CPU Management for Multimedia Applications: A case study

Martin T. Šetek

November 1, 2006

Abstract

This paper is submitted as the final work for obtaining the title Cand. Scient. at the Dept. of Informatics at the University of Oslo, Norway.

The paper is a case study of working implementations of soft real-time features for Linux. The problems raised by multimedia applications running alongside normal applications is discussed. A comparison of the architecture of modern Linux versus traditional UNIX is given, and the implications for soft real-time applications are highlighted.

Finally, some contemporary projects that implement various soft real-time features for Linux are compared in a practical experiment involving video, and software to support this activity is developed. A workload is designed, and the use of a sophisticated profiling tool is explained.

Contents

1 Introduction **4**

1.1 Background . . . 4

1.2 Problem description and scope . . . 5

1.3 Objectives . . . 7

1.3.1 Solution requirements (with rationale) . . . 8

1.4 Method . . . 9

2 Resource management **11**

2.1 Why resource management . . . 11

3 Overview of UNIX and Linux **15**

3.1 Introduction . . . 15

3.2 CPU scheduling in UNIX . . . 16

3.2.1 Swapping . . . 18

3.2.2 Timing: ticks . . . 18

3.2.3 The CPU scheduling algorithm . . . 19

3.3 Linux: a UNIX clone . . . 21

3.3.1 Linux and version numbers . . . 23

3.4 Linux 2.4 . . . 23

3.5 Linux 2.6 . . . 26

3.5.1 The scheduler is $O(1)$ . . . 26

3.6 System calls related to scheduling in Linux . . . 26

3.6.1 System calls related to scheduling . . . 26

3.7 Priorities – From the user level . . . 27

3.8 Linux soft-RT . . . 28

3.8.1 Preemptible kernel . . . 29

3.8.2 Interrupts . . . 30

3.8.3 Interrupt responses . . . 31

4 Real-time and Linux **33**
4.1 Introduction 33
4.2 Standard: POSIX.4 – Soft Real-time 33
4.3 Why Linux? 34
4.3.1 Source code availability 34
4.3.2 Multimedia and RT: several projects 34
4.4 About patchsets 34
4.4.1 Patches 36

5 The projects **37**
5.1 Vanilla Linux 37
5.2 Linux with rt-preempt 38
5.2.1 Configuration of rt-preempt 38
5.3 Linux with Class-based Resource Kernel Management 39
5.3.1 Class configuration 39
5.3.2 Configuration of ckrm 40
5.4 Linux with CKRM and rt-preempt 40

6 Test cases – workloads **42**
6.1 Introduction 42
6.1.1 Actions performed by the workload program 42
6.2 Case 1:xine/mplayer with xboard 43
6.3 Case 2: xine/mplayer with two cpuhogs 43
6.4 Case 3: xine/mplayer with three cpuhogs 45

7 Measurements **46**
7.1 Measurement tools 46
7.1.1 OProfile 46
7.1.2 A sample flat profile 48
7.1.3 A sample kernel profile 49
7.2 Choice of tool 50
7.3 Setup of test environment 51
7.3.1 Selection of workload 51

8 Conclusion **53**
8.1 Results of the experiments 53
8.2 Future work 54

A Configuration file for Linux 2.6.18 **56**

B Workload program **94**

List of Figures

3.1 Chart showing the evolution of some descendants of UNIX . . 17
3.2 Process priority queues in UNIX 22
3.3 Linux 1-1 thread scheduling model 25
3.4 Nested interrupt invocations in Linux 2.6 32

7.1 The cpuhog program 52

Chapter 1

Introduction

1.1 Background

Today, many people use their computers for more than just computing. Actually, most people seem to use computers for activities that have very little to do with the term "computing"[1], like surfing the web, reading and sending e-mail, writing documents, playing games, listening to music and watching movies.

Modern desktop computers have reached a level of performance that most of the activities just mentioned can be accomplished without machine resources, such as CPU, memory or hard disk speed and capacity being an issue. Of the activities mentioned, the only one that can make heavy demands on the computer resources is playing some types of games. And these games mostly make demands on the graphic processing capabilities of the machine[2].

There are even computers that are customized for the purpose of playing music and movies, so called media centers[3]. Some media centers can be used as normal computers, while others are more or less dedicated to their multimedia purpose. Most of these systems are connected to some kind of network, usually with Internet access. These machines often don't resemble desktop computers at all, but are often connected to television sets and look like a component of a stereo system.

Some of these computers run Linux, an operating system that is a clone of another operating system called Unix[4]. At the time Unix was designed,

[1]the procedure of calculating

[2]the performance of these games is mostly determined by the GPU on the graphics card

[3]also known as Home Theater PC's

[4]Unix has been called "the most important operating system you may never use."

computers were not on the desktop – the weight would break it. The algorithms that were chosen to implement it reflect the fact that computers were expensive machines that needed to be kept busy. Therefore, the throughput of the system was important. Yet, the system had users that worked on it interactively in a timesharing manner. Users don't like to be kept waiting for anything. So its response time was also important. The result was a system that would perform well for the typical mix of tasks at the time – jobs run interactively and long jobs doing "number crunching"[5].

In recent years, the developers of Linux have begun experimenting and implementing kernel features that reflect the new situation – that many machines in use are not servers. This includes some features that haven't been available before, because they were not in Unix and haven't been standard in Unix's descendants.

One example of this is the adjustable tick rate. On contemporary Linux systems[6], one can configure the rate for the primary timer interrupt before compiling the kernel. There are three settings at the time of writing: $100Hz$, $250Hz$ and $1000Hz$. The slower setting is for servers, because with it the system generates less interrupts. This means that the time that would otherwise be spent on processing them can be used on actual work. This again means that the throughput improves. The fastest setting is for desktop machines, because it shortens the time between a event is noticed in a Interrupt Service Routine (or ISR for short), and the time it is handled in some bottom-half routine, thereby increasing the responsiveness of the system. The third setting is simply a compromise when one wants a little bit of both (reasonably good throughput *and* reasonably good responsiveness).

Another factor that has prompted the inclusion of new features in Linux and some other descendants of Unix is the growing acceptance of the POSIX standards. POSIX stands for "Portable Operating System Interface"[7].

1.2 Problem description and scope

This thesis will attempt to investigate whether recently developed CPU scheduling and process handling technology in Linux can help improve system performance when multimedia applications are involved and there is pressure on system resources.

An aspect of multimedia applications that makes it particularly hard to

[5]performing complex and lengthy numerical calculations

[6]2.6 series

[7]The "X" is silent. Joke aside, the "X" really doesn't stand for anything in this abbreviation according to [Gal95]

measure such things is that the definition of “acceptable performance” is a matter of subjective judgment and expectations.

Take the playback of an audio stream, for example. People have very different opinions on what is acceptable sound quality. Moreover, the opinions on what is acceptable change according to the scenario and expectations that the person has. For example, most people don’t expect the same sound “quality” from a car radio and a home “hi-fi” audio system. Another example can be taken from the domain of computers. To most people, the perception of quality will be different for an audio stream that is listened to over the Internet, and one that is played back on a computer one has physical access to.

To avoid such difficulties, we will simply try to describe what is observed, and leave the difficult judgment of what “acceptable quality” is alone.

To limit the scope of the problem, we will focus our attention on video playback with accompanying sound[8] on a computer system. Assuming that an acceptable level of quality for video playback can be achieved, what strategy should one take to utilize the resources on the computer system? In other words: If the video playback is acceptable, what computer resource allocation strategy gives other applications running alongside it good performance?

One nice aspect of this way of looking at the problem is that it is decidedly simpler to measure the quality of “regular”[9] computer programs than of those dealing with video. Most applications deal with tangible results and their level of performance can be stated in relatively simple terms, like total running time. For a user of a regular computer program, its run-time performance, correctness and ease of use can be taken as reasonable measures of quality.

To somehow measure the performance of video playback, one could of course introduce concepts like the number of dropped frames and then use this as a measure of performance. But in this case it is only indicative of the perceived quality. The real judge of video quality will always have to be a human being.

It is generally understood that multimedia applications benefit from having operating system features that specifically support their activities. Because such applications have specific timing requirements, it is beneficial if the machine resources are allocated in such a way that takes those requirements into account. A model of how this can be done is given in [SN95], where a source-to-sink model is presented. The idea is that one should be

[8]when we say video playback, we will assume that it is accompanied by sound as well from now on

[9]for lack of a better term

able to control the allocation of all the resources that are used by the multimedia application from the source (typically disk storage or video and audio capture equipment) through memory and networks to the sink (typically the part of the application seen by the user). This control should typically manifest itself as some sort of upper bound on the latency experienced when accessing the resource.

Unfortunately, systems that let the application exercise this kind of control are often rather specialized to specific problem domains, or require the application to be specifically written for the specialized system.

To further limit the scope of the problem, this thesis will study the allocation of one specific computer resource in the context just described: the Central Processing Unit (CPU). The reasons for choosing the CPU is primarily because this a resource where some solutions to the resource allocation problem already exist under Linux. Another reason is that the control over this resource can be controlled[10] *outside* the application that requires the allocation. This is very important, because it means that applications do not have to be modified or rewritten to take advantage of this feature. This is somewhat analogous to applications not being explicitly written to support multitasking as opposed to writing applications using co-operative multitasking[11].

1.3 Objectives

This paper aims to evaluate solutions[12] to the problem of running multimedia applications on Linux when the system is running other applications alongside the multimedia application. The problem has two aspects:

- The OS should allow allocation of enough CPU to the multimedia application for it to perform adequately.
- The OS should give the rest of the resources to the other applications.

The objective is to *evaluate* possible solutions. For a solution to be a candidate for evaluation, the requirements mentioned in 1.3.1 must be met, in addition to the obvious requirement of somehow addressing the issues discussed in

[10]to some extent

[11]Which was quite common in the past. All programs written for MacOS before MacOS X and Windows before Windows 95 had to use this technique

[12]sometimes referred to as projects

1.3.1 Solution requirements (with rationale)

1. The solution must be runnable

   It is very hard to compare designs when some of them don't have a runnable implementation. Many systems that seem good on paper are never realized, and of those that are, few make it beyond a "proof of concept" implementation. While systems that lack an implementation can be interesting from a theoretical point of view, they will not be considered here. Simply put, the solution must have code that can be compiled and run, and this step must not involve unreasonable effort[13].

2. The solution must work in an Linux 2.6 environment

   This is a further restriction on the "must be runnable" requirement above. Of all the requirements, this is perhaps the most controversial one. The reason for it that there is a lot of code that could be a candidate for evaluation as given in the objectives (see 1.3), but that is based on some[14] version of the Linux 2.4 kernel series.

   Running an 2.4 kernel today represents some practical difficulties. Since the 2.4 series is no longer actively developed, most Linux distributions have abandoned it in favor of the more modern 2.6 series[15]. It is not trivial to run a 2.4 kernel on a system that has been designed for the 2.6 series. More important is the fact that since the 2.4 kernels are no longer being developed, support for newer hardware is progressively getting worse. New device drivers are sometimes only written for the 2.6 series (but then sometimes backported to the 2.4 series). This can make it problematic or inconvenient to run these kernels on machines with new hardware.

   In any case, basing a solution on a kernel that isn't being actively developed (because the developers have moved on) seems like a bad idea.

3. The solution must not require substantial reprogramming of applications

   This requirement is somewhat fuzzy, but it basically means that one should be able to take existing and widely used applications and run

[13]Sitting down and writing a new kernel or doing substantial reprogramming of an existing one is not considered reasonable

[14]often quite old

[15]To the author's knowledge, only Slackware still use the 2.4 kernel

them with the allocations required. If it is necessary to modify the application to make use of the CPU allocation feature, then those modifications should be minor and obvious.

4. The solution must be freely available

   This requirement is reasonable in the context of a thesis such as this one. It might not be in another context.

5. The solution should be actively maintained

   This is not an actual requirement[16], but solutions that are viable and useful to a large enough group of people will be maintained by someone. An actively maintained solution is attractive in the sense that it will typically be updated to match the developments of the rest of Linux. If one wants to build other software on top of such a solution, then this is very important.

   Projects that are serious about solving the kind of problems that are presented in this paper will often have ambitions about becoming a part of "mainline"[17] Linux. This can happen when the code has reached a level of maturity that it is acceptable to the Linux kernel developers and that it addresses problems that are of interest to a big enough group of its users. Getting to be a part of the Linux "mainline" kernel is a big bonus, because it means that whenever someone make a bugfix or changes a part of the kernel that affects the project, those people will make sure that the affected code continues to work. Also, it takes some effort from the maintainers of a project outside the mainline kernel to "chase kernels".

1.4 Method

In order to compare various projects for the particular case that is tested in this paper, some method of testing must be established. As already mentioned in 1.3.1, the various projects will be run in a particular test environment. In [Jai91], there is a description of how one should do evaluations in general, and how to perform evaluations like this one. The method of evaluation in this case is one of measurement, in the terminology used in [Jai91].

To perform this measurement, we will be using a monitoring system called Oprofile. This system will be described further in 7.1.1. Monitoring systems

[16]since it says should, not must

[17]This is what one often calls the standard Linux kernel currently being developed

work by letting the system to be studied run normally, and while it is running, collect information about its behavior.

In [Jai91], a distinction is made between:

Hardware monitors

> This is any equipment that is physically connected to the system, and can consist of probes, counters, timers, etc. and sometimes also storage for the results. It can often record data about events or take samples at regular intervals.

Firmware monitors

> Firmware monitors modify the processor microcode to perform monitoring functions.

Software monitors

> Software monitors typically use trap instructions of some kind to divert execution at points of interest in the code to a trap-handler. In that handler, data is collected.
>
> They can also use trace-mode, sometimes called debug-mode, because the same mechanism is often used by debuggers to single-step through code. This is a processor mode where the processor generates a exception after each instruction. The monitor records data in the handler for the exception. Since such modes make the intervals between instructions extremely long, this is only useful for answering some type of questions. Examples are "how many time is a specific point reached" and "what sequence of instructions is most common". Note that no useful timing can be performed in such a mode.
>
> Another mode often used by software monitors is to use an operating system controlled timer interrupt handler to call it at regular intervals in which it samples data.

Hybrid

> A hybrid monitor is a monitor that uses a combination of hardware and/or firmware and software to accomplish its goals.

Oprofile fits into the classification of a hybrid system, since it uses hardware features to trigger on events, but then uses software in the handling of the event itself.

Chapter 2

Resource management

2.1 Why resource management

Since Internet connectivity has become commonplace, a lot of software has become dependent on the computer running for long periods of time online. Many computers are set up to perform chores at odd hours (e.g. indexing files or downloading corrections for installed software), under the assumption that the user(s) are not using the computer at the time. As a consequence of this, many computers are not switched off even if not in use.

Therefore, there is a situation today where there exist a lot of relatively powerful computers that are:

- Always on.
- Connected to some network, usually with Internet access.
- Mostly idle.

For the sake of brevity, the following definition[1] of "job" will be used from now on:

> All the activities involved in completing any project on a computer from start to finish. A job may involve several processes and several programs.[2]

In this definition, "project" is to be taken as something the computer must finish *without* human involvement. So developing software is not a "job" in this sense.

[1]taken from the foldoc online dictionary

[2]this definition is useful because it focuses on work as users perceive it, not how it is organized by software, as e.g. one or more processes, threads, etc.

People have for some time realized that these unused resources can be tapped into and used to run jobs that are suited for distributed computing and where additional resources are always needed to cut down the running time. Examples of such projects are BOINC[3] that rely on volunteers to install their software on their computers and thereby donating resources to their projects, and condor[4], which is for sites which have many idle workstations that they own.

All of these networked machines typically act as clients in a client-server relationship with some server machines. These servers will usually service many clients simultaneously and sometimes run several services on the same host. It is also common that the server runs one service (e.g. a web service), but acts as a server for several distinct user groups (e.g. a multi-homed web server).

Increasingly, people are also running peer-to-peer[5] software on their computers. These programs form ad-hoc networks where each node simultaneously act as "client" and "server" in relationship to other nodes. All nodes in the network contribute bandwidth, disk space and machine resources to the "system". Often, such networks will have redundancy as a inherent trait, and there is no single point of failure. P2P networks are typically used for file sharing.

In light of this situation, some users will have the need to exercise control over the use of the machine resources in two different ways:

Reserve resources for some jobs

> On servers, there are often many jobs running simultaneously that all compete for the same resources. However, some jobs might be deemed more important than others, so that if resources are scarce, the most important job should be prioritized at the expense of other jobs. The decision on which job is most important isn't necessarily technical[6]. Therefore, this decision must often be made by human beings.
>
> In the case of workstations, the user may be compelled to do similar things. For example, a user may require the result of some job as soon as possible, and therefore wish to arrange things so that this particular job is preferred *without* having to stop other jobs that may already be running.

[3]http://boinc.berkeley.edu

[4]http://www.cs.wisc.edu/condor

[5]often written as P2P

[6]In a commercial setting, jobs belonging to the customer that pays the highest fee might be considered most important

Another likely scenario is that some job may run without difficulty when the machine is lightly loaded[7], but may experience problems when the computer resources are shared with other jobs. An example of this is the playback of video. Under normal circumstances on a lightly loaded workstation, this usually works quite well. However, if additional jobs are running simultaneously playback becomes "choppy" with intermittent and abrupt pauses. In such situations, the user will want to assign enough resources to the video playback job so that it can run smoothly.

Limit the resources consumed by some jobs

When a server processes requests from clients, it often runs some job on the behalf of the client. For example, a HTTP[8] request will often require processing by several processes. Typically a HTTP server will parse the request and then hand it off to a program that will process the request and use a connection to a DBMS[9] to retrieve or store data. The whole job usually ends with a response from the HTTP server. Faced with the proposition of serving many such requests at once, administrators might want to limit the resources given to some jobs. Again, which job(s) should be limited is a decision that may require the judgment of a human.

At first, it might seem that limiting the access to resources and reserving resources are two sides of the same coin. This is however not the case. Reserving resources implies some sort of lower bound or minimum level of access to machine resources, while limiting the access to resources implies some sort of upper bound or maximum level of access to machine resources. In fact, it is perfectly reasonable to both limit *and* reserve resources for a particular job.

In applications where multimedia data is to be processed in complicated ways and where the system doing the processing is distributed, the need for allocating sufficient resources for processing on each node of the distributed system arises. Also, in processing of multimedia data, (especially video), there are timing constraints that don't work well with the scheduling of processes on traditional operating systems. The most commonly used operating systems today have schedulers that try to maximize *throughput* – the number of processes run per unit of time and strive for *fairness* so that no process

[7]the concept of system load will be discussed later
[8]Hypertext Transfer Protocol
[9]Database Management System

can monopolize the processor (or processors in a multiprocessor or multicore system) at the expense of other processes.

The problem of scheduling processes with strict timing constraints is often solved by using hard real-time scheduling algorithms as described in [LL73]. Systems that employ such algorithms are typically used when the problem is such that all deadlines must be met and that *any failure* to meet such a deadline is a *catastrophic event*, and the system cannot continue to run and must employ emergency shutdown procedures if necessary. To meet the deadlines of the processes, the scheduler has to be told about the deadlines, there are constraints on the possible interdependencies of the running processes, and system resources are often under-utilized because the system must allocate them with regard to worst-case utilization. Also, there are few opportunities (if any) of running other processes once the system is running.

Clearly, in most problems involving the processing of multimedia data, an occasional deadline miss isn't catastrophic. In fact, it might not even be noticeable to the user of the system. The use of hard real-time systems for such problems would *work*, but the computer resources would be very under-utilized. The system would also most likely have to be dedicated to running only that job, and the user would have to reconfigure the entire system if some deadline should change.

To accommodate problems where there are timing constraints that need to be met most of the time, but where an occasional deadline miss isn't catastrophic, some operating systems have support for so called soft real-time scheduling. There doesn't seem to be any consensus to what exactly soft real-time scheduling is, other than that it doesn't give any guarantees as to deadlines *always* being met. They also favor some processes (the ones with the timing constraints) over other processes.

The benefits of using soft real-time operating systems are many. A node can be set up with some processes scheduled for soft real-time constraints. While these processes are running, it can run other processes that don't have any particular timing constraints[10]. Since this setup is unintrusive[11] from the point of view of other processes running on that node, a system can be distributed in a network where that network isn't dedicated to that particular system. Most soft real-time systems also have the possibility of reconfiguring the soft real-time scheduling dynamically[12].

[10]The majority of programs are of this type

[11]Except that it consumes resources on the node

[12]e.g. by adjusting the priority of the process

Chapter 3

Overview of UNIX and Linux

3.1 Introduction

About three decades ago, powerful computer systems were rather expensive machines that were mostly found in government institutions and industrial research facilities. The operating systems for these systems were written with the understanding that the computing resources were scarce and that the resources would have to be shared among the different people using them. Therefore, it made sense to share the powerful and expensive machines among many users in so called timesharing [1] systems.

During the late 1980's and early 1990's, workstation computers[2] and later personal computers reached a level of performance and low cost that made it feasible to let each user have a workstation at his exclusive disposal while working a less resource-consuming work, while still having more powerful machines available for more resource-intensive tasks and for central administration.

This situation has not really changed that much in the last 10 years. It is true that there are now a lot of wireless networks available[3], and people have very powerful laptop PC's at their disposal. But the situation is still that one has a relatively powerful PC at ones disposal, and that one is (sometimes) connected to more powerful machines via some network.

In academia, one of the most popular operating system that is in use today is UNIX, or to be more precise, one of several UNIX-like system. The UNIX system was developed in 1969 by Dennis Ritchie, Ken Thompson, Doug

[1]the difference between timesharing and multitasking is that timesharing is multitasking for multiple users

[2]a computer designed to be used by one person at a time with good processing power and high performance graphics

[3]which creates some new challenges

McIlroy and Rudd Canaday. It became very popular in universities and some companies, and was extended and changed in many ways by many people and businesses, until there were several different variants available. Some of these variants are still in use and are popular today[4]. More information about the history of these systems can be found at [MBKQ96] and `http://en.wikipedia.org/wiki/UNIX`.

Despite the trend of PC's instead of time-shared servers, when UNIX runs on a PC it is still in many ways in "timesharing mode". This is partly due to the fact that hardware improves much more rapidly than software, but mostly due to the fact that most users don't mind running a well designed "server" OS on their PCs.

The situation becomes somewhat different when users want to run multimedia jobs on their PC's. But to understand the kind of changes that are needed for multimedia work, it is necessary to know what UNIX-like systems traditionally have done with regard to allocating resources to jobs. This chapter will investigate the strategies taken towards CPU scheduling in UNIX and some popular descendants (the BSD's) and the UNIX clone Linux. Figure 3.1[5], shows the history of some of the descendants of Unix. A more detailed chart can be found in [MBKQ96].

The UNIX CPU scheduler is also interesting because it lays the foundation for many of the concepts and features found in practically all modern UNIX-like operating systems. Also, it is interesting to note that most of the problems and solutions that are discussed in [Bac90] are still relevant today.

3.2 CPU scheduling in UNIX

There isn't really any standard that defines CPU scheduling under UNIX, but the description in [Bac90] comes close. Also, the description in [Lio96] gives many useful insights into how the original UNIX operating system behaved and was designed, even though it describes an older version (6th edition) of UNIX than [Bac90] does. A very high-level and brief description is also given in [Tho78].

From now on, the scheduler as described in [Bac90] will be referred to simply as "the UNIX scheduler", while the CPU scheduler used in Linux 2.6 will be called "the Linux scheduler". Important differences and similarities between the two schedulers will be noted where appropriate.

[4]One common commercial variant is Solaris by Sun Microsystems. Another one is MacOS X

[5]taken from http://en.wikipedia.org/wiki/Image:Unix.svg

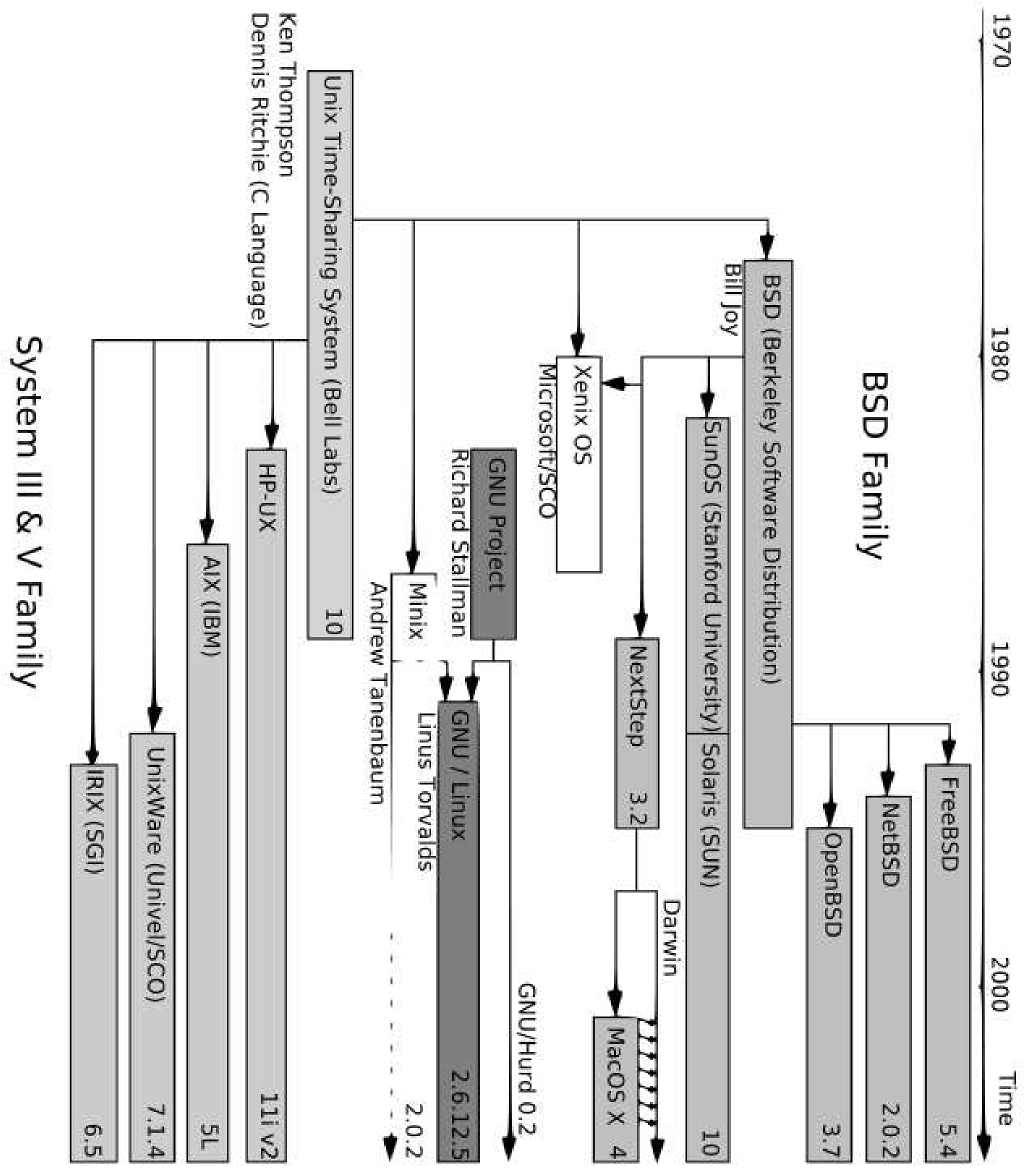

Figure 3.1: Chart showing the evolution of some descendants of UNIX

The choice of memory management for processes affects the kind of states a process can be in, and thus affect the decisions made by the CPU scheduler. UNIX used swapping as its memory management technique for processes at first, but later most implementations switched over to demand paging for their process memory management[6]. This is the memory management technique that most of today's machines support and that the various descendants of UNIX use. It is generally seen as being superior to swapping. One particular advantage is that it makes it possible to run processes that have a memory image that is larger that the physical memory available.

3.2.1 Swapping

When UNIX was invented, machines had very limited resources compared to todays computers. Since memory was a very scarce resource[7], the code and data for a typical process would consume a significant portion of memory (on the PDP-11[8], a process could take up to 64KB).

UNIX solved the problem of multitasking on such computers by loading entire processes into memory as they became runnable and sharing the CPU between these processes. When there were more runnable processes than there was available memory (a normal situation back then), after running the in-memory processes for a while, UNIX would choose a process to throw out, then write the memory image[9] of that process to a portion of secondary storage set aside for this purpose (the so-called swap space), and use the now free memory to load a new process into memory. This technique is appropriate when memory is scarce, and is still used in such situations by UNIX-like systems (see [MBKQ96]).

The process of swapping is very expensive, since it has to work a disk speeds. Therefore, whether a process is swapped in (in memory) or not is an important factor that the UNIX scheduler takes into account when choosing which process to run next. Swapped out processes will not be chosen, since they cannot run from swap space.

3.2.2 Timing: ticks

A very important mechanism that interacts with the way processes are scheduled in UNIX is a hardware timer generating interrupts at a constant rate.

[6]The Berkeley Software Distribution version 4.0 was the first major implementation of UNIX to use demand paging instead of swapping

[7]memories on these computers was measured in kilobytes

[8]one of the first machines UNIX ran on

[9]this is a bit of a oversimplification, but sufficiently accurate in this context

Whenever the timer interrupt occurs, the CPU goes into kernel mode and after doing some time bookkeeping, follows the step describes in "After an interrupt has been handled" in 3.2.3. This timer interrupt is always generated and is what makes the system "preempt processes"[10]. The exact rate of the interrupt can vary, but a rate of about $50Hz$ or $60Hz$ was common[11]. The clock interrupt has the highest priority of all interrupts in the system. Each occurrence of the timer interrupt is called a "clock tick" or simply a "tick".

3.2.3 The CPU scheduling algorithm

The UNIX scheduler is a *round robin scheduler with multilevel feedback*, which basically means that it schedules processes by repeating these steps[Bac90]:

1. Pick a process to run from one of several priority queues.
2. Transfer control to it.
3. Preempt the process in one of the cases mentioned in 3.2.3.
4. Feed the preempted process back into one of the priority queues (thus rescheduling the process).

The UNIX scheduler determines which process is picked to run next by assigning a priority to each process and choosing the one with the highest priority. Processes which have the same priority are scheduled in round-robin order, so that the one that has been waiting the longest for the CPU gets chosen first (to avoid starvation). If no process is ready to run the system puts the machine in a "idle state" (how this happens is of course hardware-dependent), and waits for the next tick to wake it up (unless a different kind of interrupt wakes it up first).

The decision about which process should get control of the CPU is taken whenever a process is about to leave kernel mode and is going to continue execution in user mode. Before control is transferred to user-space code, a rescheduling flag (called "runrun" in [Lio96]) is checked. If it is set, then all the "ready to run" processes are examined and the one with the highest priority is given control of the CPU. Specifically, this happens in one of the following situations:

[10]forcibly taking away CPU control from a process

[11]According to [Lio96], this choice was dependent on the power supply

After an interrupt has been handled

When some interrupt has happened, control is transferred to the next highest priority process[12]. The previously running process is still in a "ready to run" state.

The currently running process sleeps

The currently running process needs to wait for some resource to become available, for some slow operation to complete or for some event to occur. This happens when a process makes a system call. The process changes state from "ready to run" to one of the sleeping states. Control is transferred to the next highest priority process.

A higher priority process is ready to run

A clock tick interrupt has occurred and the currently running process is ready to resume execution in user mode. During this time, if a process with a higher priority is ready to run, control will be transferred to it. The process that had control stays in a "ready to run" state. According to [Lio96], this is done by checking a flag before resuming execution in user mode. If the flag ("runrun") is set, then the code to find the next highest priority process is executed. According to [Bac90], this is done in the interest of fairness.

When the currently running process exits

The currently running process has decided that it is done and has called *exit*(2).

In this way, a process will always be ready to run whenever it can make any progress, and thus be a candidate for getting control of the CPU. But if there are higher priority processes that are ready to run, they will *always* get control of the CPU instead. Therefore, the priority of each process needs to be changed over time in order to avoid starvation and ensure some degree of fairness. This is done once every second (or after about 50 or 60 ticks) as shown in equation 3.4.

Notice that processes are not explicitly given CPU time, but that they are given execution time implicitly by the continuous adjustment of priorities. At every tick, the kernel will increment a per-process value as shown in equation 3.1 (in [Lio96], it is called p_cpu and lives in struct proc). In this way, the kernel counts the number of ticks "used" by a particular process.

[12]there can be more than one – it could also be the same one; in that case it would be given back control of the CPU

$$cpu\_usage_{tick} = cpu\_usage_{tick-1} + 1 \tag{3.1}$$

Once every second (which will from now be called the "period"[13]), cpu_usage for *all* processes is updated as shown in equation 3.2. This rewards inactive processes, because they will have their priorities raised in every period. This also avoids starvation, because equation 3.2 gives rise to a geometric sequence (see 3.3) that exponentially decays toward 0. So a "starving" process with $nice = 0$[14], will eventually be given $priority = base\_priority$ (user-level 0), and since processes at equal priority are run in a round-robin fashion, it is guaranteed to eventually take control of the CPU.

$$cpu\_usage_{period} = \frac{cpu\_usage_{period-1}}{2} \tag{3.2}$$

$$\left\{\frac{cpu\_usage}{2^{period}}\right\} \tag{3.3}$$

After cpu_usage has been decayed, the priorities of all processes that are ready to run are recalculated as shown in equation 3.4. This has the effect that processes that use many ticks have their priority lowered.

$$priority = min(127, (cpu\_usage_{period} + base\_priority + nice)) \tag{3.4}$$

The base_priority of equation 3.4 is an offset that ensures that the priority of a process never goes below "user level 0", or above the thick line separating user level and kernel level priorities in figure 3.2.3.

Figure 3.2.3 illustrates the priority queues in UNIX rather well[15]. The processes in the queues above the thick line are assigned priorities statically. They came to be in one of these queues because they needed to sleep while in kernel mode.

3.3 Linux: a UNIX clone

Linux is a clone of UNIX created[16] by Linus Torvalds and developed with substantial help from others. While it is not a descendant of UNIX, its design *is* taken from UNIX. It is being actively developed today, and runs on many different machines. It is the system that will be studied in this paper.

[13]The authors own terminology

[14]processes with $nice > 0$ can (in theory) starve forever

[15]It is redrawn from a figure in [Bac90]

[16]the first version appeared in 1991

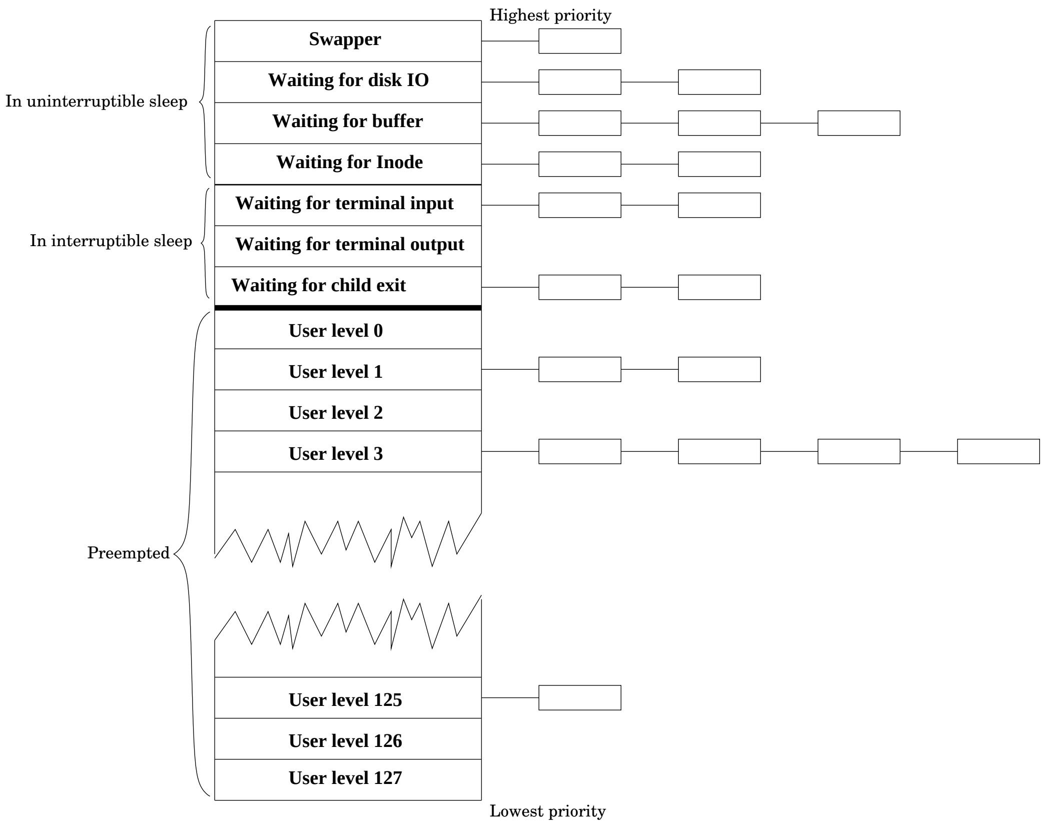

Figure 3.2: Process priority queues in UNIX

3.3.1 Linux and version numbers

Presently, the version numbers for Linux kernels follow this pattern: $W.X.Y.Z$. W is the kernel version, X is the patchlevel (also called the major revision number) , Y is the sublevel (also called the minor revision number). The kernel version number changes *very* rarely. The minor revision number is changed when new drivers or features are implemented. The extraversion (Z), is incremented when the nature of the changes are bugfixes or security patches.

Sometimes the extraversion has a string of the form "-string" appended to it. These letters can have various meanings, but usually identify some independently developed branch[17] of the kernel source. These branches are often made to evaluate experimental features or test new code. Sometimes, code from a branch is accepted into the kernel that Linus Torvalds maintains, and thus ends up being part of the "official" kernel.

Some Linux distributions maintain the kernel they use in their distributions – effectively making their own branch. Therefore, the kernels that Linus Torvalds releases are often referred to as "vanilla" kernels. Sometimes, the kernels released by Torvalds are also called the "mainline".

It is customary to refer to a specific series with the kernel version and major revision number only, as in "$W.X$".

At this time, the 2.6 series of the kernel is released in a couple of versions:

A stable release

> This is a version of the kernel that is tested and deemed stable enough to be of interest to most users. It has a version number as described above

A prepatch release

> This is a set of kernel patches that are being actively tested. Some of these releases have the string "-rcX" (where X is a number) appended to the extraversion. This is a so-called "release candidate", which means that if it is turns out to be in good condition, it will become the next stable release.

3.4 Linux 2.4

The Linux 2.4 is a series that started in January 2001, and is being used to this day. It is mature and can run on a lot of different hardware. It is still

[17]also called a fork

being maintained, but no longer developed. Only bug and security fixes are added. The Linux 2.4 kernel has a number of notable features[18] that are not found in UNIX, and are not commonly found in UNIX-like systems:

Symmetrical MultiProcessor support (SMP)

> The kernel can make use of and run threads on all the CPU's in a Symmetrical MultiProcessor system[19].

Loadable kernel modules

> The kernel can load new code into the address space of the kernel while running. This code will run in kernel mode, and can call on other kernel code, just as if it was compiled into the kernel itself. In some cases, it is also possible for the kernel to unload previously loaded code. The loading and unloading of modules can be done by "root" via userspace tools.

kernel support for user threads

> Threads in a user application are schedulable entities i a 1-1 (1 kernel entity per process thread) scheduling model. This is superior[20] to systems that only offer threads as user-space libraries, because in a 1-1 system, a thread can perform a blocking system call without affecting the progress of other threads in the same process.
>
> The Linux kernel calls all schedulable entities "tasks". A tasks can have an address space for itself - such tasks is a "process". Tasks can also share an address space, in which case they are called "threads". Figure 3.4 illustrates how this works for a single-CPU system. In this case, process 1 has two threads, while process 2 has only one thread. To the kernel, it does not usually matter if a task plays the role of one of several threads in userspace or if it plays the role of the lone "thread" of a traditional process.

Some of these features affect the way the CPU scheduler works. Support for SMP has a profound effect on the kernel code and the synchronization primitives used. Thread support in the kernel also affects many design decisions[21].

[18]Many of the features mentioned were available in Linux 2.2 as well

[19]basically, a machine with several identical CPU's that share the bus

[20]Some people claim that it is also better than a M-N model. See http://people.redhat.com/drepper/glibcthreads.html for an opinion on this subject

[21]For example, should each thread have its own unique ID?

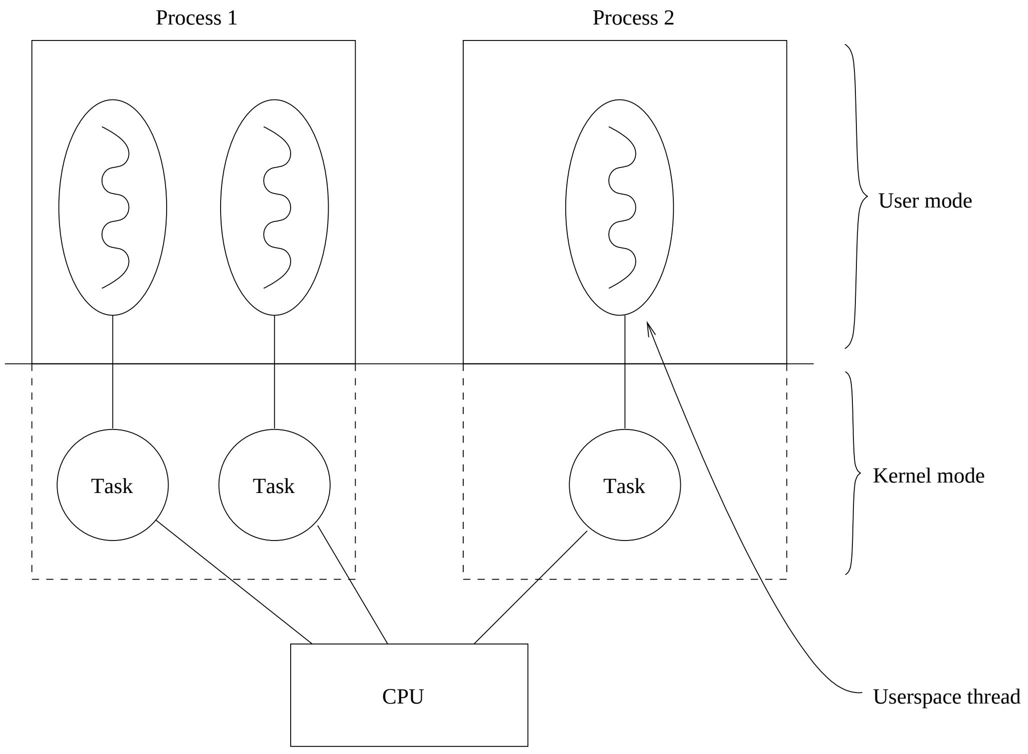

Figure 3.3: Linux 1-1 thread scheduling model

3.5 Linux 2.6

The Linux 2.6 kernel series has added a number of features and made significant changes to the way things are done internally in the kernel.

3.5.1 The scheduler is $O(1)$

This means that evey single time the scheduler runs, its operation has an upper bound that is a constant. In older versions of Linux, several routines in the scheduler had a running time bounded by $O(n)$, where n could be the numbner of tasks on the system. For example after all tasks had used up their timeslice, *all of the tasks* (running or not), would be examined so that their (goodness(), i.e. priority could be recalculated). Now, the execution of the scheduler must be bounded by (hopefully small) constant.

3.6 System calls related to scheduling in Linux

The Linux kernel provides system calls related to scheduling as described in 3.6.1. Some of these system call are used in the program 4.2

3.6.1 System calls related to scheduling

getpriority ()
: Get the nice value for a process, process group or user.

setpriority ()
: Sets the nice value for a process, process group or user.

nice ()
: Increment or decrement the nice value of a process.

sched_getparam()
: Get the scheduling parameters for a process. The interpretation of the parameters depends on whether the process is in the SCHED_FIFO, SCHED_RR or SCHED_OTHER scheduling policy.

sched_setparam()
: Sets the scheduling parameters for a process. The interpretation of the parameters depends on whether the process is in the SCHED_FIFO, SCHED_RR or SCHED_OTHER scheduling policy.

sched_getscheduler()
: Gets the scheduling policy and scheduling parameters for a process.

sched_setscheduler()
: Gets the scheduling policy and scheduling parameters for a process.

sched_get_priority_min()
: Gets the minimum priority value for a given scheduling policy.

sched_get_priority_max()
: Gets the maximum priority value for a given scheduling policy.

sched_rr_get_interval ()
: Returns information about the timeslice for a specified process that is running under the SCHED_RR scheduling policy.

sched_yield()
: The calling task gives up the processor. It is placed last in the queue corresponding to its static priority.

sched_getaffinity ()
: Get the CPU affinity mask for a specified task. The affinity mask is a bitmask that determines which CPUs the scheduler should be allowed to schedule the task on (so called "hard processor affinity").

sched_setaffinty ()
: Set the CPU affinity mask.

3.7 Priorities – From the user level

The Linux user can affect the scheduling of a task[22] by adjusting its *priority.* The process priority concept in Linux comes from UNIX, and is called *nice value.* The nice value a task can take on (also sometimes referred to as "priority"), are values in $[-20, 19]$ – a range of 40 values, where -20 is most important (least nice) and 19 is least important (nicest). The default nice value is 0. The larger the value, the "nicer" the task is. Exactly what a nice value is, is left up to the particular operating system, as long as tasks with a smaller nice value are deemed more important than those with larger values. I'll use the term "nice value" instead of "priority", since there are so

[22]Since the Linux scheduler treats threads and processes more or less the same, I'll use the term task to mean either process or thread. When there is need to differentiate between the two, I'll use the terms process and thread explicitly

many different (unrelated) things which are called “priority” in Linux and since nice values as a term is much more intuitive than when viewed as a priority value – a value of -10 is obviously not as nice as one of 12. The nice value of a process can be set initially with the user command *nice*(1) and changed while it is running with *renice*(1). These programs use the system calls *getpriority*(2) and *setpriority*(2) to do this.

The book [Gal95] claims that nice values are inefficient for real time programming. This may be true, but there’s no discussion of its appropriateness for soft real-time work.

In addition to the vague concept of nice values, Linux also implements two additional scheduling policies that are specified by the POSIX.4 standard – SCHED_FIFO (queue) and SCHED_RR (round robin). POSIX.4 also has the SCHED_OTHER scheduler, which is usually just a synonym for the “standard” time sharing scheduler – typically with nice values.

Both SCHED_FIFO and SCHED_RR give a scale of at least 32 priority values. The programmer can get hold of the maximum and minimum values in this range with the calls *sched_get_priority_max*(2) (most important) and *sched_get_priority_min*(2) (least important).

3.8 Linux soft-RT

The 2.6 series of the Linux kernel has support for soft real-time scheduling of processes using the POSIX.1b[23] API[24]. Basically, Linux divides tasks into three categories:

SCHED_FIFO Processes in this category are only preempted by higher priority processes, when blocking on I/O or when voluntarily giving up the processor[25]. No time-slicing is performed.

SCHED_RR This category is like SCHED_FIFO, except that time-slicing is done.

SCHED_OTHER The default scheduling policy of Linux. It favors I/O bound (interactive) processes to some degree. It implement the traditional UNIX scheduling (nice values).

From now on, we will refer to POSIX.1b as POSIX.4[26].

[23]formerly known as POSIX.4

[24]the sched_setscheduler(2) man-page describes the scheme used by Linux

[25]by calling sched_yield()

[26]because it is easier to type

3.8.1 Preemptible kernel

One notable change is that the kernel itself, and not only processes, should be preemptible. In the older 2.4 series, code in the kernel could call a function called cli()[27] to make sure that no other interrupt handlers on this or any other CPU could run until a corresponding sti() function was called. In this way, one could make a critical region for shared data, but with very high overhead. The mechanism corresponds to having a global lock. This is a very coarse synchronization mechanism, since even code that in no way is manipulating the data in question has to wait until the "lock" is released (the interrupts are enabled).

All of this has been done away with. Now, to protect data against concurrent access[28], one needs to use one of many blocking synchronization mechanisms that exist in Linux (like various types of semaphores), and in code where blocking isn't allowed, one must use spinlocks and exercise control over interrupts. These two mechanisms are now explained further:

Spinlocks

> Spinlocks are a form of locks. As any other locking mechanism, this means that only *one* flow of control can ever "hold the lock", and all other flows must wait until the holder releases it to have a chance of getting to "hold the lock". In many locking implementations for operating systems made for Single CPU systems, the flows of control that don't get the lock, simply get blocked (stop running) until the lock is released. This makes perfect sense on these systems, because the flows of control that dont have the lock, cannot make progress until it is released by the holder. Furthermore, on a single CPU system, there is only *one physical flow of control*[29], the notion of more than one such flow is simulated by context switching. So it makes sense to only simulate the flow that can make progress (the holder of the lock).
>
> On SMP systems, the situation is different. On such systems, there are several physical flows of control (one per CPU). To make an efficient system, the operations that need to be made atomic, should be as few as possible. This means that the critical region for some lock is typically very short (a couple of instructions). Also, a situation where more than one physical flow of control wants to execute these instructions at the

[27]this function (and the corresponding sti() function are named after specific x86 assembler instructions; they work differently, though

[28]this is an important synchronization principle: "protect data, not code"

[29]The author's own terminology – It simply means any instance of code being executed, no matter what context

same time is quite infrequent. Therefore, it will pay off for the physical flows that don't get the lock to simply spin in a tight loop and try to get the lock. This will whaste some CPU cycles, but since the holder will try to release the lock as soon as it can, it will usually be fewer cycles than would be wasted if the flow was put to sleep. Putting a flow to sleep (i.e. context switching and placing it on some queue) can be quite expensive. Also, when the flow of control is revived (because the lock was released), the cache on the CPU it is running on is most likely no longer hot, so it will execute slower for a while. This topic is discussed further in great detail in [Sch94].

Control over local interrupts

In Linux, the term "local" is used when referring to something that exist on the particular CPU that some code is executing on. So calling a function to disable a "local interrupt" means disabling an interrupt on the CPU that the function is executing on. This is totally unrelated to the same function executing on a different CPU (which would disable the interrupt in question on *that* CPU).

Disabling all of the local interrupts is a way to make sure that the CPU is only executing on behalf of one context (simulating one flow of control) as long as the interrupts are disabled. Since the CPU cannot context switch, the only thing that can mess things up is a flow of control coming from another CPU. As explained before, this can be dealt with by using spinlocks. Since disabling interrupts makes that CPU unable to process them, this technique cannot be used often and for long periods of time.

There are many other synchronization mechanisms in Linux, but most of them are variations on blocking semaphores, hardware enforced ordering barriers and atomic operations on data. These are discussed further in [Lov05].

3.8.2 Interrupts

Kernel control paths that are created to service interrupts may be arbitrarily nested in Linux. This is illustrated in figure 3.8.3.

In Linux, the interrupt handler can be configured to use:

The kernel-mode stack

The interrupt handler can use the kernel-mode stack of the process that just happened to have the CPU that the interrupt was sent to when the interrupt occurred.

A per-CPU interrupt stack

The interrupt handler can use a stack that is separate from all other kernel-mode stacks. Each CPU will have one such stack that is exclusively used for interrupts.

Either way, the stack that the interrupt handler(s) run on are quite *small* (typically 4 or 8 KB in size), and since the interrupt handlers can nest, it is imperative that the interrupt handlers use as little stack as possible. The reason why the stacks are so small is that each task must have one such kernel-level stack, and this stack must always stay in memory and cannot be paged out to swapspace. Now, 8 KB doesn't seem like a lot of memory, but if a system has, lets say, 1024[30] tasks (which is large, but not unthinkable, especially if there are many multithreaded programs running at the same time), the memory requirements for just the kernel-level stacks is 8MB. Whether all of these tasks are running or not is irrelevant, because even tasks that sleep most of the time will have this block of memory resident at all time. Therefore, the option of using a separate interrupt stack was introduced.

The decision of using the task's kernel-mode stack for interrupts has some consequences. One is that an interrupt handler cannot block (and thereby schedule some other task) in any way, since the data to restore (due to the nested control paths) is on the kernel stack of the current task. Therefore, the current task must not change until all interrupt processing has finished and the outermost interrupt handler is ready to resume the userspace execution of the current task. For the same reason, the delivery of signals to a task must wait until the kernel is ready to go back to user mode (see the arrow in figure 3.8.3). The system needs to know if it has reached nesting level 0 when returning from an interrupt handler. It does this by keeping count of which level it has reached at any given moment in the current task.

The nesting of control paths is allowed to improve performance, since the hardware issuing the interrupt and the interrupt controller wait until the CPU acknowledges the interrupt (while the interrupt handler is running, new signals on the IRQ line it services are ignored – which could come from *other* devices sharing the line). Therfore, the interrupt handler must reenable interrupts at the earliest possible moment.

3.8.3 Interrupt responses

Based on the appropriate response to an interrupt, one can classify where the response should be executed.

[30]an arbitrarily chosen number

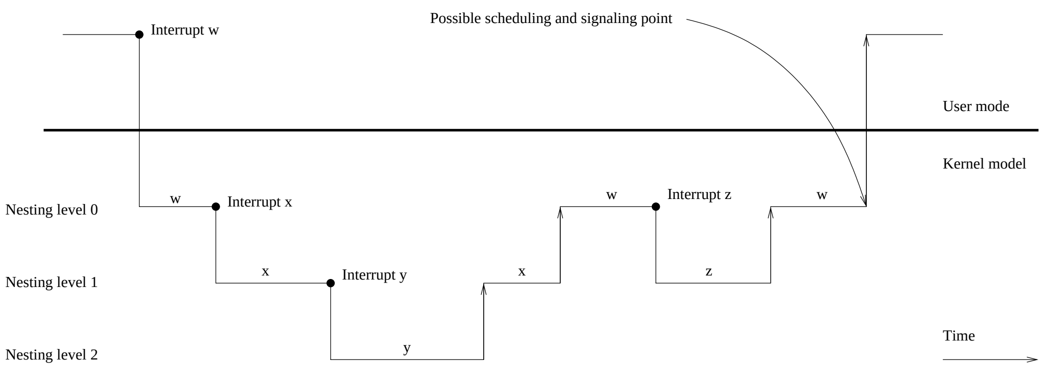

Figure 3.4: Nested interrupt invocations in Linux 2.6

Critical

> Acknowledging the interrup controller, reprogramming the interrupt controller, updating data structures shared between interrupting device and CPU. All of these are critical and must be done immediately in the handler itself with local interrupts disabled.

Noncritical

> Updating data structures that are only used bu the CPU. Must also complete quickly. Runs within the handler with interrupts enabled.

Deferrable

> Copying data from kernel space to user space (run in Softirq's and Tasklets)

The interrupt controllers never have their priority values raised or lowered, which means that the kernel will handle any kind of interrupt all the time (except when all interrupts are temporarily off). A kernel thread kirqd, tries to balance interrupts onto CPU's.

Chapter 4

Real-time and Linux

4.1 Introduction

As mentioned earlier, the UNIX operating system didn't have any kind of real-time features (soft or hard). Various people and businesses have added real-time or real-time like features to various variants of UNIX. Most of these are no longer being used, because a standard for what features a UNIX-like system should have is given in the various POSIX standards. One of these standards, POSIX.4, deals with soft real-time features. The book [Gal95] discusses this standard in great detail.

4.2 Standard: POSIX.4 – Soft Real-time

This standard is a API for C programs, and it specifies the prototypes for functions, related datatypes and constants. It fits well with the rest of the POSIX standards and UNIX in general.

The idea is that if one writes a program that uses some of this functionality, then one can recompile the program on a different system which also supports POSIX.4, ideally without having to change the code at all. In reality, code may have to be changes somewhat, and not all features of the standard are supported equally well on all platforms. It is not a complete standard in the sense that to create substantial programs, one will often have to rely on mechanisms it doesn't have.

Even so, it is still a useful basis on which to build programs.

To get an idea of what POSIX.4 offers to programmers, here is a simple program written by the author, which displays the range of priorities available to processes in the scheduling classes given in POSIX.4. It was written because its output was needed when choosing a priority for the video playback application in the test cases. The program is shown in 4.2.

4.3 Why Linux?

It could be argued that this paper is too Linux-centric, and that it should also have studied the (soft) real-time features offered by other operating systems. What follows is an explanation of why Linux was chosen.

4.3.1 Source code availability

The source code for the entire system is readily available on the Internet. This includes the kernel and all programs necessary to build new kernels.

Source code for all programs that run ot top of the kernel is also available. Although there does exist proprietary software (that comes without source code) for Linux, it is perfectly normal to not have any such programs installed. The same is mostly true of device drivers for the kernel. Only in some rare cases is there a need to use a proprietary driver.

This might not seem like a big deal, but this is a very valuable feature for programmers and students who want to study the code Another bonus is that development tools for making userspace applications are also available, in source code.

4.3.2 Multimedia and RT: several projects

Another reason for choosing Linux was that there are several interesting projects relating to soft real-time processes and multimedia that are being actively developed for it. Some of these could even become a part of standard Linux.

4.4 About patchsets

Since the code for Linux is so accessible to anyone, there exist *many* extensions for it. These are typically called patchsets, because they are distributes as sets of “patches”.

```
#define _GNU_SOURCE

#include <stdio.h>
#include <err.h>
#include <stdlib.h>
#include <sched.h>

#define NELEM(a) (sizeof(a) / sizeof(*a))

int main(int argc, char *argv[])
{
        struct { int value; char *name; } priority_policy[] =
                { { SCHED_FIFO,  "SCHED_FIFO" },
                  { SCHED_RR,    "SCHED_RR" },
                  { SCHED_OTHER, "SCHED_OTHER" },
                  { SCHED_BATCH, "SCHED_BATCH" } };
        int i, min, max;

        printf("policy\t\tmin/lowest\tmax/highest\n");
        for (i = 0; i < NELEM(priority_policy); i++) {
                min = sched_get_priority_min(priority_policy[i].value);
                if (min == -1) {
                        warnx("unable to get minimum priority for policy: %s",
                              priority_policy[i].name);
                        continue;
                }
                max = sched_get_priority_max(priority_policy[i].value);
                if (max == -1) {
                        warnx("unable to get maximum priority for policy: %s",
                              priority_policy[i].name);
                        continue;
                }
                printf("%s\t\t%d\t%d\n", priority_policy[i].name, min, max);
        }

        return EXIT_SUCCESS;
}
```

4.4.1 Patches

Traditionally, changes to the kernel source code has been made available via files that are created by the *patch*(1)[1] program.

The basic principle behind this program is that one in general can create a textual recipe of how to turn one text file into another. Such a recipe is generated by the companion program, called $diff(1)$. Its input is (usually) two texts, and its output is the recipe, called a “diff”. This is typically the file that is distributed. If one has a “diff” and one of the original files, one can use them with the patch program to generate the other original file. So if a developer makes a change to some code (let’s say its the Linux kernel), then for each modified file, he can generate a “diff” by giving patch the original files and his updated version. Then he can distribute these “diffs’ . People can get his changes into their source code by downloading his “diffs”, and using them with patch and the original files (which they have) to get his changes. It is also possible to make a “diff” that spans many files. Such a diff is generated by giving diff two file trees with text files to compare. The resulting diff is one file that can transform one of the trees into the other.

[1]This notation is the standard way to refer to documentation in UNIX. The number in parenthesis means that documentation for “patch” can be found in section 1 of the UNIX manual (1 is user commands)

Chapter 5

The projects

In this paper, the following solutions were chosen as cases to test and study. When examining the vast number of solution available on the Internet, there were not that many that fulfilled *all* of the criteria given in this paper. The three project that were selected were picked because they fulfilled all of them. After some preliminary testing, they all seemed like possible solutions to the problem at hand. Finally, one additional solution was "generated" by the author of this paper.

5.1 Vanilla Linux

This is the "standard" (a.k.a. "Vanilla") Linux kernel that can be downloaded from `http://www.kernel.org/`. The features in this kernel that we are most interested in testing is the CPU scheduler and its support for POSIX.4 soft real-time scheduling classes.

Most Linux distributions come with pre-compiled kernels. For the testing done in this paper, there is a need to compile custom kernels. Since there are plenty of good instructions on the Internet and in books like [Lov05] on how to do this, this paper will only specify the configuration parameters used for each kernel.

A kernel configuration file is quite long. Therefore, the configuration file for Linux 2.6.18 has been put into appendix A.

Since the other projects are based on this kernel, space has been saved by only listing the configuration settings that it each project adds, changes or removes with regard to the configuration found in appendix A, from now on called the "base config".

Please note that such a configuration is taylored to a particular PC and its hardware. Some of the settings will need to be changed to run on a different

PC. Which settings this applies to should be fairly obvious from context.

Some features were left out because they are not supported by all the projects (e.g. Power Managment doesn't work in rt-preempt) and because they are not relevant to the workload.

5.2 Linux with rt-preempt

This is a experimental kernel that aims to give Linux hard real-time capabilities by making the kernel as preemptible as possible, thereby reducing latencies to a minimum. Also, all interrupts and software interrupts (softirqs and tasklets) are handled by kernel-threads instead. It also incorporates high-resolution timers that are supported via the system calls for nanosleep(), the interval timers (getitimer() and setitimer()) and POSIX timers. It is being developed by some very experienced and capable Linux kernel developers. It is distributed as a patch to the vanilla Linux kernel.

5.2.1 Configuration of rt-preempt

Uses the base config and the following settings:

- CONFIG_HIGH_RES_TIMERS yes
- CONFIG_NO_HZ yes
- CONFIG_PREEMPT_RT yes
- CONFIG_PREEMPT_RCU yes
- CONFIG_VMSPLIT_3G yes
- GENERIC_TIME_VSYSCALL no (since ntp isn't relevant for this paper)
- CONFIG_BLOCKER yes (in case we want to run pi_test suite)
- CONFIG_WAKEUP_TIMING yes
- CONFIG_WAKEUP_LATENCT_HIST yes (interesting feature)

These generate warning about increasing overhead and latencies at bootup, so for our testing purposes they should be *disabled*.

- CRITICAL_PREEMPT_TIMING

- CONFIG_PREEMPT_OFF_HIST
- CONFIG_CRITICAL_IRQSOFF_TIMING
- CONFIG_INTERRUPT_OFF_HIST
- CONFIG_LATENCY_TRACE

The version used in this paper was: patch-2.6.18-rt7.

5.3 Linux with Class-based Resource Kernel Management

This project has as its goal to allows root to define classes of applications, and then allocate resources, such as CPU, memory pages, and disk bandwith.

The project has been in development for over 3 years, and has been considered for inclusion in mainline Linux.

In the current version (f0.8-2.6.18, the one used in this paper), there is support for:

- allocating CPU
- limiting the number of tasks a group can spawn
- limiting the rate of calls to fork() that a group can do
- control the numbner of LRU pages used by a group

In this paper, only the CPU allocation feature will be explored.

5.3.1 Class configuration

The CKRM system uses the relatively new configfs special file system (in Linx since 2.6.14) for the configuration of classes. It makes it possible for userspace programs to change and view kernel configuration data as *files*. Hierarchical structuring of the data is provided by the directories of the "filesystem". The filesystem doesn't have any backing store, but is instead backed by kernel data structures.

For example, in this paper we need a class (a.k.a. resource group) for the video application. To create it, one simply creates a directory where the filesystem has been mounted (/config in our case). The kernel responds by filling it with some "text files", where each file controls some aspect of the configuration[1]. These can then be read and manipulated with standard tools (such as cat and echo). For example, to make a task a member of a resource group, write its pid into the members file in the directory for the group in question.

5.3.2 Configuration of ckrm

Uses the base config and the following settings:

- CONFIG_CPU_RC (no way of not selecting in xconfig)
- CONFIG_RES_GROUPS (for obvious reasons)
- CONFIG_RGCS (User interface for resource groups.. needed Y or M)
- CONFIG_RES_GROUPS_NUMTASKS (resource group that limits number of tasks in a group)
- CONFIG_RES_GROUPS_MEM_RC (collects info about phys. memory usage)
- CONFIG_RES_GROUP_CPU (obviously)
- CONFIG_VMSPLIT_3G (not relevant, but a choice has to be made - make it same as in rt-preempt)

5.4 Linux with CKRM and rt-preempt

This "project" came to be based on a crazy idea the author of this paper had. It seemed that both the ckrm and rt-preempt projects were addressing different kernel issues relevant to multimedia applications. Low latency seemed to be beneficial for video playback, but so did CPU reservation.

What if one could have both? This *had* to be explored. The patches for ckrm were applied to a vanilla Linux 2.6.18. Then the patches for rt-preempt. After fixing some very simple rejects[2], a bootable and surprisingly stable kernel was created. The only problems known to the author, is that it will hang if one tries to put a task into the POSIX.4 scheduling class SCHED_RR. This isn't a big problem, since one has ckrm-classes instead (which are used in the workload).

Uses the base config and the following settings:

[1]This is somewhat analogous to "normal" filesystems automatically adding the entries "." and ".." in every directory

[2]patch will generate a reject file when a part of a patch cannot be applied cleanly

- All the settings from the rt-preempt config
- All the settings from the ckrm config
- CONFIG_INOTIFY (replaces CONFIG_DNOTIFY) yes
- CONFIG_INOTIFY_USER (see CONFIG_INOTIFY) yes

Chapter 6

Test cases – workloads

6.1 Introduction

Three different test cases were conceived and run. A program was written to automate the process and to measure the time elapsed for each run. The program, called workload, can be found in B[1].

Since it was necessary for a human to look and pay attention to the video that was played back, the video file was shortened to 5 minutes in length[2].

Data for the video file (as reported by mplayer):

```
VIDEO:  MPEG1  320x240  (aspect 2)  29.970 fps  2774.0 kbps (346.8 kbyte/s)
AUDIO: 48000 Hz, 2 ch, s16le, 224.0 kbit/14.58% (ratio: 28000->192000)
```

6.1.1 Actions performed by the workload program

The workload program will:

- Set up resource group if needed.
- Start the profiler (OProfile)
- Record the starting time
- Launch some instances of xboard or cpuhog
- Launch mplayer or xine with special priorities (see below)
- Take down the xboard or cpuhog instances

[1]Somehow, the program turned out to be really ugly, but it does the job

[2]this was found to be sufficiently long as to give meaningful results, but not so long as to bore the human watching it

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (97.9%)	poll_idle (78,4%)	poll_idle (95.7%)	poll_idle (90.6%)
Elapsed time	301.94s	301.90s	302.67s	301.06s
Top user prog	gnuchess (78.15%)	gnuchess (74.02%)	gnuchess (78.01%)	gnuchess (74.72%)

Table 6.1: Summary for case 1 – xine

- Record the ending time
- Stop the profiler
- Read and store the profiling data and elapsed time to some files.

For the kernels that support class based resource management (ckrm and ckrm-rt7), a resource group called "mm" is created, and the video playback application (mplayer or xine) is placed in it.

For the other kernels (vanilla 2.6.18 and rt-preempt), the video playback application (mplayer or xine) is run in the POSIX.4 scheduling class SCHED_RR with a priority of 99 (the best priority obtainable in Linux for this scheduling class).

All other programs (xboard or cpuhog) are run without any special preparation (SCHED_OTHER, nice value 0) in all cases.

6.2 Case 1:xine/mplayer with xboard

All the kernels were subjected to having one of xine or mplayer and xboard run together as just described. Xboard was run so that it would play against itself for 100 matches (a loong time), and The video was watched and notes about any playback problems were written down. In this case, there were none. All kernels had no trouble with this test.. The data for case 1 is shown in tables 6.1 and 6.2. No surprises here. Everything is going smoothly, the machine is mostly idle.

6.3 Case 2: xine/mplayer with two cpuhogs

All the kernels were run with one of xine or mplayer and two instances of the program cpuhog (as shown in 7.1). Again, the video was watched and notes taken. See tables 6.3 and 6.4.

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (97.9%)	poll_idle (68.2%)	poll_idle (94.8%)	poll_idle (88.7%)
Elapsed time	300.80s	302.34s	303.35s	305.94s
Top user prog	gnuchess (77.76%)	gnuchess (74.51%)	gnuchess (77.93%)	gnuchess (74.65%)

Table 6.2: Summary for case 1 – mplayer

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (91.3%)	poll_idle (92.4%)	poll_idle (87.3%)	poll_idle (81.5%)
Elapsed time	302.46s	301.90s	314.90s	303.45s
Top user prog	libc-2.4.so (59.86%)	libc-2.4.so (57.40%)	libc-2.4.so (60.43%)	libc-2.4.so (56.83%)
Second user prog	cpuhog (19.80%)	cpuhog (18.58%)	cpuhog (19.42%)	cpuhog (19.33%)
Playback problems	Some skipping	OK	Some skipping	OK

Table 6.3: Summary for case 2 – xine

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (89.7%)	poll_idle (89.1%)	poll_idle (80.5%)	poll_idle (72.3%)
Elapsed time	432.27s	302.17s	302.54s	s
Top user prog	libc-2.4.so (63.78%)	libc-2.4.so (57.82%)	libc-2.4.so (59.73%)	libc-2.4.so (58.11%)
Second user prog	cpuhog (21.12%)	cpuhog (19.11%)	cpuhog (19.67%)	cpuhog (18.98%)
Playback problems	Got stuck!	OK	OK	OK

Table 6.4: Summary for case 2 – mplayer

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (86.5%)	poll_idle (76.8%)	poll_idle (85.1%)	poll_idle (74.3%)
Elapsed time	307.40s	304.52s	308.10s	305.34s
Top user prog	libc-2.4.so (60.57%)	libc-2.4.so (63.71%)	libc-2.4.so (59.27%)	libc-2.4.so (57.92%)
Second user prog	cpuhog (19.96%)	cpuhog (20.23%)	cpuhog (20.53%)	cpuhog (18.61%)
Playback problems	Skipping	Slow choppy	Choppy	OK

Table 6.5: Summary for case 3 – xine

	vanilla	rt-preempt	ckrm	ckrm-rt
Top kernel function	poll_idle (84.7%)	poll_idle (65.9%)	poll_idle (78.7%)	poll_idle (70.3%)
Elapsed time	184.94s	302.34s	351.79s	301.87s
Top user prog	libc-2.4.so (68.57%)	libc-2.4.so (57.86%)	libc-2.4.so (62.11%)	libc-2.4.so (58.16%)
Second user prog	cpuhog (22.69%)	cpuhog (19.05%)	cpuhog (20.17%)	cpuhog (19.01%)
Playback problems	Stuck!	OK	Pauses	OK

Table 6.6: Summary for case 3 – mplayer

6.4 Case 3: xine/mplayer with three cpuhogs

All the kernels were run with one of xine or mplayer and three instances of the program cpuhog (as shown in 7.1). Again, the video was watched and notes taken. See tables 6.5 and 6.6.

Chapter 7

Measurements

7.1 Measurement tools

7.1.1 OProfile

The primary measurement tool that was used for the experiments in this paper was OProfile

Reasons for choosing OProfile[1]:

Need a profile of an application and its shared libraries

> This applies to all of the test cases used in this paper. Mplayer uses a *lot*[2] of shared libraries. If significant amounts of time is spent in one of the libraries its using, we don't want the profiling system to miss it.

Need to capture the performance behavior of entire system

> This also applies to the test cases. The overall behavior of the system is also interesting.

Low overhead during sampling

> Although all profiling systems will claim to have a low overhead, the OProfile system really has quite low overhead because it uses built-in features in the CPU to do the profiling if they are available. Many inexpensive and very common CPU's *do* have these features.

The particular CPU used in this paper is a "Sempron" made by Advanced Micro Devices (AMD). It is a budget[3] variant of AMD's "Athlon" CPU.

[1]points slightly paraphrased from the OProfile manual http://oprofile.sourceforge.net/doc/introduction.html

[2]try this in the shell "ldd 'which mplayer'"

[3]Basically, it is cheaper and slower than the "Athlon"

This processor has a instruction set compatible with the Intel's x86 series of processors[4]. The reason this is even mentioned here is that the kind of monitoring that can be done with OProfile is dependent on the particular model of CPU used. This particular CPU works well with OProfile, since it has the hardware performance counters that OProfile needs.

When OProfile is using the CPU performance counters (as is the case on the "Athlon"), one has the opportunity configure the counters. Under normal operation of OProfile, the counters are loaded with a value. The counter will then increment for each event it is programmed to trigger on until the count loaded is reached. When this happens, a Non-maskable interrupt is triggered. This is rather nice, since it it means that the interrupt will occur even if the kernel has disabled all local interrupts. The handler will then record the event. OProfile will record the instruction pointer and and associate a counter with that pointer. In this way, the counters are used by OProfile to sample events. OProfile uses the passing of a clock cycle as the default event to have the count increased, thereby sampling the program counter at regular intervals (multiples of clock cycles).

The basic operation of OProfile works like this:

Setup

OProfile needs to know about the image of the currently running kernel if it is to generate correct profiling data for it.

Clean out

The OProfile system will retain old profiling data and add new data to it by default. Therefore, one needs to perform this step to get a fresh profile.

Start the sampling

OProfile needs to be told that it must start to sample events. It starts up oprofiled (the oprofile daemon), that handles the communication with the oprofile kernel driver.

Start the application to collect data about

The application (or applications) that should be profiled must run.

Stop the sampling

Oprofiled must be told that it is to stop the sampling.

[4]Probably the worlds most commonly used instruction set. All programs for all PC's use it

Generate profiles

Now, various profiles can be generated. Currently, OProfile supports a flat profile (as shown in 7.1.2), annotated source code profile for tasks (both C and assembler output), and a profile for the kernel (shown in 7.1.3).

7.1.2 A sample flat profile

Here is some sample output from a profile that ran for 1 minute on a lightly loaded system (only the first 20 lines are shown). The numbers under the column "samples" are the number of times a non-maskable interrupt happened while the instruction pointer was inside the "code" named in the third column. The second column is the ratio of the samples for that line and the total number of samples made, expressed as a percentage. For code that is associated with tasks, there is additional output indented below it.

```
CPU: Athlon, speed 1102.51 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Cycles outside of halt state) with a \
unit mask of 0x00 (No unit mask) count 100000
CPU_CLK_UNHALT...|
  samples|      %|
------------------
   558267 84.6943 vmlinux-2.6.18
    61829  9.3800 libc-2.4.so
    14650  2.2225 libmad.so.0.2.1
     6199  0.9404 libasound.so.2.0.0
     2558  0.3881 emacs.emacs-21
        CPU_CLK_UNHALT...|
          samples|      %|
        ------------------
             2540 99.2963 emacs.emacs-21
               18  0.7037 anon (tgid:5046 range:0xb7f5b000-0xb7f5c000)
     2542  0.3856 oprofiled
        CPU_CLK_UNHALT...|
          samples|      %|
        ------------------
             2538 99.8426 oprofiled
                4  0.1574 anon (tgid:4933 range:0xb7f67000-0xb7f68000)
     2519  0.3822 Xorg
        CPU_CLK_UNHALT...|
          samples|      %|
        ------------------
             2459 97.6181 Xorg
               60  2.3819 anon (tgid:4412 range:0xb7f02000-0xb7f03000)
     1419  0.2153 libglib-1.2.so.0.0.10
     1325  0.2010 xmms
        CPU_CLK_UNHALT...|
          samples|      %|
        ------------------
             1219 92.0000 xmms
              106  8.0000 anon (tgid:6937 range:0xb7f7c000-0xb7f7d000)
     1301  0.1974 libpthread-2.4.so
     1075  0.1631 libfb.so
     1014  0.1538 libX11.so.6.2.0
      820  0.1244 libxaa.so
      765  0.1161 libxmmsmad.so
      755  0.1145 nv_drv.so
```

In this case, the Linux kernel was the busiest part of the system (which is natural for a lightly loaded system). We see that the busiest userspace component was the c library (which nearly every program uses in some way). Right in third place is a library called “mad”. The user was listening to music with xmms (which is listed further down), but it in turn uses the “mad” library to decode mp3 data. the first task listed is “emacs” (which the user was running to type this document). The oprofiled is of course also running, and therefore shows up in the profile.

7.1.3 A sample kernel profile

This is some sample output for a kernel profile. The first column shows the number of ticks spent in

```
10137627 total                                      3.7940
9956907 poll_idle                                311153.3438
 93609 __do_softirq                             650.0625
 11489 handle_IRQ_event                         119.6771
  5057 schedule                                   2.8733
  3637 nv_nic_irq                                 5.6828
  2633 __copy_to_user_ll                         23.5089
  2277 do_select                                  1.9495
  2253 snd_pcm_period_elapsed                     3.3527
  2181 sysenter_past_esp                         18.0248
```

```
1843 fget_light                                    10.4716
1807 unix_poll                                      8.6875
1719 mmx_copy_page                                  5.9688
1605 do_sys_poll                                    1.5199
1148 __d_lookup                                     3.1196
1134 hrtimer_run_queues                             3.3750
1080 number                                         1.3776
1070 run_timer_softirq                              2.3060
1064 __link_path_walk                               0.2692
1027 __wake_up                                     10.6979
```

7.2 Choice of tool

In the context of the problem described here, OProfile seems like the natural tool to choose. It can take samples anywhere, *including* inside the kernel and the shared object file libraries that are dynamically linked into tasks.

It is also capable of identifying samples in a way that makes it easy to see where in a particular program or library the sample was taken[5].

It imposes a low overhead on the system, because it takes advantage of particular hardware features. The performance counters do have a weakness, however. Because of the way the performance counters interact with the way the CPU executes instructions, the samples are not always 100% accurate. The inaccuracy means that a particular sample can sometimes yield an incorrect program counter (it can miss by a few instructions). This isn't a big problem, since the broader picture is usually what one is interested in anyway, and the sample will most likely still "point" into the right part of the system.

A disadvantage of OProfile is that it has no notion of tasks. It only keeps track of where the instruction pointer has been. This means that if there are several threads that execute the same code, then this will only show up as higher numbers for that piece of code. There is no easy way to see counts per thread.

[5]the name of a component is much better than just giving some address in hex

7.3 Setup of test environment

7.3.1 Selection of workload

In [Jai91], one argues that it is important to make the synthetic workload one constructs representative with regard to how the system will be used "in the field". In our case, this is somewhat difficult, because most users know from experience that running a resource-intensive job while a video is being played back is a bad idea. Consequently, users modify their behavior, and simply don't do such things.

It might sound farfetched that users modify their behavior in this way. It is however not that uncommon. For example, before buffer-underrun protection[6] was common in CD-RW drives, people would stop using their computer while a CD was being recorded[7].

Another problem in selecting representative workload is that most programs that are CPU-intensive are written in a way that makes it relatively easy for a scheduler to identify them. They will typically use up their timeslices many times over (often while doing some hefty calculation in a loop). On the other hand, a process that mostly sleeps, will only intermittently be in a runnable state, and are therefore not an issue for the scheduler.

In order to test how the schedulers behaves when there is contention among tasks to use the processor, there is a need to somehow gererate this "pressure" on the CPU. The way that was chosen for this workload was to write a program that would try to defy classification by the scheduler, and thus not loose priority[8].

It turned out that constructing a process with this characteristic (constantly being schedulable with a relatively high priority) was quite easy. The resulting C program is shown in figure 7.1.

Perhaps surprisingly, only three instances were needed to make the computer this test was conducted on seem *very slow* and unresponsive.

[6]So called Burn-proof

[7]If they didn't, they risked making a rather expensive coaster

[8]As noted earlier, the Linux 2.6 scheduler will mostly try to punish non-interactive tasks

```
#include <stdlib.h>
#include <time.h>

#define LOOPS 100000
#define ARRAY_SIZE 1000

int myrand(void)
{
        return rand() % ARRAY_SIZE;
}

int main(int argc, char *argv[])
{
        struct timespec sleeptime = { 0, 1000000 };
        int data[ARRAY_SIZE];
        int i;

        /*
         * We're sleeping and doing some useless work, in the
         * hopes of being mistaken for a interactive process.
         */
        for (;;) {
                for (i = 0; i < LOOPS; i++)
                        /*
                         * usless computation that we think the
                         * compiler cannot optimize away.
                         */
                        data[myrand()] *= ++data[myrand()];
                nanosleep(&sleeptime, NULL);
        }
}
```

Figure 7.1: The cpuhog program

Chapter 8

Conclusion

8.1 Results of the experiments

The tests show that:

The C library is heavily used

> The C runtime library is heavily used because the cpuhog uses it. Here one of the disadvantages of OProfile is shown. One could assume that most of the time, cpuhog was calling it, and simply add the two percentages up.

Comparable treatment of other programs

> It doesn't seem that any implementation favors the multimedia application too much. So the only useful guideline is to look at the playback problems. From that point of view it would seem that ckrm-rt would be the preferable choice. It has comparable performance and gives a better viewing experience than the others.

The OS isn't busy?

> This is perhaps the most troubling result. Why is the idle routine in the kernel (poll_idle()) executed so frequently, when other applications can't run to our satisfaction? Is the data simply wrong, or does the *really* CPU really call it that often? OProfile is very consistent in this – every single invocations of it will give you poll_idle at the top.

Being a real-time application doesn't matter?

> This is troubling too. Even a task that has the maximum priority (99) in the scheduling class SCHED_RR can experience that other normal

programs can with a much lower priority can spoil everything (like cpuhog does). This should not have happened.

Maybe the profile is right, and timeliness and low latency is much more important for multimedia than everything else, including CPU scheduling.

8.2 Future work

The problems surrounding the real-time piority in SCHED_RR should be investigated. So should the high count for (poll_idle).

Bibliography

[Bac90] Maurice J. Bach. *The design of the UNIX operating system.* Prentice Hall, 1990.

[Gal95] Bill O. Gallmeister. *POSIX.4 – Programming for the real world.* O'Reilly & Associates, INC., first edition edition, January 1995.

[Jai91] Raj Jain. *The art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[Lio96] John Lions. *Lions' Commentary on UNIX 6th Edition with Source Code.* Peer-to-Peer Communications, 1996.

[LL73] L. C. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20, January 1973.

[Lov05] Robert Love. *Linux Kernel Development.* Novell Press, second edition edition, 2005.

[MBKQ96] Marshall Kirk McKusic, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System.* Addison-Wesley Longman, 1996.

[Sch94] Curt Schimmel. *UNIX Systems for Modern Architectures.* Addison Wesley, 1994.

[SN95] Ralf Steinmetz and Klara Nahrstedt. *Multimedia: Computing, Communications & Applications.* Prentice Hall, 1995.

[Tho78] K. Thompson. Unix implementation. 1978.

Appendix A

Configuration file for Linux 2.6.18

```
# Automatically generated make config: don’t edit Linux kernel
#version: 2.6.18 Sat Oct 28 13:18:10 2006 CONFIG_X86_32=y
#CONFIG_GENERIC_TIME=y CONFIG_LOCKDEP_SUPPORT=y
#CONFIG_STACKTRACE_SUPPORT=y CONFIG_SEMAPHORE_SLEEPERS=y CONFIG_X86=y
#CONFIG_MMU=y CONFIG_GENERIC_ISA_DMA=y CONFIG_GENERIC_IOMAP=y
#CONFIG_GENERIC_HWEIGHT=y CONFIG_ARCH_MAY_HAVE_PC_FDC=y CONFIG_DMI=y
#CONFIG_DEFCONFIG_LIST="/lib/modules/$UNAME_RELEASE/.config"

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y
CONFIG_BROKEN_ON_SMP=y
CONFIG_LOCK_KERNEL=y
CONFIG_INIT_ENV_ARG_LIMIT=32

#
# General setup
#
CONFIG_LOCALVERSION=""
# CONFIG_LOCALVERSION_AUTO is not set
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_POSIX_MQUEUE=y
CONFIG_BSD_PROCESS_ACCT=y
# CONFIG_BSD_PROCESS_ACCT_V3 is not set
```

```
CONFIG_TASKSTATS=y
CONFIG_TASK_DELAY_ACCT=y
# CONFIG_AUDIT is not set
CONFIG_IKCONFIG=y
CONFIG_IKCONFIG_PROC=y
CONFIG_RELAY=y
CONFIG_INITRAMFS_SOURCE=""
# CONFIG_CC_OPTIMIZE_FOR_SIZE is not set
# CONFIG_EMBEDDED is not set
CONFIG_UID16=y
CONFIG_SYSCTL=y
CONFIG_KALLSYMS=y
# CONFIG_KALLSYMS_ALL is not set
# CONFIG_KALLSYMS_EXTRA_PASS is not set
CONFIG_HOTPLUG=y
CONFIG_PRINTK=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_EPOLL=y
CONFIG_SHMEM=y
CONFIG_SLAB=y
CONFIG_VM_EVENT_COUNTERS=y
CONFIG_RT_MUTEXES=y
# CONFIG_TINY_SHMEM is not set
CONFIG_BASE_SMALL=0
# CONFIG_SLOB is not set

#
# Loadable module support
#
CONFIG_MODULES=y
CONFIG_MODULE_UNLOAD=y
CONFIG_MODULE_FORCE_UNLOAD=y
# CONFIG_MODVERSIONS is not set
# CONFIG_MODULE_SRCVERSION_ALL is not set
# CONFIG_KMOD is not set

#
# Block layer
```

```
#
# CONFIG_LBD is not set
# CONFIG_BLK_DEV_IO_TRACE is not set
# CONFIG_LSF is not set

#
# IO Schedulers
#
CONFIG_IOSCHED_NOOP=y
# CONFIG_IOSCHED_AS is not set
# CONFIG_IOSCHED_DEADLINE is not set
CONFIG_IOSCHED_CFQ=y
# CONFIG_DEFAULT_AS is not set
# CONFIG_DEFAULT_DEADLINE is not set
CONFIG_DEFAULT_CFQ=y
# CONFIG_DEFAULT_NOOP is not set
CONFIG_DEFAULT_IOSCHED="cfq"

#
# Processor type and features
#
# CONFIG_SMP is not set
CONFIG_X86_PC=y
# CONFIG_X86_ELAN is not set
# CONFIG_X86_VOYAGER is not set
# CONFIG_X86_NUMAQ is not set
# CONFIG_X86_SUMMIT is not set
# CONFIG_X86_BIGSMP is not set
# CONFIG_X86_VISWS is not set
# CONFIG_X86_GENERICARCH is not set
# CONFIG_X86_ES7000 is not set
# CONFIG_M386 is not set
# CONFIG_M486 is not set
# CONFIG_M586 is not set
# CONFIG_M586TSC is not set
# CONFIG_M586MMX is not set
# CONFIG_M686 is not set
# CONFIG_MPENTIUMII is not set
# CONFIG_MPENTIUMIII is not set
# CONFIG_MPENTIUMM is not set
# CONFIG_MPENTIUM4 is not set
# CONFIG_MK6 is not set
CONFIG_MK7=y
# CONFIG_MK8 is not set
# CONFIG_MCRUSOE is not set
# CONFIG_MEFFICEON is not set
# CONFIG_MWINCHIPC6 is not set
# CONFIG_MWINCHIP2 is not set
# CONFIG_MWINCHIP3D is not set
# CONFIG_MGEODEGX1 is not set
# CONFIG_MGEODE_LX is not set
# CONFIG_MCYRIXIII is not set
# CONFIG_MVIAC3_2 is not set
# CONFIG_X86_GENERIC is not set
CONFIG_X86_CMPXCHG=y
CONFIG_X86_XADD=y
CONFIG_X86_L1_CACHE_SHIFT=6
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y
CONFIG_X86_CMPXCHG64=y
CONFIG_X86_GOOD_APIC=y
CONFIG_X86_INTEL_USERCOPY=y
CONFIG_X86_USE_PPRO_CHECKSUM=y
CONFIG_X86_USE_3DNOW=y
CONFIG_X86_TSC=y
# CONFIG_HPET_TIMER is not set
# CONFIG_PREEMPT_NONE is not set
# CONFIG_PREEMPT_VOLUNTARY is not set
CONFIG_PREEMPT=y
CONFIG_PREEMPT_BKL=y
CONFIG_X86_UP_APIC=y
CONFIG_X86_UP_IOAPIC=y
CONFIG_X86_LOCAL_APIC=y
CONFIG_X86_IO_APIC=y
CONFIG_X86_MCE=y
CONFIG_X86_MCE_NONFATAL=y
# CONFIG_X86_MCE_P4THERMAL is not set
CONFIG_VM86=y
```

```
# CONFIG_TOSHIBA is not set
# CONFIG_I8K is not set
# CONFIG_X86_REBOOTFIXUPS is not set
# CONFIG_MICROCODE is not set
# CONFIG_X86_MSR is not set
# CONFIG_X86_CPUID is not set

#
# Firmware Drivers
#
# CONFIG_EDD is not set
# CONFIG_DELL_RBU is not set
# CONFIG_DCDBAS is not set
CONFIG_NOHIGHMEM=y
# CONFIG_HIGHMEM4G is not set
# CONFIG_HIGHMEM64G is not set
CONFIG_PAGE_OFFSET=0xC0000000
CONFIG_ARCH_FLATMEM_ENABLE=y
CONFIG_ARCH_SPARSEMEM_ENABLE=y
CONFIG_ARCH_SELECT_MEMORY_MODEL=y
CONFIG_SELECT_MEMORY_MODEL=y
CONFIG_FLATMEM_MANUAL=y
# CONFIG_DISCONTIGMEM_MANUAL is not set
# CONFIG_SPARSEMEM_MANUAL is not set
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
CONFIG_SPARSEMEM_STATIC=y
CONFIG_SPLIT_PTLOCK_CPUS=4
# CONFIG_RESOURCES_64BIT is not set
# CONFIG_MATH_EMULATION is not set
CONFIG_MTRR=y
CONFIG_REGPARM=y
CONFIG_SECCOMP=y
# CONFIG_HZ_100 is not set
# CONFIG_HZ_250 is not set
CONFIG_HZ_1000=y
CONFIG_HZ=1000
# CONFIG_KEXEC is not set
CONFIG_PHYSICAL_START=0x100000
CONFIG_COMPAT_VDSO=y
```

```
#
# Power management options (ACPI, APM)
#
# CONFIG_PM is not set

#
# ACPI (Advanced Configuration and Power Interface) Support
#
# CONFIG_ACPI is not set

#
# CPU Frequency scaling
#
# CONFIG_CPU_FREQ is not set

#
# Bus options (PCI, PCMCIA, EISA, MCA, ISA)
#
CONFIG_PCI=y
# CONFIG_PCI_GOBIOS is not set
# CONFIG_PCI_GOMMCONFIG is not set
# CONFIG_PCI_GODIRECT is not set
CONFIG_PCI_GOANY=y
CONFIG_PCI_BIOS=y
CONFIG_PCI_DIRECT=y
# CONFIG_PCIEPORTBUS is not set
# CONFIG_PCI_MSI is not set
# CONFIG_PCI_DEBUG is not set
CONFIG_ISA_DMA_API=y
# CONFIG_ISA is not set
# CONFIG_MCA is not set
# CONFIG_SCx200 is not set

#
# PCCARD (PCMCIA/CardBus) support
#
# CONFIG_PCCARD is not set

#
# PCI Hotplug Support
#
```

```
# CONFIG_HOTPLUG_PCI is not set

#
# Executable file formats
#
CONFIG_BINFMT_ELF=y
# CONFIG_BINFMT_AOUT is not set
# CONFIG_BINFMT_MISC is not set

#
# Networking
#
CONFIG_NET=y

#
# Networking options
#
# CONFIG_NETDEBUG is not set
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_UNIX=y
CONFIG_XFRM=y
# CONFIG_XFRM_USER is not set
# CONFIG_NET_KEY is not set
CONFIG_INET=y
# CONFIG_IP_MULTICAST is not set
# CONFIG_IP_ADVANCED_ROUTER is not set
CONFIG_IP_FIB_HASH=y
# CONFIG_IP_PNP is not set
# CONFIG_NET_IPIP is not set
# CONFIG_NET_IPGRE is not set
# CONFIG_ARPD is not set
# CONFIG_SYN_COOKIES is not set
CONFIG_INET_AH=y
CONFIG_INET_ESP=y
CONFIG_INET_IPCOMP=m
CONFIG_INET_XFRM_TUNNEL=m
CONFIG_INET_TUNNEL=m
CONFIG_INET_XFRM_MODE_TRANSPORT=y
CONFIG_INET_XFRM_MODE_TUNNEL=y
CONFIG_INET_DIAG=y
```

```
CONFIG_INET_TCP_DIAG=y
# CONFIG_TCP_CONG_ADVANCED is not set
CONFIG_TCP_CONG_BIC=y
# CONFIG_IPV6 is not set
# CONFIG_INET6_XFRM_TUNNEL is not set
# CONFIG_INET6_TUNNEL is not set
# CONFIG_NETWORK_SECMARK is not set
# CONFIG_NETFILTER is not set

#
# DCCP Configuration (EXPERIMENTAL)
#
# CONFIG_IP_DCCP is not set

#
# SCTP Configuration (EXPERIMENTAL)
#
# CONFIG_IP_SCTP is not set

#
# TIPC Configuration (EXPERIMENTAL)
#
# CONFIG_TIPC is not set
# CONFIG_ATM is not set
# CONFIG_BRIDGE is not set
# CONFIG_VLAN_8021Q is not set
# CONFIG_DECNET is not set
# CONFIG_LLC2 is not set
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_X25 is not set
# CONFIG_LAPB is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set

#
# QoS and/or fair queueing
#
# CONFIG_NET_SCHED is not set

#
```

```
# Network testing
#
# CONFIG_NET_PKTGEN is not set
# CONFIG_NET_TCPPROBE is not set
# CONFIG_HAMRADIO is not set
# CONFIG_IRDA is not set
# CONFIG_BT is not set
# CONFIG_IEEE80211 is not set

#
# Device Drivers
#

#
# Generic Driver Options
#
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
# CONFIG_FW_LOADER is not set
# CONFIG_DEBUG_DRIVER is not set
# CONFIG_SYS_HYPERVISOR is not set

#
# Connector - unified userspace <-> kernelspace linker
#
# CONFIG_CONNECTOR is not set

#
# Memory Technology Devices (MTD)
#
# CONFIG_MTD is not set

#
# Parallel port support
#
CONFIG_PARPORT=y
CONFIG_PARPORT_PC=y
# CONFIG_PARPORT_SERIAL is not set
# CONFIG_PARPORT_PC_FIFO is not set
# CONFIG_PARPORT_PC_SUPERIO is not set
# CONFIG_PARPORT_GSC is not set
# CONFIG_PARPORT_AX88796 is not set
# CONFIG_PARPORT_1284 is not set

#
# Plug and Play support
#

#
# Block devices
#
CONFIG_BLK_DEV_FD=y
# CONFIG_PARIDE is not set
# CONFIG_BLK_CPQ_DA is not set
# CONFIG_BLK_CPQ_CISS_DA is not set
# CONFIG_BLK_DEV_DAC960 is not set
# CONFIG_BLK_DEV_UMEM is not set
# CONFIG_BLK_DEV_COW_COMMON is not set
CONFIG_BLK_DEV_LOOP=y
# CONFIG_BLK_DEV_CRYPTOLOOP is not set
# CONFIG_BLK_DEV_NBD is not set
# CONFIG_BLK_DEV_SX8 is not set
# CONFIG_BLK_DEV_UB is not set
# CONFIG_BLK_DEV_RAM is not set
# CONFIG_BLK_DEV_INITRD is not set
# CONFIG_CDROM_PKTCDVD is not set
# CONFIG_ATA_OVER_ETH is not set

#
# ATA/ATAPI/MFM/RLL support
#
CONFIG_IDE=y
CONFIG_BLK_DEV_IDE=y

#
# Please see Documentation/ide.txt for help/info on IDE drives
#
# CONFIG_BLK_DEV_IDE_SATA is not set
# CONFIG_BLK_DEV_HD_IDE is not set
CONFIG_BLK_DEV_IDEDISK=y
# CONFIG_IDEDISK_MULTI_MODE is not set
CONFIG_BLK_DEV_IDECD=y
```

```
# CONFIG_BLK_DEV_IDETAPE is not set
# CONFIG_BLK_DEV_IDEFLOPPY is not set
# CONFIG_BLK_DEV_IDESCSI is not set
# CONFIG_IDE_TASK_IOCTL is not set

#
# IDE chipset support/bugfixes
#
# CONFIG_IDE_GENERIC is not set
# CONFIG_BLK_DEV_CMD640 is not set
CONFIG_BLK_DEV_IDEPCI=y
# CONFIG_IDEPCI_SHARE_IRQ is not set
# CONFIG_BLK_DEV_OFFBOARD is not set
CONFIG_BLK_DEV_GENERIC=y
# CONFIG_BLK_DEV_OPTI621 is not set
# CONFIG_BLK_DEV_RZ1000 is not set
CONFIG_BLK_DEV_IDEDMA_PCI=y
# CONFIG_BLK_DEV_IDEDMA_FORCED is not set
CONFIG_IDEDMA_PCI_AUTO=y
# CONFIG_IDEDMA_ONLYDISK is not set
# CONFIG_BLK_DEV_AEC62XX is not set
# CONFIG_BLK_DEV_ALI15X3 is not set
CONFIG_BLK_DEV_AMD74XX=y
# CONFIG_BLK_DEV_ATIIXP is not set
# CONFIG_BLK_DEV_CMD64X is not set
# CONFIG_BLK_DEV_TRIFLEX is not set
# CONFIG_BLK_DEV_CY82C693 is not set
# CONFIG_BLK_DEV_CS5520 is not set
# CONFIG_BLK_DEV_CS5530 is not set
# CONFIG_BLK_DEV_CS5535 is not set
# CONFIG_BLK_DEV_HPT34X is not set
# CONFIG_BLK_DEV_HPT366 is not set
# CONFIG_BLK_DEV_SC1200 is not set
# CONFIG_BLK_DEV_PIIX is not set
# CONFIG_BLK_DEV_IT821X is not set
# CONFIG_BLK_DEV_NS87415 is not set
# CONFIG_BLK_DEV_PDC202XX_OLD is not set
# CONFIG_BLK_DEV_PDC202XX_NEW is not set
# CONFIG_BLK_DEV_SVWKS is not set
# CONFIG_BLK_DEV_SIIMAGE is not set
# CONFIG_BLK_DEV_SIS5513 is not set
# CONFIG_BLK_DEV_SLC90E66 is not set
# CONFIG_BLK_DEV_TRM290 is not set
# CONFIG_BLK_DEV_VIA82CXXX is not set
# CONFIG_IDE_ARM is not set
CONFIG_BLK_DEV_IDEDMA=y
# CONFIG_IDEDMA_IVB is not set
CONFIG_IDEDMA_AUTO=y
# CONFIG_BLK_DEV_HD is not set

#
# SCSI device support
#
# CONFIG_RAID_ATTRS is not set
CONFIG_SCSI=y
# CONFIG_SCSI_PROC_FS is not set

#
# SCSI support type (disk, tape, CD-ROM)
#
CONFIG_BLK_DEV_SD=y
# CONFIG_CHR_DEV_ST is not set
# CONFIG_CHR_DEV_OSST is not set
# CONFIG_BLK_DEV_SR is not set
# CONFIG_CHR_DEV_SG is not set
# CONFIG_CHR_DEV_SCH is not set

#
# Some SCSI devices (e.g. CD jukebox) support multiple LUNs
#
# CONFIG_SCSI_MULTI_LUN is not set
# CONFIG_SCSI_CONSTANTS is not set
# CONFIG_SCSI_LOGGING is not set

#
# SCSI Transport Attributes
#
# CONFIG_SCSI_SPI_ATTRS is not set
# CONFIG_SCSI_FC_ATTRS is not set
# CONFIG_SCSI_ISCSI_ATTRS is not set
# CONFIG_SCSI_SAS_ATTRS is not set
```

```
#
# SCSI low-level drivers
#
# CONFIG_ISCSI_TCP is not set
# CONFIG_BLK_DEV_3W_XXXX_RAID is not set
# CONFIG_SCSI_3W_9XXX is not set
# CONFIG_SCSI_ACARD is not set
# CONFIG_SCSI_AACRAID is not set
# CONFIG_SCSI_AIC7XXX is not set
# CONFIG_SCSI_AIC7XXX_OLD is not set
# CONFIG_SCSI_AIC79XX is not set
# CONFIG_SCSI_DPT_I2O is not set
# CONFIG_SCSI_ADVANSYS is not set
# CONFIG_MEGARAID_NEWGEN is not set
# CONFIG_MEGARAID_LEGACY is not set
# CONFIG_MEGARAID_SAS is not set
# CONFIG_SCSI_SATA is not set
# CONFIG_SCSI_HPTIOP is not set
# CONFIG_SCSI_BUSLOGIC is not set
# CONFIG_SCSI_DMX3191D is not set
# CONFIG_SCSI_EATA is not set
# CONFIG_SCSI_FUTURE_DOMAIN is not set
# CONFIG_SCSI_GDTH is not set
# CONFIG_SCSI_IPS is not set
# CONFIG_SCSI_INITIO is not set
# CONFIG_SCSI_INIA100 is not set
# CONFIG_SCSI_PPA is not set
# CONFIG_SCSI_IMM is not set
# CONFIG_SCSI_SYM53C8XX_2 is not set
# CONFIG_SCSI_IPR is not set
# CONFIG_SCSI_QLOGIC_1280 is not set
# CONFIG_SCSI_QLA_FC is not set
# CONFIG_SCSI_LPFC is not set
# CONFIG_SCSI_DC395x is not set
# CONFIG_SCSI_DC390T is not set
# CONFIG_SCSI_NSP32 is not set
# CONFIG_SCSI_DEBUG is not set

#
# Multi-device support (RAID and LVM)
#
```

```
# CONFIG_MD is not set

#
# Fusion MPT device support
#
# CONFIG_FUSION is not set
# CONFIG_FUSION_SPI is not set
# CONFIG_FUSION_FC is not set
# CONFIG_FUSION_SAS is not set

#
# IEEE 1394 (FireWire) support
#
# CONFIG_IEEE1394 is not set

#
# I2O device support
#
# CONFIG_I2O is not set

#
# Network device support
#
CONFIG_NETDEVICES=y
# CONFIG_DUMMY is not set
# CONFIG_BONDING is not set
# CONFIG_EQUALIZER is not set
# CONFIG_TUN is not set

#
# ARCnet devices
#
# CONFIG_ARCNET is not set

#
# PHY device support
#
# CONFIG_PHYLIB is not set

#
# Ethernet (10 or 100Mbit)
```

```
#
CONFIG_NET_ETHERNET=y
CONFIG_MII=y
# CONFIG_HAPPYMEAL is not set
# CONFIG_SUNGEM is not set
# CONFIG_CASSINI is not set
# CONFIG_NET_VENDOR_3COM is not set

#
# Tulip family network device support
#
# CONFIG_NET_TULIP is not set
# CONFIG_HP100 is not set
CONFIG_NET_PCI=y
# CONFIG_PCNET32 is not set
# CONFIG_AMD8111_ETH is not set
# CONFIG_ADAPTEC_STARFIRE is not set
# CONFIG_B44 is not set
CONFIG_FORCEDETH=y
# CONFIG_DGRS is not set
# CONFIG_EEPRO100 is not set
# CONFIG_E100 is not set
# CONFIG_FEALNX is not set
# CONFIG_NATSEMI is not set
# CONFIG_NE2K_PCI is not set
# CONFIG_8139CP is not set
# CONFIG_8139TOO is not set
# CONFIG_SIS900 is not set
# CONFIG_EPIC100 is not set
# CONFIG_SUNDANCE is not set
# CONFIG_TLAN is not set
# CONFIG_VIA_RHINE is not set
# CONFIG_NET_POCKET is not set

#
# Ethernet (1000 Mbit)
#
# CONFIG_ACENIC is not set
# CONFIG_DL2K is not set
# CONFIG_E1000 is not set
# CONFIG_NS83820 is not set
# CONFIG_HAMACHI is not set
# CONFIG_YELLOWFIN is not set
# CONFIG_R8169 is not set
# CONFIG_SIS190 is not set
# CONFIG_SKGE is not set
# CONFIG_SKY2 is not set
# CONFIG_SK98LIN is not set
# CONFIG_VIA_VELOCITY is not set
# CONFIG_TIGON3 is not set
# CONFIG_BNX2 is not set

#
# Ethernet (10000 Mbit)
#
# CONFIG_CHELSIO_T1 is not set
# CONFIG_IXGB is not set
# CONFIG_S2IO is not set
# CONFIG_MYRI10GE is not set

#
# Token Ring devices
#
# CONFIG_TR is not set

#
# Wireless LAN (non-hamradio)
#
# CONFIG_NET_RADIO is not set

#
# Wan interfaces
#
# CONFIG_WAN is not set
# CONFIG_FDDI is not set
# CONFIG_HIPPI is not set
# CONFIG_PLIP is not set
# CONFIG_PPP is not set
# CONFIG_SLIP is not set
# CONFIG_NET_FC is not set
# CONFIG_SHAPER is not set
# CONFIG_NETCONSOLE is not set
# CONFIG_NETPOLL is not set
# CONFIG_NET_POLL_CONTROLLER is not set

#
# ISDN subsystem
#
# CONFIG_ISDN is not set

#
# Telephony Support
#
# CONFIG_PHONE is not set

#
# Input device support
#
CONFIG_INPUT=y

#
# Userland interfaces
#
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_PSAUX=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1280
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=1024
# CONFIG_INPUT_JOYDEV is not set
# CONFIG_INPUT_TSDEV is not set
# CONFIG_INPUT_EVDEV is not set
# CONFIG_INPUT_EVBUG is not set

#
# Input Device Drivers
#
CONFIG_INPUT_KEYBOARD=y
CONFIG_KEYBOARD_ATKBD=y
# CONFIG_KEYBOARD_SUNKBD is not set
# CONFIG_KEYBOARD_LKKBD is not set
# CONFIG_KEYBOARD_XTKBD is not set
# CONFIG_KEYBOARD_NEWTON is not set
CONFIG_INPUT_MOUSE=y
CONFIG_MOUSE_PS2=y
```

```
# CONFIG_MOUSE_SERIAL is not set
# CONFIG_MOUSE_VSXXXAA is not set
# CONFIG_INPUT_JOYSTICK is not set
# CONFIG_INPUT_TOUCHSCREEN is not set
# CONFIG_INPUT_MISC is not set

#
# Hardware I/O ports
#
CONFIG_SERIO=y
CONFIG_SERIO_I8042=y
# CONFIG_SERIO_SERPORT is not set
# CONFIG_SERIO_CT82C710 is not set
# CONFIG_SERIO_PARKBD is not set
# CONFIG_SERIO_PCIPS2 is not set
CONFIG_SERIO_LIBPS2=y
# CONFIG_SERIO_RAW is not set
# CONFIG_GAMEPORT is not set

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_HW_CONSOLE=y
# CONFIG_VT_HW_CONSOLE_BINDING is not set
# CONFIG_SERIAL_NONSTANDARD is not set

#
# Serial drivers
#
CONFIG_SERIAL_8250=y
# CONFIG_SERIAL_8250_CONSOLE is not set
CONFIG_SERIAL_8250_PCI=y
CONFIG_SERIAL_8250_NR_UARTS=4
CONFIG_SERIAL_8250_RUNTIME_UARTS=4
# CONFIG_SERIAL_8250_EXTENDED is not set

#
# Non-8250 serial port support
#
```

```
CONFIG_SERIAL_CORE=y
# CONFIG_SERIAL_JSM is not set
CONFIG_UNIX98_PTYS=y
CONFIG_LEGACY_PTYS=y
CONFIG_LEGACY_PTY_COUNT=256
CONFIG_PRINTER=y
# CONFIG_LP_CONSOLE is not set
# CONFIG_PPDEV is not set
# CONFIG_TIPAR is not set

#
# IPMI
#
# CONFIG_IPMI_HANDLER is not set

#
# Watchdog Cards
#
# CONFIG_WATCHDOG is not set
# CONFIG_HW_RANDOM is not set
# CONFIG_NVRAM is not set
CONFIG_RTC=y
# CONFIG_DTLK is not set
# CONFIG_R3964 is not set
# CONFIG_APPLICOM is not set
# CONFIG_SONYPI is not set

#
# Ftape, the floppy tape device driver
#
# CONFIG_FTAPE is not set
CONFIG_AGP=y
# CONFIG_AGP_ALI is not set
# CONFIG_AGP_ATI is not set
# CONFIG_AGP_AMD is not set
# CONFIG_AGP_AMD64 is not set
# CONFIG_AGP_INTEL is not set
CONFIG_AGP_NVIDIA=y
# CONFIG_AGP_SIS is not set
# CONFIG_AGP_SWORKS is not set
# CONFIG_AGP_VIA is not set
# CONFIG_AGP_EFFICEON is not set
# CONFIG_DRM is not set
# CONFIG_MWAVE is not set
# CONFIG_PC8736x_GPIO is not set
# CONFIG_NSC_GPIO is not set
# CONFIG_CS5535_GPIO is not set
# CONFIG_RAW_DRIVER is not set
# CONFIG_HANGCHECK_TIMER is not set

#
# TPM devices
#
# CONFIG_TCG_TPM is not set
# CONFIG_TELCLOCK is not set

#
# I2C support
#
CONFIG_I2C=y
CONFIG_I2C_CHARDEV=y

#
# I2C Algorithms
#
CONFIG_I2C_ALGOBIT=y
# CONFIG_I2C_ALGOPCF is not set
# CONFIG_I2C_ALGOPCA is not set

#
# I2C Hardware Bus support
#
# CONFIG_I2C_ALI1535 is not set
# CONFIG_I2C_ALI1563 is not set
# CONFIG_I2C_ALI15X3 is not set
# CONFIG_I2C_AMD756 is not set
# CONFIG_I2C_AMD8111 is not set
# CONFIG_I2C_I801 is not set
# CONFIG_I2C_I810 is not set
# CONFIG_I2C_PIIX4 is not set
CONFIG_I2C_NFORCE2=y
# CONFIG_I2C_OCORES is not set
# CONFIG_I2C_PARPORT is not set
# CONFIG_I2C_PARPORT_LIGHT is not set
# CONFIG_I2C_PROSAVAGE is not set
# CONFIG_I2C_SAVAGE4 is not set
# CONFIG_SCx200_ACB is not set
# CONFIG_I2C_SIS5595 is not set
# CONFIG_I2C_SIS630 is not set
# CONFIG_I2C_SIS96X is not set
# CONFIG_I2C_STUB is not set
# CONFIG_I2C_VIA is not set
# CONFIG_I2C_VIAPRO is not set
# CONFIG_I2C_VOODOO3 is not set
# CONFIG_I2C_PCA_ISA is not set

#
# Miscellaneous I2C Chip support
#
# CONFIG_SENSORS_DS1337 is not set
# CONFIG_SENSORS_DS1374 is not set
# CONFIG_SENSORS_EEPROM is not set
# CONFIG_SENSORS_PCF8574 is not set
# CONFIG_SENSORS_PCA9539 is not set
# CONFIG_SENSORS_PCF8591 is not set
# CONFIG_SENSORS_MAX6875 is not set
# CONFIG_I2C_DEBUG_CORE is not set
# CONFIG_I2C_DEBUG_ALGO is not set
# CONFIG_I2C_DEBUG_BUS is not set
# CONFIG_I2C_DEBUG_CHIP is not set

#
# SPI support
#
# CONFIG_SPI is not set
# CONFIG_SPI_MASTER is not set

#
# Dallas's 1-wire bus
#

#
# Hardware Monitoring support
```

```
#
# CONFIG_HWMON is not set
# CONFIG_HWMON_VID is not set

#
# Misc devices
#
# CONFIG_IBM_ASM is not set

#
# Multimedia devices
#
CONFIG_VIDEO_DEV=y
CONFIG_VIDEO_V4L1=y
CONFIG_VIDEO_V4L1_COMPAT=y
CONFIG_VIDEO_V4L2=y

#
# Video Capture Adapters
#

#
# Video Capture Adapters
#
# CONFIG_VIDEO_ADV_DEBUG is not set
# CONFIG_VIDEO_VIVI is not set
# CONFIG_VIDEO_BT848 is not set
# CONFIG_VIDEO_BWQCAM is not set
# CONFIG_VIDEO_CQCAM is not set
# CONFIG_VIDEO_CPIA is not set
# CONFIG_VIDEO_CPIA2 is not set
# CONFIG_VIDEO_SAA5246A is not set
# CONFIG_VIDEO_SAA5249 is not set
# CONFIG_TUNER_3036 is not set
# CONFIG_VIDEO_STRADIS is not set
# CONFIG_VIDEO_ZORAN is not set
CONFIG_VIDEO_SAA7134=y
CONFIG_VIDEO_SAA7134_ALSA=y
# CONFIG_VIDEO_MXB is not set
# CONFIG_VIDEO_DPC is not set
# CONFIG_VIDEO_HEXIUM_ORION is not set
# CONFIG_VIDEO_HEXIUM_GEMINI is not set
# CONFIG_VIDEO_CX88 is not set

#
# Encoders and Decoders
#
# CONFIG_VIDEO_MSP3400 is not set
# CONFIG_VIDEO_CS53L32A is not set
# CONFIG_VIDEO_TLV320AIC23B is not set
# CONFIG_VIDEO_WM8775 is not set
# CONFIG_VIDEO_WM8739 is not set
# CONFIG_VIDEO_CX2341X is not set
# CONFIG_VIDEO_CX25840 is not set
# CONFIG_VIDEO_SAA711X is not set
# CONFIG_VIDEO_SAA7127 is not set
# CONFIG_VIDEO_UPD64031A is not set
# CONFIG_VIDEO_UPD64083 is not set

#
# V4L USB devices
#
# CONFIG_VIDEO_PVRUSB2 is not set
# CONFIG_VIDEO_EM28XX is not set
# CONFIG_USB_VICAM is not set
# CONFIG_USB_IBMCAM is not set
# CONFIG_USB_KONICAWC is not set
# CONFIG_USB_QUICKCAM_MESSENGER is not set
# CONFIG_USB_ET61X251 is not set
# CONFIG_VIDEO_OVCAMCHIP is not set
# CONFIG_USB_W9968CF is not set
# CONFIG_USB_OV511 is not set
# CONFIG_USB_SE401 is not set
# CONFIG_USB_SN9C102 is not set
# CONFIG_USB_STV680 is not set
# CONFIG_USB_ZC0301 is not set
# CONFIG_USB_PWC is not set

#
# Radio Adapters
#
# CONFIG_RADIO_GEMTEK_PCI is not set
# CONFIG_RADIO_MAXIRADIO is not set
# CONFIG_RADIO_MAESTRO is not set
# CONFIG_USB_DSBR is not set

#
# Digital Video Broadcasting Devices
#
# CONFIG_DVB is not set
CONFIG_VIDEO_TUNER=y
CONFIG_VIDEO_BUF=y
CONFIG_VIDEO_IR=y
# CONFIG_USB_DABUSB is not set

#
# Graphics support
#
CONFIG_FIRMWARE_EDID=y
CONFIG_FB=y
CONFIG_FB_CFB_FILLRECT=y
CONFIG_FB_CFB_COPYAREA=y
CONFIG_FB_CFB_IMAGEBLIT=y
# CONFIG_FB_MACMODES is not set
# CONFIG_FB_BACKLIGHT is not set
CONFIG_FB_MODE_HELPERS=y
# CONFIG_FB_TILEBLITTING is not set
# CONFIG_FB_CIRRUS is not set
# CONFIG_FB_PM2 is not set
# CONFIG_FB_CYBER2000 is not set
# CONFIG_FB_ARC is not set
# CONFIG_FB_ASILIANT is not set
# CONFIG_FB_IMSTT is not set
# CONFIG_FB_VGA16 is not set
# CONFIG_FB_VESA is not set
# CONFIG_FB_HGA is not set
# CONFIG_FB_S1D13XXX is not set
# CONFIG_FB_NVIDIA is not set
CONFIG_FB_RIVA=y
CONFIG_FB_RIVA_I2C=y
# CONFIG_FB_RIVA_DEBUG is not set
# CONFIG_FB_I810 is not set
# CONFIG_FB_INTEL is not set
# CONFIG_FB_MATROX is not set
# CONFIG_FB_RADEON is not set
# CONFIG_FB_ATY128 is not set
# CONFIG_FB_ATY is not set
# CONFIG_FB_SAVAGE is not set
# CONFIG_FB_SIS is not set
# CONFIG_FB_NEOMAGIC is not set
# CONFIG_FB_KYRO is not set
# CONFIG_FB_3DFX is not set
# CONFIG_FB_VOODOO1 is not set
# CONFIG_FB_CYBLA is not set
# CONFIG_FB_TRIDENT is not set
# CONFIG_FB_GEODE is not set
# CONFIG_FB_VIRTUAL is not set

#
# Console display driver support
#
CONFIG_VGA_CONSOLE=y
# CONFIG_VGACON_SOFT_SCROLLBACK is not set
# CONFIG_VIDEO_SELECT is not set
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=y
# CONFIG_FRAMEBUFFER_CONSOLE_ROTATION is not set
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y

#
# Logo configuration
#
# CONFIG_LOGO is not set
# CONFIG_BACKLIGHT_LCD_SUPPORT is not set

#
# Sound
#
CONFIG_SOUND=y

#
# Advanced Linux Sound Architecture
```

```
#
CONFIG_SND=y
CONFIG_SND_TIMER=y
CONFIG_SND_PCM=y
CONFIG_SND_SEQUENCER=y
# CONFIG_SND_SEQ_DUMMY is not set
# CONFIG_SND_MIXER_OSS is not set
# CONFIG_SND_PCM_OSS is not set
# CONFIG_SND_SEQUENCER_OSS is not set
CONFIG_SND_RTCTIMER=y
CONFIG_SND_SEQ_RTCTIMER_DEFAULT=y
# CONFIG_SND_DYNAMIC_MINORS is not set
# CONFIG_SND_SUPPORT_OLD_API is not set
# CONFIG_SND_VERBOSE_PROCFS is not set
# CONFIG_SND_VERBOSE_PRINTK is not set
# CONFIG_SND_DEBUG is not set

#
# Generic devices
#
CONFIG_SND_AC97_CODEC=y
CONFIG_SND_AC97_BUS=y
# CONFIG_SND_DUMMY is not set
# CONFIG_SND_VIRMIDI is not set
# CONFIG_SND_MTPAV is not set
# CONFIG_SND_SERIAL_U16550 is not set
# CONFIG_SND_MPU401 is not set

#
# PCI devices
#
# CONFIG_SND_AD1889 is not set
# CONFIG_SND_ALS300 is not set
# CONFIG_SND_ALS4000 is not set
# CONFIG_SND_ALI5451 is not set
# CONFIG_SND_ATIIXP is not set
# CONFIG_SND_ATIIXP_MODEM is not set
# CONFIG_SND_AU8810 is not set
# CONFIG_SND_AU8820 is not set
# CONFIG_SND_AU8830 is not set
# CONFIG_SND_AZT3328 is not set
# CONFIG_SND_BT87X is not set
# CONFIG_SND_CA0106 is not set
# CONFIG_SND_CMIPCI is not set
# CONFIG_SND_CS4281 is not set
# CONFIG_SND_CS46XX is not set
# CONFIG_SND_CS5535AUDIO is not set
# CONFIG_SND_EMU10K1 is not set
# CONFIG_SND_EMU10K1X is not set
# CONFIG_SND_ENS1370 is not set
# CONFIG_SND_ENS1371 is not set
# CONFIG_SND_ES1938 is not set
# CONFIG_SND_ES1968 is not set
# CONFIG_SND_FM801 is not set
# CONFIG_SND_HDA_INTEL is not set
# CONFIG_SND_HDSP is not set
# CONFIG_SND_HDSPM is not set
# CONFIG_SND_ICE1712 is not set
# CONFIG_SND_ICE1724 is not set
CONFIG_SND_INTEL8X0=y
# CONFIG_SND_INTEL8X0M is not set
# CONFIG_SND_KORG1212 is not set
# CONFIG_SND_MAESTRO3 is not set
# CONFIG_SND_MIXART is not set
# CONFIG_SND_NM256 is not set
# CONFIG_SND_PCXHR is not set
# CONFIG_SND_RME32 is not set
# CONFIG_SND_RME96 is not set
# CONFIG_SND_RME9652 is not set
# CONFIG_SND_SONICVIBES is not set
# CONFIG_SND_TRIDENT is not set
# CONFIG_SND_VIA82XX is not set
# CONFIG_SND_VIA82XX_MODEM is not set
# CONFIG_SND_VX222 is not set
# CONFIG_SND_YMFPCI is not set

#
# USB devices
#
# CONFIG_SND_USB_AUDIO is not set
# CONFIG_SND_USB_USX2Y is not set
```

```
#
# Open Sound System
#
# CONFIG_SOUND_PRIME is not set

#
# USB support
#
CONFIG_USB_ARCH_HAS_HCD=y
CONFIG_USB_ARCH_HAS_OHCI=y
CONFIG_USB_ARCH_HAS_EHCI=y
CONFIG_USB=y
# CONFIG_USB_DEBUG is not set

#
# Miscellaneous USB options
#
CONFIG_USB_DEVICEFS=y
# CONFIG_USB_BANDWIDTH is not set
# CONFIG_USB_DYNAMIC_MINORS is not set
# CONFIG_USB_OTG is not set

#
# USB Host Controller Drivers
#
CONFIG_USB_EHCI_HCD=y
# CONFIG_USB_EHCI_SPLIT_ISO is not set
# CONFIG_USB_EHCI_ROOT_HUB_TT is not set
# CONFIG_USB_EHCI_TT_NEWSCHED is not set
# CONFIG_USB_ISP116X_HCD is not set
CONFIG_USB_OHCI_HCD=y
# CONFIG_USB_OHCI_BIG_ENDIAN is not set
CONFIG_USB_OHCI_LITTLE_ENDIAN=y
# CONFIG_USB_UHCI_HCD is not set
# CONFIG_USB_SL811_HCD is not set

#
# USB Device Class drivers
#
# CONFIG_USB_ACM is not set
# CONFIG_USB_PRINTER is not set
```

```
#
# NOTE: USB_STORAGE enables SCSI, and 'SCSI disk support'
#

#
# may also be needed; see USB_STORAGE Help for more information
#
CONFIG_USB_STORAGE=y
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_DATAFAB is not set
# CONFIG_USB_STORAGE_FREECOM is not set
# CONFIG_USB_STORAGE_ISD200 is not set
# CONFIG_USB_STORAGE_DPCM is not set
# CONFIG_USB_STORAGE_USBAT is not set
# CONFIG_USB_STORAGE_SDDR09 is not set
# CONFIG_USB_STORAGE_SDDR55 is not set
# CONFIG_USB_STORAGE_JUMPSHOT is not set
# CONFIG_USB_STORAGE_ALAUDA is not set
# CONFIG_USB_LIBUSUAL is not set

#
# USB Input Devices
#
# CONFIG_USB_HID is not set

#
# USB HID Boot Protocol drivers
#
# CONFIG_USB_KBD is not set
# CONFIG_USB_MOUSE is not set
# CONFIG_USB_AIPTEK is not set
# CONFIG_USB_WACOM is not set
# CONFIG_USB_ACECAD is not set
# CONFIG_USB_KBTAB is not set
# CONFIG_USB_POWERMATE is not set
# CONFIG_USB_TOUCHSCREEN is not set
# CONFIG_USB_YEALINK is not set
# CONFIG_USB_XPAD is not set
# CONFIG_USB_ATI_REMOTE is not set
# CONFIG_USB_ATI_REMOTE2 is not set
# CONFIG_USB_KEYSPAN_REMOTE is not set
# CONFIG_USB_APPLETOUCH is not set

#
# USB Imaging devices
#
# CONFIG_USB_MDC800 is not set
# CONFIG_USB_MICROTEK is not set

#
# USB Network Adapters
#
# CONFIG_USB_CATC is not set
# CONFIG_USB_KAWETH is not set
# CONFIG_USB_PEGASUS is not set
# CONFIG_USB_RTL8150 is not set
# CONFIG_USB_USBNET is not set
# CONFIG_USB_MON is not set

#
# USB port drivers
#
# CONFIG_USB_USS720 is not set

#
# USB Serial Converter support
#
# CONFIG_USB_SERIAL is not set

#
# USB Miscellaneous drivers
#
# CONFIG_USB_EMI62 is not set
# CONFIG_USB_EMI26 is not set
# CONFIG_USB_AUERSWALD is not set
# CONFIG_USB_RIO500 is not set
# CONFIG_USB_LEGOTOWER is not set
# CONFIG_USB_LCD is not set
# CONFIG_USB_LED is not set
# CONFIG_USB_CYPRESS_CY7C63 is not set
# CONFIG_USB_CYTHERM is not set
# CONFIG_USB_PHIDGETKIT is not set
# CONFIG_USB_PHIDGETSERVO is not set
# CONFIG_USB_IDMOUSE is not set
# CONFIG_USB_APPLEDISPLAY is not set
# CONFIG_USB_SISUSBVGA is not set
# CONFIG_USB_LD is not set
# CONFIG_USB_TEST is not set

#
# USB DSL modem support
#

#
# USB Gadget Support
#
# CONFIG_USB_GADGET is not set

#
# MMC/SD Card support
#
# CONFIG_MMC is not set

#
# LED devices
#
# CONFIG_NEW_LEDS is not set

#
# LED drivers
#

#
# LED Triggers
#

#
# InfiniBand support
#
# CONFIG_INFINIBAND is not set

#
```

```
# EDAC - error detection and reporting (RAS) (EXPERIMENTAL)
#
# CONFIG_EDAC is not set

#
# Real Time Clock
#
# CONFIG_RTC_CLASS is not set

#
# DMA Engine support
#
# CONFIG_DMA_ENGINE is not set

#
# DMA Clients
#

#
# DMA Devices
#

#
# File systems
#
CONFIG_EXT2_FS=y
# CONFIG_EXT2_FS_XATTR is not set
# CONFIG_EXT2_FS_XIP is not set
CONFIG_EXT3_FS=y
# CONFIG_EXT3_FS_XATTR is not set
CONFIG_JBD=y
# CONFIG_JBD_DEBUG is not set
# CONFIG_REISERFS_FS is not set
# CONFIG_JFS_FS is not set
# CONFIG_FS_POSIX_ACL is not set
# CONFIG_XFS_FS is not set
# CONFIG_OCFS2_FS is not set
# CONFIG_MINIX_FS is not set
# CONFIG_ROMFS_FS is not set
# CONFIG_INOTIFY is not set
# CONFIG_QUOTA is not set
```

```
CONFIG_DNOTIFY=y
# CONFIG_AUTOFS_FS is not set
# CONFIG_AUTOFS4_FS is not set
# CONFIG_FUSE_FS is not set

#
# CD-ROM/DVD Filesystems
#
CONFIG_ISO9660_FS=y
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_ZISOFS_FS=y
# CONFIG_UDF_FS is not set

#
# DOS/FAT/NT Filesystems
#
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_FAT_DEFAULT_CODEPAGE=850
CONFIG_FAT_DEFAULT_IOCHARSET="iso8859-1"
# CONFIG_NTFS_FS is not set

#
# Pseudo filesystems
#
CONFIG_PROC_FS=y
CONFIG_PROC_KCORE=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
# CONFIG_HUGETLBFS is not set
# CONFIG_HUGETLB_PAGE is not set
CONFIG_RAMFS=y
CONFIG_CONFIGFS_FS=y

#
# Miscellaneous filesystems
#
# CONFIG_ADFS_FS is not set
# CONFIG_AFFS_FS is not set
# CONFIG_HFS_FS is not set
# CONFIG_HFSPLUS_FS is not set
# CONFIG_BEFS_FS is not set
# CONFIG_BFS_FS is not set
# CONFIG_EFS_FS is not set
# CONFIG_CRAMFS is not set
# CONFIG_VXFS_FS is not set
# CONFIG_HPFS_FS is not set
# CONFIG_QNX4FS_FS is not set
# CONFIG_SYSV_FS is not set
# CONFIG_UFS_FS is not set

#
# Network File Systems
#
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
# CONFIG_NFS_V3_ACL is not set
# CONFIG_NFS_V4 is not set
# CONFIG_NFS_DIRECTIO is not set
# CONFIG_NFSD is not set
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_NFS_COMMON=y
CONFIG_SUNRPC=y
# CONFIG_RPCSEC_GSS_KRB5 is not set
# CONFIG_RPCSEC_GSS_SPKM3 is not set
CONFIG_SMB_FS=y
# CONFIG_SMB_NLS_DEFAULT is not set
CONFIG_CIFS=y
# CONFIG_CIFS_STATS is not set
# CONFIG_CIFS_WEAK_PW_HASH is not set
# CONFIG_CIFS_XATTR is not set
# CONFIG_CIFS_DEBUG2 is not set
# CONFIG_CIFS_EXPERIMENTAL is not set
# CONFIG_NCP_FS is not set
CONFIG_CODA_FS=y
# CONFIG_CODA_FS_OLD_API is not set
# CONFIG_AFS_FS is not set
# CONFIG_9P_FS is not set
```

```
#
# Partition Types
#
# CONFIG_PARTITION_ADVANCED is not set
CONFIG_MSDOS_PARTITION=y

#
# Native Language Support
#
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-15"
# CONFIG_NLS_CODEPAGE_437 is not set
# CONFIG_NLS_CODEPAGE_737 is not set
# CONFIG_NLS_CODEPAGE_775 is not set
CONFIG_NLS_CODEPAGE_850=y
# CONFIG_NLS_CODEPAGE_852 is not set
# CONFIG_NLS_CODEPAGE_855 is not set
# CONFIG_NLS_CODEPAGE_857 is not set
# CONFIG_NLS_CODEPAGE_860 is not set
# CONFIG_NLS_CODEPAGE_861 is not set
# CONFIG_NLS_CODEPAGE_862 is not set
# CONFIG_NLS_CODEPAGE_863 is not set
# CONFIG_NLS_CODEPAGE_864 is not set
# CONFIG_NLS_CODEPAGE_865 is not set
# CONFIG_NLS_CODEPAGE_866 is not set
# CONFIG_NLS_CODEPAGE_869 is not set
# CONFIG_NLS_CODEPAGE_936 is not set
# CONFIG_NLS_CODEPAGE_950 is not set
# CONFIG_NLS_CODEPAGE_932 is not set
# CONFIG_NLS_CODEPAGE_949 is not set
# CONFIG_NLS_CODEPAGE_874 is not set
# CONFIG_NLS_ISO8859_8 is not set
# CONFIG_NLS_CODEPAGE_1250 is not set
# CONFIG_NLS_CODEPAGE_1251 is not set
# CONFIG_NLS_ASCII is not set
CONFIG_NLS_ISO8859_1=y
# CONFIG_NLS_ISO8859_2 is not set
# CONFIG_NLS_ISO8859_3 is not set
# CONFIG_NLS_ISO8859_4 is not set
# CONFIG_NLS_ISO8859_5 is not set
# CONFIG_NLS_ISO8859_6 is not set
# CONFIG_NLS_ISO8859_7 is not set
# CONFIG_NLS_ISO8859_9 is not set
# CONFIG_NLS_ISO8859_13 is not set
# CONFIG_NLS_ISO8859_14 is not set
CONFIG_NLS_ISO8859_15=y
# CONFIG_NLS_KOI8_R is not set
# CONFIG_NLS_KOI8_U is not set
CONFIG_NLS_UTF8=y

#
# Instrumentation Support
#
CONFIG_PROFILING=y
CONFIG_OPROFILE=y
CONFIG_KPROBES=y

#
# Kernel hacking
#
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
# CONFIG_PRINTK_TIME is not set
CONFIG_MAGIC_SYSRQ=y
# CONFIG_UNUSED_SYMBOLS is not set
CONFIG_DEBUG_KERNEL=y
CONFIG_LOG_BUF_SHIFT=14
# CONFIG_DETECT_SOFTLOCKUP is not set
CONFIG_SCHEDSTATS=y
# CONFIG_DEBUG_SLAB is not set
# CONFIG_DEBUG_PREEMPT is not set
# CONFIG_DEBUG_RT_MUTEXES is not set
# CONFIG_RT_MUTEX_TESTER is not set
# CONFIG_DEBUG_SPINLOCK is not set
# CONFIG_DEBUG_MUTEXES is not set
# CONFIG_DEBUG_RWSEMS is not set
# CONFIG_DEBUG_LOCK_ALLOC is not set
# CONFIG_PROVE_LOCKING is not set
# CONFIG_DEBUG_SPINLOCK_SLEEP is not set
# CONFIG_DEBUG_LOCKING_API_SELFTESTS is not set
# CONFIG_DEBUG_KOBJECT is not set
CONFIG_DEBUG_BUGVERBOSE=y
# CONFIG_DEBUG_INFO is not set
```

```
CONFIG_DEBUG_FS=y
# CONFIG_DEBUG_VM is not set
# CONFIG_FRAME_POINTER is not set
CONFIG_UNWIND_INFO=y
CONFIG_STACK_UNWIND=y
CONFIG_FORCED_INLINING=y
# CONFIG_RCU_TORTURE_TEST is not set
CONFIG_EARLY_PRINTK=y
# CONFIG_DEBUG_STACKOVERFLOW is not set
# CONFIG_DEBUG_STACK_USAGE is not set
# CONFIG_DEBUG_PAGEALLOC is not set
# CONFIG_DEBUG_RODATA is not set
# CONFIG_4KSTACKS is not set
CONFIG_X86_FIND_SMP_CONFIG=y
CONFIG_X86_MPPARSE=y
CONFIG_DOUBLEFAULT=y

#
# Security options
#
# CONFIG_KEYS is not set
# CONFIG_SECURITY is not set

#
# Cryptographic options
#
CONFIG_CRYPTO=y
CONFIG_CRYPTO_HMAC=y
# CONFIG_CRYPTO_NULL is not set
# CONFIG_CRYPTO_MD4 is not set
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_SHA1=y
# CONFIG_CRYPTO_SHA256 is not set
# CONFIG_CRYPTO_SHA512 is not set
# CONFIG_CRYPTO_WP512 is not set
# CONFIG_CRYPTO_TGR192 is not set
CONFIG_CRYPTO_DES=y
# CONFIG_CRYPTO_BLOWFISH is not set
# CONFIG_CRYPTO_TWOFISH is not set
# CONFIG_CRYPTO_SERPENT is not set
# CONFIG_CRYPTO_AES is not set
# CONFIG_CRYPTO_AES_586 is not set
# CONFIG_CRYPTO_CAST5 is not set
# CONFIG_CRYPTO_CAST6 is not set
# CONFIG_CRYPTO_TEA is not set
# CONFIG_CRYPTO_ARC4 is not set
# CONFIG_CRYPTO_KHAZAD is not set
# CONFIG_CRYPTO_ANUBIS is not set
CONFIG_CRYPTO_DEFLATE=y
# CONFIG_CRYPTO_MICHAEL_MIC is not set
# CONFIG_CRYPTO_CRC32C is not set
# CONFIG_CRYPTO_TEST is not set

#
# Hardware crypto devices
#
# CONFIG_CRYPTO_DEV_PADLOCK is not set

#
# Library routines
#
# CONFIG_CRC_CCITT is not set
# CONFIG_CRC16 is not set
CONFIG_CRC32=y
# CONFIG_LIBCRC32C is not set
CONFIG_ZLIB_INFLATE=y
CONFIG_ZLIB_DEFLATE=y
CONFIG_PLIST=y
CONFIG_GENERIC_HARDIRQS=y
CONFIG_GENERIC_IRQ_PROBE=y
CONFIG_X86_BIOS_REBOOT=y
CONFIG_KTIME_SCALAR=y
```

Appendix B

Workload program

```
#!/usr/bin/perl
#
# workload - Starts a workload for the thesis
#

use strict;
use warnings;

use Config; # for signal numbers
use Time::HiRes qw(gettimeofday);

use Readonly; # Easy read-only variables (much better than "use
              # constant")

use Cwd;

usage() unless (@ARGV == 3);

# <player> - video player (mplayer or xine
# <other> - chess or cpuhog
# <num> - number of chess or cpuhog to run
my ($player, $other, $count) = @ARGV;

Readonly my $MOVIE_FILE => $ENV{MOVIE_FILE} || '/tmp/Lecture-1a-5min.mpg';
Readonly my $RES_GROUP => $ENV{RES_GROPU} || 'mm';
Readonly my $CPU_SHARE => $ENV{CPU_SHARE} || 'res=cpu,min_shares=80';
my $workdir =  getcwd;
Readonly my $RESULTS_DIR_PREFIX => $ENV{RESULTS_DIR_PREFIX} ||
  "$workdir/results";

my $video_playback_command = $ENV{VIDEO_PLAYBACK_COMMAND} || $player;

# these are used to measure the time used by the workload.
my $start_time = 0;
my $end_time = 0;

# create measurements dir if it doesn't exist already
unless (-d $RESULTS_DIR_PREFIX) {
  mkdir $RESULTS_DIR_PREFIX or
    die "can't make directory $RESULTS_DIR_PREFIX: $!";
}

# Add /usr/games to path (so that xboard and gnuchess can be started)
$ENV{PATH} .= ':/usr/games/bin';

# Get signal names and numbers (see perldoc Config)
my %sig_num;
my @sig_name;

unless ($Config{sig_name} && $Config{sig_num}) {
  die "No sigs?";
} else {
  my @names = split ' ', $Config{sig_name};
  @sig_num{@names} = split ' ', $Config{sig_num};
  foreach (@names) {
    $sig_name[$sig_num{$_}] ||= $_;
  }
}

init_profiler();
init_workload();
start_profiler();
$start_time = gettimeofday();
run_workload();
$end_time = gettimeofday();
save_results();
cleanup_workload();
cleanup_profiler();
```

```
### Subs ###
sub usage {
    print "workload <mplayer or xine> <chess or cpuhog> <count>\n";
    exit 0;
}

sub run_program {
system(@_) == 0
    or die "Program exited badly <@_>: $?";
}

sub run_xine {
    # Argh! xine plays the movie and then dies with a segfault when
    # *told* (via. --auto-play=q) to quit after playback!!!
    # this sub is a ugly workaround: It does not check for errors...x
    system(@_);
}

# Run program passed (with args) as a separate process, return its pid.
sub fork_program {
  $| = 1; # just to be safe (see perldoc -f fork)
  my $pid = fork;

  die "fork was unsuccessful" if (!defined $pid);

  if ($pid == 0) {
    # we're in the child
    exec(@_)
  }

  return $pid;
}

# fork, register in resource group and *then* run the
# program.. Makes sure we are in the correct resource group
# before launching the program
sub fork_program_resreg {
  $| = 1; # just to be safe (see perldoc -f fork)
  my $pid = fork;

  die "fork was unsuccessful" if (!defined $pid);
  if ($pid == 0) {
    # we're in the child
    # register us in the proper resource group
    my $fh;
    open($fh, ">/config/res_groups/$RES_GROUP/members")
      or die "Cannot write to class member file: $!";
    print $fh "$$\n"; # add child pid to members
    close($fh);
    exec(@_) # liftoff!

  }

  return $pid;
}

sub have_ckrm {
  my $extraver = kernel_version();

  return ($extraver eq '2.6.18-ckrm') || ($extraver eq '2.6.18-mongrel');
}

sub kernel_version {
  chomp(my $output = `uname -r`);
  return $output;
}

sub run_workload {
  # let the video playback determine the length of the test

  start_other_work($other, $count);

  start_video(); # will wait until its done
  # end the other children in the process group (except ourselves)
  {
    local $SIG{HUP} = 'IGNORE';
    kill HUP => -$$;
  }
}

sub start_video {
```

```
# Bah! The video program xine needs this extra option to
# automatically quit after playback: --auto-play=q. And that option
# makes it die from a SIGSEGV when done! Therefore we need to give it
# special treatment here... Mplayer will ignore the empty argument.
  my $is_xine = ($video_playback_command =~ /^xine/) ? 1 : 0;
  my $quit_option = $is_xine ? '--auto-play=q' : '';

  if (have_ckrm()) {
    my $pid = fork_program_resreg($video_playback_command,
  $quit_option,
  $MOVIE_FILE);
    waitpid($pid, 0); # wait for it to finish...
  } else {
    # Set it up to run with RT priority
    # We can just wait for this one ..
    if ($is_xine) {
# oops, its xine.. do the ugly thing
run_xine('chrt', '--rr', 99,
 $video_playback_command,
 $quit_option,
 $MOVIE_FILE);
    } else {
  run_program('chrt', '--rr', 99,
      $video_playback_command,
      $MOVIE_FILE);
    }
    # modify for settings file
    $video_playback_command = "chrt --rr 99 $video_playback_command";
  }
}

sub start_other_work {
  my ($arg, $count) = @_;
  my @prog;
  if ($arg eq 'chess') {
    @prog = ('xboard', '-matchGames', '100', '-iconic');
  } else {
    @prog = ('./cpuhog');
  }
   # Let gnuchess play with itself for a looong time
  # This process will get killed later.
  my $i = 0;
  my $pid;
  while ($i < $count) {
    $pid = fork_program(@prog);
    if (!defined($pid)) {
      die "Fork failed: $!\n";
    }
    ++$i;
  }
}

sub save_results {
  # create a results dir every time, so that old results never get
  # overwritten. Append a new number to the name.

  my $count = 0;
  my $newdir = $RESULTS_DIR_PREFIX . "/measurements-$count";
  # find next available dir name (slightly dangerous approach)
  while (-d $newdir) {
    ++$count;
    $newdir = $RESULTS_DIR_PREFIX . "/measurements-$count";
  }

  mkdir $newdir or die "Can't make dir $newdir: $!";

  save_usermode_profile($newdir);
  save_kernelmode_profile($newdir);

  # now save the settings we ran with
  my $fh;
  open($fh, ">$newdir/settings.txt")
    or die "Can't open new file $newdir/settings: $!";

  print $fh "start time: $start_time\n";
  print $fh "end time: $end_time\n";
  print $fh "elapsed time: ", ($end_time - $start_time),"\n";

  print $fh "moviefile: $MOVIE_FILE\n";
  print $fh "video playback command: $video_playback_command\n";
  if (have_ckrm()) {
    print $fh "resource group: $RES_GROUP\n";
    print $fh "cpu_share: $CPU_SHARE\n";
  }
  close($fh);
}

sub save_usermode_profile {
  my $dir = shift;
  my $kernel_version = kernel_version();
  my $user_oprof_out = `opreport`;
  my $taskout = $dir . "/tasks-${kernel_version}.oprofile";
  my $fh;
  open($fh, ">$taskout") or die "Can't open $taskout: $!";
  print $fh $user_oprof_out;
  close($fh);
}

sub save_kernelmode_profile {
  my $dir = shift;
  my $kernel_version = kernel_version();
  my $kernel_oprof_out = `readprofile -m /boot/System.map-$kernel_version`;
  my $kernelout = $dir . "/kernel-${kernel_version}.oprofile";
  my $fh;
  open($fh, ">$kernelout") or die "Can't open $kernelout: $!";
  print $fh $kernel_oprof_out;
  close($fh);
}

sub init_profiler {

  # mount /boot unless it has already been done
  -d '/boot/grub' or run_program('mount', '/boot');

  # set up OProfile
  run_program('opcontrol', '--setup', "--vmlinux=/boot/vmlinux-" . `uname -r`);

  # failure is an option here...
  system('opcontrol', '--reset');
}

sub start_profiler {
  run_program('opcontrol', '--start');
```

```
}

sub cleanup_profiler {
  # failure is an option here...
  system('opcontrol', '--shutdown');
}

sub init_workload {
  # do we have a class-based resource managment patch?
  if (have_ckrm()) {

    # is the configfs filesystem mounted on /config ?
    unless (-d "/config/res_groups") {
      mount_configfs();
    }

    # is there a class for the video playback application?
    unless (-d "/config/res_groups/$RES_GROUP") {
      # create it
      mkdir "/config/res_groups/$RES_GROUP"
or die "Can\'t make resource group $RES_GROUP: $!"
    }

    # configure the CPU share
    my $shares_filename = "/config/res_groups/$RES_GROUP/shares";
    my $fh;
    open($fh, ">$shares_filename") or die "Cannot open: $shares_filename: $!";
    print $fh $CPU_SHARE;
    close($fh);
    print "Done initalizing the workload\n";
  }
}

sub cleanup_workload {
  if (have_ckrm()) {
    rmdir "/config/res_groups/$RES_GROUP" or
      die "Cannot remove resouce group $RES_GROUP: $!"
    }
}

sub mount_configfs {
  run_program('/bin/mount', '-t', 'configfs', 'none', '/config');
}

__END__

=head1 NAME

workload - Starts a workload for the thesis

=head1 SYNOPSIS

B<workload> <mplayer or xine> <chess or cpuhog> <count>

=head1 DESCRIPTION

Looks at the output of 'uname -r' to determine how to set up the
workload. Stores the measurements obtained in results under a new subdirectory.
Count is the numnber of chess or cpuhogs to run.

=head1 DEPENDENCIES

Needs the following CPAN modules:
Readonly

Needs the following programs:
which

mplayer (this can be overridden with the VIDEO_PLAYBACK_COMMAND
 environment variable)

=head1 BUGS

Error checking is very rudimentary.

=head1 AUTHOR

Martin Setek <martitse@ifi.uio.no>

=cut
```