Bossa release notes

- **November 1, 2004:** A more precise set of event types are available for use via the compiler option -pb. These are described here, and distinguish cases where the policy decides it does not want to run a process from cases where the process is unable to run due to its recent kernel behavior.

- **May 2, 2004:** Some notes on the implementation of virtual schedulers:

  Virtual schedulers manipulate schedulers rather than processes. For this, they can use two constructs that are not allowed in process schedulers: next(p) and exp => forwardImmediate(). In next(p), p is an expression evaluating to a process, such as e.target. next(p) returns the child scheduler that is directly or indirectly managing the process p. exp => forwardImmediate() forwards the current event notification to the scheduler indicated by exp.

  Like a process scheduler, a virtual scheduler defines a set of states. Each state is declared to be in one of the classes RUNNING, READY, or BLOCKED. The state of a child scheduler must coincide with the scheduler state of the child scheduler, as computed from the states of the processes that it is directly or indirectly managing. For example, a scheduler that is directly or indirectly managing some processes in READY states but no process in a RUNNING state has scheduler state READY, and thus must be a state in the READY class of its virtual scheduler.

  The main complexity in the definition of a virtual scheduler is in the semantics of the operation exp => forwardImmediate(). Forwarding an event notification to a child scheduler may ultimately cause some changes in process states, and thus a change in the scheduler state of the scheduler exp. This implies that the state of the scheduler exp must also change to a state of the new class. The problem is that there is nothing in the syntax exp => forwardImmediate() that indicates what this new class should be. Because virtual schedulers tend to be quite simple, it is often the case that there is only one state in the given class, and thus the child scheduler is simply put in this state. If there is more than one state, one must be designated as public and the others as private, using the public and private keywords in the state declaration (public is the default). exp => forwardImmediate() always move the scheduler exp to the public state that corresponds to its scheduler state after the forwarding operation. The only exception is when the scheduler is initially in a state whose class is the same as its scheduler state after the forwarding operation. In this case, the scheduler remains where it is. If the scheduler is moved to a public state and actually a private state is desired, the scheduler can be moved explicitly to the new state after the exp => forwardImmediate() operation. This is illustrated by the following example, extracted from the implementation of the fixed priority virtual scheduler:

```
  states = {
    RUNNING running : scheduler;
    READY ready : sorted queue;
    READY yield : private scheduler;
    BLOCKED blocked : queue;
  }

...

      On yield.system.pause.*, yield.user.* {
      if (next(e.target) in running) {
        scheduler s = running;
        bool toyield = !empty(ready);
        s => forwardImmediate();
```

```
      if (toyield) {
        s => yield;
      }
    }
  }
```

Except in the case of bossa.schedule and preempt, the operation exp => forwardImmediate() can only be used when exp evaluates to the scheduler that is managing the target process (or target scheduler in the case of the expiration of a timer that is associated with a scheduler, but even in this case, the proper scheduler is obtained by next(e.target)). exp => forwardImmediate() always forwards the event indicated by the handler containing this code. Otherwise, the only state change operations (ie, exp => state) that are allowed are from a state in the RUNNING class to a state in the READY class or from a state in some class to another state in the same class. A state change operation from a state in the RUNNING class to a state in the READY class implicitly forwards a preempt event notification to the affected scheduler.

exp => forwardImmediate() can only be executed once within the execution of an event handler.

- **April 20, 2004:** Processes can be specified to be added to the beginning or end of a sorted queue that is statically sorted using the following syntax:
  - exp => queue.head, to add the process represented by exp to the front of the queue queue.
  - exp => queue.tail, to add the process represented by exp to the end of the queue queue.
- **January 13, 2004:** The names of the functions for manipulating time have been changed. The following functions are defined for both the version of Bossa with high-resolution timers and for the Bossa without high-resolution timers:
  - now() : unit -> time
    The current time.
  - start_relative_timer(timer,offset) : timer * time -> unit
    Set a timer for offset time units in the future.
  - start_absolute_timer(timer,time) : timer * time -> unit
    Set a timer for the time time.
  - stop_timer(timer) : timer -> unit
    Stop a timer.
  - time_to_ticks(t) : time -> int
    Convert a time to a number of ticks (on Bossa with high-resolution timers, this is equivalent to time_to_jiffies, but is included for portability).
  - ticks_to_time(n) : int -> time
    Convert a number of ticks to a time.

  The following functions are only defined for the version of Bossa with high-resolution timers:
  - make_time(sec,nsec) : int * int -> time
    Convert a pair of a number of seconds and a number of nanoseconds to the corresponding time.
  - make_cycle_time(jiffies,cycles) : int * cycles -> time
    Convert a pair of a number of jiffies and a number of cycles to the corresponding time.
  - make_cycles(n) : int -> cycles
    Cast an integer to a number of cycles.
  - time_to_jiffies(t) : time -> int
    Drop the subjiffies component of a time.

- time_to_subjiffies(t) : time -> cycles
  Drop the jiffies component of a time.

The following functions are planned, but are unfortunately not currently implemented:

- time_to_seconds(t) : time -> int
  Drop the nanoseconds component of a time.
- time_to_nanoseconds(t) : time -> int
  Drop the seconds component of a time.

- **January 9, 2004:** In the version of Bossa with high-resolution timers, **Configure high-resolution-timers** must be set to Y in the **General setup** menu. Use the **Help** button to determine the appropriate value for **Clock source?** for your machine.

- **January 8, 2004:** The function error can be used in an interface function with a string argument to give an error message (to appear in the kernel message log, eg /var/log/messages) and abort the interface function. error should not be used once the interface function has made any side effects to the policy state, such as changing the state of a processes or assigning to a process attribute, as these changes will not be undone by the error. In particular, when error is used in the attach function after a state change operation, the policy will be left in an incoherent state.

- **December 3, 2003:** A virtual scheduler can now define interface functions. The function mount, defined in load.c now returns an integer indicating the index associated with the child scheduler. This integer should be given in the position of a scheduler argument of a virtual scheduler interface function, just as the pid (or 0, to indicate the current process) is given in the position of a process argument of a process scheduler.

  The manager program mentioned in the release notes for April 20 is no longer part of the kernel hierarchy, but is available here: manager.c.

  The language now includes some types and functions for manipulating times that are particularly useful when writing policies for the version of Bossa with high-resolution timers. These are:

  - The type cycles, to represent a number of cycles.
  - The function time_to_ticks : time -> int, to convert a time to a number of ticks (an integer). In the version of Bossa without high resolution timers, this function just amounts to a type cast. In the version of Bossa with high resolution timers, this function drops the sub-jiffie component of the time information.
  - ticks_to_time : int -> time to convert an integer, intended to represent a number of ticks, to a time. In the version of Bossa without high resolution timers, this function just amounts to a type cast. In the version of Bossa with high resolution timers, this function sets the sub-jiffie component of the time information to 0.
  - cycles_to_time : cycles -> time, to convert a number of cycles to a time. This function is only defined in the version of Bossa with high-resolution timers.
  - drop_jiffies : time -> cycles, drops the jiffies part of a time value, leaving only the number of cycles. This function is only defined in the version of Bossa with high-resolution timers.
  - make_time : int * int -> time, to create a time from a pair of integers. The first integer is the number of jiffies and the second is the number of cycles. This function is only defined in the version of Bossa with high-resolution timers.

- **October 4, 2003:** If the magic sysrq key (bound to z by default in Bossa) does not work, check that the initialization file /etc/sysctl.conf does not set kernel.sysrq to 0.

- **September 15, 2003:** A Bossa policy can now define any number of timers, and specify separate handlers for each one. Timers can be defined as global variables or as being local to a child process (for a process scheduler) or to a child scheduler (for a virtual scheduler). An example of a policy that defines multiple timers is given in Timer.bossa. An example of the use of this policy is given in the file timer.c in the directory timer_demo.tar.gz. For each timer, there should be a handler unblock.timer.XXX where XXX is the name of the variable containing the timer

  Timers are created and destroyed automatically. A timer declared to be in a global variable is created when the scheduler is loaded into the hierarchy and destroyed when the scheduler is unloaded from the hierarchy. A timer associated with a process or scheduler is created when the process or scheduler attaches to its parent and destroyed when the process or scheduler departs its parent or is terminated.

  The following functions are available for use in a policy, as illustrated in Timer.bossa:

  - void start_absolute_timer(timer,time);: set timer for the absolute time time.
  - void start_relative_timer(timer,time);: set timer for time units (jiffies) in the future.
  - void stop_timer(timer);: stop timer.
  - time now();: obtain the current time.

- **August 18, 2003:** It is now possible to change schedulers from user space. The permissions on the "device" (/dev/bossa) have changed to make this possible. If you have not used Bossa before, it should work immediately. If you have used Bossa before, it may be that your installation of devfs remembers the previous permissions associated with /dev/bossa. To check for this, look in your devfs configuration file (/etc/devfs/devfsd.conf) for the variable RESTORE. If this variable is defined, delete the information about Bossa in the directory that is the value of this variable.

  The /proc interface has been improved to give information about the hierarchy. This information is available in the directory /proc/bossa. This directory contain one immediate subdirectory, named according to the scheduler at the root of the hierarchy. This subdirectory itself contains nested subdirectories reflecting the hierarchy structure. Each subdirectory is associated with an "info" file, that gives information about the schedulers or processes that are the immediate children of the scheduler associated with the subdirectory. "info" files are also present at the /proc/bossa level. For a scheduler that is loaded into the hierarchy, such a file is just a symbolic link to the actual info file at the appropriate position in the subdirectory tree. For a scheduler that is not loaded into the hierarchy, the "info" is actually present in /proc/bossa.

  An easy way to get an overview of the hierarchy is to type "ls -l" in /proc/bossa. The root is the scheduler for which there is both an "info" file and a subdirectory. The positions of other schedulers in the hierarchy are indicated by the symbolic links. There are no symbolic links for the schedulers that are not yet loaded into the hierarchy.

- **April 20, 2003:** The following is some information on using virtual schedulers:

  - **Writing a scheduler for use in a hierarchy:**

    A process scheduler must define the interface functions attach and detach to allow a process to attach itself to the scheduler and to detach itself from the scheduler, respectively. The first argument to either function is the affected process. The is invoked explicitly, either by the affected process or by some other process, and can take other arguments. The detach function of the scheduler currently managing the affected process is invoked implicitly by the attach function of the new scheduler; this function does not take any arguments other than the affected process. The attach function should explicitly place the affected process in either a READY or BLOCKED state. The detach function should not explicitly affect the state of any process.

    A virtual scheduler must define an attach function, which is used when a new child scheduler is added to the virtual scheduler. Currently, it is not possible to remove any sort of scheduler from the hierarchy, so there is no detach function in this case.

– **Adding a new scheduler to Bossa:**
Currently, all schedulers must be statically linked with the kernel. The easiest way to add a new scheduler to the system is to simply replace the file of an existing scheduler with the new definition. The available file names are:
  * kernel/Linux.c
  * kernel/EDFLinux.c
  * kernel/EDFRTLinux.c
  * kernel/Progress.c
  * kernel/Fixed_priority.c
  * kernel/Proportion.c
  * kernel/Proportion_prio.c

  In each case, the scheduler should be given a name that corresponds to that of the file to be replaced. For example, one could define the scheduler EDFRTLinux, and then replace kernel/EDFRTLinux.c with the C code generated for this scheduler.

– **The default scheduler:**
Every hierarchy has a default process scheduler, which is the scheduler that is loaded at boot time and that is the initial parent of all new processes. This scheduler must be declared to be "default" and must define the event handlers process.new.initial_process, which is invoked on creation of the first process, and process.new.fork, which is invoked on creation of all subsequent processes. The event process.new.fork may refer to e.source (where e is the name of the handler's event argument) to access the parent of the new process if this parent is also managed by the default scheduler. This property must be checked using the primitive srcOnSched before referencing e.source.

  Every virtual scheduler in a hierarchy that is an ancestor of the default process scheduler must itself be declared to be "default". Such virtual schedulers must contain a process.new.fork event handler, but not a process.new.initial_process handler, because this event only occurs at boot time when no virtual scheduler is yet loaded.

  The default process scheduler is chosen using the "bossa" boot-line argument. Possible values are as follows:
  * edf: The scheduler in the file kernel/EDFLinux.c is the default scheduler.
  * edfrt: The scheduler in the file kernel/EDFRTLinux.c is the default scheduler.
  * progress: The scheduler in the file kernel/Progress.c is the default scheduler.
  * Anything else, or no "bossa" argument: The scheduler in the file kernel/Linux.c is the default scheduler.

– **Creating a hierarchy:**
The program bossa/manager.c provides a small interactive menu that allows the creation of a hierarchy. This program can be compiled using gcc with no arguments. Executing this program should give the following output:

```
Available schedulers:
0. Fixed_priority (VS, not loaded, default)
1. EDFLinux (PS, not loaded, default)
2. EDFRTLinux (PS, not loaded, default)
3. Progress (PS, not loaded, default)
4. Linux (PS, root, default)
5. Proportion (VS, not loaded, default)
6. Proportion_prio (VS, not loaded, default)

Default path:
Linux
Connect? (parent, child)
```

"Available schedulers" indicates the schedulers that are compiled with the kernel and for each one the following information: whether the scheduler is a process scheduler (PS) or virtual scheduler (VS), whether the scheduler has been loaded, and whether the scheduler is able to be a default (process or virtual) scheduler. "Default path" indicates the schedulers on the path from the root scheduler to the default process scheduler. In the example, only the Linux scheduler has been loaded, so it is both the root and the default scheduler. Finally, the program prompts the user for two schedulers to connect, to be indicated by their numbers. The parent scheduler comes first. One of the mentioned schedulers must already be loaded and the other must not yet be loaded. If a new child is added a virtual scheduler, the user is prompted for the various arguments of the attach function of the virtual scheduler.

If running this program gives an error message rather than the list of available scheduler, the problem is probably that devfs was not selected in the configuration. Select devfs and then recompile the kernel. The use of devfs is not compatible with the name of the mouse device assumed by the standard configuration file for X windows on some versions of Linux. A possible solution is to make a link from /dev/psaux to /dev/mouse, however we do not guarantee the correctness or effectiveness of this solution.

- **Observing the structure of a hierarchy:**

  The manager program mentioned above gives some information about the hierarchy, including the path from the root scheduler to the default process scheduler. This program can safely be terminated by ˆC, and thus it can be used to obtain information about the structure of the hierarchy as well as to add schedulers to the hierarchy.

  Some information about the root scheduler is available via the /proc file system. Namely, when the hierarchy contains a virtual scheduler at the root, the file /proc/sys/bossa/scheduler lists the states defined by this scheduler and the states of its child schedulers. No information about the child schedulers is available at present, however.

- **Moving between schedulers:**

  For any scheduler named XXX, the Bossa compiler generates the files user_XXX.c and user_XXX.h, which are stub functions for the interface functions defined by the scheduler and the corresponding header files, respectively. Moving between schedulers is done by simply calling the attach function of the desired scheduler, or in practice the stub function bossa_XXX_attach. The arguments are the pid of the affected process (or 0 for the current process) and any other arguments specified by the scheduler definition. The names and types of these arguments can be found in the stub prototypes contained in the user_XXX.h header file.

  The stub function header file user_XXX.h contains any type declarations contained in the policy. A notable example is the set of enum type declarations. The Bossa compiler makes no effort to give enum type elements unique names, precisely because they can be part of the user interface. But this lack or renaming can lead to conflicts if several schedulers use the same names, and a single program would like to move among these schedulers, or more generally use interface functions from several such schedulers.

- **April 8, 2003:** Bossa accepts the boot-time arguments bossa=edf, bossa=edfrt, and bossa=progress to select the scheduler defined in the file kernel/EDFLinux.c, kernel/EDFRTLinux.c, or kernel/Progress.c as the root scheduler. If any other value for bossa is provided or if no value is provided, the scheduler defined in the file kernel/Linux.c is selected.

- **February 16, 2003:** Bossa relies critically on the use of devfs. Thus, devfs must be turned on in the Linux configuration file (config.in) for Bossa to boot. Furthermore, the use of devfs implies that the mouse driver is placed in the directory /dev/misc/psaux. For more information on devfs, consult Documentation/filesystems/devfs/README

- **December 18, 2002:** The event unblock.schedule has been removed, as its effect is essentially the same as an unblock followed by a yield.system.immediate.

The program bossa/manager.c allows building a hierarchy with a fixed priority virtual scheduler at the root and any of the process schedulers that are included in the system as children. The support for this hierarchy is rather minimal, however. Currently, new processes are assigned to process schedulers in a round-robin fashion, and there is no way to move a process from one scheduler to another.

Internally, the structure of process attributes has been moved from a separate structure into the task_struct structure maintained by Linux for each process. The amount of space is controlled by the variable CONFIG_BOSSA_DATA_SIZE that can be initialized during the kernel configuration process (e.g. make xconfig). There is no verification that enough space has been reserved. By default, there is enough space for 20 integer-sized values.

- **October 23, 2002:** The language provides a trace declaration that creates a trace of the targets of a specified set of events. An example of such a trace declaration is as follows:

```
bool trace_on = false;

trace 100 { bossa.schedule, block.* }
          { priority, ticks }
  { trace_on }
```

The trace declaration should appear just after the ordering criteria declaration and contains four parts:

  - The number of events that are recorded in the trace.
  - The names of the events that are included in the trace.
  - The process structure fields and global variables whose values are stored in the trace for each event.
  - An expression, indicating whether events should currently be traced. This expression can only refer to global variables.

Interface functions can be defined to control the expression argument, eg as follows:

```
void start_trace() { trace_on = true; }

void end_trace() { trace_on = false; }
```

The fourth argument is optional; if it is absent, tracing begins at boot time.

The trace can be reset interactively at any time using sysrq-t (note that in the Bossa kernel, the letter "z" has been rebound to sysrq, so this should be "zt" in practice). At this time, the contents of the trace is printed using printk, and the trace counter is set to 0. Unfortunately, the amount of information that when printed with printk actually shows up in the log files (eg, /var/log/messages) seems to be limited, so tracing more than a few hundred events does not seem to be useful.

- **October 22, 2002:** The kernel provides some functions for manipulating timers that can be declared as external functions and used in a Bossa policy, as illustrated in the EDF policy. These functions are:

  - system struct timer_list no_timer(): returns a null value of timer type.
  - system struct timer_list init_bossa_timer(process, int): allocates the timer structure, initializes its timeout, and returns the timer structure.
  - void end_bossa_timer(system struct timer_list): if the timer argument is not NULL, calls del_timer_sync and frees the timer structure.
  - void reset_bossa_timer(system struct timer_list, int): calls mod_timer to reset the times to time out at time indicated by the second argument.

– int now(): returns the current value of jiffies.

No verification is performed of the timer functions. In particular, neglecting to call end_bossa_timer when a process associated with a timer ends can crash the system.

- **October 17, 2002:** The compiler takes one argument, which is the name of the file containing the policy. The compiler creates three files: c-code/policy.c, c-code/user_policy.h, and c-code/user_policy.c, where policy is the name of the policy. c-code/policy.c is the compiled policy, and should be placed in the kernel subdirectory of the kernel source tree. c-code/user_policy.h and c-code/user_policy.c are are a header file and an implementation of the user interface functions, and should be linked with any application that would like to use the interface functions. **Warning:** the compiler is a bit slow.

- **October 9, 2002:** The current version of Bossa provides two possible schedulers: a Bossa reimplementation of the Linux scheduler and an implementation of a combined Linux and EDF scheduler. These can be selected using make xconfig. Reservations can be made using the combined Linux and EDF scheduler, using the IOCTL interface defined in kernel/bossa_edf_dev.c.

  The system calls nice and setpriority affect the Bossa priority of a process.