lkmonitor Architecture Review

lkmonitor Architecture Review

1. Introduction

lkmonitor is a Gnome application, meaning that lkmonitor employs the features offered by the glib and gtk libraries to support most of its capabilities.

lkmonitor shows information regarding to various system characteristics, as CPU or memory usage. This information is obtained from the /proc pseudo-file system. lkmonitor's main task is to read data from the aforementioned files, parse it and extract the relevant information. Under this point of view, lkmonitor performs as a simple file reader (Figure 1).

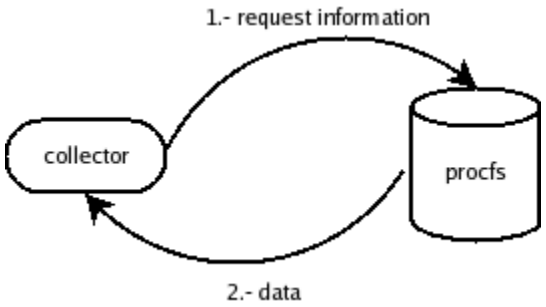

Figure 1 Basic lkmonitor functioning cycle

Although the description of the procfs internals is not the goal of this document, it's important to know how it works.

2. The procfs pseudo-filesystem

procfs is a pseudo-filesystem, that is, the files under /proc do not exist physically in the hard drive, but the information they contain is calculated on demand. As the rest of the filesystems in Linux, procfs is supported by the VFS (Virtual File System), a kernel layer that offers a level of abstraction when we work with filesystems, handling the differences between them while offering a common interface.

Although other UNIX - like operating systems also provide a /proc filesystem (BSD, for example), the format is different between them. While Linux offers a plain text based format for the majority of its files, FreeBSD and others offer less information in text format and more in binary format. The first case suits best shell scripting, the second being closer to programming.

Both general information and process-specific information can be found under /proc. Linux distinguishes the different types of information through the i-node number. Under Linux, this i-node number is a 32-bit integer, whereas a PID (Process ID) is represented with a 16-

bit integer. Linux therefore splits the i-node number in two 16-bit halves, the higher-order bits interpreted as the PID which the information is being given about, and the lower-order bits as the type of information to be given. As a PID of zero is not valid, it's used to mark those files that contain general information about the system.

Figure 2 shows what the kernel does when making a cat /proc/cpuinfo (for example). The process created by the shell first asks for information by reading the file. VFS catches the query and established that the file to be read is one of the pseudo-files in the procfs. The procfs filesystem then looks up in the kernel tables for the information required by the reading process.

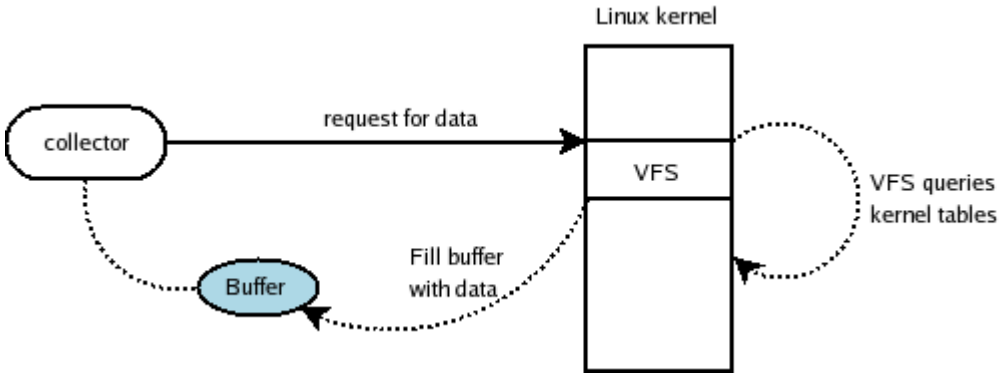

Figure 2 Kernel response to the reading of a file in procfs

Consulted kernel data structures depend on the kind of information desired (global, specific, about the CPU, about a single process, etc.). The process' buffer is filled with the collected data

The single most important fact about the data collection is that it's absolutely transparent under an external point of view.

3. lkmonitor architecture

lkmonitor implements two different components as two separate threads: the main thread controls the event loop and the collector thread obtains the updated information. Both threads share some data structures, but the gtk objects are the most important. gtk library does not offer a secure concurrent access in multithreading, making synchronism and mutual exclusion necessary in order to access objects like labels, panels or buttons in a secure way.

Fortunately enough, gtk offers two primitives to lock threads and implement critical sections. Those are gdk_threads_enter() and gdk_threads_leave(). This avoids the possibility of the main thread accessing the objects while the collector thread is updating the information on them. The collector thread behaviour is shown in Figure 3.

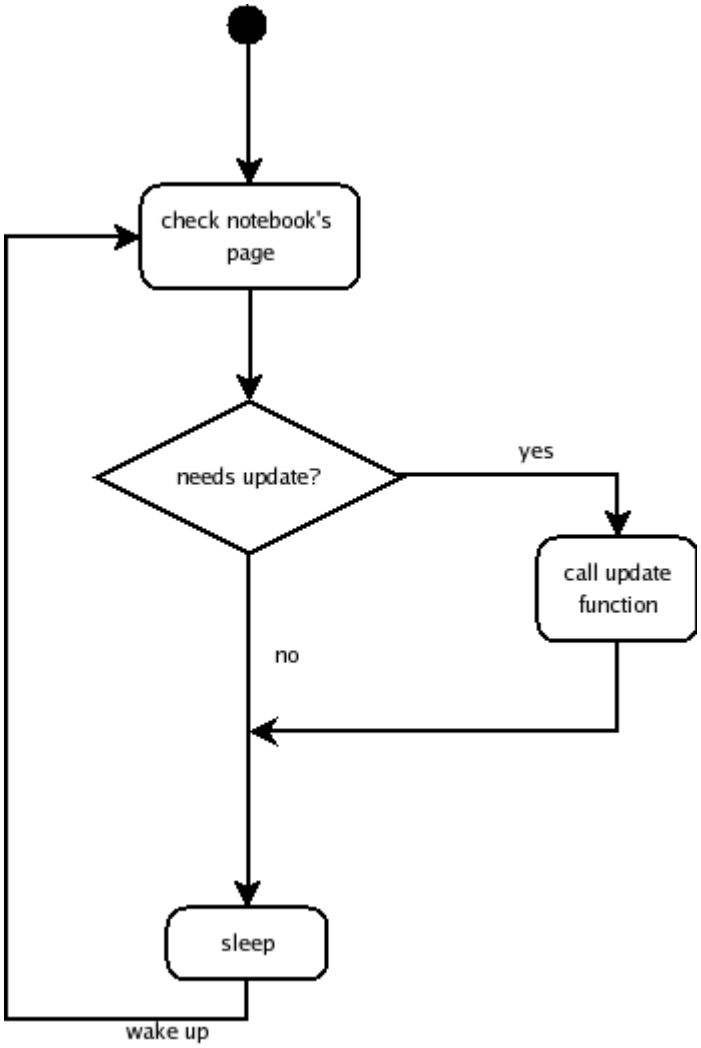

Figure 3 Collector thread behaviour

lkmonitor is an application that show information and updates it constantly. The collector loop is simple: it determines if the shown information is dynamic (e.g., the CPU panel has static info), obtains the updated info from procfs and shows it in the main window. Then, the loop starts again.

As seen in Figure 4, the application starts creating some data structures where the retrieved information is going to be stored. Afterwards, the threading subsystem is initialized so as to allow the use of the thread-locking primitives. Then, the main window is created, the collector thread is launched and the event control loop is started.

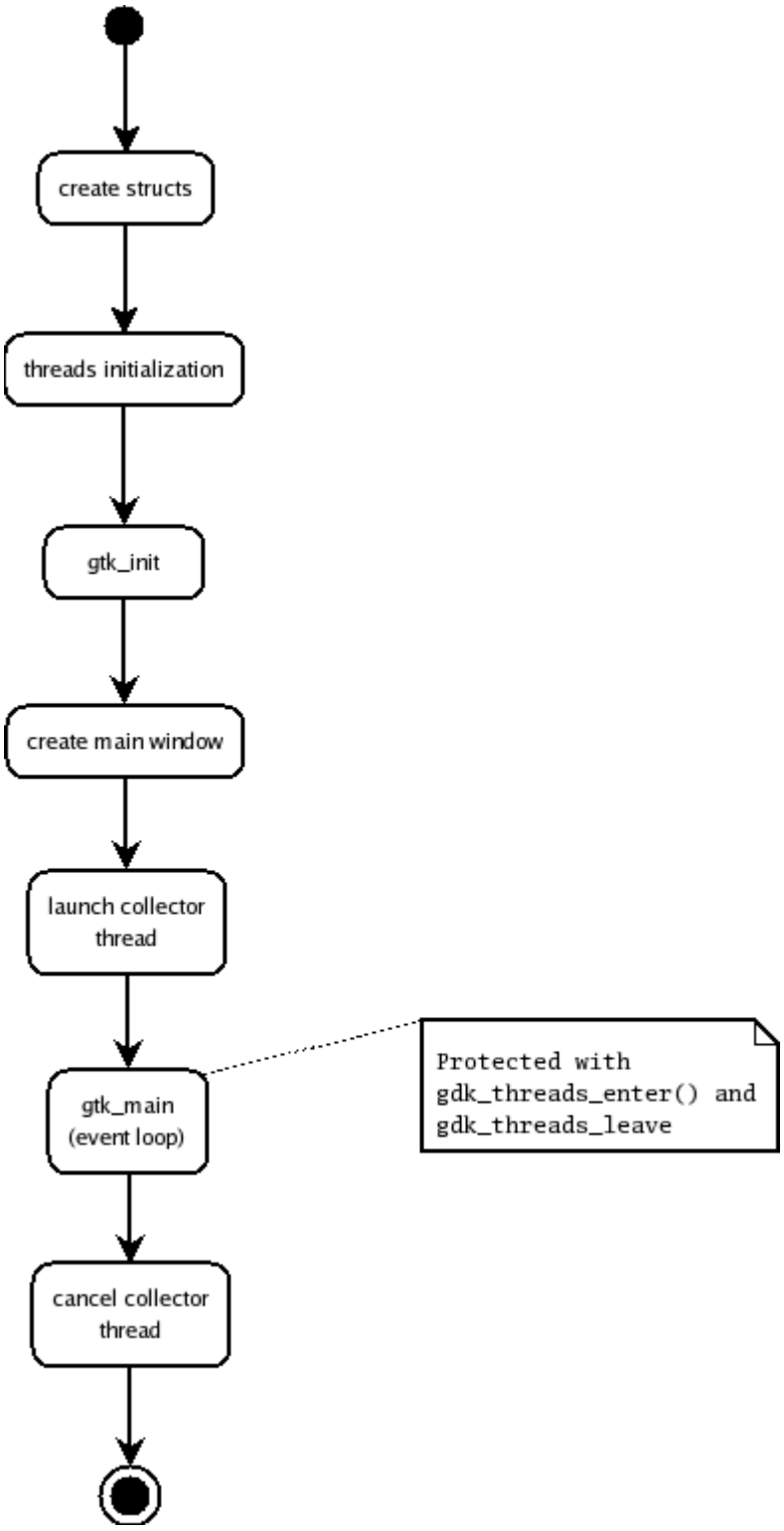

Figure 4

The collector thread calls to the update function family (update_cpu, update_mem, etc.), which updates the information. The data is first obtained with the get_*_info functions.

Those functions do the actual work of reading and parsing the file in order to extract the related information. When this work is done, the update functions avoid the unwanted access to the gtk objects locking the other threads by means of gtk_threads_enter(). Once the window is updated, the critical section is unlocked.

The behaviour of the update functions is shown in Figure 5.

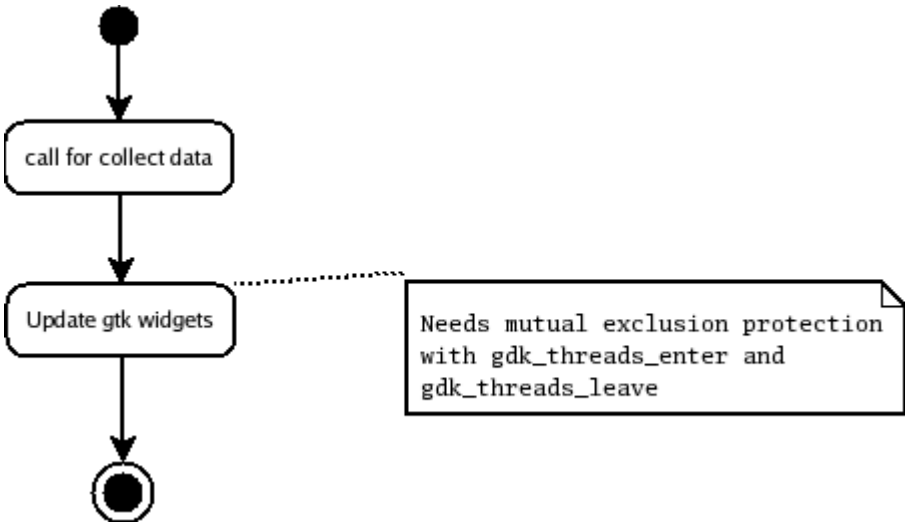

Figure 5 Behaviour of the update family of functions