

The Monitor System of the LHCb On-Line Farm

LHCb Technical Note

Issue: Draft
Revision: 1

Reference: LHCb 2005-51 DAQ
Created: 18 Nov. 2004
Last modified: 31 Aug. 2005

Prepared By: F. Bonifazi, D. Bortolotti, A. Carbone, D. Galli, D. Gregori, U. Marconi,
G. Peco, V. Vagnoni.

Abstract

The aim of the LHCb on-line farm Monitor System is to keep under control all the working indicators which are relevant for the farm operation, and to set the appropriate alarms whenever an error or a critical condition comes up. Since the most stressing tasks of the farm are the data transfer and processing, relevant indicators includes the CPU and the memory load of the system, the network interface and the TCP/IP stack parameters, the rates of the interrupts raised by the network interface card and the detailed status of the running processes. The monitoring of computers' physical conditions (temperatures, fan speeds and motherboard voltages) are the subject of a separate technical note, since they are accessed in a different way, by using the IPMI protocol.

Document Status Sheet

Table 1 Document Status Sheet

1. Document Title: The Monitor System of the LHCb On-Line Farm			
2. Document Reference Number: LHCb 2005-51 DAQ			
3. Issue	4. Revision	5. Date	6. Reason for change
Draft	0	18 Nov. 2004	First version
1	1	31 Aug. 2005	First released version

Table of Contents

LHCb TECHNICAL NOTE.....	I
ISSUE: DRAFT	I
ABSTRACT.....	I
DOCUMENT STATUS SHEET.....	I
TABLE 1 DOCUMENT STATUS SHEET.....	I
LIST OF FIGURES.....	V
1. REQUIREMENTS.....	1
2. MONITOR IMPLEMENTATION.....	3
2.1. DIM.....	3
2.2. MONITOR SERVERS	4
2.3. SENSORS	4
2.4. PROCFS AND SYSFS	5
2.5. DEVELOPMENT GUIDE-LINES	5
3. THE MONITOR SERVERS.....	7
3.1. INFORMATION ABOUT THE CPUS (CPUINFOSRV)	7
3.1.1. Synopsis	7
3.1.2. Description.....	7
3.1.3. Command line options	7
3.1.4. Environment.....	8
3.1.5. Services provided.....	8
3.2. STATISTICS ABOUT THE CPUS (CPUSTATSRV)	10
3.2.1. Synopsis	10
3.2.2. Description.....	10
3.2.3. Command line options	10
3.2.4. Environment.....	10
3.2.5. Services provided.....	11
3.3. STATISTICS ABOUT INTERRUPTS (IRQSRV)	12
3.3.1. Synopsis	13
3.3.2. Description.....	13
3.3.3. Command line options	13
3.3.4. Environment.....	13
3.3.5. Services provided.....	14
3.3.6. Commands provided.....	15
3.4. STATISTICS ABOUT MEMORY USAGE (MEMSRV).....	16
3.4.1. Synopsis	17
3.4.2. Description.....	17
3.4.3. Command line options	17
3.4.4. Environment.....	17
3.4.5. Services provided.....	18
3.5. STATISTICS ABOUT NETWORK INTERFACES (NIFSRV)	22
3.5.1. Synopsis	22

3.5.2.	Description.....	22
3.5.3.	Command line options	22
3.5.4.	Environment	23
3.5.5.	Services provided.....	23
3.5.6.	Commands provided.....	26
3.6.	STATISTICS ABOUT NETWORK INTERFACES COALESCENCE (COALSRV).....	26
3.6.1.	Synopsis	27
3.6.2.	Description.....	27
3.6.3.	Command line options	27
3.6.4.	Environment	27
3.6.5.	Services provided.....	28
3.6.6.	Commands provided.....	29
3.7.	STATISTICS ABOUT TCP/IP STACK (TCPIP_SRV).....	29
3.7.1.	Synopsis	30
3.7.2.	Description.....	30
3.7.3.	Command line options	30
3.7.4.	Environment	30
3.7.5.	Services provided.....	31
3.7.6.	Commands provided.....	35
3.8.	STATISTICS ABOUT PROCESSES (PSSRV)	36
3.8.1.	Synopsis	37
3.8.2.	Description.....	37
3.8.3.	Command line options	37
3.8.4.	Environment	38
3.8.5.	Services provided.....	38
4.	THE PVSS MONITOR CLIENTS.....	45
5.	REFERENCES	68

List of Figures

Figure 1. The Monitor System deployment.....	2
Figure 2. The Monitor System's collaboration diagram. s1: Server start-up; c1-c3: client start-up; r1-r3 data flow.....	3
Figure 3. The CPU information PVSS client. By right-clicking a field in this window, a help window, which contains the field description (shown, for example, in Figure 4), pops-up.....	45
Figure 4. The CPU information PVSS help window, obtained by right-clicking the <code>bogomips</code> field in the CPU information PVSS client, shown in Figure 3.....	46
Figure 5. The CPU statistics PVSS client. The tabbed pane to switch to the alarm setting panel (shown in Figure 7) and the button to open context switch rate vs time plot (shown in Figure 6) are emphasized.....	47
Figure 6. The CPU statistics PVSS client. Plot of the time evolution of the global CPU context switch rate.....	48
Figure 7. The CPU statistics PVSS client. Alarm setup panel, which allows setting alarm thresholds for the FSM.	49
Figure 8. The Interrupts statistics PVSS client. These data refer to the CPU 1 (emphasized tabbed pane) of a PC running the kernel 2.4 as can be recognized by the timer interrupts rate (100 s^{-1} , emphasized). The timer interrupts are all handled by the CPU 1 (this behavior can be changed by editing <code>/proc/irq/<IRQ#>/smp_affinity</code> i-node). The button to reset the average and maximum values and the text area showing the time elapsed since the last reset are also emphasized.	50
Figure 9. The Interrupts statistics PVSS client. These data refer to the CPU 0 (emphasized tabbed pane) of a PC running kernel 2.6. The timer interrupts (1000 s^{-1} in the kernel 2.6) are shared between CPUs 0 and 1 (compare this Figure with Figure 10). This behavior can be changed by editing <code>/proc/irq/<IRQ#>/smp_affinity</code> i-node.....	51
Figure 10. The Interrupts statistics PVSS client. These data refer to the CPU 1 (emphasized tabbed pane) of a PC running kernel 2.6. The timer interrupts (1000 s^{-1} in the kernel 2.6) are shared between CPUs 0 and 1 (compare this Figure with Figure 9). This behavior can be changed by editing <code>/proc/irq/<IRQ#>/smp_affinity</code> i-node.....	52
Figure 11. The Memory statistics PVSS client (the monitored PC is running a 2.4 kernel).	53
Figure 12. The Memory statistics PVSS client (the monitored PC is running a 2.6 kernel).	54

Figure 13. The Memory statistics PVSS help window, obtained by right-clicking the Committed_AS field in the memory statistics PVSS client, shown in Figure 11 or Figure 12.....	55
Figure 14. The Network Interfaces PVSS client.....	56
Figure 15. The Network Interface PVSS client. Plot of the time evolution of the I/O bit rate.....	57
Figure 16. The Network Interface Interrupt Coalescence PVSS Client.....	58
Figure 17. The TCP/IP stack PVSS client. IP tabbed pane.	59
Figure 18. The TCP/IP stack PVSS client. TCP tabbed pane.....	60
Figure 19. The TCP/IP stack PVSS client. UDP tabbed pane.	61
Figure 20. The processes statistics PVSS client. Simple view: panel which shows the main information. The monitored PC is running the kernel 2.6 (the threads information are therefore provided).....	62
Figure 21. The processes statistics PVSS client. Advanced view: panel which shows extended information (1/5, information about UTGID, TGID, CPU and memory share, command name and command line, PPID and number of threads). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).	63
Figure 22. The processes statistics PVSS client. Advanced view: panel which shows extended information (2/5, size information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).....	64
Figure 23. The processes statistics PVSS client. Advanced view: panel which shows extended information (3/5, scheduling information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).	65
Figure 24. The processes statistics PVSS client. Advanced view: panel which shows extended information (4/5, time and signal information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).....	66
Figure 25. The processes statistics PVSS client. Advanced view: panel which shows extended information (5/5, signal, user, terminal and execution information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).....	67

1. Requirements

The LHCb on-line farm Monitor System should be able to retrieve, once every few tens of seconds, the indicators which are relevant for the farm operation, and to collect them centrally, in order to show them in tables and plots on a separate monitor PC and to set the appropriate alarm condition for the experiment's Finite State Machine (FSM) in case an error or a critic farm condition comes up.

Since the most stressing tasks of the farm are the data transfer and processing, relevant indicators includes:

- The type and the characteristics of the system CPUs (clock rate, cache size, performance).
- The CPU operation statistics (context switch rate, time percentages spent in user/kernel mode, in serving hard and soft interrupts, in waiting for an I/O operation).
- The memory operation statistics (used/free memory and low-memory, virtual memory and memory overbooking).
- The hardware interrupts statistics.
- The network interface statistics (I/O byte/frame rates, error fractions).
- The network interface frame-coalescence statistics.
- The TCP/IP stack statistics (I/O datagram rates, datagram fragmentation and reassembling, error fractions).
- The statistics about running processes (thread identifier, thread group identifier, CPU and memory usage, scheduler type and priority, number of threads, CPU on which the process is currently running, start time, elapsed time, etc.).

All these indicators have to be retrieved from the operating system of each farm node and transmitted, through the network, to the monitor PC. The processes which necessarily must run on every farm node must however be light-weight, in order not to overload the node resources of the on-line farm (which is already heavily loaded by the data acquisition and trigger tasks).

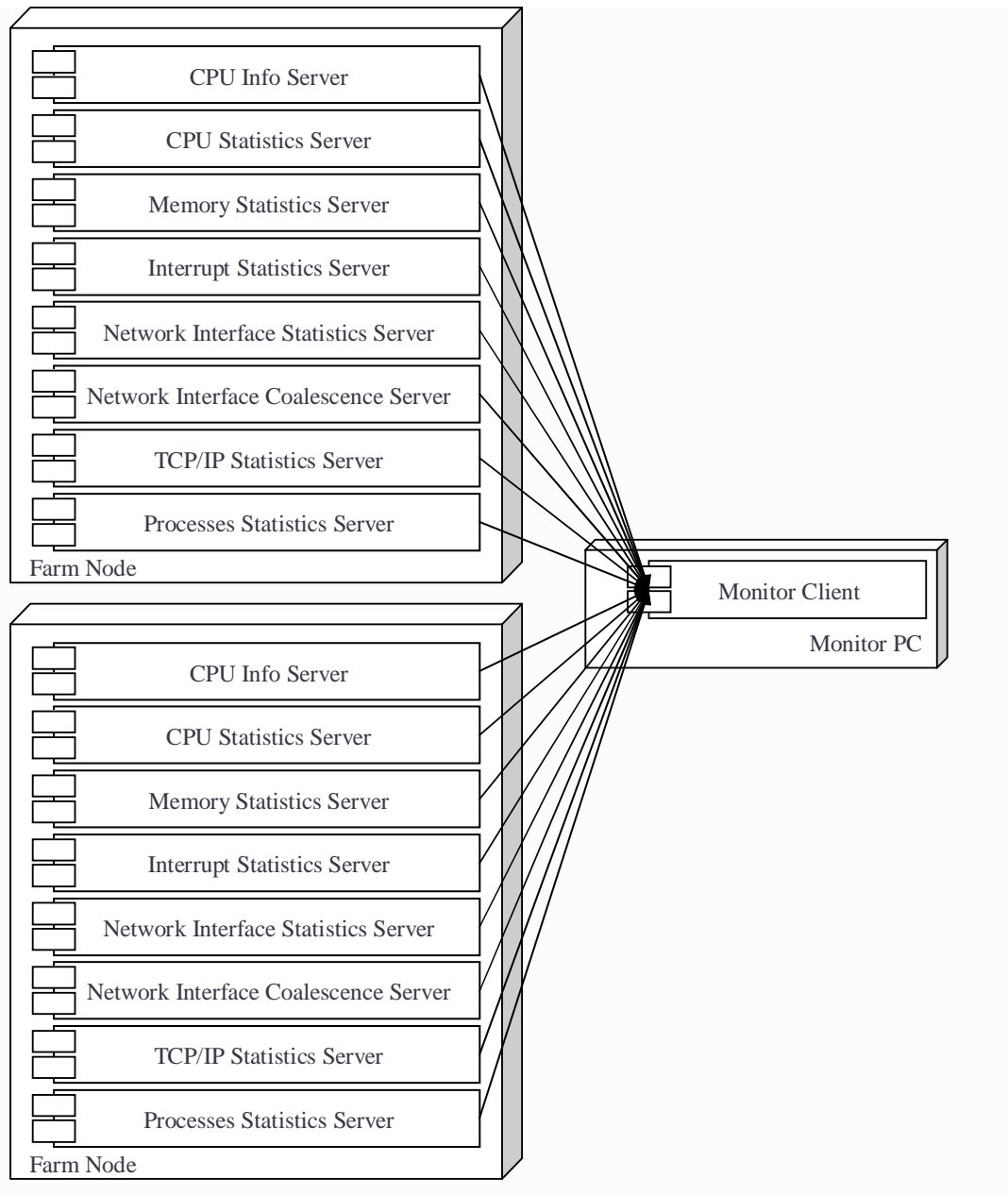


Figure 1. The Monitor System deployment.

The monitoring of physical operation condition of the nodes (temperatures, fan speeds and motherboard voltages) are the subject of a separate technical note, since they are accessed in a different way, by-passing the operating system by using the IPMI protocol, implemented by the network interface hardware itself.

2. Monitor implementation

The LHCb on-line farm Monitor System is based on the DIM (Distributed Information Management System) inter-process communication layer [1].

2.1. DIM

DIM has a client/server architecture and uses a Name Server mechanism to publish/subscribe services.

Each farm node runs eight DIM Monitor Server (whose executable name are `coalSrv`, `cpuinfoSrv`, `cpustatSrv`, `irqSrv`, `memSrv`, `nifSrv`, `psSrv`, `tcpipSrv`), which register monitor services (which provide monitor data) and commands (which provide reset handle) with the DIM Name Server and make them available to the monitor client (s1 in Figure 2).

The Monitor Client asks the DIM Name Server which server makes the required services and commands available (c1 in Figure 2); once got the answer (c2), the monitor client contacts directly the server (c3) to subscribe to services (to bring itself up-to-date about the monitor data, r2, and to require counter reset execution, r3).

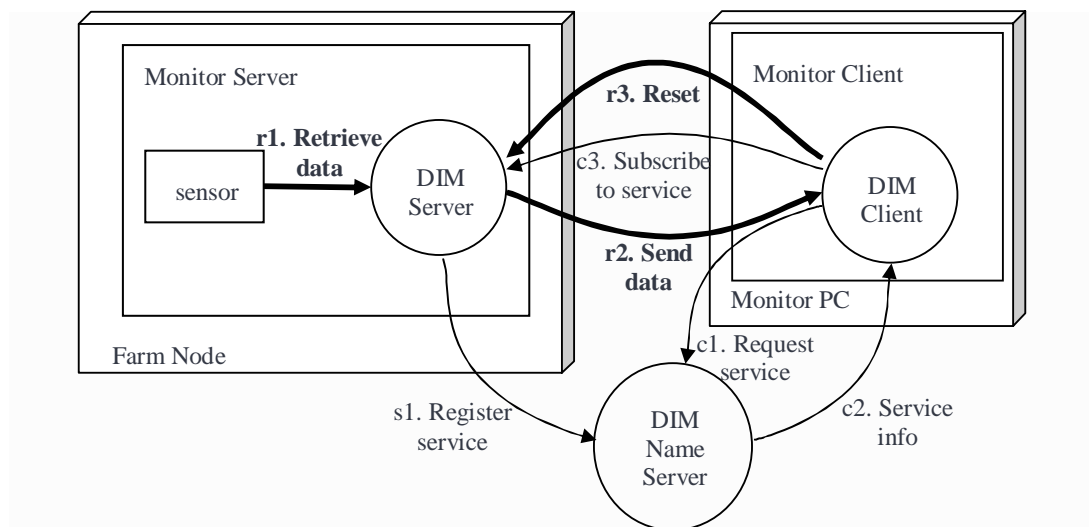


Figure 2. The Monitor System's collaboration diagram. s1: Server start-up; c1-c3: client start-up; r1-r3: data flow.

2.2. Monitor Servers

DIM Monitor Servers consists in a **main loop**, executed **periodically**, which calls at any iteration a **sensor** (a function which retrieves data from the operating system) and updates the published data, grouped in one or more **services**.

Together with instantaneous values (the mean values in the interval between two successive service updates), several sensors produce also **average** and **maximum** values, to keep information about error counters whose increment is discontinuous. Average and maximum values are computed starting from the monitor server process start-up.

To give the client the possibility to reset average and maximum data (for example to check if a certain error disappear after an action has been taken to fix the problem), the sensors that publish such data publish also a **command** to **reset** them. On user request the client contacts the servers and sends them the reset command request. A counter reset is also triggered by a SIGHUP signal sent to the monitor server process.

2.3. Sensors

Sensors retrieve data from the operating system by accessing the i-nodes in the `proc` file system and in the `sys` file system. Very often the operating system data are integer **counters** (e.g. Ethernet sent frame counter, counter of time (expressed in `USER_HZ`) the CPU has spent performing different kinds of work, interrupt counters, etc.).

To produce understandable values, the sensors evaluate from this numbers data in the form of **rates** (e.g. the number of Ethernet frames received for time unit) and **ratios** (e.g. the ratio between the number of received Ethernet frame which have been dropped by the network interface due to fifo overrun and the total number of received Ethernet frames).

To compute rates, the difference between counter values in two successive instants is divided by the time interval between the two instants; to compute ratios, the difference between counter values in two successive instants is divided by the difference between the reference counter values in the same two successive instants.

A potential problem could arise from the fact that the operating system uses 32-bit counters even for quantities which increments very quickly. In order to correctly compute counter differences, in the time interval between two successive readings one counter reset at maximum is allowed. In the 32-bit counters of the bytes sent/received by a gigabit Ethernet network interface transferring data at full load, the counter reset can happen as frequently as once every 34 seconds. So, to correctly compute counter differences, the sensor reading must happened more frequently than once every 34 s.

2.4. Procfs and Sysfs

Until Linux kernel 2.4, the **proc file system** is the only file system-like interface to the internal data structures in the kernel (the other interface is `ioctl()/sysctl()`).

Starting from Linux kernel 2.6, **sys file system** has been added to shows all of the devices (virtual and real) and their inter-connectedness within the system. Many data are now going to move from `procfs` to `sysfs`.

The trend, in the forthcoming kernel version, is to store in `procfs` only process-specific statistics and to store in `sysfs` all devices statistics (network interfaces, temperatures and fan speed, SCSI interfaces, etc.).

At present, in kernel 2.6, network interfaces data are accessible both on `procfs` and `sysfs`; TCP/IP stack, CPU states and memory data are accessible only on `procfs`.

Monitor sensors at present use `procfs`.

In the future probably most of the sensors (all but process sensor) should be modified to access `sysfs`.

2.5. Development guide-lines

During the farm operation, computing nodes will be very busy in event processing and data transfer. The need of accurate monitoring arises from the demand of optimization of the computing resources: it become useless if the monitor itself wastes resources.

Therefore the Monitor Server implementation on computing nodes must be **light-weight**: processes must occupy as less computer resources as possible. The function are so written in plain C, putting care in resource saving. For examples the shallow copy (i.e. pointer copy) is preferred with respect to the deep copy; if the deep copy is needed, the low-level array copy (`memcpy(3)`) is preferred with respect to the high level array copy (looping on array elements); when possible, the `malloc(3)` library call (and therefore the rather expensive `brk(2)` system call), is called only during server initialization; otherwise the `realloc(3)` call, which optimize memory allocation, is preferred with respect to `free(3)/malloc(3)`; and so on.

A particular care must be placed in accessing the `procfs` and the `sysfs` i-node contents: as a matter of fact, each time a `procfs` or a `sysfs` i-node is accessed – either to read a single character or a large block – a handler function is called by the kernel to gather on the fly the entire set of data referenced by the i-node. Therefore, in order to save CPU resources, **procfs and sysfs data must be read in one shot only**. This is not the behavior of the high-level stream I/O calls (`fopen(3)`, `fgets(3)`, `fscanf(3)`), which, by default, read data up to fill a transparently allocated buffer, then read data again

as many time as needed (by refilling each time the same allocated buffer), to access the following data. To access `procfs/sysfs` data in one shot only, the low-level I/O calls (`open(2)` and `read(2)`) must be used instead of the high-level (stream) I/O calls and the read buffer must be sized to contain the whole i-node content.

The Monitor Server `psSrv`, which publishes the status of the running processes, deserve a particular reference. It has to access many `procfs` i-nodes (one per process), whose location has been changed passing from kernel 2.4 to kernel 2.6 (from `/proc/<pid>/...` to `/proc/<tgid>/task/<tid>/...`). To cope with the present differences between 2.4 and 2.6, and with future changes in kernel, we decided to access processes information by means of a maintained library (`libprocps-3 [5]`).

3. The monitor servers

In this section we will show the eight monitor server, with a short description of the monitored quantities.

3.1. Information about the CPUs (`cpuinfoSrv`)

The `cpuinfoSrv` is a DIM server which provides information about the CPU or the CPUs of the PC. Information provided by this server is static, i.e. is read at server start-up and published once using DIM. These data can be useful, for example, to evaluate, as a weighted average, some aggregate parameters of the farm.

3.1.1. Synopsis

```
cpuinfoSrv [-d...] -l logger  
cpuinfoSrv -h
```

3.1.2. Description

Starts the System CPU Information Monitor Server, sending its log messages to the logger facility `logger`. The number of the logged messages can be increased by means of the `-d` (repeatable) option.

3.1.3. Command line options

- `-h` Print the program usage and exit immediately.
- `-l logger`
Use `logger` as logging facility. Values allowed for `logger` are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to <code>stderr</code> .
0x4	DIMSVC	Send the messages to a specific DIM service.
- `-d` (repeatable) Increase the number of the logged messages.

3.1.4. Environment

The program `cpuinfoSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.1.5. Services provided

In the following list, the string “XX” must be substituted with a two-digit number containing the processor ordinal number (e.g.: 00, 01, 02 and 03, for a dual-processor PC with hyper-threading activated).

`/<HOSTNAME>/cpuinfo/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getCpuinfoFromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/cpuinfo/server_version` (string).

Returns the RCS identifier of the DIM server (`cpuinfoSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/cpuinfo/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`). This service publishes log messages sent by the server.

`/<HOSTNAME>/cpuinfo/success` (integer).

This dummy service always returns 1. It is used by PVSS-DIM to check if `cpuinfoSrv` process is running.

`/<HOSTNAME>/cpuinfo/cpuXX_vendorId` (string).

The CPU brand identifier string. E.g.: “GenuineIntel”, “AuthenticAMD”, “CyrixInstead”, “GenuineTMx86”.

`/<HOSTNAME>/cpuinfo/cpuXX_modelName` (string).

The CPU model identifier string. E.g.: “Intel(R) Pentium(R) III CPU family 1133MHz”, “Intel(R) Xeon(TM) CPU 2.40GHz”, “AMD Athlon (TM) 64 Processor 3700+”.

`/<HOSTNAME>/cpuinfo/cpuXX_family` (integer).

The CPU family identifier. Value 6 stands for Intel Pentium III, AMD K7, Cyrix M2, VIA C3; value 15 stands for Intel Pentium IV and AMD K8, etc. [1].

`/<HOSTNAME>/cpuinfo/cpuXX_model` (integer).

The CPU model identifier. E.g.: value 1 can stand for P4 0.18 μm , value 2 can stand for P4 0.13 μm , value 3 can stand for P4 0.09 μm , value 5 can stand for AMD K8 Opteron 0.13 μm . [1].

`/<HOSTNAME>/cpuinfo/cpuXX_stepping` (integer).

The sub-version identifier (a.k.a. revision number) of a CPU. The stepping number increase with manufactory improvement or bug fixes of the same CPU model. [1].

`/<HOSTNAME>/cpuinfo/cpuXX_cpuMHz` (floating point).

The clock frequency of the CPU, measured in megahertz.

`/<HOSTNAME>/cpuinfo/cpuXX_kbCacheSize` (integer).

The CPU cache size, measured in kilobytes.

`/<HOSTNAME>/cpuinfo/cpuXX_physicalId` (integer or N/A).

The physical processor ID as return by a CUID call.

`/<HOSTNAME>/cpuinfo/cpuXX_siblings` (integer or N/A).

The number of hyper-threading cores in that physical CPU. A sibling value of 1 indicates that hyper-threading is disabled.

`/<HOSTNAME>/cpuinfo/cpuXX_bogomips` (floating point).

This number is a very raw measure of the CPU computational speed. Etymology has origin by the words bogus (something which is a fake, not genuine) and MIPS. MIPS is an acronym for “Millions of Instruction per Second” and is the number of millions of machine code instruction which a CPU is able to execute in a second. Bogomips are measured at boot time by running a program loop which does absolutely nothing (hence the definition of Bogomips as “the number of million times per second a processor can do absolutely nothing”). Bogomips ranges from 0.02 (Intel 8088 at 4.77 MHz) to 6717.44 (Intel Pentium 4 at 3.4 GHz). Bogomips/CPU clock ratio ranges from 0.004 (Intel 8088) to 3.97 (Intel Xeon with hyper-threading). Bogomips can be used as a weight for the evaluation of farm aggregate values as a weighted average [2].

3.2. Statistics about the CPUs (`cpustatSrv`)

The `cpustatSrv` is a DIM server which collects, and evaluates as a percentage, both the aggregate values and the specific per-CPU values of the fraction of time spent by the CPUs performing different kinds of work. Moreover it collects the global context switch rate (useful to check the operation of process scheduling and process affinity).

3.2.1. Synopsis

```
cpustatSrv [-d...][-u refresh_time_interval] -l logger
cpustatSrv -h
```

3.2.2. Description

Starts the System CPU Statistics Monitor Server, sending its log messages to the logger facility `logger` and refreshing data every `refresh_time_interval` seconds. The number of the logged messages can be increased by means of the `-d` (repeatable) option.

3.2.3. Command line options

- `-h` Print the program usage and exit immediately.
- `-l logger`
Use `logger` as logging facility. Values allowed for `logger` are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVCSVC	Send the messages to a specific DIM service.
- `-u refresh_time_interval`
Refresh data every `refresh_time_interval` seconds. Default: 20 s. Allowed values: >1 s.
- `-d` (repeatable) Increase the number of the logged messages.

3.2.4. Environment

The program `cpustatSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.2.5. Services provided

`/<HOSTNAME>/cpustat/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getCpustatFromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/cpustat/server_version` (string).

Returns the RCS identifier of the DIM server (`cpustatSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/cpustat/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`). This service publishes log messages sent by the server.

`/<HOSTNAME>/cpustat/success` (integer).

This service returns 1 if published data are meaningful, and 0 if published data are meaningless due to error in sensor operation. It is also used by PVSS-DIM to check if `cpustatSrv` process is running.

`/<HOSTNAME>/cpustat/ctxtRate` (floating point).

Total number of context switches per second across all the CPUs.

`/<HOSTNAME>/cpustat/total` (floating point array).

`/<HOSTNAME>/cpustat/cpuXX` (floating point array).

These services provide the aggregate statistics (`total`) and the specific per-CPU statistics (`cpuXX`), of the fraction of time spent by the CPUs performing different kinds of work. The string “XX” must be substituted with a two-digit number containing the processor ordinal number (e.g.: 00, 01, 02 and 03, for a dual-processor PC with hyper-threading activated). The floating point array returned by the service provides the following quantities in the order: `user`, `system`, `nice`, `idle`, `iowait`, `irq`, `softirq`.

- **user** (floating point). Time percentage the CPU has spent in normal processes executing in user mode.
- **system** (floating point). Time percentage the CPU has spent in processes executing in kernel mode.
- **nice** (floating point). Time percentage the CPU has spent in niced processes in user mode. Time spent in niced tasks will also be counted in system and user time.
- **idle** (floating point). Time percentage the CPU has spent not working.
- **iowait** (floating point). Time percentage the CPU has spent in waiting for I/O to complete.
- **irq** (floating point). Time percentage the CPU has spent in servicing interrupts (top halves). Available only with kernel version ≥ 2.5 (with previous kernel version this number is set to zero).
- **softirq** (floating point). Time percentage the CPU has spent in servicing softirqs (bottom halves). Available only with kernel version ≥ 2.5 (with previous kernel version this number is set to zero).

3.3. Statistics about interrupts (**irqSrv**)

The **irqSrv** is a DIM server which collects the interrupt rates issued by the hardware device drivers. Data are partitioned on **per-CPU** and **per-IRQ-line** basis.

The available IRQ-lines was 8 in the old 8088 PC XT equipped with a 8259 PIC (Programmable Interrupt Controller), is 15 (starting with the PC AT) in Intel-based PCs equipped with two 8259 PIC (Programmable Interrupt Controller) connected in cascade, and is typically 24 and up to 64 in more recent hardware, equipped with APICs (Advanced Programmable Interrupt Controllers).

The APICs can also distribute interrupts across multiple processors in an intelligent and programmable way (**IRQ-to-CPU affinity**: the interrupts risen, e.g., by the different network interfaces can be served by different processors or partitioned amongst different processors). In Linux the IRQ-to-CPU affinity can be controlled by means of the settings in the `/proc/irq/<IRQ#>/smp_affinity` i-node [8].

The data provided by the **irqSrv** Monitor Server can be very useful to check the operation of IRQ-to-CPU affinity.

The server can cope with a change of the number of the interrupt sources (it can happen while inserting a new device, e.g.: connecting a serial device or starting the audio subsystem).

Average values and maximum values (since DIM server start-up) are evaluated and published. The minimum and average values can be reset by means of a DIM command published by this server or by means of a SIGHUP signal sent to the server.

3.3.1. Synopsis

```
irqSrv [-d...] [-u refresh_time_interval] -l logger  
irqSrv -h
```

3.3.2. Description

Starts the Hardware Interrupt Statistics Monitor Server, sending its log messages to the logger facility **logger** and refreshing data every **refresh_time_interval** seconds. The number of the logged messages can be increased by means of the **-d** (repeatable) option.

3.3.3. Command line options

- h** Print the program usage and exit immediately.
- l logger**
Use **logger** as logging facility. Values allowed for **logger** are in the range 0..7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVC	Send the messages to a specific DIM service.
- u refresh_time_interval**
Refresh data every **refresh_time_interval** seconds. Default: 20 s. Allowed values: > 1 s.
- d** (repeatable) Increase the number of the logged messages.

3.3.4. Environment

The program `irqSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.3.5. Services provided

`/<HOSTNAME>/irq/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getIrqFromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/irq/server_version` (string).

Returns the RCS identifier of the DIM server (`irqSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/irq/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`). This service publishes log messages sent by the server.

`/<HOSTNAME>/irq/success` (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `irqSrv` process is running.

`/<HOSTNAME>/irq/tss` (string).

This service will return a formatted string containing the time elapsed since server start-up or since the last server reset.

`/<HOSTNAME>/irq/irqNumber` (long integer array).

This service publishes the list of the ordinal number of the exploited IRQ lines.

`/<HOSTNAME>/irq/irqName` (character array containing a sequence of NULL-terminated strings).

This service publishes a list of NULL-terminated strings. Each string corresponds to an exploited IRQ line, and is the comma-separated list of the names of the devices which share such an IRQ line. Typically, on APICs-equipped PC every device is assigned to a different IRQ-line, while on PIC-equipped PC the same IRQ line can be shared among different devices (e.g.: `"aic7xxx, usb-uhci, usb-uhci, eth0"`).

Well known interrupt-source devices are:

- **timer** (IRQ line 0) – On a single processor PC the 16-bit Programmable Interval Timer (PIT); on a SMP system the 32-bit local Advanced Programmable Interrupt Counter (local APIC).
- **keyboard** or **i8042** (IRQ line 1) – The keyboard controller.
- **cascade** (IRQ line 2) – Historically, this was the line of the first Intel 8259 PIC connected to the second Intel 8259 PIC. The connection of a second 8-lines PIC in cascade to the first 8-lines PIC allowed the PC AT to have 15 interrupt lines instead of the 8 interrupt lines of a PC XT. In modern hardware, equipped with APICs, which can manage up to 64 lines, this datum is meaningless.
- **serial0/serial1** (IRQ lines 4/3) – The serial ports.
- **rtc** (IRQ line 8) – The real time clock.
- **PS/2 Mouse** (IRQ lines 12) – The mouse connected to the PS/2 mouse socket.
- **ide0/ide1** (IRQ lines 14/15) – The primary/secondary IDE channel controllers.
- **eth0/eth1/...** – The Ethernet network interfaces.
- **aic7xxx** – The Adaptec SCSI interface.
- **sym53c8xx** – The NCR SCSI interface 53C8XX.
- **usb-uhci** – The USB Universal Host Controller Interface.
- **usb-ohci** – The USB Open Host Controller Interface.
- **usb-ehci** – The USB Enhanced Host Controller Interface.
- **Intel ICH2** – The Intel I/O Controller Hub 2 (ICH2).

`/<HOSTNAME>/irq/cpuXX_Rate` (float array).

This service publishes the list of the instantaneous interrupt rates (interrupts per second) issued to the CPU number XX, partitioned on per-IRQ-line base. The data order is the same of `/<HOSTNAME>/irq/irqName` service.

`/<HOSTNAME>/irq/cpuXX_aRate` (float array).

This service publishes the list of the average interrupt rates (interrupts per second) issued to the CPU number XX, partitioned on per-IRQ-line base. The data order is the same of `/<HOSTNAME>/irq/irqName` service. These values are reset by the `/<HOSTNAME>/irq/reset` command.

`/<HOSTNAME>/irq/cpuXX_mRate` (float array).

This service publishes the list of the maximum interrupt rates (interrupts per second) issued to the CPU number XX, partitioned on per-IRQ-line base. The data order is the same of `/<HOSTNAME>/irq/irqName` service. These values are reset by the `/<HOSTNAME>/irq/reset` command.

3.3.6. Commands provided

`/<HOSTNAME>/irq/reset`

This command set to zero the cumulative interrupt counters used to evaluate the average values published in `/<HOSTNAME>/irq/cpuXX_aRate` and the maximum values published in `/<HOSTNAME>/irq/cpuXX_mRate`.

3.4. Statistics about memory usage (`memSrv`)

The `memSrv` is a DIM server which collects the statistics about the memory usage. Since the provided data fields depend on kernel version, we choose to include the fields for all the recent kernel versions (2.4 and 2.6) and to set to `-1` the fields which are not available.

The data about the used and free memory are partitioned into low memory and high memory. The **high memory** is all the memory above 896 MiB of physical memory. The high memory areas can be used by the user-space programs, or for the page-cache (but not for kernel data structures). The kernel must use some tricks to access the high memory (it temporarily maps pages from high memory into the lower page tables), making it slower to access than low memory. The **low memory**, on the other side, is memory which can be used for everything that high memory can be used for, but it is also available for the kernel's use for its own data structures (e.g. kernel slabs). The kernel cannot work if it is out of low memory.

Datum `Committed_AS` refers to the Linux kernel memory **over-committing** policy. When a process dynamically allocate memory calling the `malloc(3)` library call, and hence the `brk(2)` system call, nothing happens, really. Only when the process start using the memory allocated with the `brk(2)`, it get real memory, and just as much it uses. The hope is that programs ask for more than they actually need or that not all the running programs need the memory they have demanded with `brk(2)` *at the same time*. If the hope is fulfilled, the kernel can run more programs in the same memory space, or can run a program that requires more virtual memory than is available. If the hope is not fulfilled, an out-of-memory (OOM) condition occurs: as a consequence, the **OOM killer** is invoked to choose a process (following certain criteria) and kill it. Since kernel version 2.5.30 the over-committing behavior is controlled by the two user-settable parameters in `/proc/sys/vm/:overcommit_memory` and `overcommit_ratio`. Since kernel 2.5.30 the parameter `overcommit_memory` can be set to three different values:

- 0: heuristic over-commit handling (default);
- 1: never refuse any `brk(2)`;
- 2: strict over-commit. Never commit a virtual address space larger than swap space plus a fraction `overcommit_ratio` (default 50%) of the physical memory.

Previous kernel allowed only values 0 and 1 for `overcommit_memory`.

Data with `vmalloc` prefix concern a mechanism to map physically non-contiguous memory areas to a contiguous area in virtual memory. The size of this region will be always at least `VMALLOC_RESERVE`, which on the x86 is 128 MiB, located between

VMALLOC_START and VMALLOC_END addresses in virtual memory. The page tables in this region are adjusted as necessary to point to physical pages which are allocated with the normal physical page allocator. Since allocations require altering the kernel page tables, there is a limitation on how much memory can be mapped with `vmalloc()` as only the virtual addresses space between VMALLOC_START and VMALLOC_END is available. Usually, it is only used for storing the swap map information and for loading kernel modules into memory.

3.4.1. Synopsis

```
memSrv [-d...] [-u refresh_time_interval] -l logger
memSrv -h
```

3.4.2. Description

Starts the System Memory Monitor Server, sending its log messages to the logger facility `logger` and refreshing data every `refresh_time_interval` seconds. The number of the logged messages can be increased by means of the `-d` (repeatable) option.

3.4.3. Command line options

- `-h` Print the program usage and exit immediately.
- `-l logger`
Use `logger` as logging facility. Values allowed for `logger` are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVC	Send the messages to a specific DIM service.
- `-u refresh_time_interval`
Refresh data every `refresh_time_interval` seconds. Default: 20 s. Allowed values: > 1 s.
- `-d` (repeatable) Increase the number of the logged messages.

3.4.4. Environment

The program `memSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.4.5. Services provided

/<HOSTNAME>/mem/sensor_version (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getMemFromProc.c`), containing the version and the last modification time.

/<HOSTNAME>/mem/server_version (string).

Returns the RCS identifier of the DIM server (`memSrv.c`), containing the version and the last modification time.

/<HOSTNAME>/mem/log (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`). This service publishes log messages sent by the server.

/<HOSTNAME>/mem/success (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `memSrv` process is running.

/<HOSTNAME>/mem/Active (long integer).

This service publishes the total amount of memory which has been used more recently and usually is not reclaimed unless absolutely necessary. Since kernel 2.4.

/<HOSTNAME>/mem/ActiveAnon (long integer).

This service publishes the total amount of program memory that has been used more recently. Specific of kernel 2.4.

/<HOSTNAME>/mem/ActiveCache (long integer).

This service publishes the total amount of cache memory that has been used more recently. Specific of kernel 2.4.

`/<HOSTNAME>/mem/Buffers` (long integer).

This service publishes the total amount of memory used for relatively temporary storage for raw disk blocks (buffer cache).

`/<HOSTNAME>/mem/Cached` (long integer).

This service publishes the total amount of memory used for the in-memory cache for the files read from the disk (the page-cache). Does not include `SwapCached`.

`/<HOSTNAME>/mem/Committed_AS` (long integer).

This service publishes an estimate of how much memory the system would need to make a 99.99% guarantee that there never will be an out of memory (OOM) condition for the present workload. In other words, the `Committed_AS` is a guesstimate of how much memory the system would need in the worst case. Since kernel 2.5.41+.

`/<HOSTNAME>/mem/Dirty` (long integer).

This service publishes the total amount of memory which is waiting to get written back to the disk. Since kernel 2.5.41+.

`/<HOSTNAME>/mem/HighFree` (long integer).

This service publishes the amount of free high memory.

`/<HOSTNAME>/mem/HighTotal` (long integer).

This service publishes the total amount of high memory.

`/<HOSTNAME>/mem/Inact_clean` (long integer).

This service publishes the amount of memory which has been less recently used and has been already copied to the swap device. Because clean pages are already synchronized with respect to their backing store, the OS can reuse `Inact_clean` pages without having to write the page to a swap device. Specific of kernel 2.4.

`/<HOSTNAME>/mem/Inact_dirty` (long integer).

This service publishes the amount of memory which has been less recently used and has not yet been copied to the swap device. When memory is required, the OS chooses to steal `Inact_clean` pages before swapping `Inact_dirty` pages. Specific of kernel 2.4.

`/<HOSTNAME>/mem/Inact_laundry` (long integer).

This service publishes the amount of memory which has been less recently used and which is being copied to the swap device. Specific of kernel 2.4.

`/<HOSTNAME>/mem/Inact_target` (long integer).

This service publishes the amount of memory which OS ought to have, to be possibly reclaimed for other purposes, calculated as the sum of `Active`, `Inact_dirty`, and `Inact_clean` divided by 5. When exceeded, the kernel will not do work to move pages from active to inactive. This is an indicator of when page cleaning should be performed. Specific of kernel 2.4.

`/<HOSTNAME>/mem/Inactive` (long integer).

This service publishes the amount of memory which has been less recently used. It is more eligible to be reclaimed for other purposes. Since kernel 2.5.41+. In kernel 2.4 this value is the sum `Inact_dirty + Inact_laundry + Inact_clean`.

`/<HOSTNAME>/mem/LowFree` (long integer).

This service publishes the amount of free low memory.

`/<HOSTNAME>/mem/LowTotal` (long integer).

This service publishes the total amount of low memory.

`/<HOSTNAME>/mem/Mapped` (long integer).

This service publishes the amount of memory used to store files which have been mmaped, such as libraries. Since kernel 2.5.41+.

`/<HOSTNAME>/mem/MemFree` (long integer).

This service publishes the amount of free memory (The sum of `LowFree + HighFree`).

`/<HOSTNAME>/mem/MemShared` (long integer).

This service publishes the amount of memory shared between processes. Set to zero since kernel 2.4.

`/<HOSTNAME>/mem/MemTotal` (long integer).

This service publishes the total amount of memory, i.e. physical ram minus a few reserved bits and the kernel binary code (= `LowTotal + HighTotal`).

`/<HOSTNAME>/mem/MemUsed` (long integer).

This service publishes the amount of used memory (the difference `MemTotal - MemFree`).

`/<HOSTNAME>/mem/PageTables` (long integer).

This service publishes the amount of memory dedicated to the lowest level of page tables.

`/<HOSTNAME>/mem/ReverseMaps` (long integer).

This service publishes the number of reverse mappings performed. Since kernel 2.5.41+.

`/<HOSTNAME>/mem/Slab` (long integer).

This service publishes the amount of memory used for in-kernel data structures cache. Since kernel 2.5.41+.

`/<HOSTNAME>/mem/SwapCached` (long integer).

This service publishes the amount of memory that once was swapped out, is swapped back in but still also is in the swap store (if memory is needed it doesn't need to be swapped out *again* because it is already in the swap store. This saves I/O).

`/<HOSTNAME>/mem/SwapFree` (long integer).

This service publishes the amount of free swap space (= `SwapTotal` – `SwapUsed`).

`/<HOSTNAME>/mem/SwapTotal` (long integer).

This service publishes the total amount of swap space available.

`/<HOSTNAME>/mem/SwapUsed` (long integer).

This service publishes the amount of memory which has been evicted from RAM, and is temporarily on the disk.

`/<HOSTNAME>/mem/VmallocChunk` (long integer).

This service publishes the largest free contiguous block of vmalloc memory area.

`/<HOSTNAME>/mem/VmallocTotal` (long integer).

This service publishes the total size of vmalloc memory area.

`/<HOSTNAME>/mem/VmallocUsed` (long integer).

This service publishes the amount of vmalloc memory area which is used.

`/<HOSTNAME>/mem/Writeback` (long integer).

This service publishes the amount of memory which is actively being written back to the disk. Since kernel 2.5.41+.

3.5. Statistics about network interfaces (`nifSrv`)

The `nifSrv` is a DIM server which collects the statistics about the Network Interface Card. Statistics are mainly I/O rates (bit rates and Ethernet frame rates) and error fraction. Average and Maximum values can be reset by mean of the `/<HOSTNAME>/nif/reset` command.

The server loop of `nifSrv` must be run at least once every 34 seconds, in order to avoid the double counter reset if a gigabit Ethernet interface is transmitting or receiving at full link load (see section 2.3).

3.5.1. Synopsis

```
nifSrv [-d...] [-u refresh_time_interval] -l logger
nifSrv -h
```

3.5.2. Description

Starts the Network Interface Statistics Monitor Server, sending its log messages to the logger facility `logger` and refreshing data every `refresh_time_interval` seconds. The number of the logged messages can be increased by means of the `-d` (repeatable) option.

3.5.3. Command line options

- `-h` Print the program usage and exit immediately.
- `-l logger`
Use `logger` as logging facility. Values allowed for `logger` are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVC	Send the messages to a specific DIM service.
- `-u refresh_time_interval`
Refresh data every `refresh_time_interval` seconds. Default: 20 s. Allowed values: 1-20.
- `-d` (repeatable) Increase the number of the logged messages.

3.5.4. Environment

The program `nifSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.5.5. Services provided

In the following list the string `<if_name>` have to be substituted with the name of the network interface, e.g. `lo`, `eth0`, `eth1`, `eth2`, etc.

`/<HOSTNAME>/nif/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getNifFromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/nif/server_version` (string).

Returns the RCS identifier of the DIM server (`nifSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/nif/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`, see section 3.5.3). This service publishes log messages sent by the server.

`/<HOSTNAME>/nif/success` (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `nifSrv` process is running.

`/<HOSTNAME>/nif/tss` (integer).

This service will return a formatted string containing the time elapsed since server start-up or since the last server reset.

`/<HOSTNAME>/nif/<if_name>` (character array containing a sequence of three NULL-terminated strings).

This service publishes: (a) the **interface name** (e.g.: "lo", "eth0", "eth1", etc.); (b) the **IP address** (e.g. "10.10.10.1") or "(not assigned)" if the network interface is not configured; (c) the **MAC address** (e.g. "01:20:ed:18:21:33") or "(not ethernet)" if the MAC address is not available (as it happens on loop-back interface).

`/<HOSTNAME>/nif/<if_name>/rate` (7-float array).

This server publishes, in the order, the following 7 rates:

- **rx_bitRate** – The total number of **bits received** in a second.
- **rx_packetsRate** – The total number of **Ethernet frames received** in a second.
- **rx_multicastRate** – The total number of multicast Ethernet frames received in a second.
- **rx_compressedRate** – The total number of compressed frames received in a second (used by cslip & hdlc).
- **tx_bitRate** – The total number of **bits transmitted** in a second.
- **tx_packetsRate** – The total number of **Ethernet frames transmitted** in a second.
- **tx_compressedRate** – The total number of compressed frames transmitted in a second (used by cslip & hdlc).

`/<HOSTNAME>/nif/<if_name>/arate` (7-float array).

This server publishes, in the same order, the average value of the 7 rates above. The value can be reset by mean of the `/<HOSTNAME>/nif/reset` command.

`/<HOSTNAME>/nif/<if_name>/mrate` (7-float array).

This server publishes, in the same order, the maximum value of the 7 rates above. The value can be reset by mean of the `/<HOSTNAME>/nif/reset` command.

`/<HOSTNAME>/nif/<if_name>/ratio` (11-float array).

This server publishes, in the order, the following 11 ratios:

- **rx_bytes4packet** – The ratio `rx_bytes/rx_packets`, where `rx_bytes` is the total number of bytes received in a time interval and `rx_packets` is the total number of Ethernet frames received in a time interval. This ratio is the **mean number of bytes contained in a received Ethernet frame**.
- **rx_errorsFrac** – The ratio `rx_errors/rx_packets`, where `rx_errors` is the total number of **bad Ethernet frames** received in a time interval and `rx_packets` is the total number of Ethernet frames received in a time interval.

- **rx_droppedFrac** – The ratio $\text{rx_dropped}/\text{rx_packets}$, where rx_dropped is the number of received **Ethernet frames dropped** by operating system due to **buffer overflows or throttling policy** in a time interval and rx_packets is the total number of Ethernet frames received in a time interval.
- **rx_fifo_errorsFrac** – The ratio $\text{rx_fifo_errors}/\text{rx_packets}$, where rx_fifo_errors is the number of receiver **fifo overrun** in a time interval and rx_packets is the total number of Ethernet frames received in a time interval.
- **rx_frame_errorsFrac** – The ratio $\text{rx_frame_errors}/\text{rx_packets}$, where rx_frame_errors is the number of **receiver frame alignment errors** in a time interval and rx_packets is the total number of Ethernet frames received in a time interval.
- **tx_bytes4packet** – The ratio $\text{tx_bytes}/\text{tx_packets}$, where tx_bytes is the total number of bytes transmitted in a time interval and tx_packets is the total number of Ethernet frames transmitted in a time interval. This ratio is the **mean number of bytes contained in a transmitted Ethernet frame**.
- **tx_errorsFrac** – The ratio $\text{tx_errors}/\text{tx_packets}$, where tx_errors is the total number of **errors due to packet transmit problems** in a time interval and tx_packets is the total number of transmitted Ethernet frames in a time interval.
- **tx_droppedFrac** – The ratio $\text{tx_dropped}/\text{tx_packets}$, where tx_dropped is the number of transmitted **Ethernet frames dropped** by operating system due to **buffer overflows or throttling policy** in a time interval and tx_packets is the total number of transmitted Ethernet frames in a time interval.
- **tx_fifo_errorsFrac** – The ratio $\text{tx_fifo_errors}/\text{tx_packets}$, where tx_fifo_errors is the number of **transmitter fifo overrun** in a time interval and tx_packets is the total number of transmitted Ethernet frames in a time interval.
- **tx_carrier_errorsFrac** – The ratio $\text{tx_carrier_errors}/\text{tx_packets}$, where tx_carrier_errors is the number of **transmission errors due to loss of carrier** in a time interval and tx_packets is the total number of Ethernet frames transmitted in a time interval. If this ratio is greater than zero there is probably a **cable/connector problem** or a **bad duplex setting**.
- **collisionsFrac** – The ratio $\text{collisions}/\text{tx_packets}$, where collisions is the total number of **Ethernet collisions** in a time interval and tx_packets is the total number of transmitted Ethernet frames in a time interval.

/<HOSTNAME>/nif/<if_name>/aratio (11-float array).

This server publishes, in the same order, the average value of the 11 ratios above. The value can be reset by mean of the `/<HOSTNAME>/nif/reset` command.

`/<HOSTNAME>/nif/<if_name>/mratio` (11-float array).

This server publishes, in the same order, the maximum value of the 11 ratios above. The value can be reset by mean of the `/<HOSTNAME>/nif/reset` command.

3.5.6. Commands provided

`/<HOSTNAME>/nif/reset`

This command set to zero the cumulative interrupt counters used to evaluate the average values published in `/<HOSTNAME>/nif/arate` and `/<HOSTNAME>/nif/aratio` and the maximum values published in `/<HOSTNAME>/nif/mrate` and `/<HOSTNAME>/nif/mratio`.

3.6. Statistics about network interfaces coalescence (`coalSrv`)

Almost all the gigabit Ethernet NICs (Network Interface Cards) implement nowadays an *interrupt moderation* mechanism: instead of transferring the single Ethernet frame to the kernel memory and rising immediately a hardware interrupt, the gigabit Ethernet NICs store up a group of frames in an internal buffer and send the frame group in a single DMA transfer, rising one interrupt only at the end of the group transfer [13] [14]. This mechanism is estimated to be able to reduce CPU utilization up to 30% in frame receiving and up to 11% in frame sending.

As a matter of fact, if a network interface transmits or receive frames one-by-one at full link load using the standard Ethernet frames of 1518 B MTU (Maximum Transmission Unit), then the huge corresponding interrupt rate could induce an *interrupt live-lock* condition in the CPU (the CPU is so busy in serving the interrupts that is no more able to process the frames it receives). The so-called Ethernet jumbo frames (9018 B MTU) improve the operations, but are not so portable across the different NIC and Ethernet switches.

The interrupt moderation mechanism can be tuned by modifying some parameters in the NIC driver (e.g.: `InterruptThrottleRate`, `TxIntDelay`, `TxAbsIntDelay`, `RxIntDelay` and `RxAbsIntDelay` in the Intel e1000 driver) or can be turned off, in order to leave the interrupt moderation task to the Linux kernel NAPI mechanism [13] [14].

The `coalSrv` is DIM server which evaluates and publishes the ratios between the number of frames sent/received by the network interfaces and the corresponding number

of interrupts raised by the network interface cards. Under light frame transfer rate this ratio is expected to be close to 1, while under heavy frame transfer rate this ratio can sensibly increase.

The server contacts both the interrupt sensor and the NIC sensor, matches the data corresponding to the same network interface and publishes the ratio.

3.6.1. Synopsis

```
coalSrv [-d...][-u refresh_time_interval] -l logger
coalSrv -h
```

3.6.2. Description

Starts the Network Interface Coalescence Monitor Server, sending its log messages to the logger facility **logger** and refreshing data every **refresh_time_interval** seconds. The number of the logged messages can be increased by means of the **-d** (repeatable) option.

3.6.3. Command line options

- h** Print the program usage and exit immediately.
- l logger**
Use **logger** as logging facility. Values allowed for **logger** are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVCS	Send the messages to a specific DIM service.
- u refresh_time_interval**
Refresh data every **refresh_time_interval** seconds. Default: 20 s. Allowed values: 1-20.
- d** (repeatable) Increase the number of the logged messages.

3.6.4. Environment

The program `coalSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFMensors-0.so`.

3.6.5. Services provided

`/<HOSTNAME>/coalescence/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getCoalescence.c`), containing the version and the last modification time.

`/<HOSTNAME>/coalescence/server_version` (string).

Returns the RCS identifier of the DIM server (`coalSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/coalescence/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`, see section 3.5.3). This service publishes log messages sent by the server.

`/<HOSTNAME>/coalescence/success` (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `coalSrv` process is running.

`/<HOSTNAME>/coalescence/tss` (integer).

This service will return a formatted string containing the time elapsed since server start-up or since the last server reset.

`/<HOSTNAME>/coalescence/name` (character array containing a sequence of NULL-terminated strings).

This service publishes a list of NULL-terminated strings. Each string corresponds to a network interface name (e.g. `eth0`, `eth1`, etc.).

`/<HOSTNAME>/coalescence/Factor` (float array).

This service publishes the list of the **current** coalescence factor of the network interfaces. The order of the data is the same of the `/<HOSTNAME>/coalescence/name` service.

`/<HOSTNAME>/coalescence/aFactor` (float array).

This service publishes the list of the **average** coalescence factor of the network interfaces. The order of the data is the same of the `/<HOSTNAME>/coalescence/name` service. These values are reset by the `/<HOSTNAME>/coalescence/reset` command.

`/<HOSTNAME>/coalescence/mFactor` (float array).

This service publishes the list of the **maximum** coalescence factor of the network interfaces. The order of the data is the same of the `/<HOSTNAME>/coalescence/name` service. These values are reset by the `/<HOSTNAME>/coalescence/reset` command.

3.6.6. Commands provided

`/<HOSTNAME>/coalescence/reset`

This command set to zero the cumulative counters of both interrupts and network interfaces, used to evaluate the average values published in `/<HOSTNAME>/coalescence/aFactor` and the maximum values published in `/<HOSTNAME>/coalescence/mFactor`.

3.7. Statistics about TCP/IP stack (`tcPIPsrv`)

The `tcPIPsrv` is a DIM server which collects the statistics about the operation of the kernel TCP/IP stack implementation. Statistics can be classified in:

- *I/O rates* (input/output/forwarded/delivered IP datagram rates, UDP input/output datagram rates, TCP input/output segment rates);
- *IP fragmentation rates* (fragmentation rate, reassembling rate);
- *IP delivery fractions* (fraction of datagrams which have been delivered, fraction of datagrams which have been forwarded);
- *IP fragmentation fractions* (fraction of transmitted datagrams which require fragmentation, mean number of fragments for received datagram);
- *error fractions* (fraction of input IP datagrams discarded for several reasons, fraction of output IP datagrams which cannot be fragmented, fraction of input IP datagram for which the reassembling failed, etc.).

The server provides also the average and maximum values for these above quantities, which can be reset by mean of the `/<HOSTNAME>/tcPIP/reset` command.

3.7.1. Synopsis

```
tcpipSrv [-d...] [-u refresh_time_interval] -l logger
```

```
tcpipSrv -h
```

3.7.2. Description

Starts the TCP/IP Stack Statistics Monitor Server, sending its log messages to the logger facility **logger** and refreshing data every **refresh_time_interval** seconds. The number of the logged messages can be increased by means of the **-d** (repeatable) option.

3.7.3. Command line options

-h Print the program usage and exit immediately.

-l logger

Use **logger** as logging facility. Values allowed for **logger** are in the range 0...7. The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVCS	Send the messages to a specific DIM service.

-u refresh_time_interval

Refresh data every **refresh_time_interval** seconds. Default: 20 s. Allowed values: 1-20.

-d (repeatable) Increase the number of the logged messages.

3.7.4. Environment

The program `tcpipSrv` needs the two environment variables:

DIM_DNS_NODE

hostname.domain of DIM dns node.

LD_LIBRARY_PATH

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so` and `libSFM sensors-0.so`.

3.7.5. Services provided

`/<HOSTNAME>/tcpip/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getTCPIPfromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/tcpip/server_version` (string).

Returns the RCS identifier of the DIM server (`tcpipSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/tcpip/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`, see section 3.5.3). This service publishes log messages sent by the server.

`/<HOSTNAME>/tcpip/success` (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `tcpipSrv` process is running.

`/<HOSTNAME>/tcpip/tss` (integer).

This service will return a formatted string containing the time elapsed since server start-up or since the last server reset.

`/<HOSTNAME>/tcpip/data-rate` (10-float array).

The floating point array returned by this service provides the following 10 rates in the following order:

- *IP rates:*
 - **InReceivesRate** [datagrams/s]: The total number of input IP datagrams received from interfaces in a second, including those received in error.
 - **InDeliversRate** [datagrams/s]: The total number of input IP datagrams successfully delivered to IP user-protocols (including ICMP) in a second.
 - **ForwDatagramsRate** [datagrams/s]: the number of input IP datagrams in a second, for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination.

- **OutRequestsRate** [datagrams/s]: The total number of IP datagrams which local IP user-protocols (including ICMP) supplied to IP in a second in requests for transmission. Note that this counter does not include any datagrams counted in `ipForwDatagrams`.
- **ReasmReqdsRate** [fragments/s]: The number of IP fragments received in a second, which needed to be reassembled at this entity.
- **FragReqdsRate** [datagrams/s]: The number of IP datagrams in a second that need to be fragmented. This is the rate of the sum of counters `FragOKs` and `FragFails`.
- *TCP rates:*
 - **InSegsRate** [segments/s]: The total number of TCP segments received in a second, including those received in error. This rate includes the segments received on currently established connections.
 - **OutSegsRate** [segments/s]: The total number of TCP segments sent in a second, including those on current connections but excluding those containing only retransmitted octets.
- *UDP rates:*
 - **InDatagramsRate** [datagrams/s]: The total number of UDP datagrams delivered to UDP users in a second.
 - **OutDatagramsRate** [datagrams/s]: The total number of UDP datagrams sent from this entity in a second.

`/<HOSTNAME>/tcpip/data-ratio` (20-float array).

The floating point array returned by this service provides the following 20 ratios in the following order:

- *IP ratios:*
 - **InHdrErrorsFrac**: The **fraction of the input datagrams** received by all the interfaces which has been **discarded due to errors in their IP headers**, (including **bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options**, etc.). It is evaluated as the ratio between the kernel counters: `InHdrErrors/InReceives`.
 - **InAddrErrorsFrac**: The **fraction of the input datagrams** received by all the interfaces which has been **discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity**. This fraction includes invalid addresses (e.g., 0.0.0.0) and addresses

of unsupported Classes (e.g., Class E). For entities which are not IP routers and therefore do not forward datagrams, this fraction includes datagrams discarded because the destination address was not a local address. It is evaluated as the ratio between the kernel counters: $\text{InAddrErrors}/\text{InReceives}$.

- **InUnknownProtosFrac**: The **fraction of the input datagrams** received by all the interfaces, which are locally-addressed but have been **discarded because of an unknown or unsupported protocol**. It is evaluated as the ratio between the kernel counters: $\text{InUnknownProtos}/\text{InReceives}$.
- **InDiscardsFrac**: The **fraction of the input datagrams** received by all the interfaces, for which no problems were encountered to prevent their continued processing, but which were **discarded** (e.g., for **lack of buffer space**). This fraction does not include any datagrams discarded while awaiting re-assembly. It is evaluated as the ratio between the kernel counters: $\text{InDiscards}/\text{InReceives}$.
- **ForwDatagramsFrac**: The **fraction of the input datagrams** received by all the interfaces, for which **this entity was not their final IP destination**, as a result of which an attempt was made to find a route to forward them to that final destination. It is evaluated as the ratio between the kernel counters: $\text{ForwDatagrams}/\text{InReceives}$.
- **InDeliversFrac**: The **fraction of the input datagrams** received by all the interfaces which have been **successfully delivered to IP user-protocols** (including ICMP). It is evaluated as the ratio between the kernel counters: $\text{InDelivers}/\text{InReceives}$.
- **ReasmReqdsFrac** [fragments/datagram]: The **mean number of IP fragments for each received input datagram**. It is evaluated as the ratio between the kernel counters: $\text{ReasmReqds}/\text{InReceives}$.
- **ReasmTimeoutFrac**: The **fraction of the input datagrams which need re-assembly** at this entity, received by all the interfaces, for which the **re-assembly algorithm fails due to reassembling time-out**. It is evaluated as the ratio between the kernel counters: $\text{ReasmTimeout}/(\text{ReasmOKs} + \text{ReasmFails})$.
- **ReasmFailsFrac**: The **fraction of the input datagrams which need re-assembly**, received by all the interfaces, for which the **re-assembly algorithm fails** (for **whatever reason**: time-out, errors, etc). Note that this is not necessarily the fraction of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received. It is evaluated as the ratio between the kernel counters: $\text{ReasmFails}/(\text{ReasmOKs} + \text{ReasmFails})$.

- **ReasmOKsFrac**: The fraction of the input datagrams which need re-assembly at this entity, received by all the interfaces which are successfully re-assembled. It is evaluated as the ratio between the kernel counters: $\text{ReasmOKs} / (\text{ReasmOKs} + \text{ReasmFails})$.
 - **OutDiscardsFrac**: the fraction of the total number of IP datagrams, which local IP user-protocols supplied to IP in requests for transmission, for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this fraction would include datagrams counted in `ipForwDatagrams` if any such packets met this (discretionary) discard criterion. It is evaluated as the ratio between the kernel counters: $\text{OutDiscards} / (\text{OutRequests} + \text{ForwDatagrams})$.
 - **OutNoRoutesFrac**: the fraction of the total number of IP datagrams, which local IP user-protocols supplied to IP in requests for transmission, discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in `ipForwDatagrams` which meet this "no-route" criterion. Note also that this includes any datagrams which a host cannot route because all of its default routers are down. It is evaluated as the ratio between the kernel counters: $\text{OutNoRoutes} / (\text{OutRequests} + \text{ForwDatagrams})$.
 - **FragReqdsFrac**: the fraction of the total number of IP datagrams, which local IP user-protocols supplied to IP in requests for transmission, that need to be fragmented at this entity. It is evaluated as the ratio between the kernel counters: $(\text{FragOKs} + \text{FragFails}) / (\text{OutRequests} + \text{ForwDatagrams})$.
 - **FragFailsFrac**: the fraction of the total number of IP datagrams, which local IP user-protocols supplied to IP in requests for transmission, that need to be fragmented at this entity that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their "Don't Fragment" flag was set. It is evaluated as the ratio between the kernel counters: $\text{FragFails} / (\text{FragOKs} + \text{FragFails})$.
 - **FragCreatesFrac** [fragments/datagram]: The mean number of IP fragments created for each transmitted output datagram which need to be fragmented. It is evaluated as the ratio between the kernel counters: $\text{FragCreates} / \text{FragOKs}$.
- *TCP ratios*:
 - **RetransSegsFrac**: the fraction of sent TCP segments which contains only retransmitted octets. It is evaluated as the ratio between the kernel counters: $\text{RetransSegs} / (\text{RetransSegs} + \text{OutSegs})$.

- **OutRstsFrac**: the fraction of sent TCP segments which contains the **RST flag**. It is evaluated as the ratio between the kernel counters: `OutRsts/OutSegs`.
- **InErrsFrac**: the fraction of received TCP segments which is in error (e.g., bad TCP checksums). It is evaluated as the ratio between the kernel counters: `InErrs/InSegs`.
- *UDP ratios*:
 - **NoPortsFrac**: the fraction of received UDP datagrams for which there was no application at the destination port. It is evaluated as the ratio between the kernel counters: `NoPorts/InDatagrams`.
 - **InErrorsFrac**: the fraction of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port. It is evaluated as the ratio between the kernel counters: `InErrors/InDatagrams`.

`/<HOSTNAME>/tcpip/data-arate` (10-float array).

The floating point array returned by this service provides the average values of the 10 rates provided by the service `/<HOSTNAME>/tcpip/data-rate` in the same order.

`/<HOSTNAME>/tcpip/data-aratio` (20-float array).

The floating point array returned by this service provides the average values of the 20 ratios provided by the service `/<HOSTNAME>/tcpip/data-ratio` in the same order.

`/<HOSTNAME>/tcpip/data-mrate` (10-float array).

The floating point array returned by this service provides the maximum values of the 10 rates provided by the service `/<HOSTNAME>/tcpip/data-rate` in the same order.

`/<HOSTNAME>/tcpip/data-mratio` (20-float array).

The floating point array returned by this service provides the maximum values of the 20 ratios provided by the service `/<HOSTNAME>/tcpip/data-ratio` in the same order.

3.7.6. Commands provided

`/<HOSTNAME>/tcpip/reset`

This command set to zero the cumulative counters used to evaluate the average values published in `/<HOSTNAME>/tcpip/data-arate` and `/<HOSTNAME>/tcpip/data-aratio` and the maximum values published in `/<HOSTNAME>/tcpip/data-mrate` and `/<HOSTNAME>/tcpip/data-mratio`.

3.8. Statistics about processes (`psSrv`)

The `psSrv` is a DIM server which collects the statistics about the processes in execution on the PC (statistics usually provided by the “`ps`” or the “`top`” utilities). For each process, statistics includes executable image name, command line, user, group, memory and CPU share, CPU time, elapsed time, number of threads, scheduler, static priority, nice level and pending/catched/ignored/blocked signal mask.

Since this server must cope with **deep changes in Linux kernel threading model**, we chose to make the sensor access kernel data in `procfs` by means of the maintained (but undocumented) library `libproc` available from <http://procps.sourceforge.net/>.

As a matter of fact in kernel 2.4.20 Linux switched from **LinuxThreads** to **NPTL** (Native POSIX Threading Library). The main differences between the two models, from the user point of view, are here summarized:

- *LinuxThreads*, Linux kernel $\leq 2.4.19$:
 - Each thread has a unique Process ID (PID).
 - The `getpid()` function therefore returns different values (PID) for the different threads of the same process.
- *NPTL*, Native POSIX Threading Library, Linux kernel $\geq 2.4.20$
 - Each thread has a unique identifier called TID (Thread Identifier) while the PID has been replaced by TGID (Thread Group Identifier).
 - The `getpid()` function therefore returns the same value (TGID) for all threads in a process.

To determine which threading library an executable file actually uses, the following bash command may be typed:

```
`ldd <executable_path> | grep libc.so.6 | cut -d' ' -f 3`
```

To force the use of LinuxThreads, with kernels $\geq 2.4.20$ the following bash command may be typed:

```
export LD_ASSUME_KERNEL=2.4.19
```

Another change has been introduced since Linux kernel 2.6: the Linux 2.4 kernel does not provide proper support for grouping threads by process, while the Linux 2.6 kernel allows for proper thread grouping and reporting. As a matter of fact, in kernel 2.4 process data in `procfs` are stored in:

```
/proc/<tid>/...
```

while in kernel 2.6 they are stored in:

```
/proc/<tgid>/task/<tid>/...
```

Hacks exist to group them anyway in kernel 2.4, but such hacks will falsely group similar tasks and will fail to group tasks due to race conditions. As none of this is acceptable in a critical system tool, **task grouping is not currently available for the 2.4 kernel in sourceforge psproc project.**

As a consequence, the monitor server `psSrv` will show correctly the statistics about the different threads in the same process only using kernel 2.6.

3.8.1. Synopsis

```
psSrv [-d...] [-u refresh_time_interval] -l logger  
psSrv -h
```

3.8.2. Description

Starts the System Processes Monitor Server, sending its log messages to the logger facility `logger` and refreshing data every `refresh_time_interval` seconds. The number of the logged messages can be increased by means of the `-d` (repeatable) option.

3.8.3. Command line options

`-h` Print the program usage and exit immediately.

`-l logger`

Use `logger` as logging facility. Values allowed for `logger` are in the range 0...7.

The value is the result of a bitwise OR of the following values:

0x0	NOLOG	Don't write log at all (default).
0x1	DIMLOGGER	Send the messages to the LHCb Message Logger.
0x2	STDERRLOG	Send the messages to stderr.
0x4	DIMSVC	Send the messages to a specific DIM service.

`-u refresh_time_interval`

Refresh data every `refresh_time_interval` seconds. Default: 20 s. Allowed values: 1-20.

-d (repeatable) Increase the number of the logged messages.

3.8.4. Environment

The program `psSrv` needs the two environment variables:

`DIM_DNS_NODE`

hostname.domain of DIM dns node.

`LD_LIBRARY_PATH`

Variable, in PATH format, which must contain the path of the shared libraries `libdim.so`, `libSFMutils.so`, `libproc-3.2.3.so` and `libSFMsensors-0.so`.

3.8.5. Services provided

`/<HOSTNAME>/procs/sensor_version` (string).

Returns the RCS (Revision Control System) identifier of the sensor (`getPsFromProc.c`), containing the version and the last modification time.

`/<HOSTNAME>/procs/server_version` (string).

Returns the RCS identifier of the DIM server (`psSrv.c`), containing the version and the last modification time.

`/<HOSTNAME>/procs/log` (string).

(Available only if the server is started with the option `-l 4`, `-l 5`, `-l 6` or `-l 7`, see section 3.5.3). This service publishes log messages sent by the server.

`/<HOSTNAME>/procs/success` (integer).

This service will return 1 if published data are meaningful, and 0 if published data are meaningless due to an error in sensor operation. It is also used by PVSS-DIM to check if `psSrv` process is running.

`/<HOSTNAME>/procs/nprocs` (integer)

This service returns the number of processes currently in execution on the PC.

`/<HOSTNAME>/procs/updates` (integer)

This service returns the number of refreshes of the process list since the server start-up.

`/<HOSTNAME>/procs/CMD` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 16 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **name of the executable image file** of the task, without arguments, truncated at 15 characters if it is longer. This is the same field labeled as “COMMAND” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/CMDLINE` (character array containing a sequence of *nprocs* NULL-terminated strings that may contain spaces, each long at most 240 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **name of the executable image file of the process, with all the command line arguments**, truncated at 239 characters if it is longer. The string may thus contain the spaces separating command from arguments and arguments among themselves. This is the same field labeled as “CMDLINE”, “CMD”, “ARGS” or “COMMAND” in `ps` and `top` utilities, in their different flavours (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/USER` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 20 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **effective user name** of the process, truncated at 19 characters if it is longer. Note that Linux distinguish among *real user name* (the user who created the process, i.e. the effective user of the calling process), *effective user name* (which determines the files that the process can access, and can be different from the real user if a `setuid(2)` has been called), *saved user name* (the effective user that the process had when it started) and *file-system user name* (applications like NFS servers set the file-system user to a different value than the effective user so they can read/write files as a user, without that user being able to send signals to the server process). This is the same field labeled as “USER”, or “EUSER” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/GROUP` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 20 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **effective group name** of the process, truncated at 19 characters if it is longer. Note that Linux distinguish between *real group name* (the group who created the process, i.e. the group of the calling process), *effective group name* (which determines the files that the process can access, and can be different from the real group if a `setgid(2)` has been called), *saved group name* (the effective group that the process had when it started) and *file-system group name* (applications like NFS servers set the file-system group to a different value than the

effective group so they can read/write files as a group, without that group being able to send signals to the server process). This is the same field labeled as “GROUP”, or “EGROUP” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/TGID` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **Process Identifier** number (PID) of the task (kernels $\leq 2.4.19$) or the **Thread Group Identifier** number of the task (kernels $\geq 2.4.20$). Note that in kernels $\leq 2.4.19$ a different PID is assigned to each thread of the same process, while in kernels $\geq 2.4.20$ each thread of the same process has the same TGID but different TID; in kernels $\leq 2.4.19$ the system call `getpid(2)` returns a different PID for each thread of the same process, while in kernels $\geq 2.4.20$ the system call `getpid(2)` returns the same TGID for each thread of the same process. This is the same field labeled as “PID”, or “TGID” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/UTGID` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 256 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **User assigned unique Thread Group Identifier** of the process, truncated at 255 characters if it is longer, read from the process environment (assigned, e.g., by the LHCb Task Manager [15] or by a script which set the UTGID environment variable before starting the process). For processes which have no UTGID set, the string “N/A” is published. This field is absent in `ps` and `top` utilities.

`/<HOSTNAME>/procs/PPID` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **Parent Process Identifier** number (PPID) of the task, i.e. the TGID of the process which forked the current task, or the TGID of the “init” task if the task which forked the current task has terminated. This is the same field labeled as “PPID” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/TID` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **Thread Identifier** number (TID) of the task, also known as **Light Weight Process ID** (LWP). This is the same field labeled as “TID”, “SPID”, or “LWP” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/NLWP` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **number of the threads** (alias light weight processes, LWP) **in the process**. This is the same field labeled as “NLWP” or “THCNT” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/SIZE` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **size [in KiB] of the core image of the task** (*code + data + stack*). This is the same field labeled as “SIZE” or “SZ” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/RSS` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **Resident Set Size [in KiB] of the task**, i.e. the **non-swapped physical memory** that the task has used. This is the same field labeled as “RSS”, “RSZ” or “RES” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/SHARE` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **amount [in KiB] of the shared memory used by the task**. This is the same field labeled as “SHARE” or “SHR” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/VSIZE` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **amount [in KiB] of the virtual memory usage of the entire process** (*lib + exe + data + stack*). This is the same field labeled as “VSZ” or “VIRT” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/PSR` (array of *nproc* integers).

This service publishes, for each task in execution on the PC, the **processor that process is currently assigned to**. It is useful to check the operation of process-to-CPU affinity setting by the LHCb Task Manager [15]. This is the same field labeled as “PSR” or “P” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/STAT` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 6 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **multi-characters process state** of the task. The meaning of the first character is:

- **D** Uninterruptible sleep (usually I/O);
- **R** Running or runnable (on run queue);
- **S** Interruptible sleep (waiting for an event to complete);
- **T** Stopped, either by a job control signal or because it is being traced;
- **X** dead (should never be seen);
- **Z** Defunct ("zombie") process, terminated but not reaped by its parent.

The meaning of the following characters is:

- **<** high-priority (not nice to other users);

- **N** low-priority (nice to other users);
- **L** has pages locked into memory (for real-time and custom IO);
- **s** is a session leader;
- **l** is multi-threaded (using CLONE_THREAD, like NPTL pthreads do);
- **+** is in the foreground process group.

This is the same field labeled as “STAT” or “S” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/%CPU` (array of *nproc* floats).

This service publishes, for each task in execution on the PC, the **task's share of the CPU time since the last update, expressed as a percentage of total CPU time per processor**. This is the same field labeled as “%CPU” or “C” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/%MEM` (array of *nproc* floats).

This service publishes, for each task in execution on the PC, the **ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage**. This is the same field labeled as “%MEM” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/CLS` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 6 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **scheduling class** of the task. The meaning of the first character is:

- “_” Not reported;
- “TS” SCHED_OTHER (time-sharing, dynamic priority, linux default);
- “FF” SCHED_FIFO (real-time, static priority, first in first out);
- “RR” SCHED_RR (real-time, static priority, round robin);
- “?” unknown value.

This is the same field labeled as “CLS”, “POL”, or “SCH” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/RTPRIO` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 6 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **real-time (static) priority** of the task. RTPRIO is defined only for real-time tasks (scheduled with SCHED_FIFO or SCHED_RR), for which it is in the range “1”... “99” and it is set to “N/A” for time-sharing tasks (scheduled with SCHED_OTHER). This is the same field labeled as “RTPRIO” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/NICE` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 6 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **nice level** of the task (used by the time sharing scheduler `SCHED_OTHER` to evaluate the dynamic priority). `NICE` is defined only for time-sharing tasks (scheduled with `SCHED_OTHER`), for which it is in the range 19 (nicest) ... -20, and it is set to “N/A” for real-time tasks (scheduled with `SCHED_FIFO` or `SCHED_RR`). This is the same field labeled as “NI” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/PRIO` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 6 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **dynamic priority** of the task. `PRIO` is defined only for time-sharing tasks (scheduled with `SCHED_OTHER`) and it is set to “RT” for real-time tasks (scheduled with `SCHED_FIFO` or `SCHED_RR`). This is the same field labeled as “PRI” or “PR” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/STARTED` (character array containing a sequence of *nprocs* NULL-terminated strings that may contain spaces, each long at most 25 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **time in which the process started** in the format: “Tue Jun 14 15:10:44 2005”. This is the same field labeled as “START” or “STARTED” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/ELAPSED` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 13 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **elapsed time since the process was started** in the format: “071-19:13:01”. This is the same field labeled as “ELAPSED” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/CPUTIME` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 13 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **cumulative CPU time** in the format: “071-19:13:01”. This is the same field labeled as “TIME” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/TTY` (character array containing a sequence of *nprocs* NULL-terminated strings, each long at most 9 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **controlling terminal (if any)**, that could be, e.g. “tty6”, “pts/14” or “?” if the process is not bound with a

terminal, as the daemon processes are. This is the same field labeled as “TTY” or “TT” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/PENDING` (character array containing a sequence of `nprocs` NULL-terminated strings, each long 17 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **mask of the pending signals**. This is the same field labeled as “PENDING” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/CATCHED` (character array containing a sequence of `nprocs` NULL-terminated strings, each long 17 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **mask of the caught signals**. This is the same field labeled as “CATCHED” or “CAUGHT” in `ps` and `top` utilities, in their different flavors (Unix98, BSD, GNU, etc.).

`/<HOSTNAME>/procs/IGNORED` (character array containing a sequence of `nprocs` NULL-terminated strings, each long 17 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **mask of the ignored signals**. This is the same field labeled as “IGNORED” in `ps` and `top` utilities.

`/<HOSTNAME>/procs/BLOCKED` (character array containing a sequence of `nprocs` NULL-terminated strings, each long 17 characters, including the terminating ‘\0’).

This service publishes, for each task in execution on the PC, the **mask of the blocked signals**. This is the same field labeled as “BLOCKED” in `ps` and `top` utilities.


4. The PVSS Monitor Clients

The PVSS Monitor Clients allows to collect centrally, from a separate Monitor PC, the farm working indicators provided by the Monitor Servers described above, to show them in tables and plots and to set the appropriate alarm condition for the experiment's Finite State Machine (FSM) in case an error or a critic farm condition comes up.

The meaning of all the parameters reported in the Monitor Clients' windows can be shown by right-clicking the parameter field.

In all the Monitor Clients an additional panel is provided, to set the alarm thresholds for the Finite State Machine.

The screen dumps, shown in the following figures, illustrate them in details.



Label	Value
bogomips	5583.66015625
cpuMHz	2800.2370605469
family	15
kbCacheSize	1024
model	4
modelName	Intel(R) Xeon(TM) CPU 2.80GHz
physicalId	0
siblings	1
stepping	1
vendorId	GenuineIntel

Figure 3. The CPU information PVSS client. By right-clicking a field in this window, a help window, which contains the field description (shown, for example, in Figure 4), pops-up.

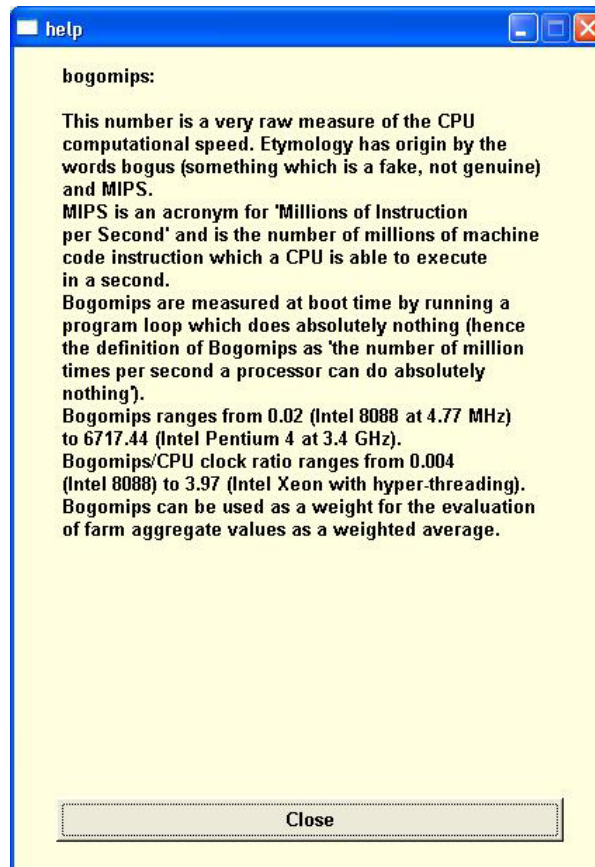


Figure 4. The CPU information PVSS help window, obtained by right-clicking the `bogomips` field in the CPU information PVSS client, shown in Figure 3.

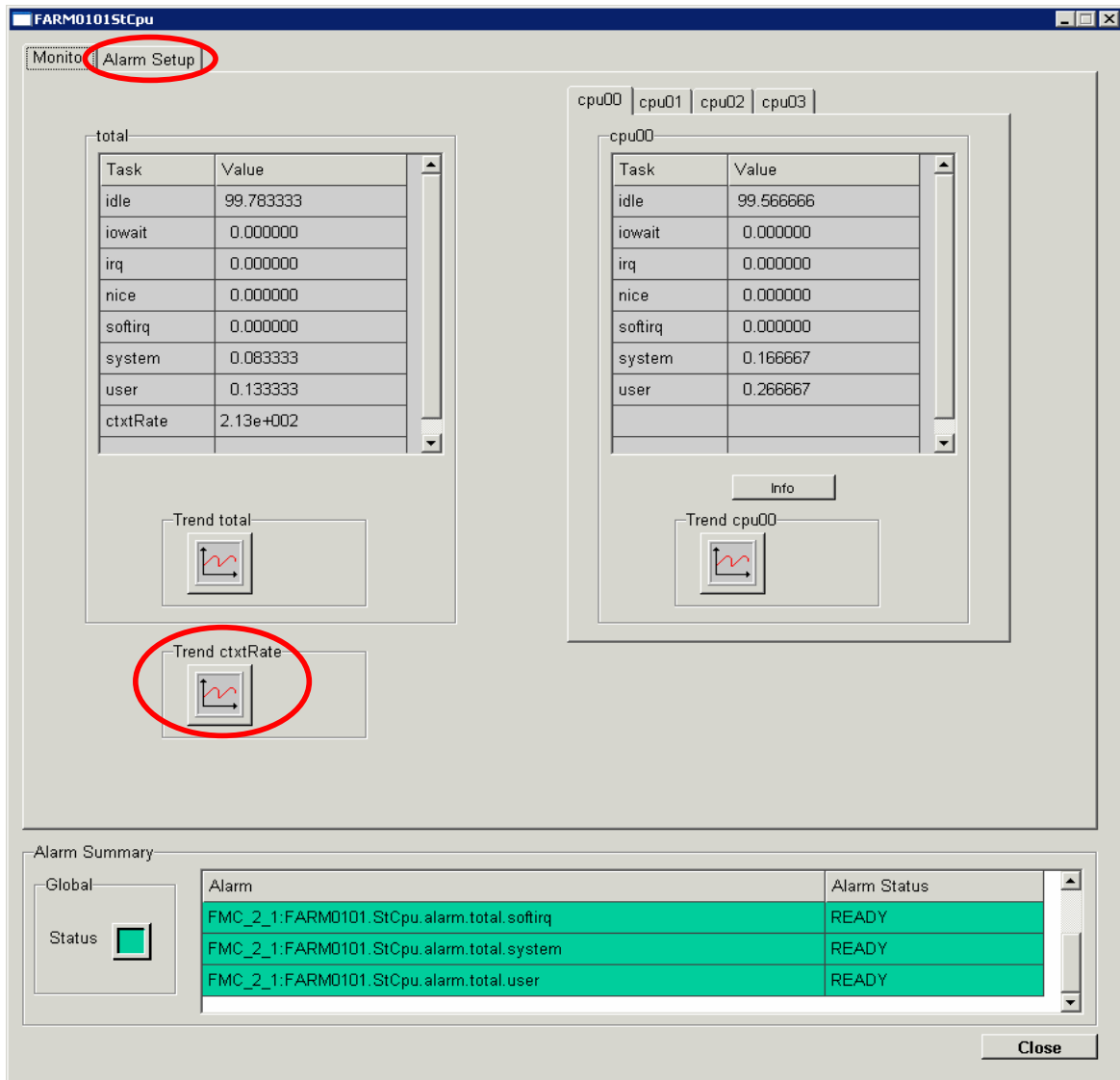


Figure 5. The CPU statistics PVSS client. The tabbed pane to switch to the alarm setting panel (shown in Figure 7) and the button to open context switch rate vs time plot (shown in Figure 6) are emphasized.

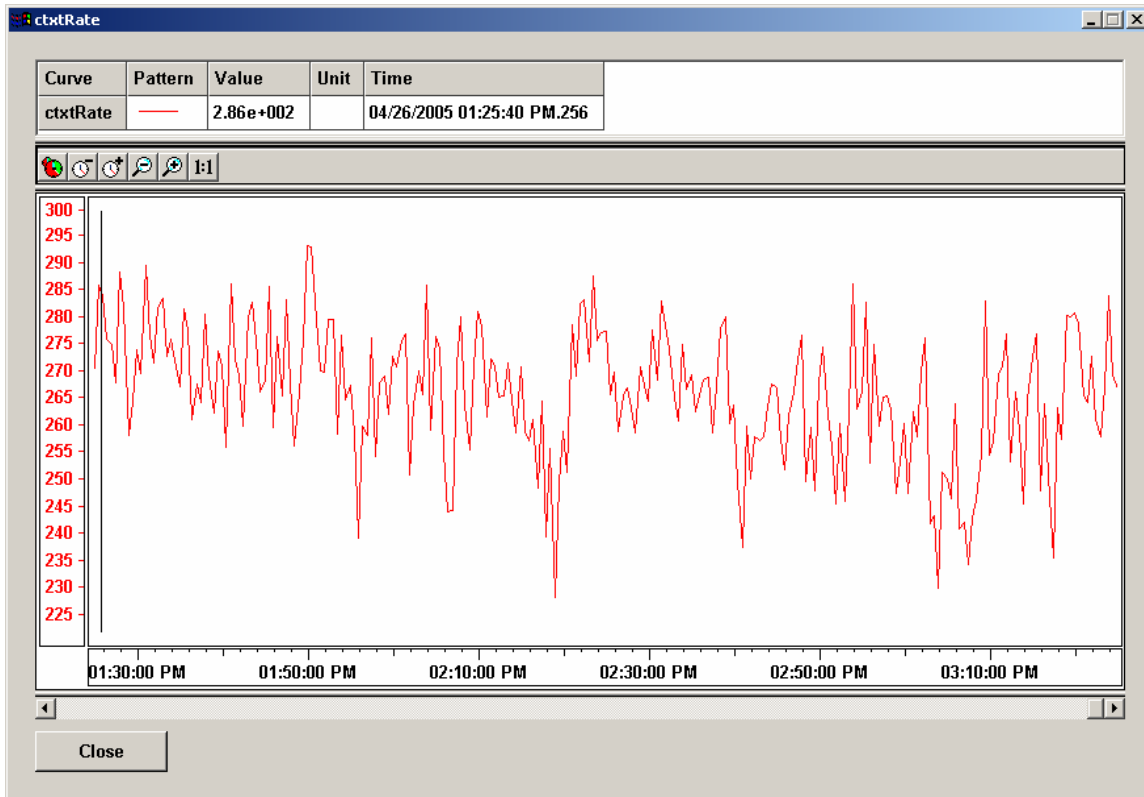


Figure 6. The CPU statistics PVSS client. Plot of the time evolution of the global CPU context switch rate.

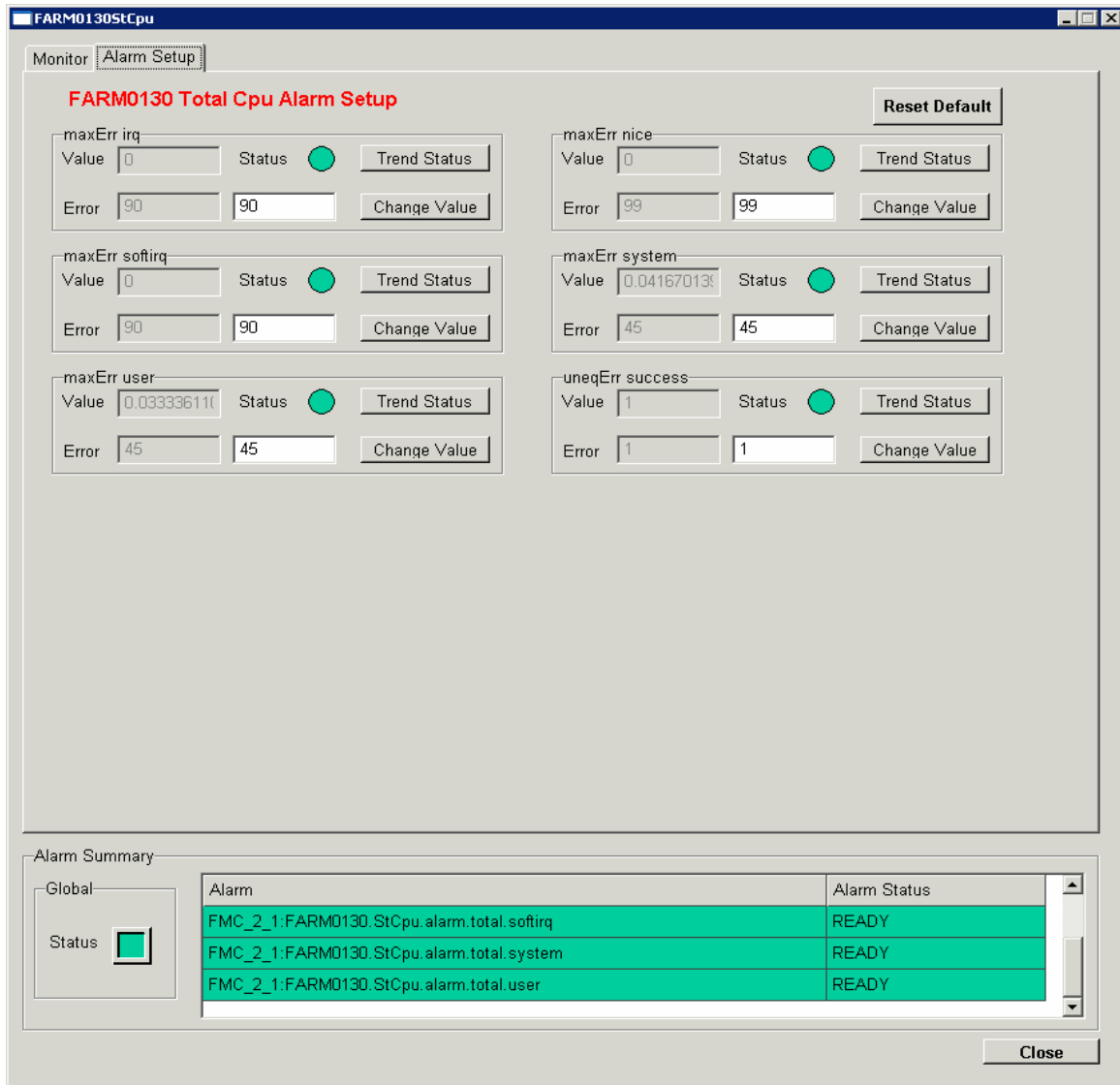


Figure 7. The CPU statistics PVSS client. Alarm setup panel, which allows setting alarm thresholds for the FSM.

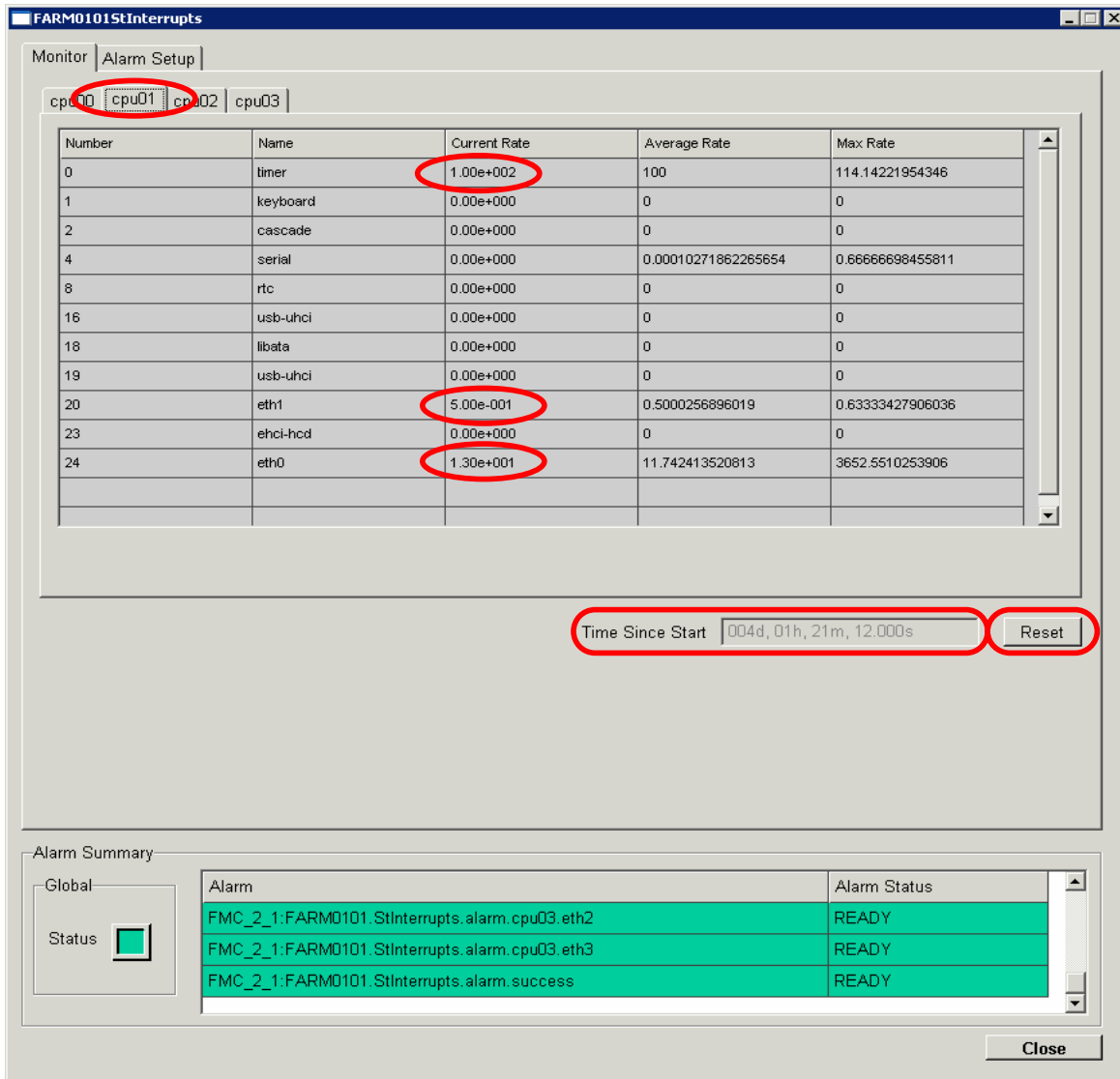


Figure 8. The Interrupts statistics PVSS client. These data refer to the CPU 1 (emphasized tabbed pane) of a PC running the kernel 2.4 as can be recognized by the timer interrupts rate (100 s^{-1} , emphasized). The timer interrupts are all handled by the CPU 1 (this behavior can be changed by editing `/proc/irq/<IRQ#>/smp_affinity` i-node). The button to reset the average and maximum values and the text area showing the time elapsed since the last reset are also emphasized.

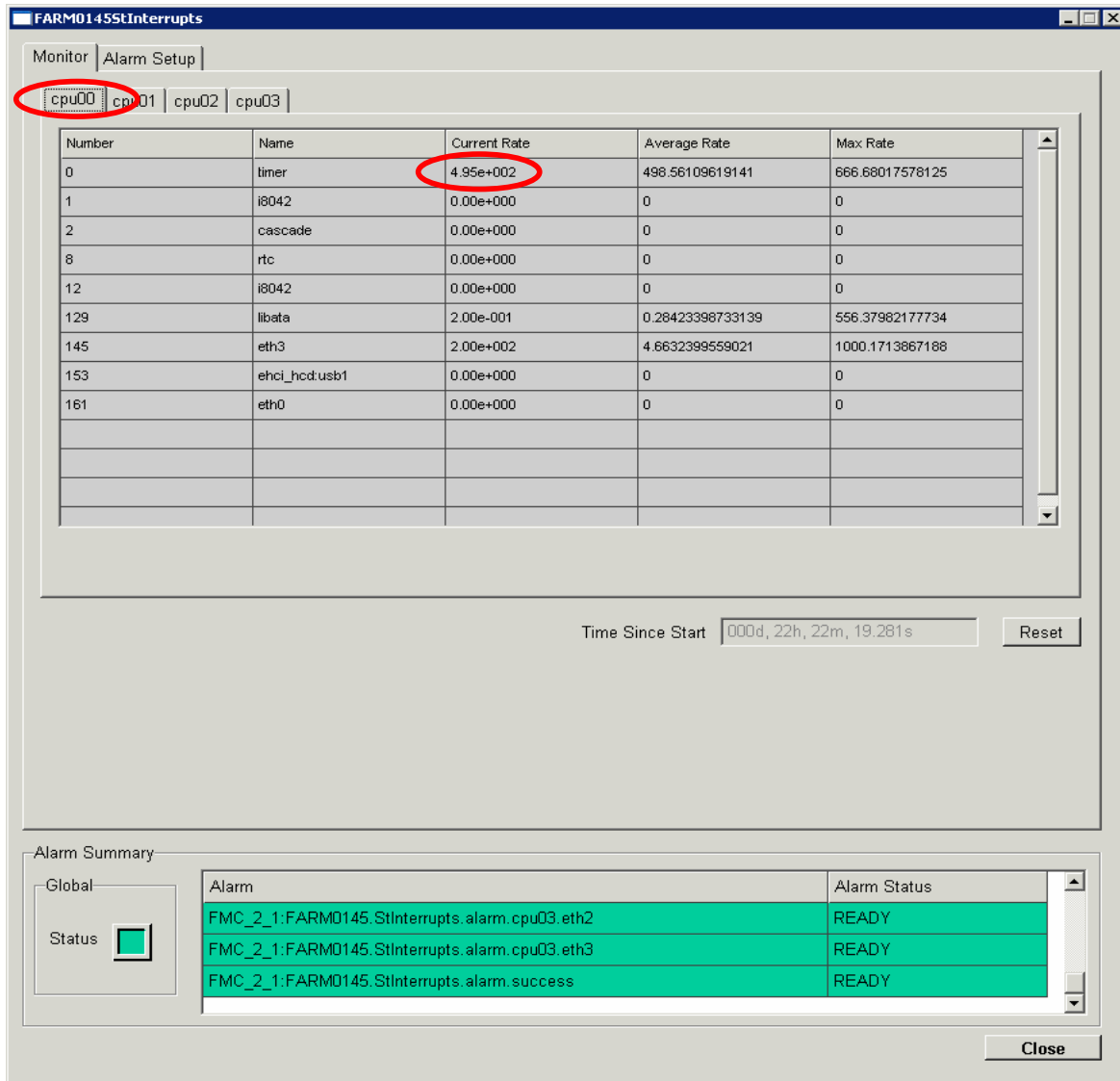


Figure 9. The Interrupts statistics PVSS client. These data refer to the CPU 0 (emphasized tabbed pane) of a PC running kernel 2.6. The timer interrupts (1000 s^{-1} in the kernel 2.6) are shared between CPUs 0 and 1 (compare this Figure with Figure 10). This behavior can be changed by editing `/proc/irq/<IRQ#>/smp_affinity` i-node.

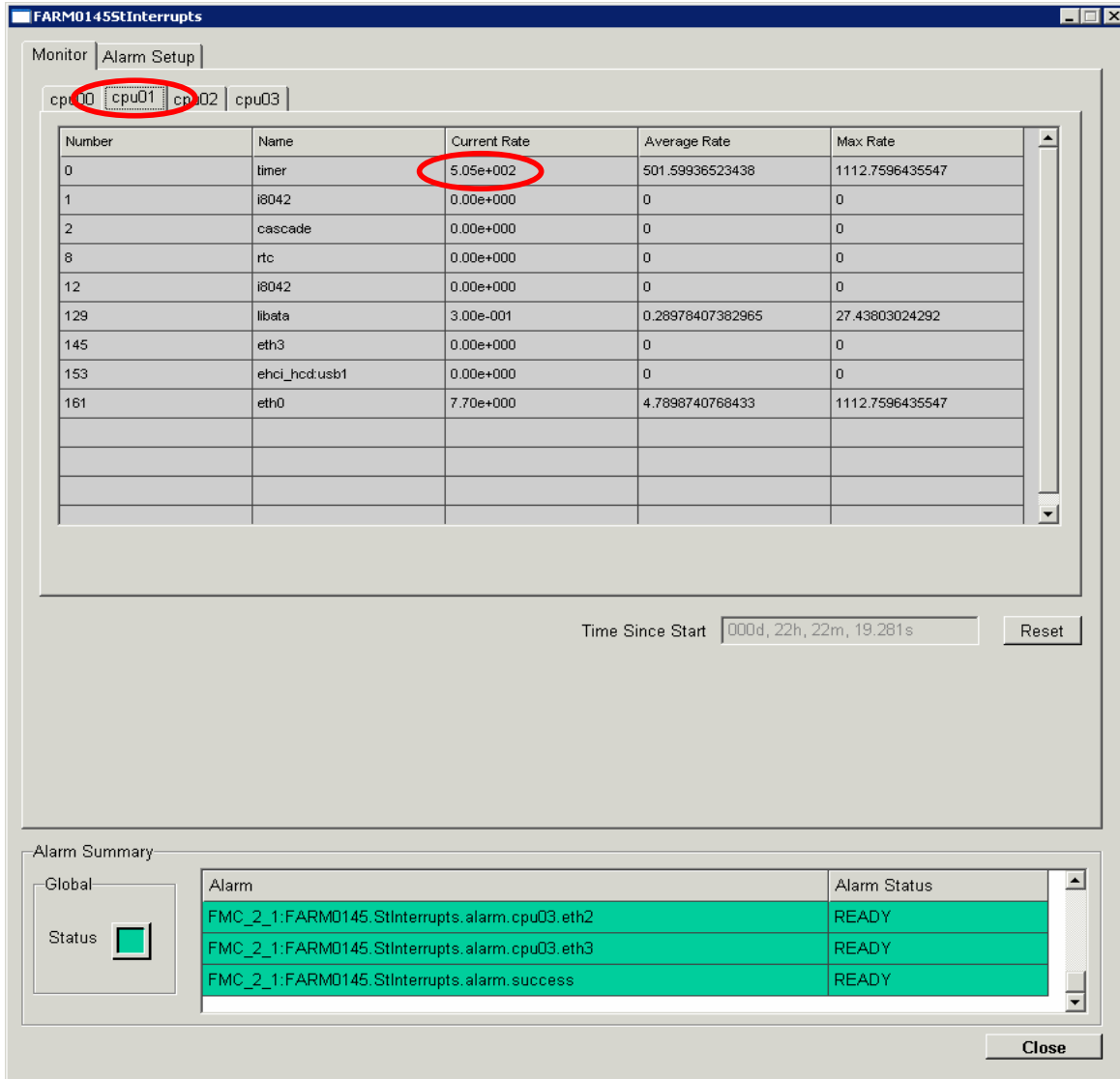


Figure 10. The Interrupts statistics PVSS client. These data refer to the CPU 1 (emphasized tabbed pane) of a PC running kernel 2.6. The timer interrupts (1000 s^{-1} in the kernel 2.6) are shared between CPUs 0 and 1 (compare this Figure with Figure 9). This behavior can be changed by editing `/proc/irq/<IRQ#>/smp_affinity` i-node.

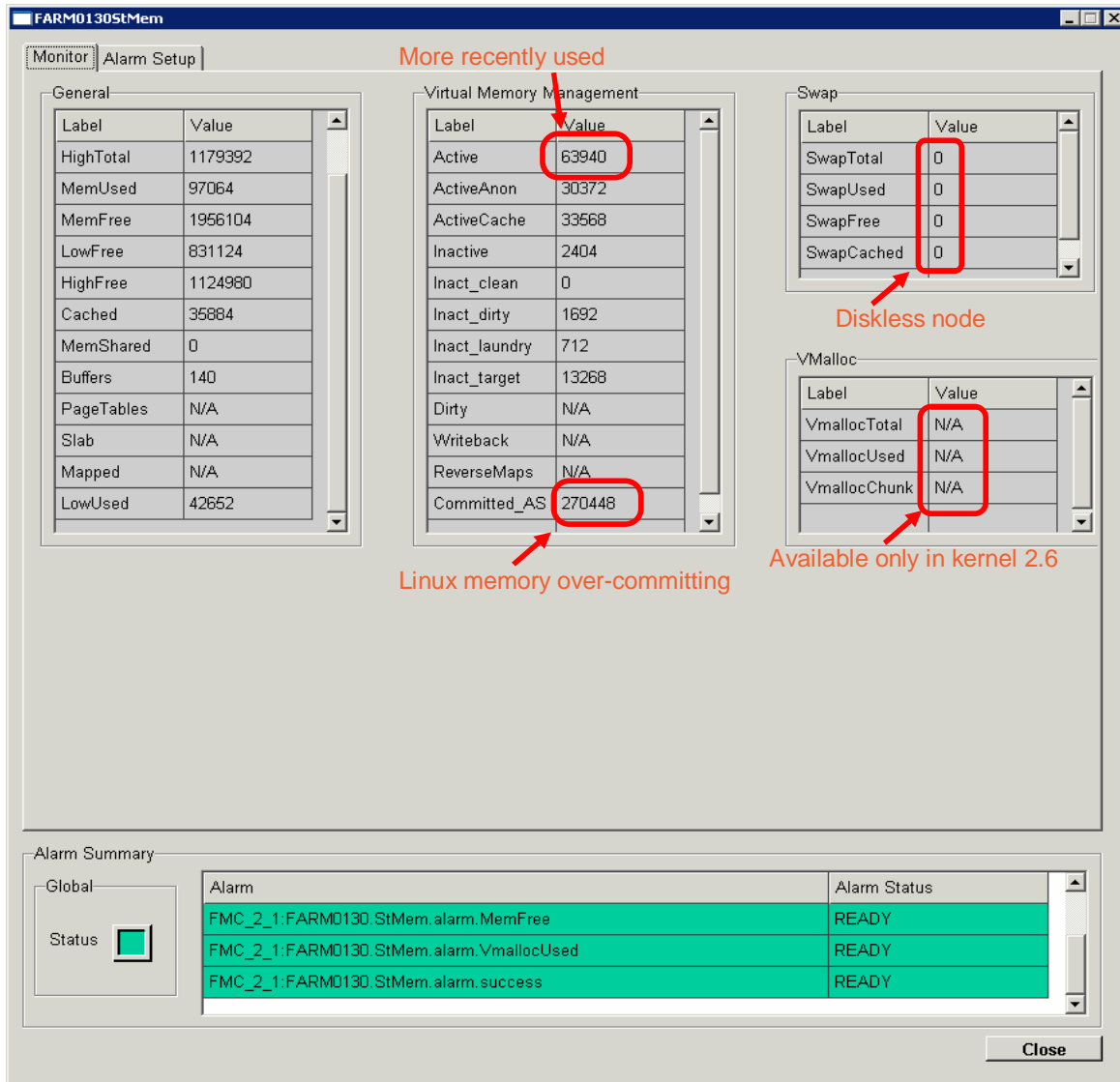


Figure 11. The Memory statistics PVSS client (the monitored PC is running a 2.4 kernel).

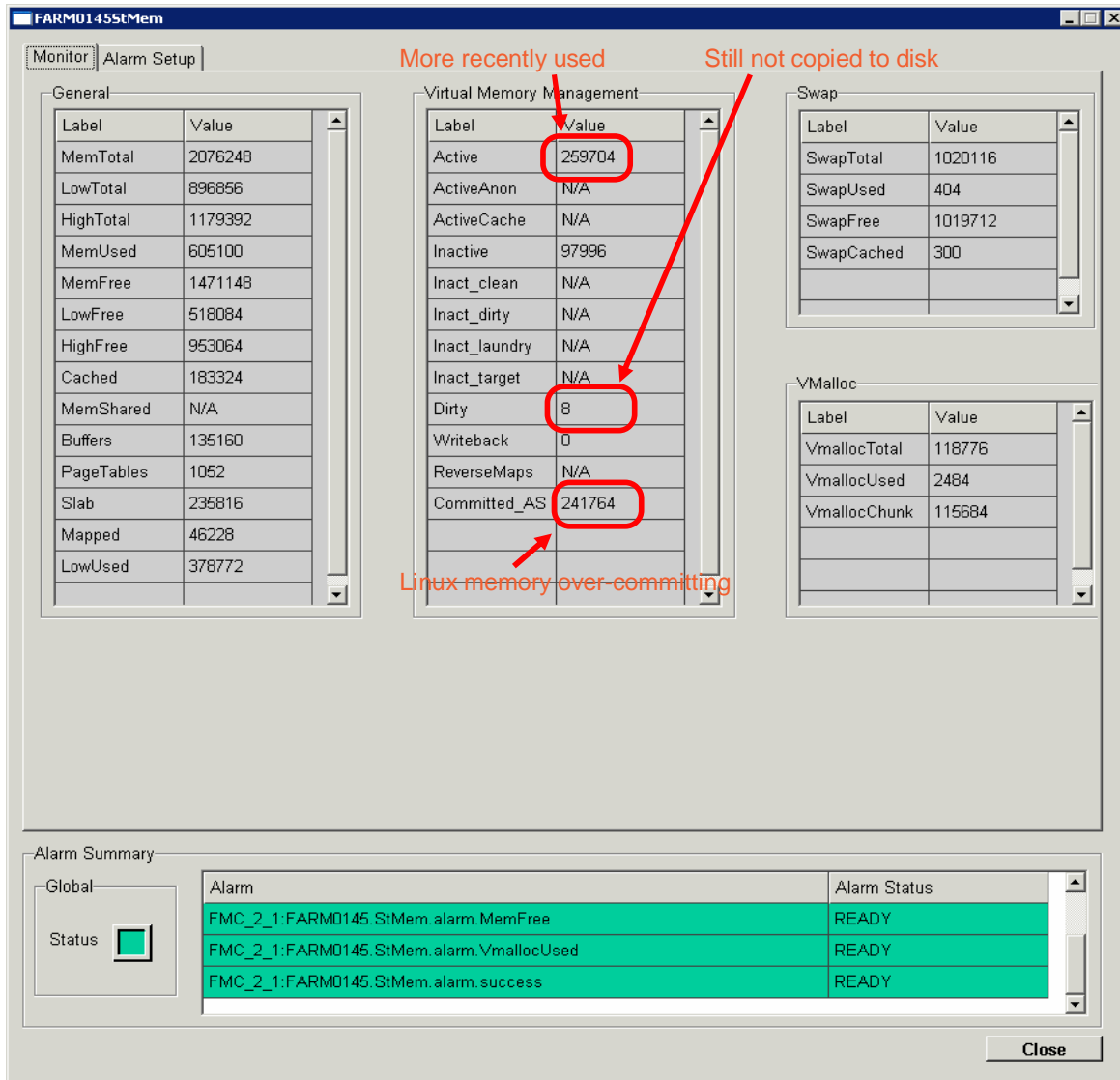


Figure 12. The Memory statistics PVSS client (the monitored PC is running a 2.6 kernel).

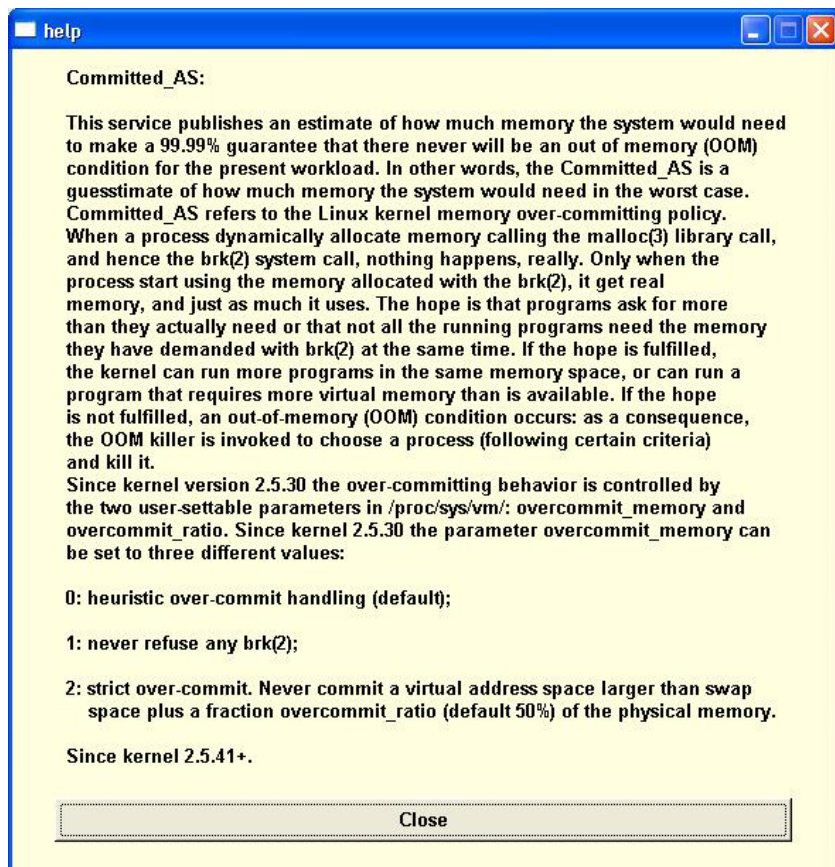


Figure 13. The Memory statistics PVSS help window, obtained by right-clicking the Committed_AS field in the memory statistics PVSS client, shown in Figure 11 or Figure 12.

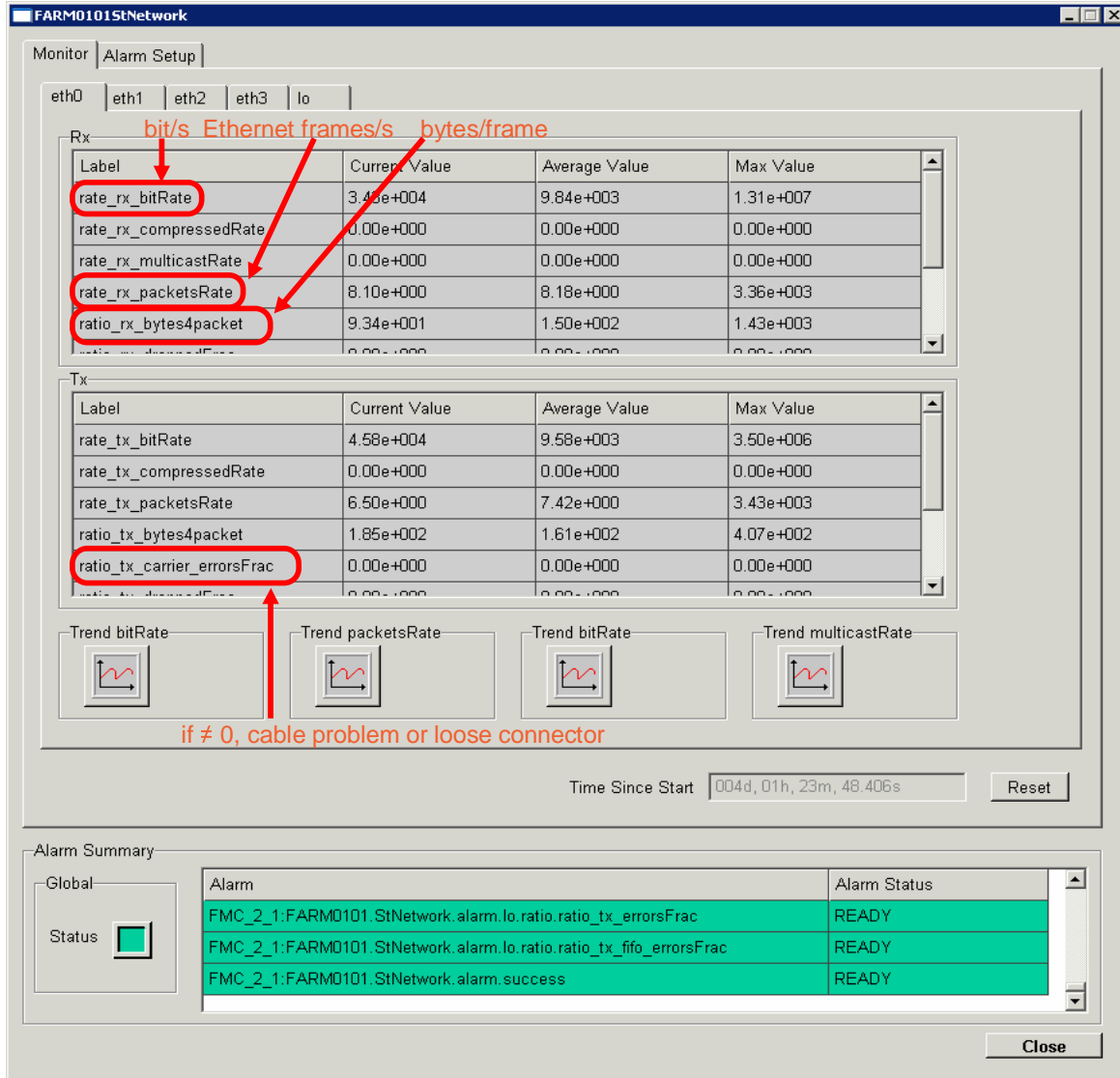


Figure 14. The Network Interfaces PVSS client.

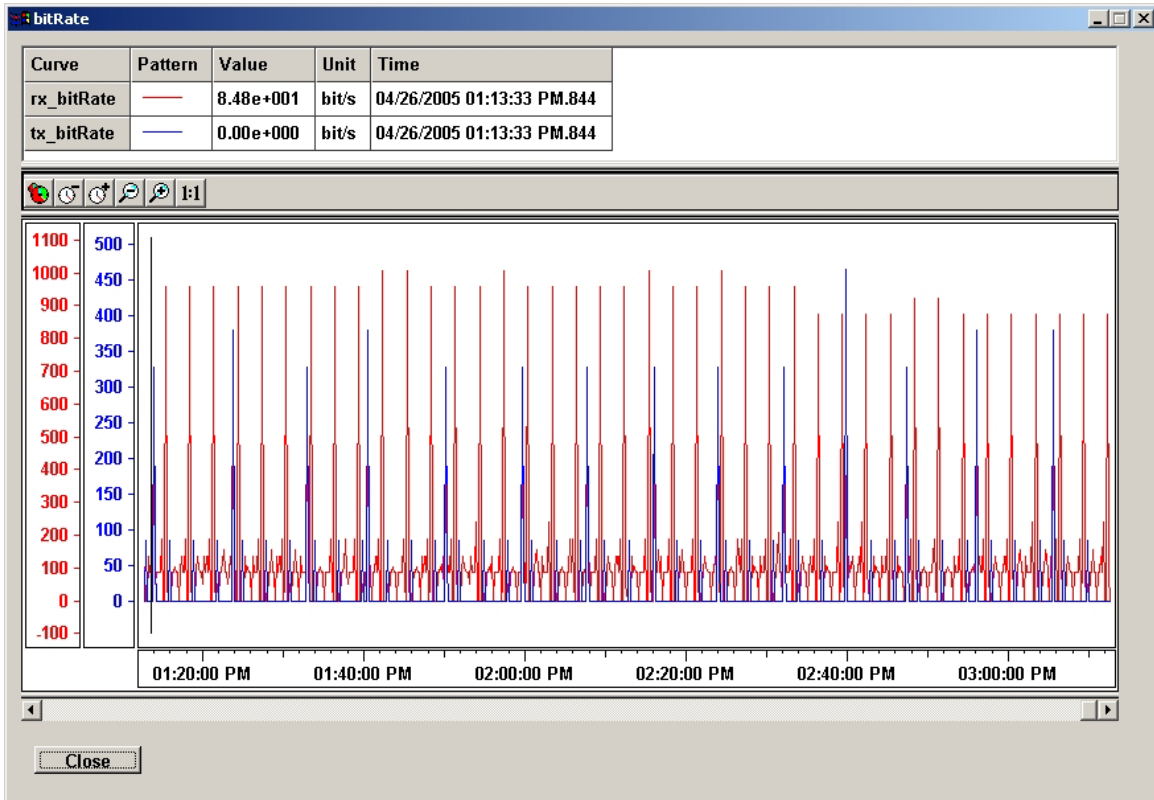


Figure 15. The Network Interface PVSS client. Plot of the time evolution of the I/O bit rate.

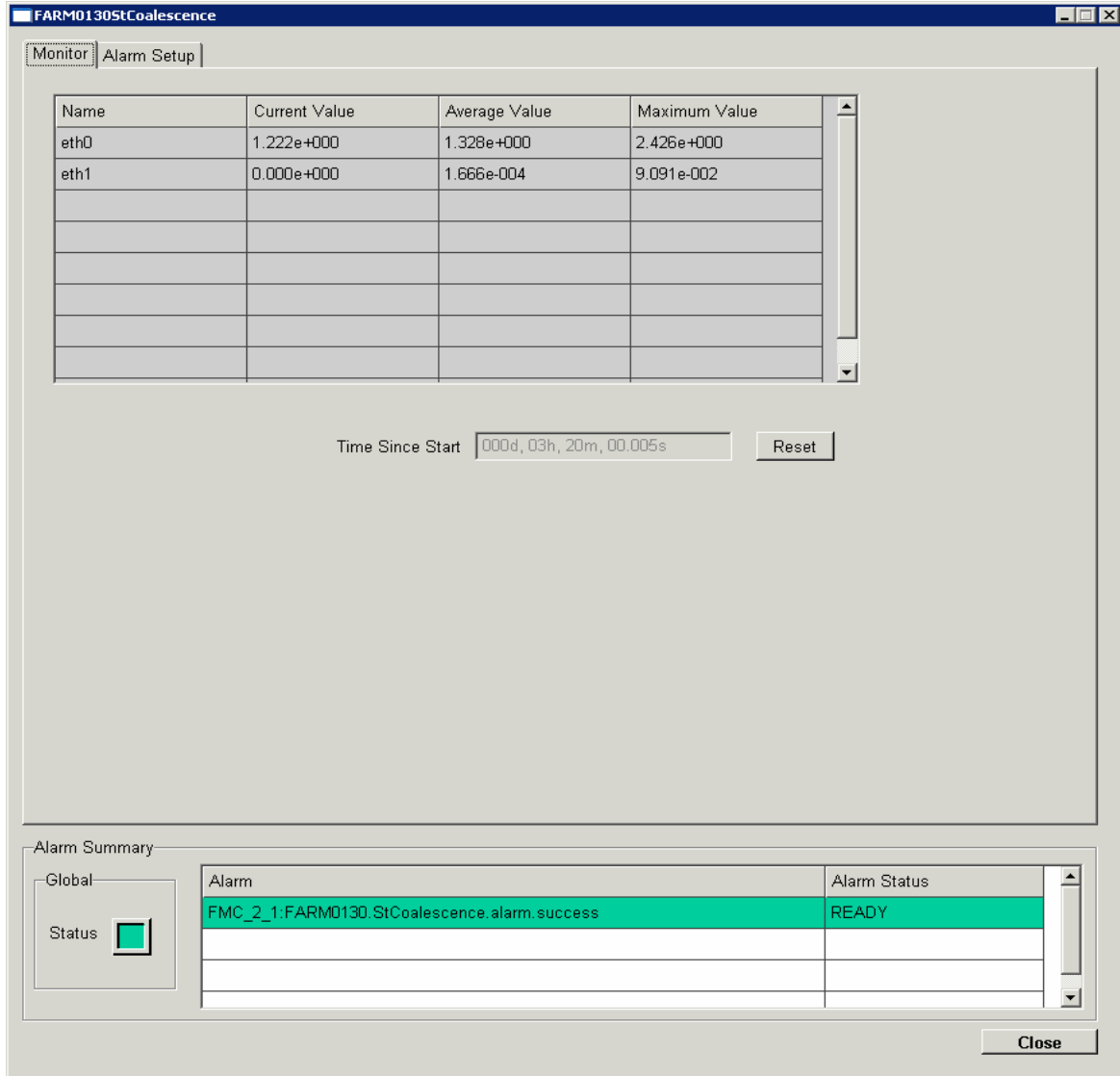


Figure 16. The Network Interface Interrupt Coalescence PVSS Client.

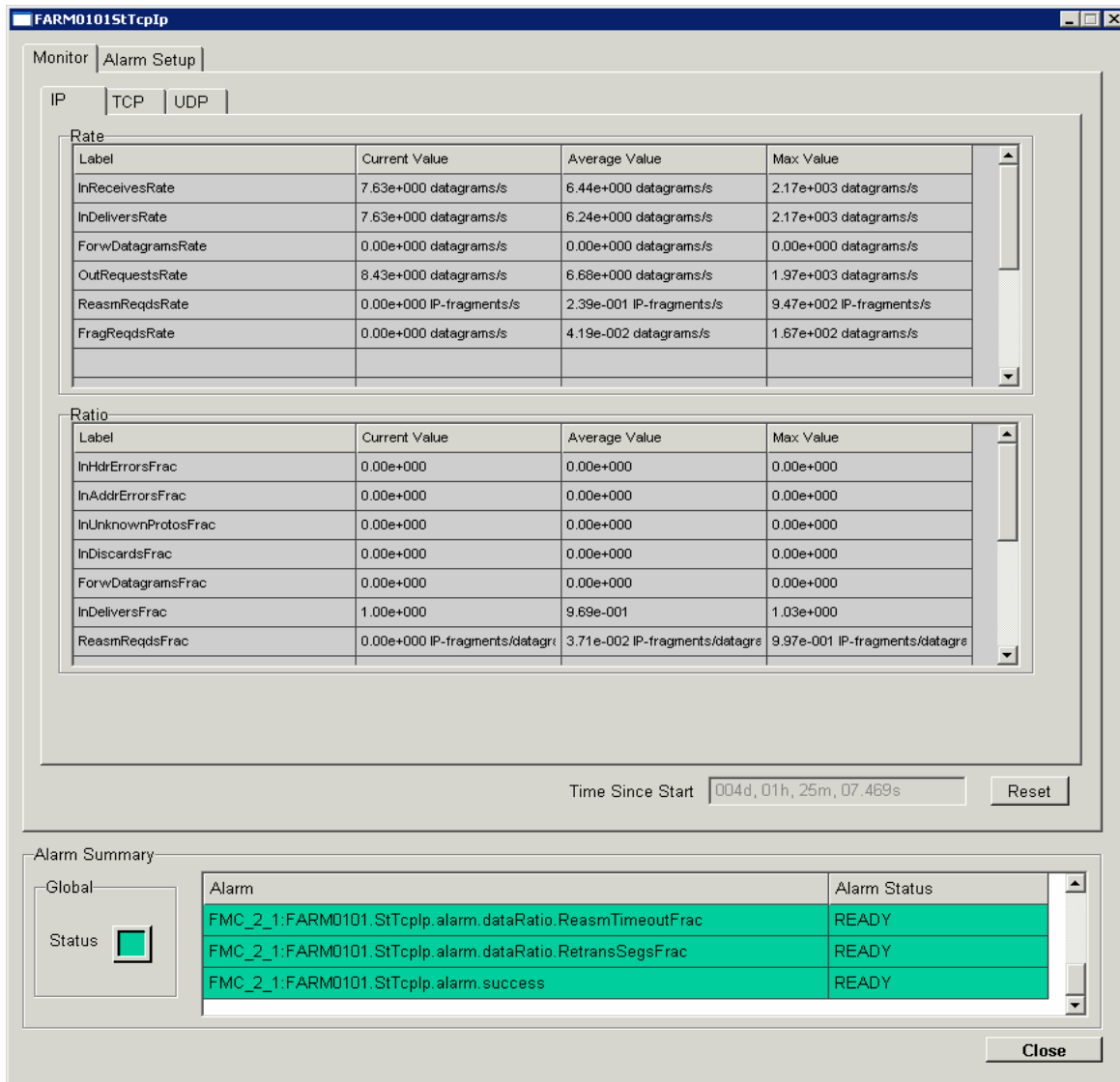


Figure 17. The TCP/IP stack PVSS client. IP tabbed pane.

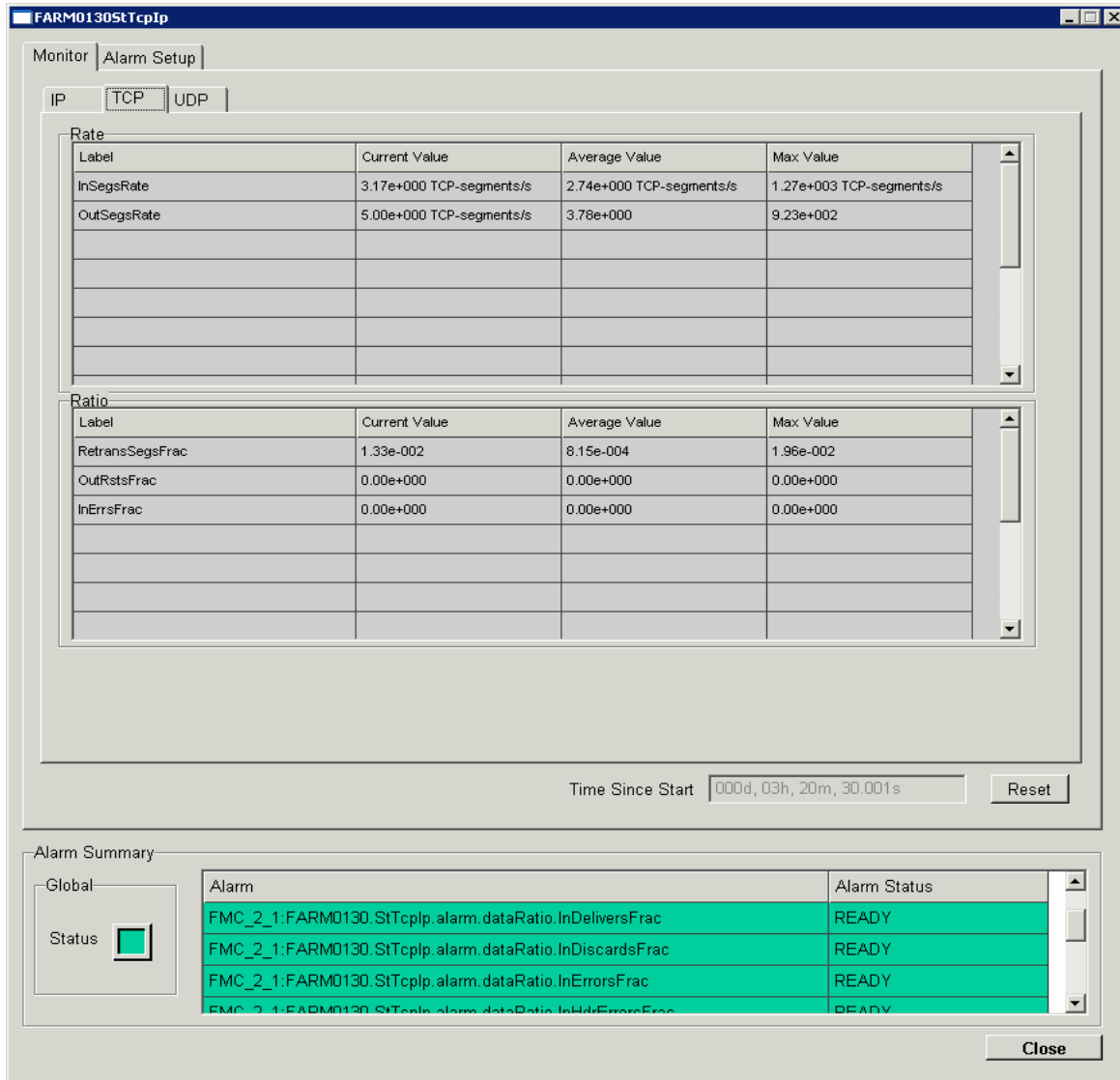


Figure 18. The TCP/IP stack PVSS client. TCP tabbed pane.

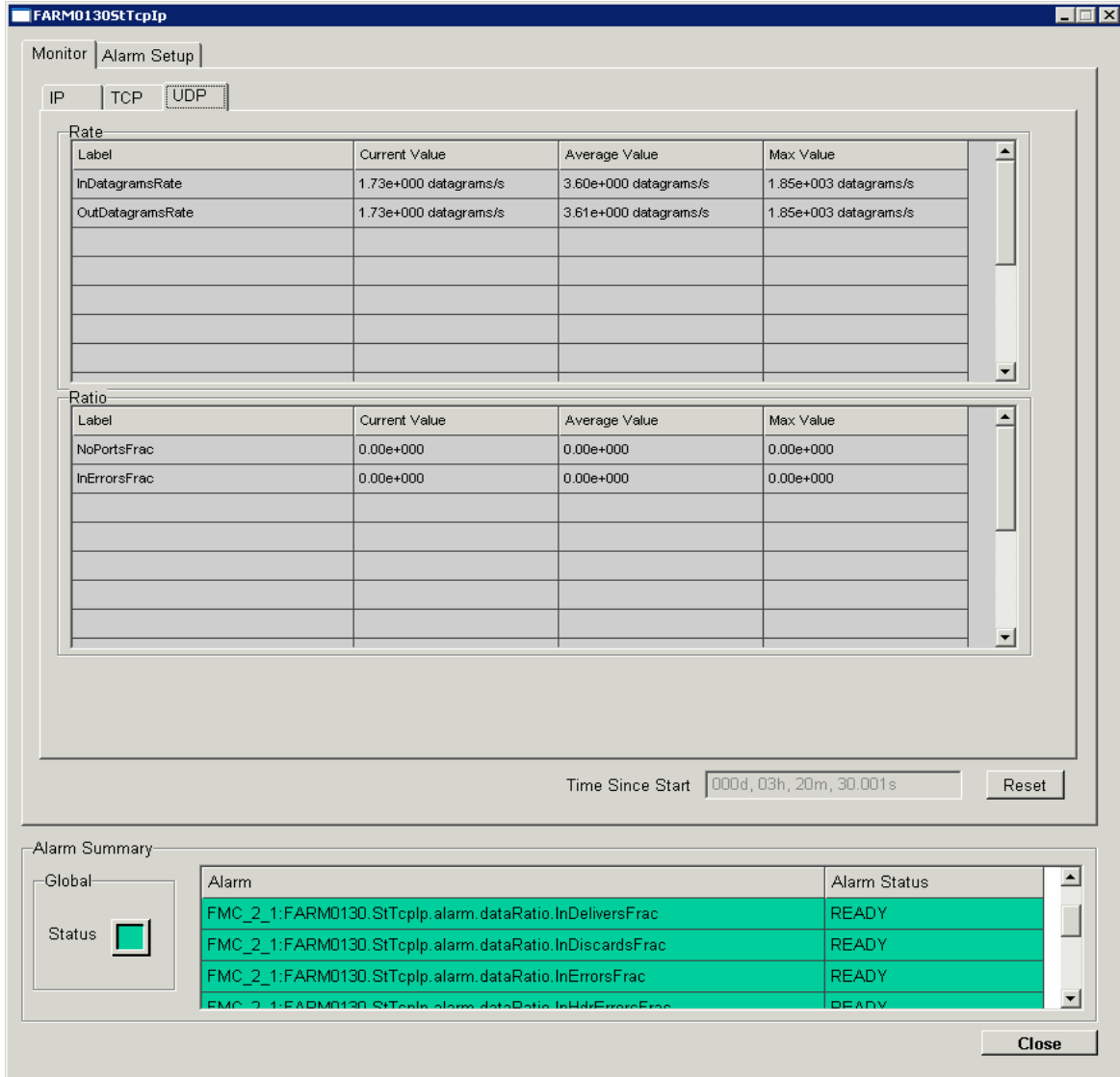


Figure 19. The TCP/IP stack PVSS client. UDP tabbed pane.

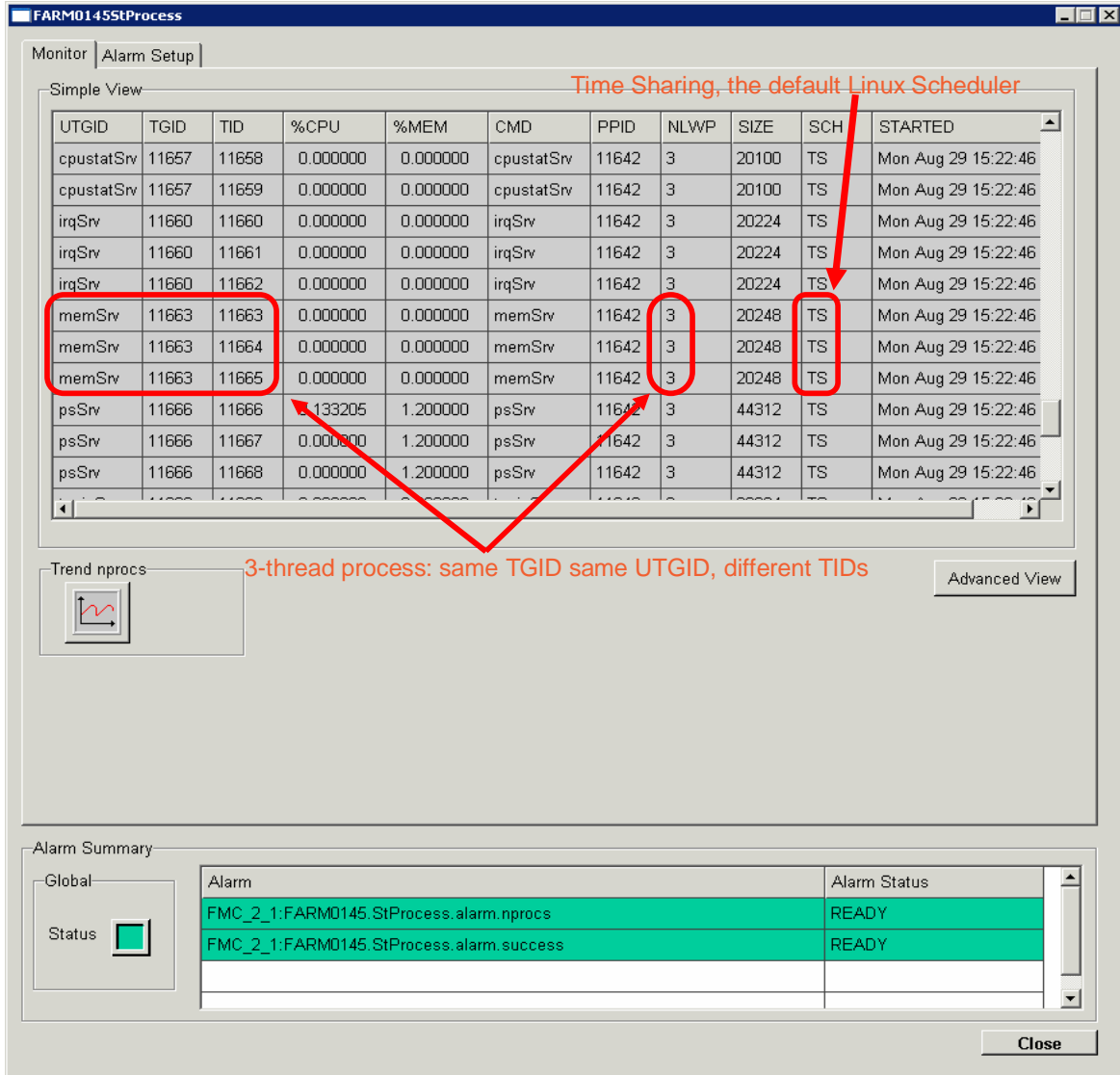


Figure 20. The processes statistics PVSS client. Simple view: panel which shows the main information. The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

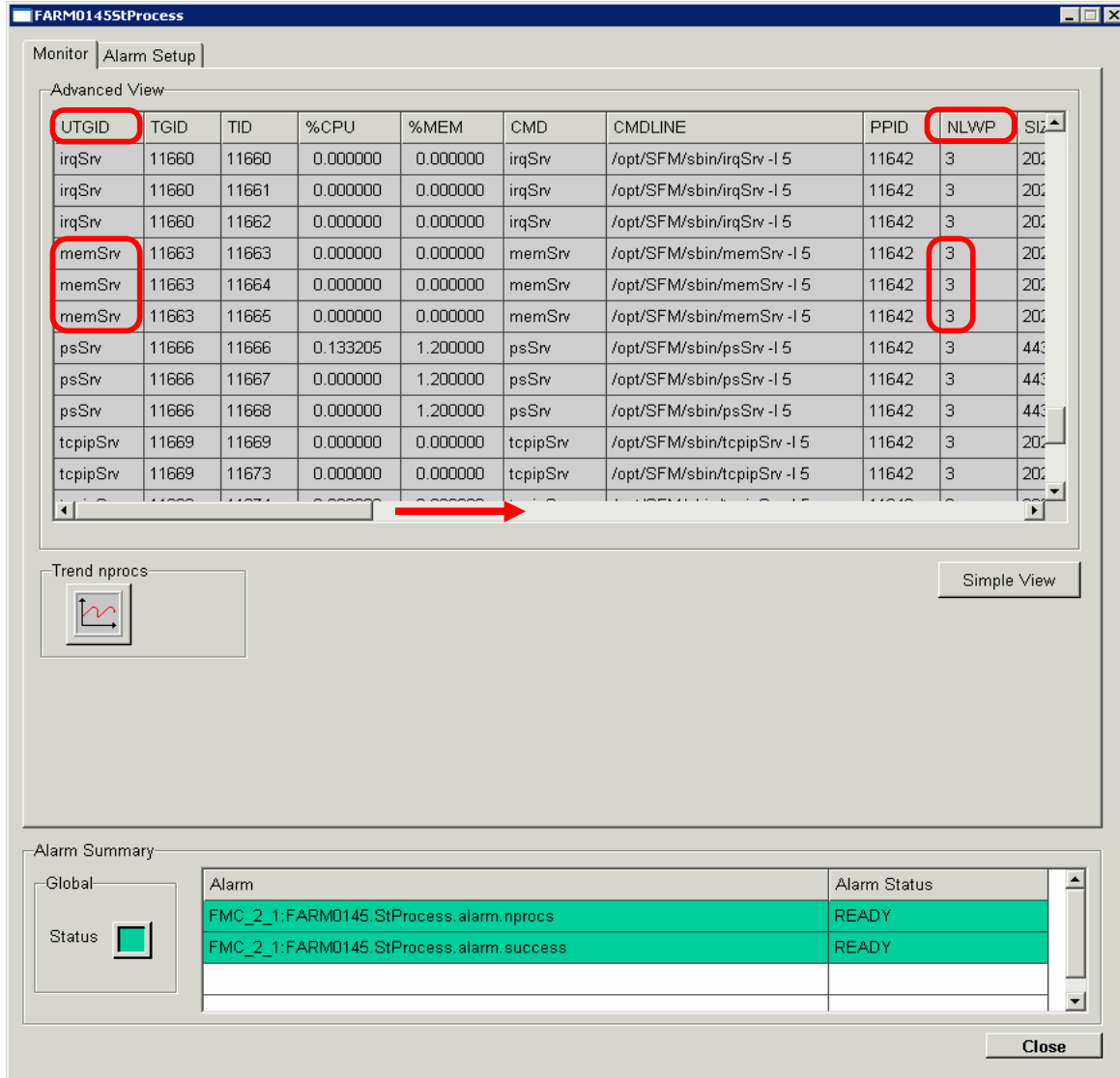


Figure 21. The processes statistics PVSS client. Advanced view: panel which shows extended information (1/5, information about UTGID, TGID, CPU and memory share, command name and command line, PPID and number of threads). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

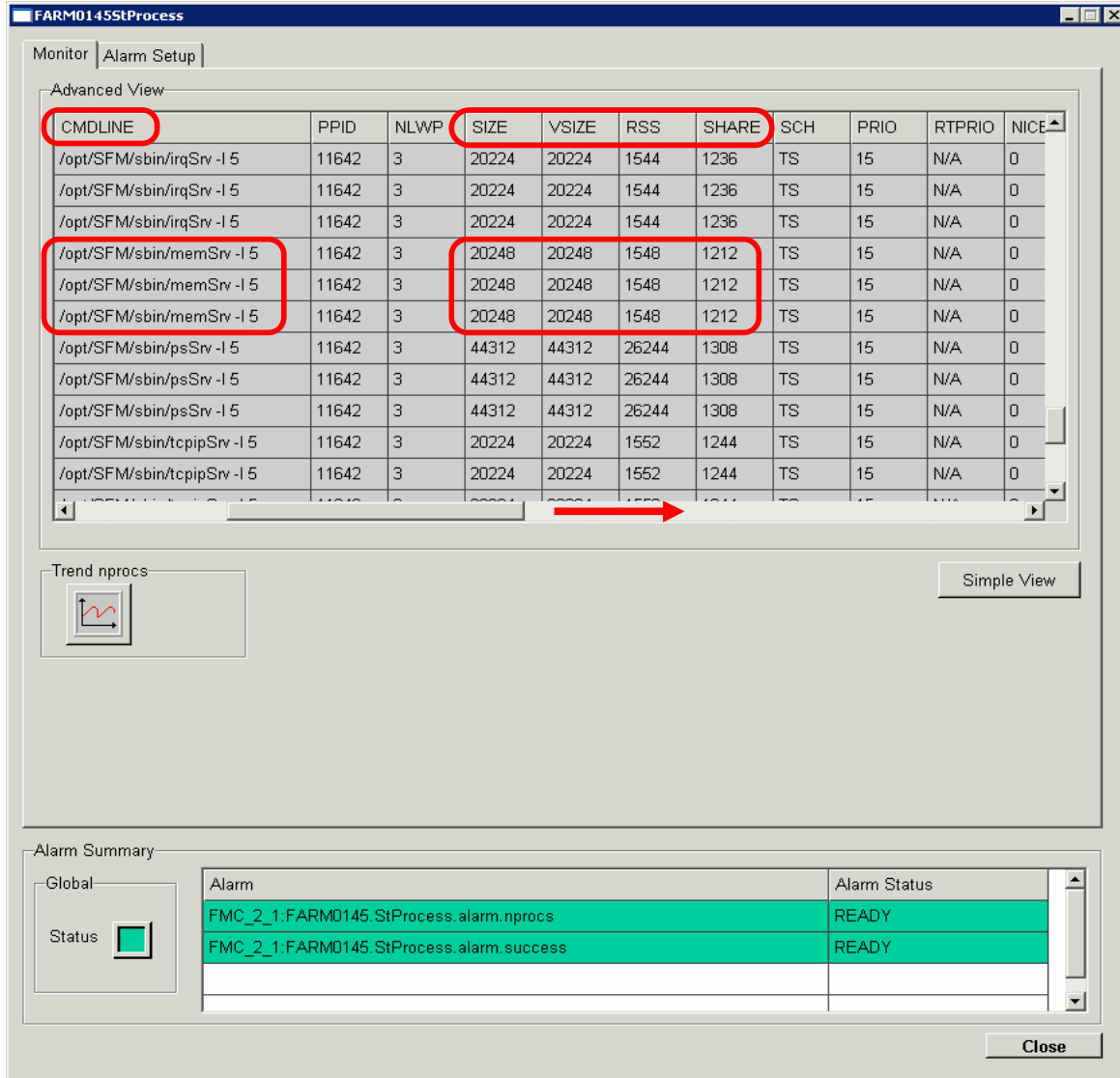


Figure 22. The processes statistics PVSS client. Advanced view: panel which shows extended information (2/5, size information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

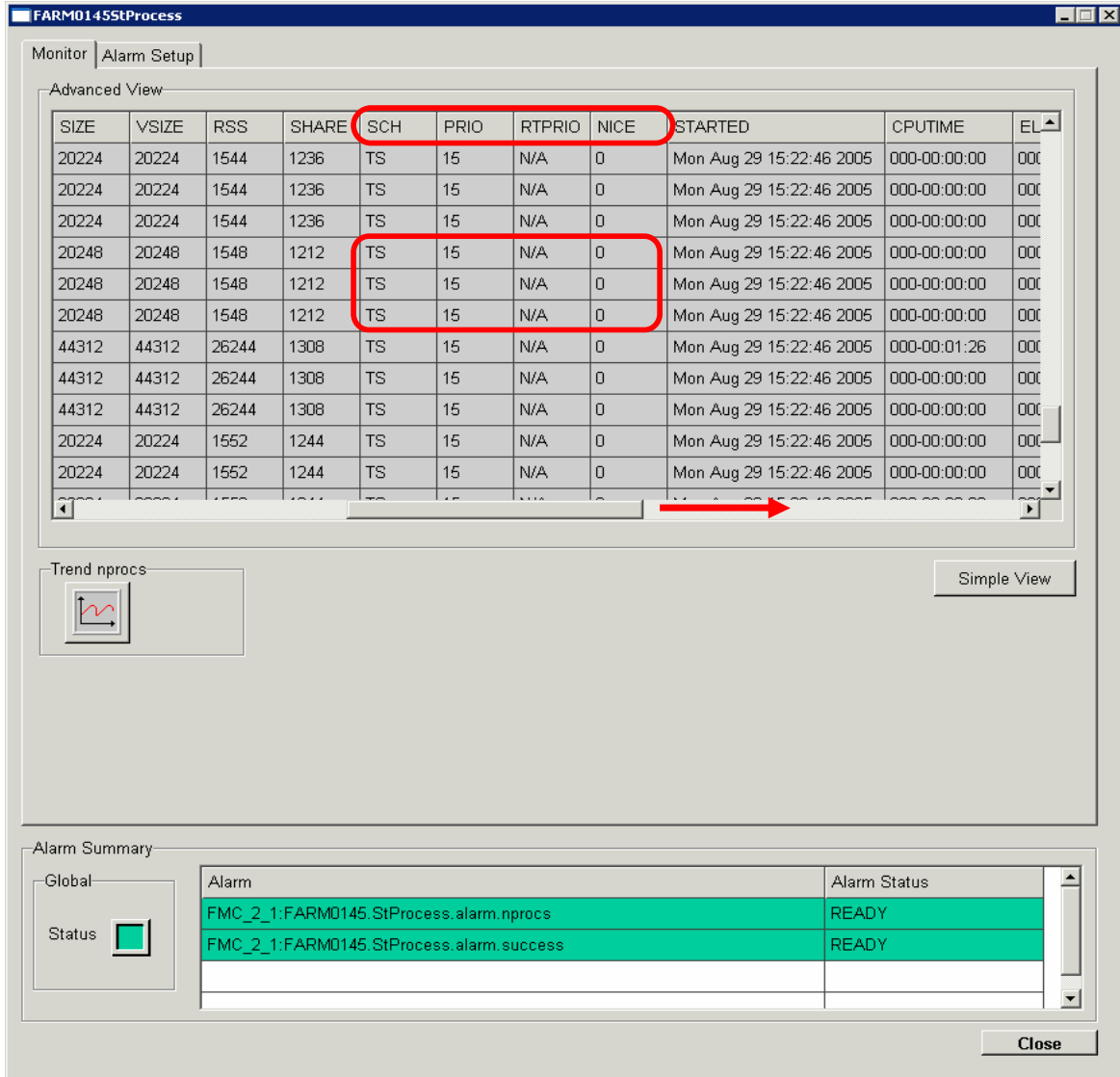


Figure 23 The processes statistics PVSS client. Advanced view: panel which shows extended information (3/5, scheduling information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

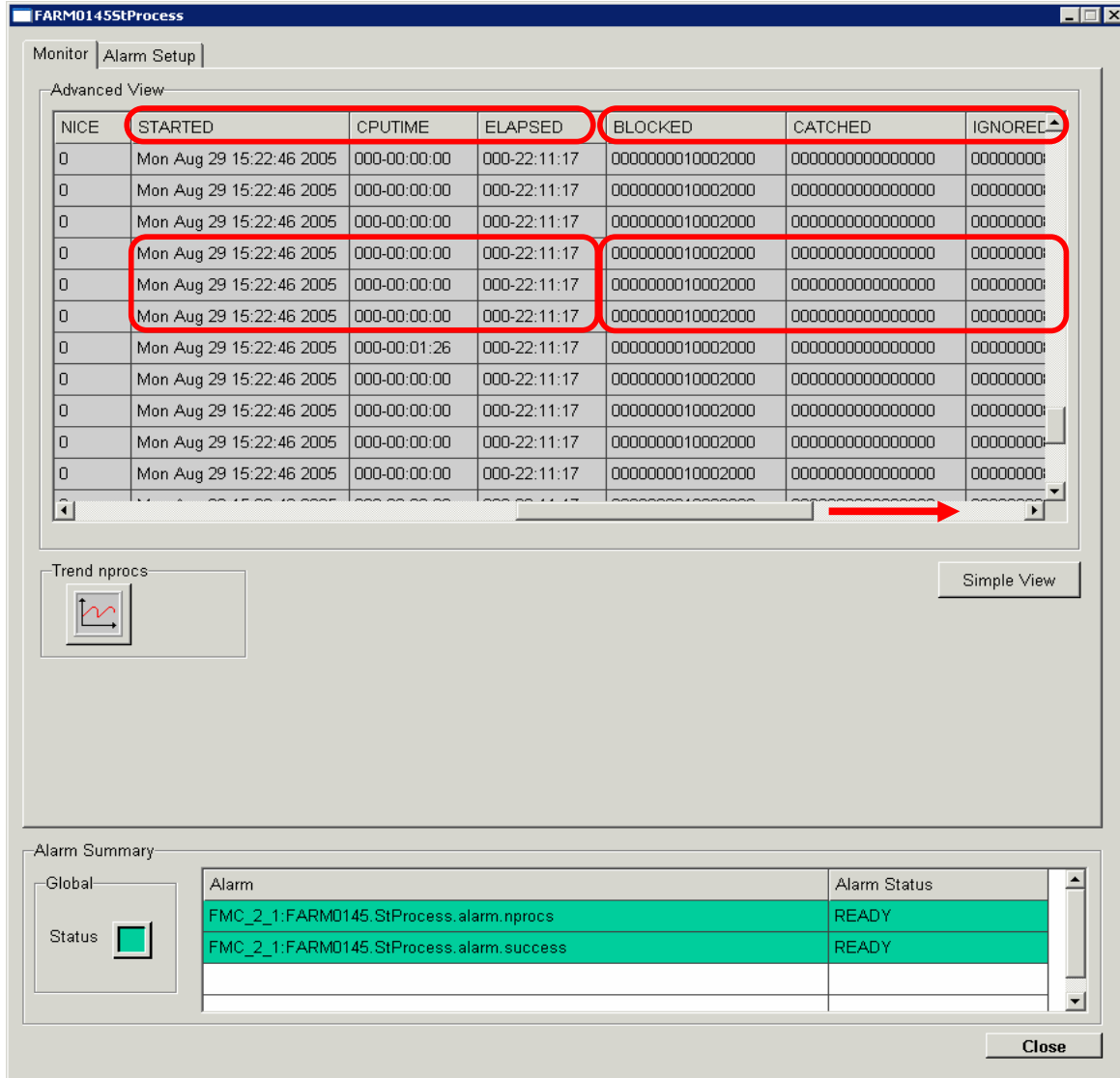


Figure 24. The processes statistics PVSS client. Advanced view: panel which shows extended information (4/5, time and signal information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

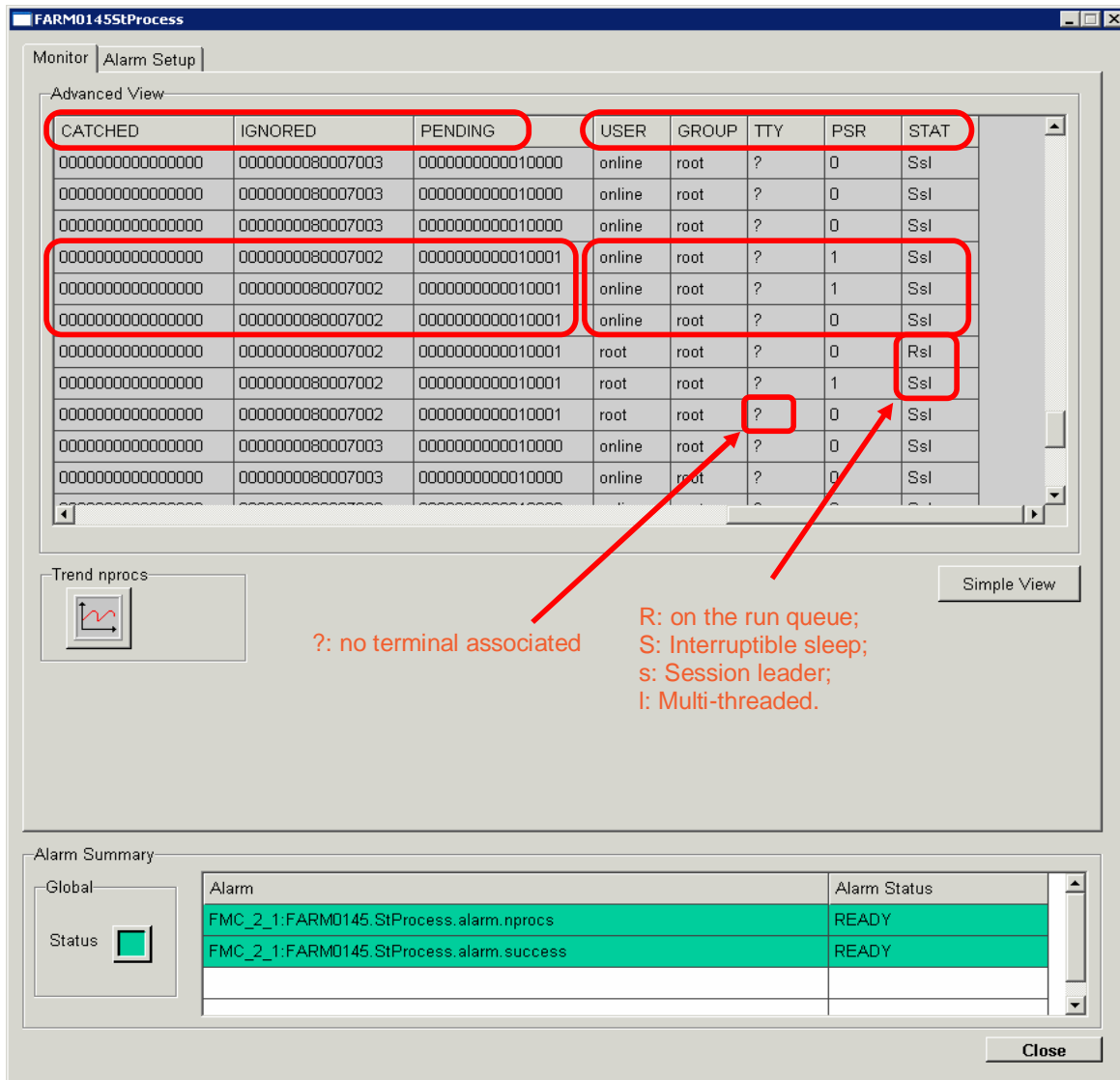


Figure 25. The processes statistics PVSS client. Advanced view: panel which shows extended information (5/5, signal, user, terminal and execution information). The monitored PC is running the kernel 2.6 (the threads information are therefore provided).

5. References

- [1] C. GASPAR, DIM, *Distributed Information Management System*: see URL <http://dim.web.cern.ch/dim/>.
- [2] C. GASPAR, *PVSS - DIM Integration*: see URL <http://clara.home.cern.ch/clara/fw/FwDim.html>.
- [3] *LHC Experiments Joint COntrol Project*: see URL <http://itco.web.cern.ch/itco/Projects-Services/JCOP/welcome.html>.
- [4] *PVSS Service*: see URL <http://itcobe.web.cern.ch/itcobe/Services/Pvss/>.
- [5] *Procps – The /proc file system utilities*: see URL: <http://procps.sourceforge.net/>.
- [6] CHRISTIAN LUDLOFF, *IA-32 architecture CPUID*: see URL: <http://www.sandpile.org/ia32/cpuid.htm>.
- [7] WIM VAN DORST, *BogoMips mini-Howto*: see URL: <http://www.clifton.nl/bogomips.html>.
- [8] INGO MOLNAR, *IRQ-affinity*, see URL: <http://cvs.sourceforge.net/viewcvs.py/linuxsh/kernel/Documentation/IRQ-affinity.txt>.
- [9] see: `/usr/src/linux/ Documentation/vm/overcommit-accounting`
- [10] MEL GORMAN, *Understanding the Linux Virtual Memory Manager*, Prentice Hall.
- [11] MEL GORMAN, *Code Commentary On The Linux Virtual Memory Manager*, see URL: <http://www.csn.ul.ie/~mel/projects/vm/guide/html/code/code-html.html>.
- [12] ANDRIES BROUWER, *The Linux kernel*, see URL: <http://www.win.tue.nl/~aeb/linux/lk/lk.html#toc9>.
- [13] A. BARCZYK, A. CARBONE, J-P. DUFÉY, D. GALLI, B. JOST, U. MARCONI, N. NEUFELD, G. PECO, V. VAGNONI, *Reliability of datagram transmission on Gigabit Ethernet at full link load*, LHCb Technical Note 2004-030 DAQ.
- [14] A. BARCZYK, D. BORTOLOTTI, A. CARBONE, J. P. DUFÉY, D. GALLI, B. GAIDIOZ, D. GREGORI, B. JOST, U. MARCONI, N. NEUFELD, G. PECO AND V. VAGNONI, *High Rate Packets Transmission on Ethernet LAN Using Commodity Hardware*, proceedings of the 14th IEEE-NPSS Real Time Conference 2005, Stockholm, june 9, 2005.

- [15] F. BONIFAZI, D. BORTOLOTTI, A. CARBONE, D. GALLI, D. GREGORI, U. MARCONI, G. PECO, V. VAGNONI, *The Task Manager for the LHCb On-Line Farm*, LHCb Technical note 2004-099 DAQ.