Scheduler Design

Project

The aim of this lab is to teach you about process scheduling. For this, you will use Bossa, a framework for developing kernel schedulers.

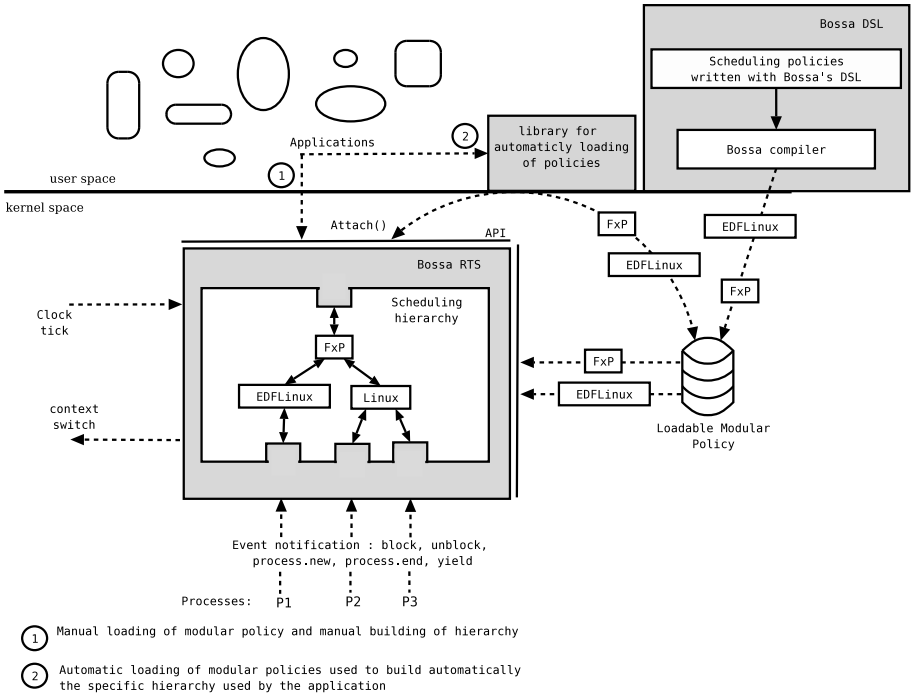


Figure 1: Bossa Framework & DSL

Requirements

This section describes the use of Bossa with the lab CD. If you would prefer to install Bossa and the associated tools directly on your machine, please skip to the next section.

On the lab CD you will find an archive that contains:

1. *The Bossa Linux kernel v2.4.24 archive.* This archive contains a Linux kernel modified according to the kernel design shown in Figure 1.

2. *the bossa_hourglass archive.* **Bossa_hourglass** is a flexible, open source (BSD-style license) utility for monitoring application scheduling behavior for Linux, FreeBSD, and

Windows 2000 running on x86 chips. It supports the standard versions of these systems as well as some real-time variants.

The basic mode of operation for Hourglass is to create a number of threads, each with a particular set of QoS parameters (such as a share, reservation, or priority) and a particular set of requirements, such as meeting a minimum specified frame rate. Each thread traces its execution, monitoring when it is scheduled and when it is not, allowing a precise record of thread execution to be produced. The trace helps the scheduler developer understand the behavior of a policy, for example, determining whether deadlines are missed and why.

3. *The jgraph archive.* **Jgraph** is a tool for drawing graphs and structured drawings. Unlike most drawing tools, jgraph is non-interactive – its takes an input file, and produces either plain or encapsulated PostScript as output. This output may be viewed on the screen (using ghostscript or ghostview), printed on a printer (using lpr), embedded into other formatters such as LaTeX or framemaker, or converted into other formats (e.g. ppm or gif).

4. *The shell script bossa_install.* This script helps you:

   (a) to compile a bossa scheduling policy (policy.bossa ⇒ policy.c). **Note** that bossa_installer creates a *c-code* directory that contains all policy.c files and stub files.

   (b) to compile the file policy.c as a kernel module (policy.c ⇒ policy.o).

   (c) to configure the file /etc/modules.conf[1] and to load the policy into the kernel.

5. *The binary file manager.* **Manager** is an application to manage the schedulers. With this application you can install or uninstall schedulers in a scheduling hierarchy.

6. *Four policies*

   (a) *Linux.bossa.* This policy is the same as the scheduling policy that you can find in Linux kernel v2.4. For more details about it see exercise 2.

   (b) *EDFu.bossa* (Earliest Deadline First). This policy is just a skeleton that will be used in exercise 3.

[1] used for the automatic loading of kernel modules

(c) *Fixed_priority.bossa*. This policy is a **Virtual scheduler**. A virtual scheduler manages other schedulers in a scheduling hierarchy. In contrast, a scheduler that manages processes directly is known as a **Process scheduler** (Linux and EDFu are process schedulers). Fixed_priority gives the processor to the active child scheduler that has the highest priority. When you add Fixed_priority to the hierarchy as the parent of some scheduler already in the hierarchy, the manager asks for the priority of the existing scheduler. Similarly, when you attach a new scheduler as a child of Fixed_priority, the manager asks for the priority of this new scheduler. Any integer value can be used for these priorities.

(d) *Proportion.bossa*. Proportion is a virtual scheduler that gives the processor to process schedulers according to their proportion. When you add Proportion to the hierarchy as the parent of some scheduler already in the hierarchy, the manager asks for the proportion of CPU time to allocate to the existing scheduler. Similarly, when you attach a new scheduler as a child of Proportion, the manager asks for the proportion of CPU time to allocate to this new scheduler. The proportion should be a value between 1 and 100.

7. *Mplayer*. Mplayer is a video player for Linux. This version of Mplayer has been modified for use with Bossa.

Installation

This section describes how to install Bossa directly on your machine. If you are using the lab CD, please skip to the next section.

Before starting your experiments, you must install Bossa as follows:

1. Install a Linux operating system.
2. Install the Bossa kernel
   (a) Untar the Bossa kernel archive using the command `tar -xvzf [bossa_file_name]`.
   (b) Clean the Bossa kernel sources using the command `make clean`.
   (c) Configure the Bossa kernel using the command `make xconfig`. You must choose:
      - **Y** for the one item in the menu **Code maturity level options**

- **Y** in the menu **File systems** for the item **/dev file system support**.
- **Y** in the menu **Bossa schedulers** for the items **Kernel support for Bossa scheduler** and **Centralized event implementation**.
- **N** in the menu **Bossa schedulers** for all policies.

(d) compile the Bossa kernel sources

i. **make dep**

ii. **make install**

iii. **make modules**

iv. **make modules_install**

Make install installs the compiler and other utility files automatically. Edit the beginning of the Makefile at the root of the kernel source tree to indicate the directory in which these files should be installed and your version of Linux.

(e) install the bossa kernel (`cp arch/i386/boot/bzImage /boot/vmlinuz-bossa`)

(f) update the boot loader. For lilo:

i. edit the file **/etc/lilo.conf**

ii. create an entry for your kernel image, save and exit

iii. run the command `lilo` at the shell prompt

(g) close all applications, restart your computer and boot bossa.

3. Install the provided policies

(a) Normally, the shell script **bossa_install** should be configured automatically by **make install**. For reference, the various variables of this script should be initialized as follows:

- **BOSSA_HOME**: The directory of the Bossa Linux source tree.
- **HR_BOSSA_HOME**: The directory of the source tree of Bossa Linux with high resolution timers. **Note** this is the lab version of Bossa.
- **BOSSA**: name of the Bossa kernel (`uname -r`)
- **HR_BOSSA**: name of the Bossa high resolution timer kernel (`uname -r`)
- **COMPILER_HOME**: The directory of the Bossa compiler, **cb.opt**.

- **LINUX_VERSION**: eg mdk, suze, debian ...

The lab version of Bossa is the version with high-resolution timers. Thus, it is important that the variables **HR_BOSSA_HOME** and **HR_BOSSA** be set correctly. The variables **BOSSA_HOME** and **BOSSA** are not relevant for the lab, and should be set to any (different) value.

(b) run **bossa_install**[2] with the Proportion policy and the Fixed_priority policy. For example: `./bossa_install Fixed_priority`

4. Install jgraph

   (a) untar the **jgraph** archive

   (b) compile **jgraph** (typing make)

   (c) copy the **jgraph** binary into /usr/local/bin (`cp jgraph /usr/local/bin`)

5. Install bossa_hourglass

   (a) untar the **bossa_hourglass** archive

   (b) **./configure**

   (c) **make**

   (d) **make install**

Using Bossa

Exercise 1:

The aim of this exercise is to build the scheduling hierarchy shown in Figure 2 and to verify its behavior. The required behavior is that the proper proportion of CPU time is given to each of the two Linux schedulers, i.e., 20% for the scheduler on the left and 80% for the scheduler on the right. An implementation of the Proportion scheduler is provided. In this implementation a child scheduler gets at least the proportion of the CPU specified; more if some CPU time is not reserved or if some reserved time is not used. Furthermore, the reservations are by default implemented within a period of 10 ticks, meaning that percentages are rounded to

[2]Don't forget to check that **bossa_install** and the bossa compiler **cb.opt** are executable (_ _ x). To change the permissions of these use the command **chmod 755 [file-name]**.

the nearest multiple of 10, according to C rounding conventions for integer division. To carry out this exercise, you must:

1. make a clone of Linux policy
2. use **manager** to create the hierarchy
3. write a program to test the hierarchy and proportions

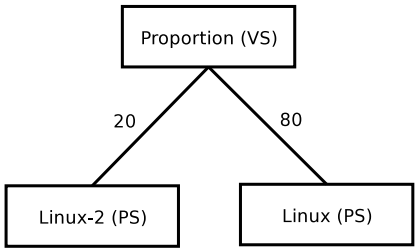


Figure 2: Hierarchy

Default scheduler: When you boot Bossa, all processes are scheduled by a default scheduler. In your case, this default scheduler is Linux. So any new program is scheduled by Linux. Consequently, it is not necessary to attach the program on the Linux scheduler.

Hierarchy checking: The directory **/proc/bossa** contains information about the current scheduling hierarchy, and thus allows you to determine whether you have used the manager correctly. There is a file **policy.info** for each policy that is currently loaded. This file contains information about all of the schedulers (for a virtual scheduler) or processes (for a process scheduler) that the scheduler currently manages. Actually, the **.info** files are symbolic links into a directory tree that follows the structure of the hierarchy. Thus, you can see the ancestors of a given scheduler using **ls -l policy.info**, and explore the hierarchy structure by looking in the subdirectory that has the name of the scheduler at the root of the hierarchy (the only thing in **/proc/bossa** that does not have the suffix **.info**).

- You can check the hierarchy in real time with the command line:

  ```
  watch -n 1 "ls -l /proc/bossa"
  ```

- You can also check if your progam is attached with the command line:
  `watch -n 1 "cat /proc/bossa/[Scheduler-name].info"`

Proportion checking: Propose and implement a program to check that the system respects the time proportion. **Note** that to attach a process to some other process scheduler `Scheduler_name`, you must include the code below in your program:

```
...
#include "[your_path]/user_[Scheduler-name].c"
#include <sys/types.h>
#include <unistd.h>
...
int pid= getpid();
...
[Scheduler-name]_attach(pid, BOSSA_SCHED_OTHER, 0, 20);
...
```

Exercise 2: Implementation of a Linux like policy

The aim of this exercise is to build a hierarchy that simulates the behavior of a Linux-like scheduler.

2.1 Design and test a First In First Out Policy: Starting from a basic Linux policy, design and test a the following policy:

1. Design a First In First Out (FIFO) policy: The FIFO policy schedules processes according to their arrival order. Check the Bossa grammar to see how to declare a FIFO queue.

2. Check scheduler behavior: Implement test programs to check that your scheduler behavior is correct.

2.2 Design and test a Round-Robin Policy: To simulate the behavior of a Linux-like scheduler you must also design and test the following policy:

1. Design a simple Round-Robin (RR) policy: The RR policy schedules processes according to the number of ticks remaining. In this version this ticks number is fixed. The number of ticks should be decremented on each clocktick event notification. If a process has not terminated when its number of ticks remaining reaches 0 it loses access to the CPU and the number of ticks is reset to the default value.

2. Check the scheduler behavior: Implement test programs to check that your scheduler behavior is correct. You can also use *bossa_hourglass*.

3. Design a Round-Robin (RR) policy with priorities: Modify the previous RR policy so that the initial number of ticks depends on the process priority. Specifically, in this version, the number of ticks is `(priority/4) + 1`. The number of ticks should be decremented on each clocktick event notification. If a process has not terminated when its number of ticks remaining reaches 0 it loses access to the CPU and the number of ticks is reset to `(priority/4) + 1`.

4. Check the scheduler behavior: Implement test programs to check that your scheduler behavior is correct. You can also use *bossa_hourglass*.

2.3 Build the hierarchy and observe its behavior:

1. To build a hierarchy that simulates the behavior of a Linux-like scheduler use the manager tool and mount your two schedulers under the Fixed_priority virtual scheduler. **Note** that in Linux the FIFO policy has a greater priority than the RR policy.

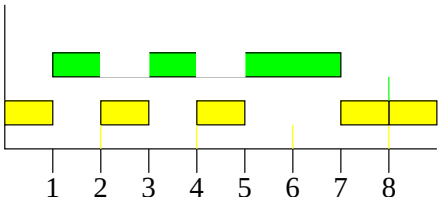
2. To observe the behavior of the hierarchy, use bossa_hourglass. There is an example of its use in the file *bossa_hourglass/htest/***test_edf.sh**. You can also find help using `hourglass -h`. To create a graph showing the results of your test, use the script *bossa_hourglass/script/***render_trace.pl**.

Exercise 3: Earliest Deadline First policy design

Earliest Deadline First (EDF) is a scheduling policy that is used to manage periodic real-time processes. Each process is associated with a period, a worst-case execution time (WCET) within each period, and a relative deadline. The relative deadline is the offset from the beginning of each period by which computation within the period should be completed (note

that the deadline must be at least as large as the WCET, but can be larger and indeed can be the same as the period). For each process, at the beginning of each period, an absolute deadline for the process is computed based on the time of the start of the period and the relative deadline associated with the process. Whenever it is necessary to elect a new process, the one with the lowest absolute deadline is chosen.

The behavior of the EDF policy is illustrated by the following diagram, where process 1 (yellow) has a period and deadline of 2 and a WCET of 1, and process 2 (green) has a period and deadline of 8 and a WCET of 4. Initially, both processes are ready to run. Process 1 is elected because its deadline is 2 while that of process 2 is 8. When process 1 yields at time 1, process 2 begins to run. It does not, however, get to complete its 4 units of computation because at time 2 the next period of process 2 begins and its deadline is 4, which is less than the at of process 2, which is still 8. For the next few time units process 1 wakes up at every other time unit and preempts process 2, which continues to try to complete the computation in its first period. At time 6, however, when process 1 wakes up, both processes have a deadline of 8. Because the deadline of process 1 is not actually earlier than that of process 2, process 2 gets to continue until it completes its computation at time 7. At that time, process 1 gets access to the CPU for its 1 unit of computation. The pattern then repeats starting at time 8.



The EDF policy can be used to manage soft real time applications such as playing a movie or mp3 file. The aim of this exercise is to design an EDF policy and attach a video player to it.

3.1 Design Policy: According to the EDF policy skeleton provided (EDFu.bossa), design an EDF policy. To complete the EDF policy you must implement the various event handlers. You can find a description of the Bossa events on the lab CD. Timers are useful in implementing this policy. You can find an explanation of Bossa timers on the lab CD.

3.2 Observe your policy behavior: To observe the behavior of your policy use the video player **mplayer**. To install it, do:

1. ./configure
2. make
3. make install

To play a movie with your policy perform the command line at the shell prompt:
`mplayer -period [n] -wcet [n] -friend [X pid] [path/movie]`
The period and wcet (worst-case execution time) should be expressed in ms.