_node(node)
        node = node + 1

main()
```

B.2 mkdisk.sh

```
#!/bin/sh

EXTRA_SPACE=1073741824 # 1Gb   8388608 # 8MB
BLOCK_SIZE=512
HEADS=16
SECTORS=63
CYL_SIZE=$(($HEADS*$SECTORS))

check_dir()
{
    if [ ! -d $DIR ]; then
echo "$DIR is not a directory"
exit 1
    fi
}

# Make a harddisk image

if [ $# -lt 1 ]; then
    echo "Usage: mkdisk.sh [-s SIZE_MB] DIR"
    exit 1
fi

if [ $1 == "-s" ]; then
    shift;
    DIR_SIZE=$(($1*1024*1024))
    shift;
    DIR=$1
    check_dir $DIR
else
    DIR=$1
    check_dir $1

#    DIR_SIZE=$((‘du -b $DIR | tail -1 | awk ’{print $1}’‘+$EXTRA_SPACE))
    DIR_SIZE=4800000000
fi

IMG_NAME=$DIR.img

CYLINDERS=$(( ( $DIR_SIZE/$BLOCK_SIZE + $CYL_SIZE) / $CYL_SIZE ))
DISK_SIZE=$(( $CYLINDERS * $CYL_SIZE ))

# Create an image
dd if=/dev/zero of=$IMG_NAME bs=$BLOCK_SIZE count=$(($DISK_SIZE))

# Partititon the image
/sbin/sfdisk -C $CYLINDERS -H $HEADS -S $SECTORS -D $IMG_NAME<<EOT
,,83,*,0,1,1


EOT
# Setup a loopback device to the image
/sbin/losetup -d /dev/loop
/sbin/losetup -o $(($SECTORS*512)) /dev/loop1 $IMG_NAME

# Create a file system
/sbin/mke2fs -m 0 /dev/loop1

# Copy data to the directory
rm -rf $IMG_NAME.dir
mkdir $IMG_NAME.dir
mount -t ext2 /dev/loop1 $IMG_NAME.dir
cp -av $DIR/* $IMG_NAME.dir
sync
umount $IMG_NAME.dir
/sbin/losetup -d /dev/loop1
rm -rf $IMG_NAME.dir
```

B.3 mkparttbl.sh

```
#!/bin/sh
# Makes a partition table to be used

if [ $# != 1 ]; then
    echo "Usage: olle.sh IMAGEFILE"
    exit 1
fi

### Configure these values!! ###
BLOCK_SIZE=512   # SECTOR_SIZE?
HEADS=16
SECTORS=63

PART_IMG=$1
PARTTBL_IMG=parttbl-$PART_IMG

PART_SIZE_FILE=‘ls -l $PART_IMG | awk ’{print $5;}’‘

CYL_SIZE=$(( $HEADS * $SECTORS ))
CYLINDERS=$(( ($PART_SIZE_FILE / $BLOCK_SIZE) / $CYL_SIZE + 1)) # +1 to round up
PART_SIZE=$(( $CYLINDERS * $SECTORS * $HEADS * $BLOCK_SIZE ))

echo "Partition size (file): $PART_SIZE_FILE"
echo "Cylinder size: $CYL_SIZE"
echo "Cylinders: $CYLINDERS"
echo "Partition size (calculated): $PART_SIZE"

dd if=/dev/zero of=$PARTTBL_IMG bs=$BLOCK_SIZE count=$SECTORS

# Partititon the image
/sbin/sfdisk -C $CYLINDERS -H $HEADS -S $SECTORS -D $PARTTBL_IMG<<EOT
,,83,*,0,1,1
EOT

PART_OFFSET=$(( $SECTORS * $BLOCK_SIZE ))

PARTTBL_SIZE=‘ls -l $PARTTBL_IMG | awk ’{print $5;}’‘ # maybe calculate?
DISK_SIZE=$(( $PARTTBL_SIZE + $PART_SIZE ))

echo ""
echo "Geometry C/H/S : $CYLINDERS / $HEADS / $SECTORS"
echo "Partition table: $PARTTBL_IMG  (file size: $PARTTBL_SIZE)"
echo "Partition image: $PART_IMG  (size: $PART_SIZE ($PART_SIZE_FILE from file), offset: $PART_OFFSET)"
echo "Total disk size: $DISK_SIZE"
echo ""
```