ProCtool User's Guide

Walter Nielsen
Morgan Herrington

1.0 Introduction to ProCtool

ProCtool is a tool for process monitoring and management. It is targeted for system administration, performance analysis, as well as general users and hackers. ProCtool encompasses the functionality provided by ps(1), renice(1), and priocntl(1). It also provides much of what dispadmin(1M) and sar(1M) do in Solaris 2.x.

The key difference between ProCtool and the above mentioned tools is that ProCtool provides an Motif graphical user interface. Also unlike ps, ProCtool will periodically sample the processes and the kernel at user specified intervals. It also provides a way of graphing the data and in general much much more functionality.

The basic idea of ProCtool was inspired by top, a freeware Unix tool available through most ftp archives that dates back to sometime during the height of the Revolutionary War.

1.1 Support, Distribution Rights, etc.

ProCtool is available in binary form for free. It is not an official Sun Microsystems Inc product though legally it is the property of Sun Microsystems Inc. There is no implied support or endorsement from Sun Microsystems. However, the authors plan to support the product on their own time. We encourage bug reports, request for enhancements, and/or comments. Of course, cash contributions are gladly accepted.

To reach the authors, e-mail to:

morgan.herrington@west.sun.com Morgan Herrington

walter.nielsen@eng.sun.com Walter Nielsen

1.2 System Requirements

Version 2.x of ProCtool requires Solaris 2.x FCS on SPARC systems only. As a rule, ProCtool is compatible only with a particular minor version of Solaris. This is because it uses unadvertised interfaces in the kernel which may change between Solaris releases. To make this relationship more explicit, the ProCtool version numbering scheme matches Solaris's i.e. ProCtool v2.4 is for Solaris 2.4. As of version 2.3.5, Proctool is Motif based, it will look for the Motif library (libXm.so.2) in /opt/SUNWmotif/lib and /usr/dt/lib.

1.3 Feature Highlights

Here are some of the features of ProCtool:

- Continually update the process and system data at a user-specified sampling interval.
- View all active processes and various parameters for each process on a system.
- View the state of various system parameters and each CPU.
- Define monitors. Monitors are agents that act on user specified criteria (see below).

- Choose which per process parameters to show in the process pane.

- Sort processes in the process pane based on various parameters.

- Change the color or hide processes based on user specified criteria (using monitors).

- Turn off or on CPUs on a MP box or bind a process to a processor.

- Selectively kill(1) or renice(1) processes.

- Send signals to a set of processes.

- Get a stack backtrace for any process.

- Graph system or process characteristics in line or bar graph mode.

- Show the list of open file descriptors for a process.

- Change the time scheduling priority for one or more processes.

- Show system wide real-time and time-sharing dispatch tables.

- Show detailed information on a process such as VM usage, I/O usage, paging rate, and memory map.

- MOTIF compatible on-line help is provided for all windows and many gadgets.

- Privileged (super-user) and non-privileged modes of operation.

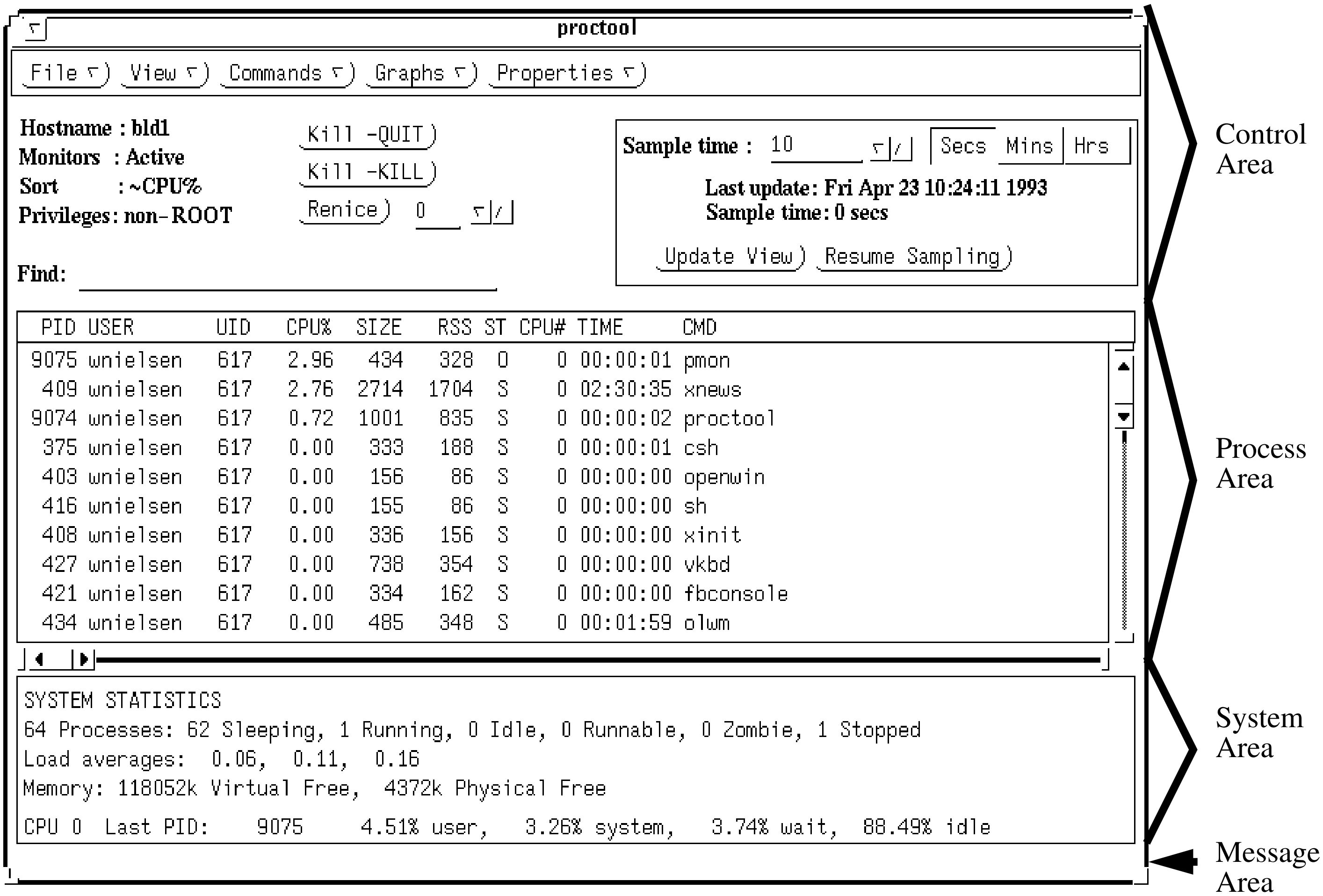
1.4 Differences from versions prior to 2.3.5

- Support for Solaris 2.3 only.
- Motif based GUI instead of OpenLook/OLIT.
- Unbind a process to a processor
- -p<number> command line option. Use this option to specify the maximum number of processes you want to display. The default limit is 300.
- -v command line option. Proctool checks which version of the Solaris OS you are running on by using the uname() call. Use the -v option to disable this check.
- Traceback for a particular LWP, if the process uses more than one LWP.
- System I/O activity window and graph.
- Support for up to 64 processors and 256 I/O devices.

.

And of course, lots of bug fixes.

FIGURE 1. ProCtool Main Window



2.0 Basic Features

ProCtool's main window is split into three main panes, the Control Area, the Process Area, and the System Area (see figure 1). The Control Area on the top contains control gadgets for Proctool. The Process Area is a list of processes similar to a ps(1) output. Processes in this window are selectable with the mouse button. The bottom pane is the System Area. This contains the system-wide information along with a per-CPU information line. The footer area of the main window is used as the message area. Most error messages are printed there.

There are a number of popup command and property windows that are also useful and interesting. The graph window is the most visually entertaining. There is a process property window that lets you examine a process in detail and a system property window that lets you delve more deeply into the system as a whole.

2.1 Monitors

A powerful mechanism provided by ProCtool is the concept of “monitors”. A monitor consists of a logical expression and a set of actions. The logical expression is composed of relational expressions where the operands are various per process parameters such as UID, Resident Set size, etc. and values. Monitors are evaluated every time an update occurs at the end of a sampling period or when a new monitor is inserted or redefined.

A typical monitor expression is as follows:

(USER == "walter") and (CMD == "csh")

which evaluates to all C-shell processes owned by the user walter.

Actions are invoked when a monitor’s logical expression evaluates to true. Actions may consist of reminders such as beeps, mail and popups. Actions may also change the visual representation of the process in proctool, for instance displaying a process in another color such as red or yellow.

Some typical use of a monitor by a system's administrator:

- Renice a user process that is paging heavily or taking up too much of the CPU.
- Popup a window when someone other than myself or root has a process running on my workstation.
- Flag all process, in RED, that have Resident Set Size greater than 1MB.
- Hide uninteresting processes in the process window such as nfsds, init.

An extremely territorial and paranoid person can set up a monitor so that if (UID!= <their UID> AND UID!= 0), then execute a command: kill -9 $PID. This will make sure no one does anything on their system, and probably win a couple of popularity contests as well.

Monitors are defined through the COMMANDS -> MONITOR menu. All monitors must have a name associated with them and must be entered into the monitor list. Monitors may be active or inactive. An inactive monitor is ignored until activated. Normally activated monitors remain active until user intervention. Monitors may also be defined so that their actions are executed exactly once. These monitors deactivate themselves once their actions have been triggered.

Monitors are explained in greater detail in Section Figure 9.0, “Monitor Window,” on page 14. The syntax of a monitor expression is shown in Appendix A.

2.2 Privileges

ProCtool can be used in both privileged or unprivileged mode. Privileged mode is granted to users who are root, i.e. UID == 0. A user can also have privileged mode by entering the root password through the COMMANDS -> SET UID menu.

Without root privileges, the user is restricted so they are unable to kill, renice, change the priority, or bring up a process property popup window on a process not owned by them.

3.0 Control Area Menus

The following is a description of the menus and menu items at the top of the main ProC-tool window.

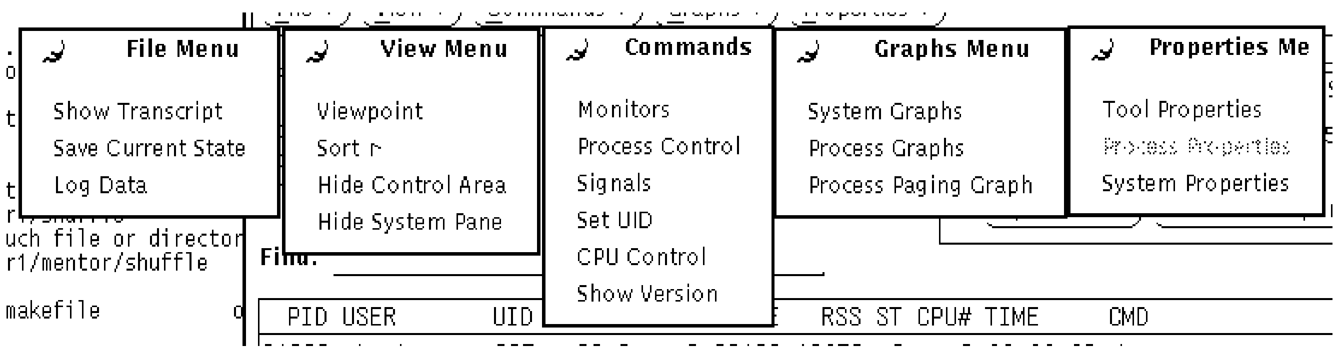


FIGURE 2. Control Area Menus

3.1 File Menu

Show Transcript - Brings up a window which holds a transcript of activities in the current proctool session. All commands that affect the state of a process, such as kill are transcripted. Monitors that are activated are also transcripted. The transcript is saved in a file.

Save Current State- Saves the current tool state into an initialization file. The current viewpoint, sort parameters, current monitors and sampling rate are saved. The initialization file is called ".proc.init". Proctool first searches the user's home directory, followed by the value of the environment variable PROCDIR for the initialization file.

Log Data - Enables one to create an ASCII log file that is updated at every sample period.

3.2 View Menu

Viewpoint - Brings up a command window (see Section 7.0 on page 12) used to select which variables to show for the processes displayed in the Process Area.

Sort... - Allows one to sort the processes shown on the main process window using a selected list of variables as keys.

Show Control Area/Hide Control Area - Toggle whether or not the control area is visible.

Show System Panel/Hide System Panel - Toggle whether the system pane is visible.

3.3 Commands Menu

Monitors - Brings up a window to define, activate, inactivate, or edit a monitor.

Process Control - Brings up a command window (Section 11.0 on page 19) to change the priorities and scheduling parameters of all time-sharing and or real-time processes. On a MP box, allows one to bind a process to a particular CPU.

Send Signal - Command window to send a signal to selected processes.

CPU Control - **(Available only on a MP box)**. Gives the user the ability to selectively turn off and on a set of CPUs.

Set UID - Brings up a popup window which allows the user to enter the root passwd. This allows the user to run ProCtool under privileged mode. See Section 12.0 on page 20.

Show Version - Shows the current version of the tool in the Message Area.

3.4 Graph Menu

See Section FIGURE 9. on page 21 for more detail on graphs.

System Graphs - Brings up a graph window showing system-wide data.

Process Graphs - Brings up a graph window which graphs variables for a set of processes.

Process Paging Graph - Brings up a graph window which graphs detail paging information for one process only.

3.5 Properties Menu

Tool Properties - Set properties applicable to all of ProCtool.

Process Properties - Brings up a property window showing a selected process's parameters. Typically a user will just double-click on a process in the Process Area instead of using this menu-item.

System Properties Brings up a detailed property window on system wide parameters. Typically, a user will just double-click anywhere in the System Area instead of using this menu-item.

4.0 Control Area

The following is description of the gadgets available in ProCtool control area, the portion between the menu area and the process pane.

Kill -Quit Button - Send a kill -QUIT command to the selected processes. A user not running in privileged mode is not allowed to kill a process that has a different effective UID.

Kill -Kill Button Send a kill -KILL command to the selected processes. A user not running in privileged mode is not allowed to kill a process that has a different effective UID.

Renice Button - Renice the selected processes by the increment specified in the text field on the right. A user not running in privileged mode is not allowed to change the priority of a process that has a different effective UID.

Sample Time Field - Set the sampling time to the interval specified. After entering a new value, type the <RETURN> key for it to take effect. The value entered must be an valid base-10 integer value.

Note: We don't recommend sampling times of less than 2 or 3 seconds. Smaller sampling times will cause ProCtool to skew the data since ProCtool itself will consume a lot of system resources. Certain operations invoked by the Process Property Window (see Section 16.0 on page 25) require a sizeable amount of CPU resources so the sampling should be set accordingly.

Update View Button. Get an update of the process and system data. This does not affect the sampling period. This will also update any visible graphs.

Resume Sampling/Suspend Sampling Button - Turn off and on sampling.

Find Text Field - WIll find and select a process in the Process Area that matches what is in this field. The user can enter either a string, in which case it will try to match by CMD. They can also enter an integer value, in which case it will try to match by PID. In the case of a string, hitting the <RETURN> key repeatedly will cycle through all processes with the same CMD value.

5.0 Process Area

The Process Area show the list of current processes running on the system. The fields displayed for each process depends upon the viewpoint selected. The order of processes displayed in this window depends on the sort criteria selected. One may also use monitors to selectively hide or color processes displayed within the Process Area.

Processes may be selected from the Process Area. Many of the property and command windows requires the user to select one or more processes first. To select a process, point to it with the mouse and clicking it with either the left or middle button. Multiple processes may be selected by sweeping with the middle button down. (The selection behavior is based roughly on OpenWindows mechanism).

Double clicking on a process will bring up the Process Property Window (see Section 16.0 on page 25). This window allows the user to examine in detail various aspects of the process.

6.0 System Pane

The system pane shows system wide and per CPU information.

The system-wide statistics show the total number of processes and the number categorized by whether they are sleeping, running, idle, etc. On a single CPU system, there will be exactly one running process, namely ProCtool itself.

ProCtool will automatically detect the number of CPUs in a system and show the utilization times for each. One a single CPU system, the last PID field is not too interesting as it will always be ProCtool. The time utilization field may be interesting on a multiple CPU system since it may show the load balance on the system.

7.0 Viewpoint Window

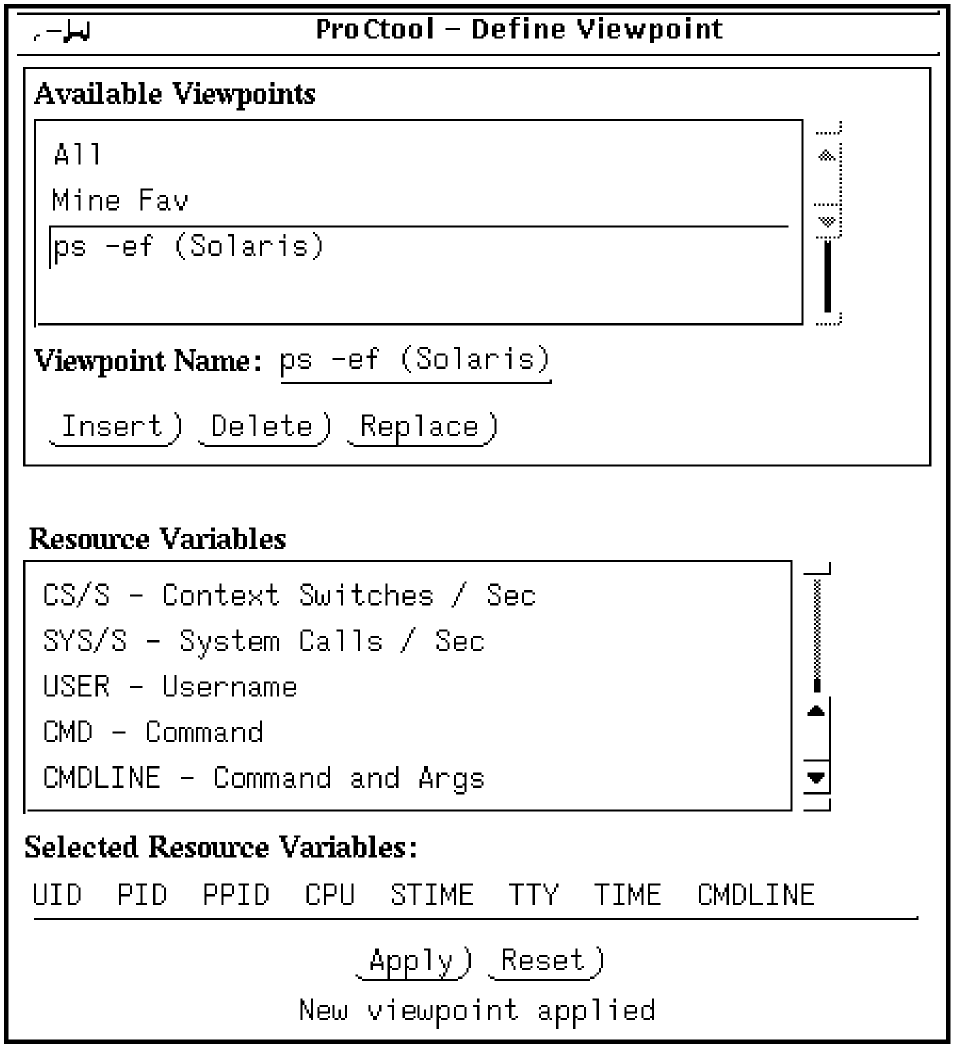


FIGURE 3. Viewpoint Command Window

The viewpoint command window allows the user to select which process parameters (resource variables) they want to see in the main process window pane. To change the current viewpoint, just enter a list of variables in the lower text field and hit the Apply button.

Viewpoints can be named and saved. This is done by entering the list of resource variables, then entering a viewpoint name in the top half of the command window, then hitting the Insert button. To replace an existing viewpoint definition, first select a viewpoint, edit the resource variable list on the bottom then hit replace. Hitting Reset resets the viewpoint to the previously set viewpoint.

8.0 Sort Window

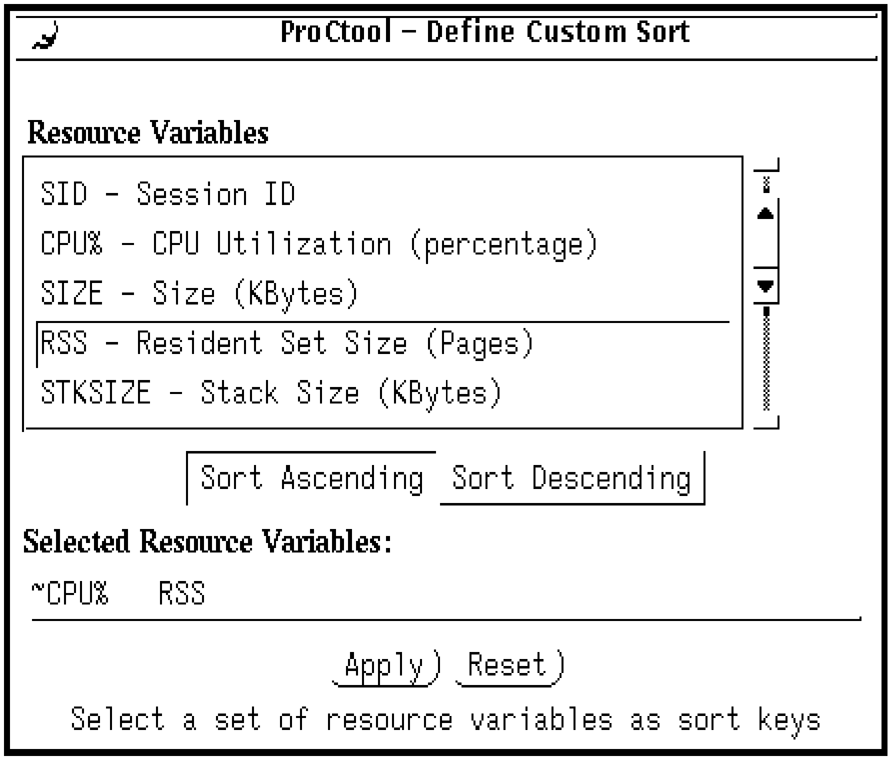


FIGURE 4. Sort Command Window

The sort command window allows one to define the sort keys used to sort the processes visible in the Process Area. The default sort key is by PID.

Multiple sort keys can be specified, up to 20. By default, ProCtool sorts in ascending order. One can choose to sort on a key in descending order by selecting the Sort Descending toggle switch before selecting a resource variable in the upper scrolling list.

Sort keys that are used in descending order are preceded by a tilde '~' in the textual representation.

For example, in Section FIGURE 4. on page 13 shows two keys, ~CPU% and RSS. This means that processes will be sorted so that the process consuming the most CPU resources (highest CPU% value) will be show first. In the event that two or more processes have the same value of CPU%, they will be sorted in ascending RSS usage.

A new set of keys specified in this window will take affect once the Apply button is selected.

9.0 Monitor Window

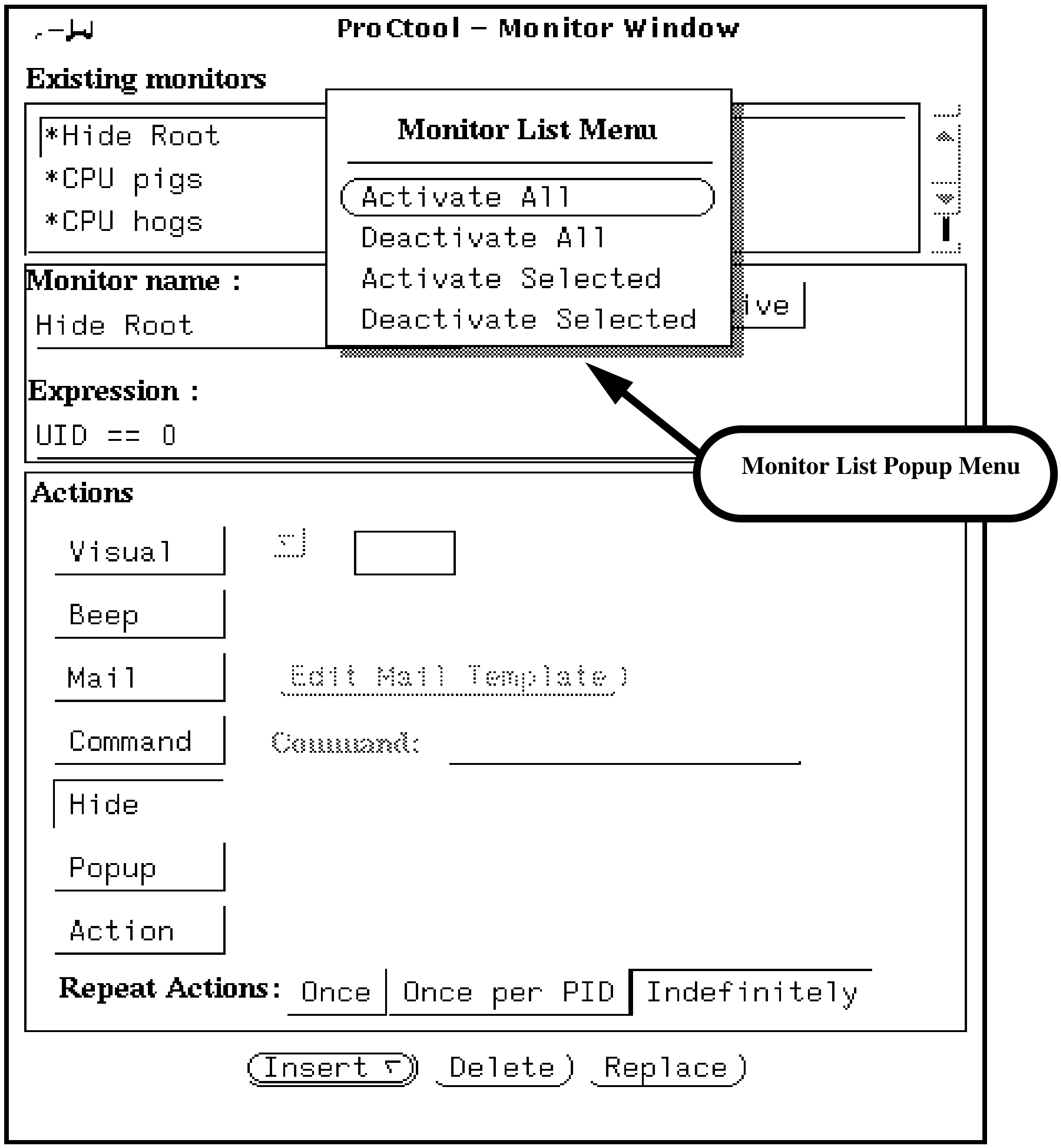


FIGURE 5. Monitor Command Window

The monitor window consists of the following components:

- *Monitor list*. A scrolling list of monitor names. Activated monitors are preceded with an asterisk '*'.
- *Monitor Popup menu* There is a menu associated with the scrolling list of monitor names. Use this to easily activate or deactivate one or all the monitors.
- *Monitor name text field* The name of the current monitor or new monitor.
- *Monitor expression text fiel*d Text field associated with the current monitor expression or a to be inserted monitor definition. You can manually enter the expression here or use the subexpression pane to do so.
- *Insert butto*n Inserts a monitor into the monitor list based on the current values in the current Monitor name, Monitor Expression, and Monitor Action fields. There can no more than one monitor with the same name.
- *Delete button* Deletes the currently selected monitor.
- *Replace button* Replaces the definition of the currently selected monitor with the values found in the Monitor name, Monitor Expression, and Monitor Action fields.

9.1 Usage

Typically one enters a new monitor by first entering a name, then an expression, and selecting a set of actions. Once this is done, the user can insert the monitor by hitting the *Insert* button. The user has the option of inserting a new monitor at the top or bottom of the list, or before and after another selected monitor. The order is important due to the order monitors are evaluated (see section on Monitor Evaluation below).

To edit the definition of an existing monitor, first select it. The name, expression, and action fields will show its current definitions. The user may then change any of these fields and then replace it by hitting the *Replace* button.

9.2 Monitor Evaluation

Monitors can be either be active or inactive. Deactivated monitors are not triggered. At every sampling update or at after each command that affects the state of a process (nice, kill, signal, priocntl), the list of active monitors is evaluated. Monitors are evaluated from top to bottom in the order they are found in the monitor list. For each monitor, the monitor expression is applied against all the processes, not just the ones that are visible. Processes are evaluated in ascending PID number for each monitor.

Conflicts may occur during monitor evaluation. For example, one monitor may want to change a process to be shown in red, another to be shown in orange. In these cases, the action of the last monitor takes precedence, using the rules described previously.

9.3 Monitor Actions

9.3.1 Visual

This monitor action is used to change the color of a process in the main process window pane. Monitors triggered with this action are not transcripted.

9.3.2 Beep

Monitors that are triggered will generate a bell (^G) on the terminal. In almost all usage, this would not be the only action selected.

9.3.3 Mail

There is a default message that is sent when a monitor mail action is triggered. The default message is shown in the figure below.

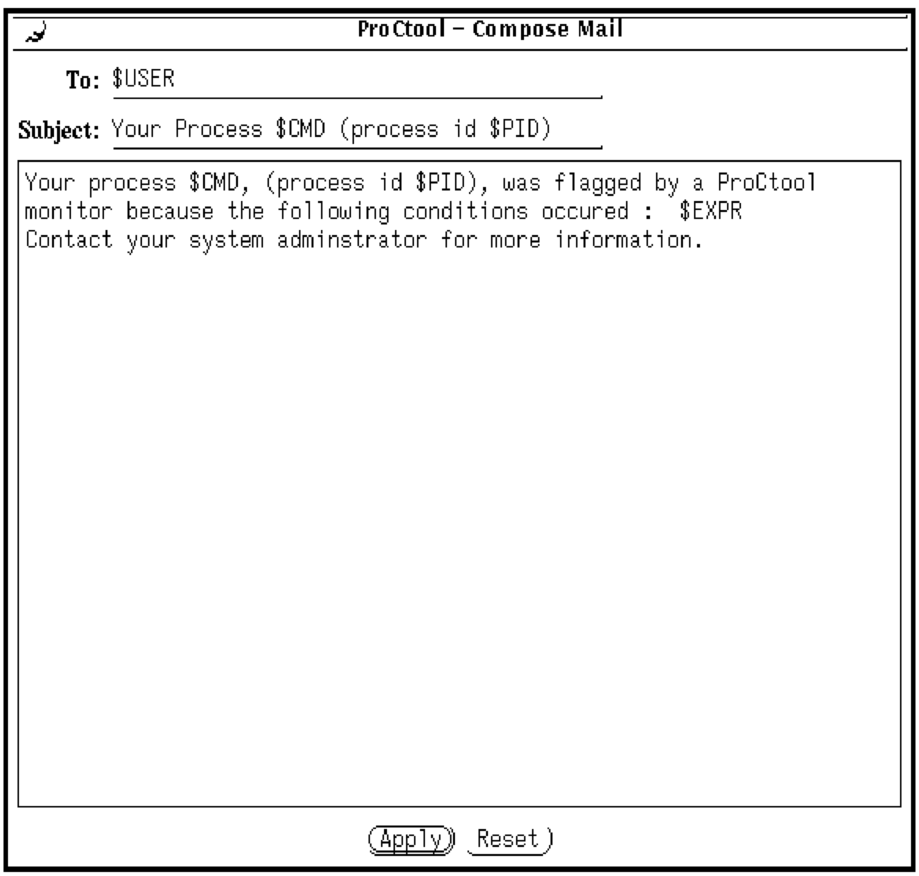


FIGURE 6. Monitor Mail Window.

The user may choose to customize the standard template. To do so, just edit this popup window. The edited template now becomes associated with the particular action. All other unedited monitor mail templates are not affected.

Note that the $USER, $CMD, $PID, and $EXPR are macros that are expanded at the time the action is invoked. They are replaced with the actual values of the process that triggered the monitor. In addition, the macro $EXPR expands to the actual monitor expression that was triggered. Note that all process resource variables may be used in the template (e.g. $RSS, $CPU, $NICE, etc.).

9.3.4 Command

Select this button to define a Monitor Action which can be any Unix command. That command will be invoked when the monitor is invoked. The command string is evaluated in the Bourne shell (/bin/sh).

The command string does macro expansion just as the Monitor Mail text does (see above).

NOTE. Discretion should be observed on what kind of command is used here. In this version of ProCtool, ProCtool will hang until the command is completed.

9.3.5 Hide

Use this action to hide a process in ProCtool's main process window pane. A typical example of this usage is to hide all processes owned by root using this expression:

(UID == 0)

9.3.6 Popup

Use this action to generate a popup when the monitor is triggered. Only one popup is triggered per monitor. The popup will show the process ID's of up to the first ten processes that triggered the monitor. The others can be seen in the transcript.

9.3.7 Repetition Count

For each monitor, you can specify how many times it should trigger. The options are:

- ONCE - Trigger once and deactivate itself.
- ONCE PER PID - Trigger once per unique PID. The monitor keeps track of each PID it triggered on.
- INDEFINITELY - Keep on triggering.

9.4 Activation

The default init file has a bunch of monitors defined. Usability hint: the easiest way to selectively activate/deactivate is to use the popup menu on the monitor scrolling list at the top of the Monitor Window. Do this by pressing the right (menu) mouse button over it.

10.0 Signal Window

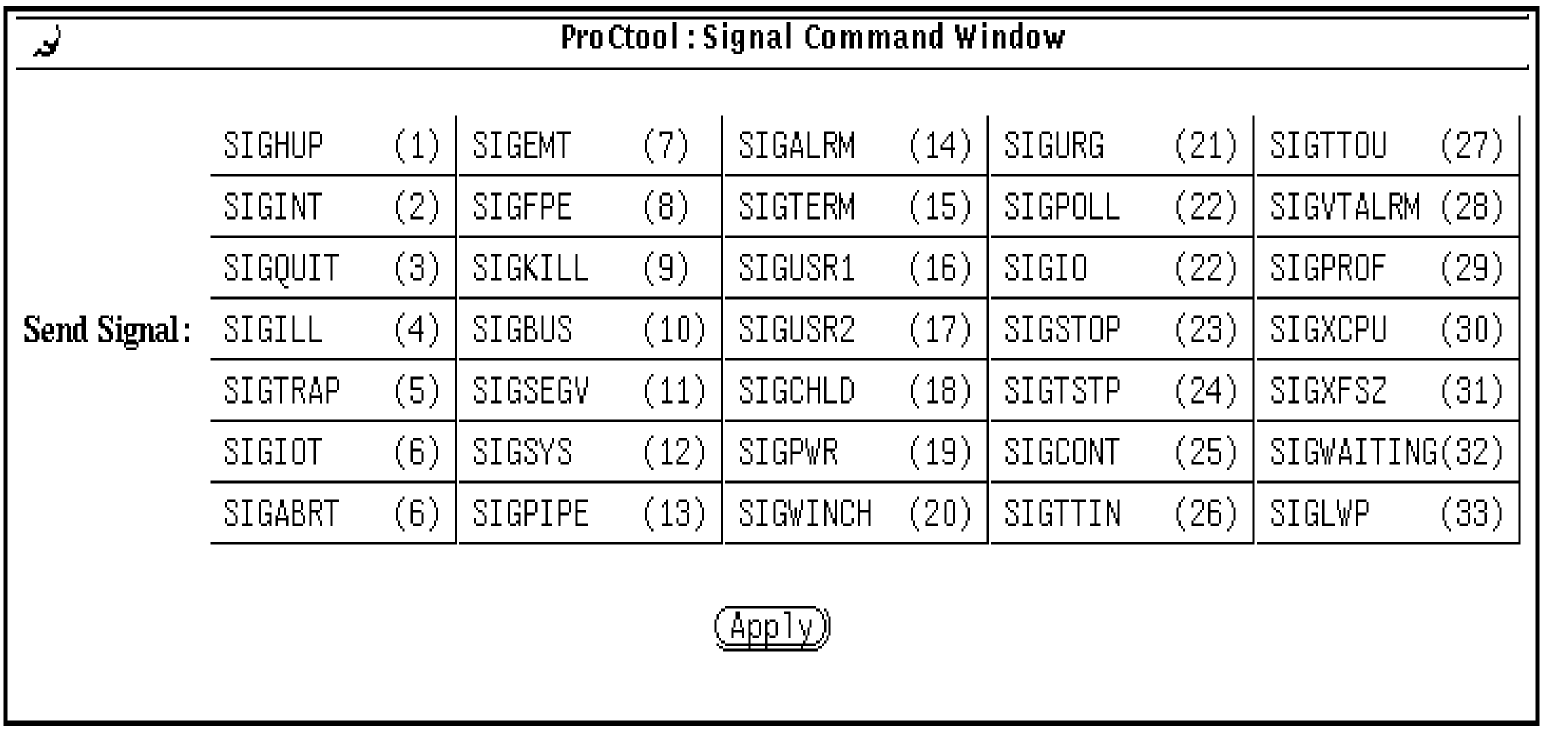


FIGURE 7. Signal Command Window

Allows one to send any signal to the list of processes selected in ProCtool main process window area. If more than one process is selected, the same signal is sent to each of the selected processes but the order is undefined. The user must have root privileges to send a signal to a process which they do not own. The results of sending the signal is shown in the transcript window. Hint - not a good idea to try this on ProCtool itself!

11.0 Process Control Window

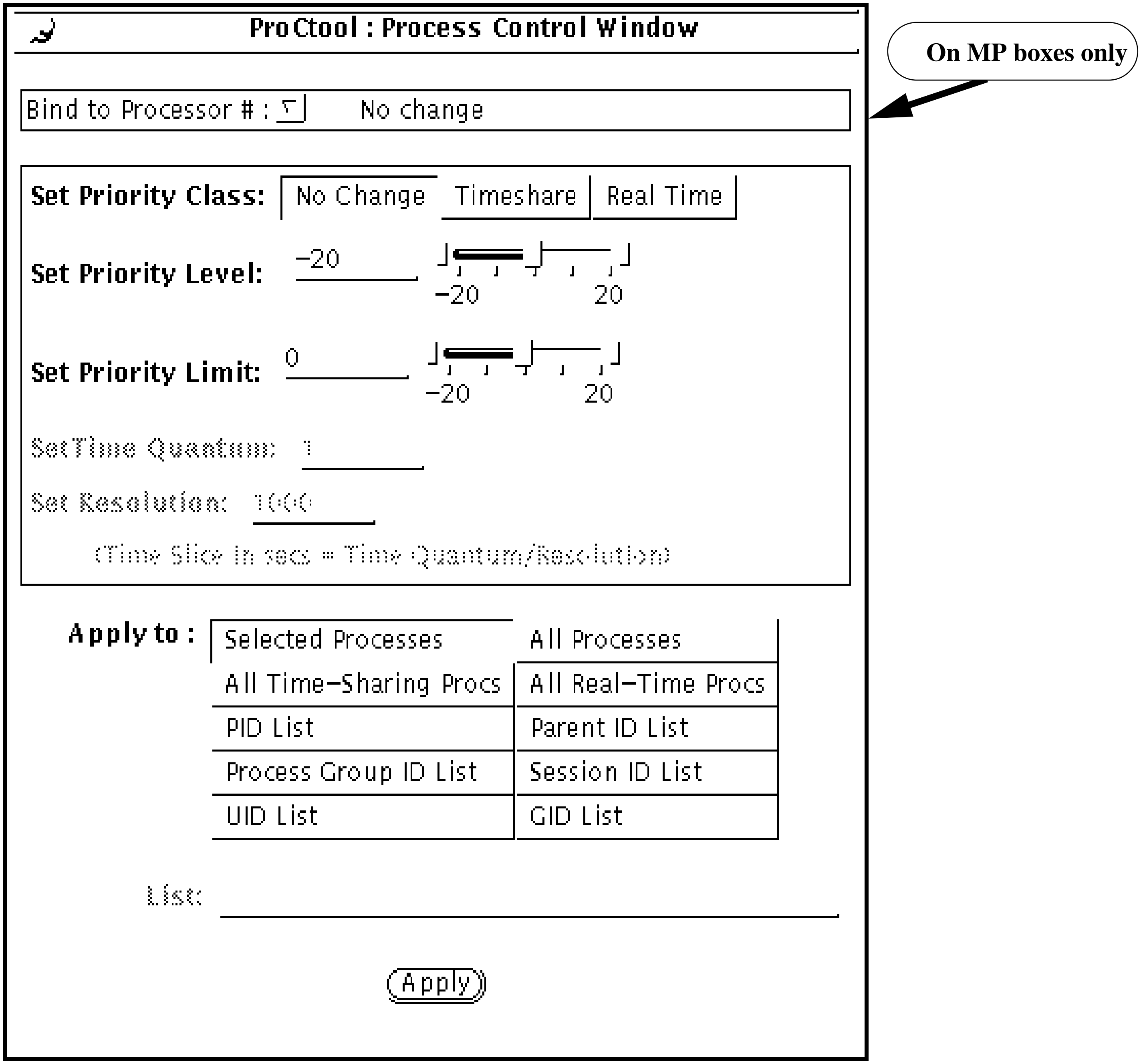


FIGURE 8. Priocntl Command Window

The Process Control Window provides two set of functions.

- Processor Binding - Allows one to bind a set of processes to a particular CPU. **This option is available only on multi-processor systems.**

- Scheduling Control - Allows one to change the scheduling parameters for a set of processes. Includes ability to change scheduling classes from Time-Sharing to and from Real-time. See priocntl(1) for more details on the various parameters.

For either one of these options, one can apply the changes to a list of processes selected in the main process area, to a list of processes with the same User ID, Parent Process ID, Group ID, etc.

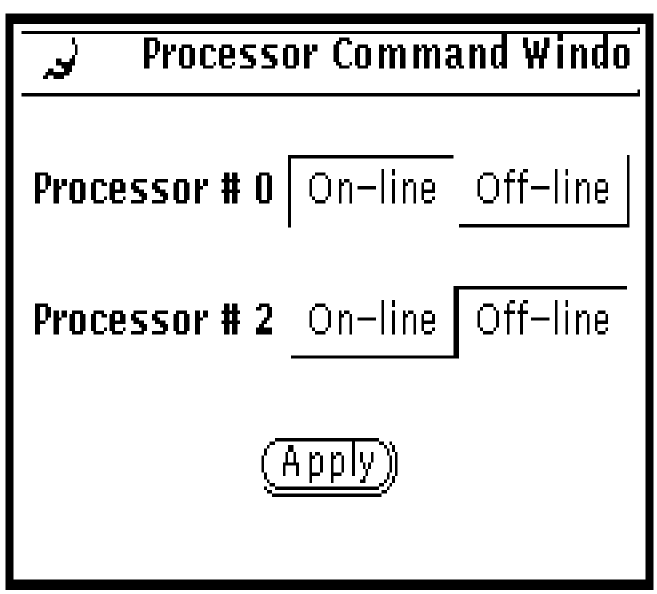
12.0 Set UID Window

Use this command window to obtain root privileges. ProCtool starts up in a non-privileged mode unless invoked as root. By entering the root password, the user is then granted super-user privileges. Having this permission allows users to nice, kill, send signals, set run-time priorities and examine processes that they do not own.

13.0 CPU Control Window

This option is available only on multi-processor systems. It allows one to selectively turn off and on processors. When using this option, ProCtool will check the status of the processors when this window is first popped up. After the user selects a change in status for one or more of the processors and hits the APPLY button, it will once again check the current status of the processors before applying the change. This is to avoid getting into situations where ALL of the processors are turned off !!

Processor ids are not necessarily numbered sequentially starting from 0. In particular, on the SS10 desktops, they seem to be numbered evenly. Perhaps to accommodate future 2 CPU modules.



14.0 Graphs

ProCtool provides three graph windows. Depending on the type of graph window, there can be up to 4 graph areas within each window. Graph windows can be used to visually display various system and process parameters. The graphs are updated at every sampling period, which is in sync with the main process window. ProCtool maintains a history of up to 60 samples per graph.

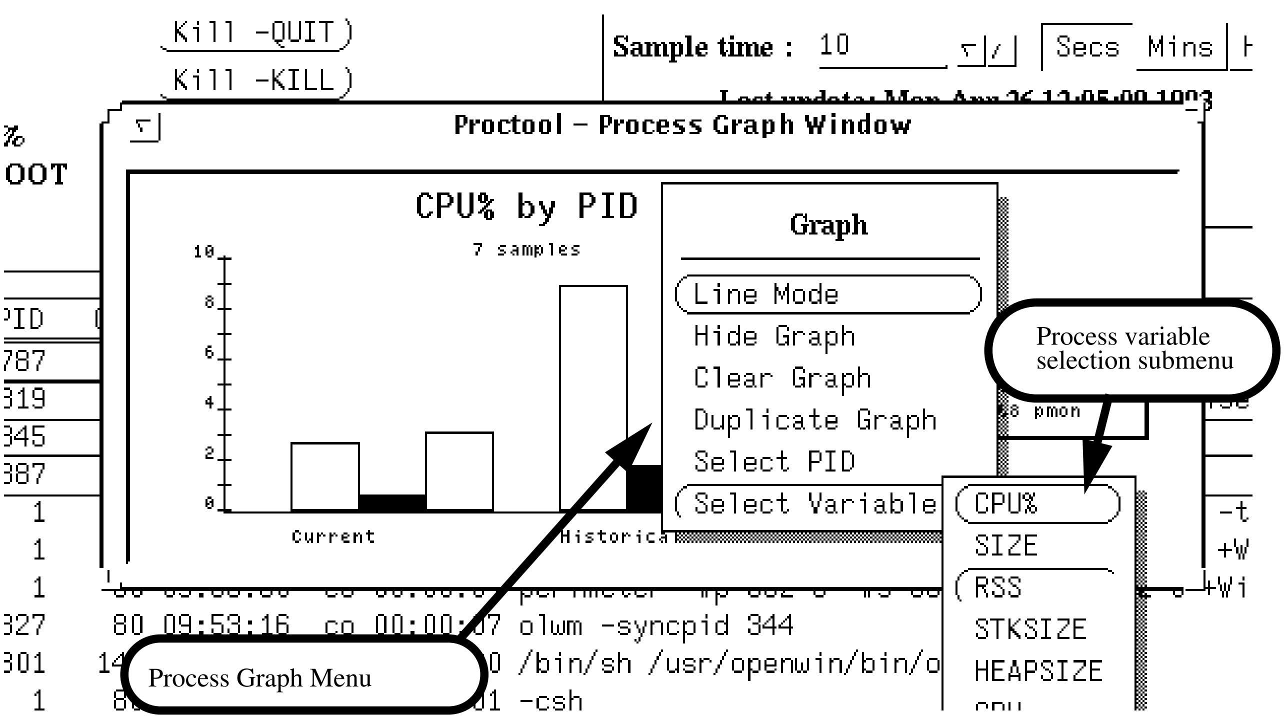
All graphs can be shown in either line mode or bar mode. The line mode charts the history, up to 60 samples. In bar mode, ProCtool will always show at least two bar graphs, one representing the most current values and the other one showing the average values for all samples in its history buffer. Currently there are no facilities for history greater than 60 samples or graph replay capabilities.

Once a graph window is opened and it starts graphing, the updates occur in sync every time the main process window is updated. If a graph window is iconized, it's data is still updated though not rendered. If an user QUITs a graph window, then updating terminates.

14.1 System Graph Window.

The can be up to three graph areas in this window.

FIGURE 9. CPU Control Window



- **CPU Graph** - Shows the processor utilization for each CPU on the system. Given in %user, %idle, %system, and %wait. ProCtool will detect how many CPUs are actually on the system and scale the graph accordingly. In bar graph mode, ProCtool displays a graph for each processor, as well a single-sample combined graph, and a multi-sample (historical) combined graph. (On a single processor system, there is no "combined" graph.)

FIGURE 10. Process Graph Window

- **Paging Graph** - Shows the paging behavior of the OS. The fields are exactly the same as what vmstat(1M) shows. The fields are:
  - page reclaims
  - minor faults
  - kilobytes paged in
  - kilobytes paged out
  - kilobytes freed
  - anticipated short-term memory shortfall (Kbytes)
  - pages scanned by clock algorithm
- **Context Switching Graph** - This graph window is NOT implemented in ProCtool 2.3.

14.2 Process Graph Window

This window allows the user to graph a variable for a set of processes. There can be up to 4 graph areas in this window. One can choose up to 8 processes for each graph area.

size	- process size (in Kbytes)
cpu%	- cpu percentage
cpu	- cpu "tick" count (as from "ps")
cpu#	- The CPU id where the process last ran (always 0 for non-MP)
io/s	- avg # blocks read/written per second by process
heap	- process heap size (in Kbytes)
lwp#	- maximum assigned LWP number
mem%	- memory utilization percentage
stack	- process stack size (in Kbytes)
rss	- resident set size (in Kbytes)
sys/s	- number of system calls per second
mpf/s	- minor page faults per second
pf/s	- major page faults per second
cs/s	- context switches per second

Here are the steps to graph a set of processes (see Figure 9, “CPU Control Window,” on page 21):

First bring up the Process Graph Window.

Next select up to 8 processes in the main window (single click plus middle button select).

Bring up the popup-menu in the graph and pick "SELECT PIDS".

Bring up the same popup-menu and pick "SELECT VARIABLE".

By selecting DUPLICATE GRAPH, one can bring up another graph area. You can graph a different set of processes on this graph and/or another variable.

14.3 Process Page Usage Graph Window

This is a specialized graph for showing detail virtual memory page access patterns for one process. On Solaris 2.x, one can instruct the kernel to keep detailed accounting of which virtual memory pages a process accesses. ProCtool breaks down the selected processes page usage with some detail, showing the of new text vs. data and read vs. write accesses on a per page granularity.

Only one process can be displayed in this graph. The selection mechanism is exactly the same as in the process graph (Figure 14.2, “Process Graph Window,” on page 22).

NOTE Using this option on a process will put a slight load on a system, it should not be used while doing a benchmark.

Also note that if you use "kill -9" to terminate ProCtool, then it will not have any opportunity to disable the accounting which is being done by the kernel. If proctool is allowed to exit normally (or via any catchable signal), then it will reset this state gracefully.

14.4 Changing Graph Colors

The default colors for the variables in a graph are GREEN, BLUE, YELLOW, RED, PURPLE, ORANGE, AQUAMARINE, and MAGENTA. You can change the colors by adding the following lines to your X resources file:

```
proctool.graph_color_0:  pink
proctool.graph_color_1: yellow
```

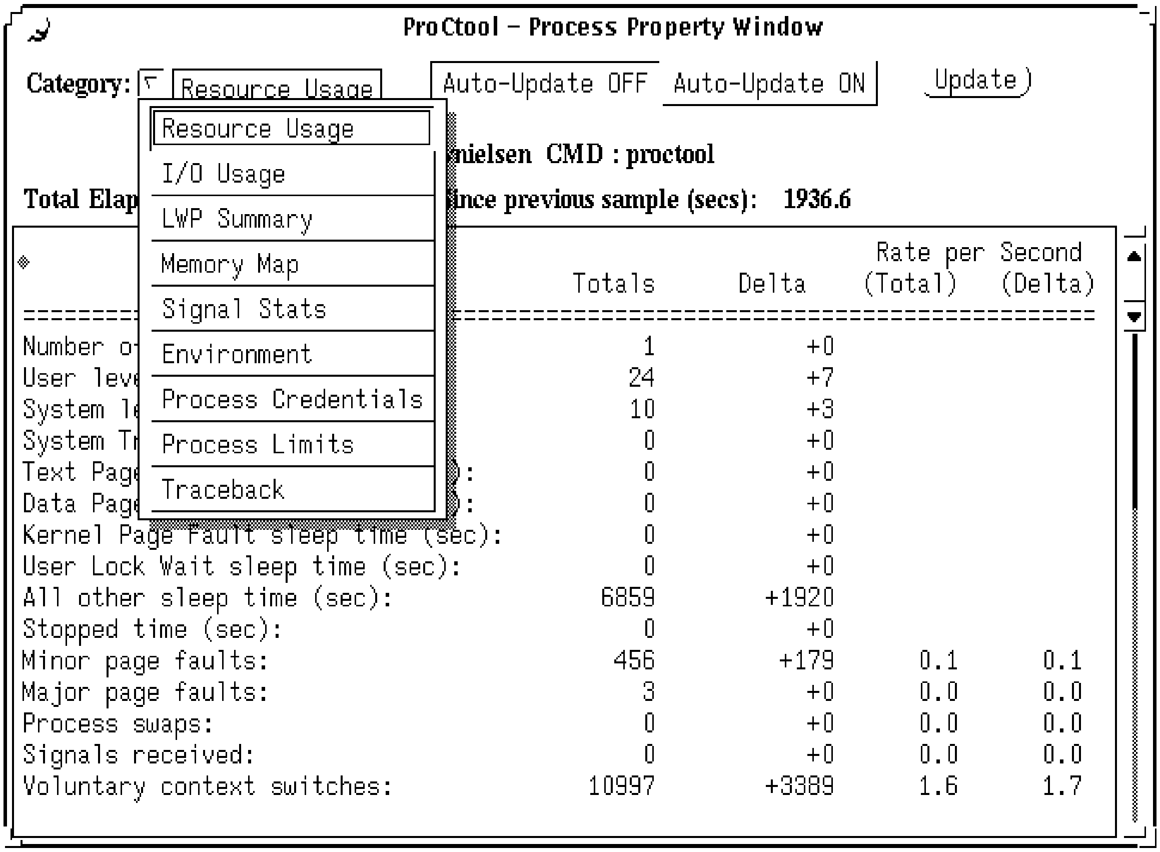
In this example, the first variable for all the graphs will be PINK instead of GREEN. And the second would be YELLOW instead of BLUE.

15.0 Tool Property Window

The tool property window allows the user to set various parameters which apply to all of ProCtool. The control gadgets in the window are described below:

- **Auto Update On/Off Toggle** After a command such as signal or renice, ProCtool will get a new sample of the process data. Reason for doing this is that usually people don't want to see a process in the screen that they just killed. For some reason, if you don't want this update to happen automatically, set this toggle to OFF.
- **Bell on Error On/Off Toggle** When ProCtool detects an error (for example, when it is unable to sample data for a particular process) it posts an error message to the status line of the main display and rings the terminal bell. In a situation where this happens frequently, the bell can be turned off.
- **Transcript Mode Append/Overwrite Toggle** The transcript file can be opened either in append or overwrite mode. The default is append but then you transcript file can get big after a few dozen sessions. Use this switch to suit yourself.
- **Transcript File Text Field** The transcript file has a preset format which appends the user name. If this is not what you want, use this option.

16.0 Process Property Window



The process property window presents information for a particular process. The kind of information presented here is generally too expensive (CPU-wise) to show for all processes in the main process window.

To invoke this window, double click on a process in the main window's process area. The contents of the Process Property Window can be updated by hitting the Update button. One can also turn on the auto-update mode. When this is done, the contents are updated after every sampling time interval.

- Traceback. This stops the process, if running, and gives a stack trace (call-chain) of the process. Upon completion it will restart the process if it was previously running. Invoking this option requires a bit of processing and may take 3-5 seconds for an average process. This feature only works for ELF executables. Note that since this changes the state of the process being inspected, you should not use "kill -9" to terminate ProCtool during a traceback.
- Process Limits. Shows scheduling class and limits if time-sharing, or time quantum if real time. Equivalent to the 'priocntl -d -i pid <pid>' command. Also displays the process resource limits (like the csh "limits" command).
- Environment. Displays the process file creation mask, the invocation arguments, and the entire list of environmental strings for the specified process.
- Signal Statistics. Shows the signals handlers associated with each signal number.
- Process Credentials. Effective/real user and group plus more.
- Resource Usage. Various resources pertaining to the process. Includes number of LWPs used by the process, paging rates, systems calls, etc.
- I/O Usage. Some numbers on I/O usage and a list of file descriptors opened.
- Memory Map. Gives the list of virtual addresses and permissions used by the process. Where one exists, it will show what shared library a particular memory segment represents. This feature only works for ELF executables.
- LWP Summary. Displays a summary of user, system, real, and lock times, as well as fault rates, I/O rates, context switch rates, and signal and system call counts for each light weight process within the selected process.

17.0 System Property Window

The system property window presents information that apply to the system as a whole. There are three available views. Selecting the Update button updates the data should it be changed from within or without ProCtool. This window may be selected through the PROPERTIES menu -> SYSTEM PROPERTIES menu item or by double clicking on the system pane.

- VM Statistics Give statistics on virtual memory and physical memory usage on system.
- Hardware Configuration. Shows the kind of hardware (clock rate, number of CPUS, etc) in the system.
- Disk I/O. Shows read/write/wait statistics for all disks in the system.
- Network I/O. Shows read/write/error statistics for all network interfaces.
- Time-sharing scheduling parameters Gives time-sharing dispatch parameters. This is the equivalent of the dispadmin (1M) command, 'dispadmin -c TS -g '
- Real-time scheduling parameters Gives real-time dispatch parameters. This is the equivalent of the dispadmin (1M) command, 'dispadmin -c RT -g '

18.0 Configuration

Configuration is done in the following steps:

1. The tool is currently distributed in a compressed tar file format and not pkgadd.
2. As root, uncompress the file and untar it to the directory of choice, possibly /usr/local/bin.
3. The following files are in the compressed tar file:

proctool	- (ELF Binary) GUI front end component.
pmon	- (ELF Binary) Backend. Must be owned by root and set UID.
proc.help.txt	- (Ascii) Help text.
README	- (Ascii) Simple installation guide (this section).
proc.help.idx	- (Ascii) Help index file.
proctool.ps	- (Postscript) Users' guide.
.proc.init	- (Binary) Optional System wide initialization file.

4. Make sure that pmon is owned by root and has the set-UID bit turned on. If not then 'chown root pmon; chmod u+s pmon'
5. Exit root mode, if desired.
6. Set the environment PROCDIR to the installation directory. If this step is omitted, proctool will default $PROCDIR to the directory where the proctool binary resides. This directory is searched for help information, and it is used to hold user transcript files.
7. Set your PATH variable to include the path to both proctool and pmon, usually $PROCDIR.
8. We've attempted to use reasonable default values for all fonts and colors, however you may specify any of the following in the user's .Xdefaults or in $HOME/Proctool or in $XAPPL-RESDIR/Proctool:

proctool.*.background:

proctool.*.title.font:

proctool.*.caption.font:

proctool.*.Caption.font:

proctool.*.font:

proctool.*.TextEdit.InputFocusColor:

proctool.*.InputFocusColor:

proctool.*.TextEdit.font:

9. We've also implemented some "special" resources which may also be specified in the same locations

proctool.process.font:

proctool.system.font:

proctool.alert.font:

proctool.process_text.font:

proctool.system_text.font:

proctool.graph_title.font:

proctool.graph_legend.font:

10. Invoke as 'proctool'.

19.0 Implementation

ProCtool is actually two processes, proctool and pmon. proctool is the GUI front-end that communicates to pmon. pmon is a setuid (to root) program that reads the /proc and kernel data structures to get the necessary information for proctool. There are a number of reasons for having two processes -

1) To preserve security of the system. Many of the operations done by pmon require root privileges hence pmon is setuid to root; proctool is not. This allows proctool, which includes many shared libraries to be safe from Trojan horses.

2) Future distributed systems. A future enhancement may involve having multiple pmons on various systems and a centralized proctool. This would allow monitoring multiple systems.

The communication protocol between the front-end and the back-end is done via pipes and shared memory. At some point, it probably makes more sense to do this using a protocol like RPC or Tooltalk.

ProCtool's GUI was implemented using Motif version 1.2 on Solaris 2.x using ANSI C.

20.0 Known bugs and Deficiencies

Graphs - In a line graph, if two or more variables have the same exact value for a more than one sampling period, they overwrite each other and only the last one is seen.

Traceback - Occasionally is unable to find an entry point, and prints it out as "???????". Does not work on a.out files running under Binary compatibility mode. Tracebacks are extremely slow and could use optimization work.

Memory map- There is some guesswork in the description of some memory map segments. For example, if a segment maps to a shared library, then if it has execute but not write privileges, it is assumed to be the text. Doesn't work on a.out files running under Binary compatibility mode.

Error messages- When displayed in the footer area of the main window or a popup, they sometimes get quickly overwritten by something else. Most errors are also written into the transcript but not all.

Customizing fonts, etc. Most fonts, colors, etc. can be customized using your standard Xt resources. In some cases, ProCtool has some fixed non-customizable resources. Examples are the font type in the process and system window. This may change in the future.

21.0 Trouble Shooting

In case you have trouble running proctool, here is a short checklist of potential problems:

- The version of ProCtool must match the version of Solaris. For version 2.X of Solaris,you must execute version 2.X.Y of ProCtool. The tool will attempt to enforce this requirement, but you can manually bypass the check with the "-v" option.
- You must have a Motif library installed. This library is part of the "SUNWmfrun" package which is included in Solaris (from 2.4 on). If you're unsure if it is installed, look for "libXm.so" in the directory "/usr/dt/lib". For Solaris 2.3, you can get a Motif library from a Solaris 2.4 system (this is legal and supported), or from a Solaris 2.3 SDK. You can recognize this failure from the error message issued by the runtime loader:
ld.so.1: proctool: fatal: libXm.so.3: can't open file: errno=2
- Your X-server must support the fonts being used. This is typically an issue if ProCtool is being remotely displayed on an X-terminal (or a workstation not running the Sun supplied X-server). In this case, you should explicitly specify a font in your ~/.Xdefaults file or in your ~/Proctool file. You can recognize this failure from an error message like:
Proctool error - Bad resource specified for proctool.process.font.
- The monitor program, "pmon", must be set-uid to "root". Make sure that the executable is owned by "root", and that the set-uid bit is turned on (ie. "chmod u+s pmon"). One other common problem is that the filesystem where the binary is installed may not be exported with the necessary permissions to run a set-uid program.

22.0 Bug Reporting

If you happen to find a bug in proctool, please report it to the authors; otherwise it probably won't be fixed! In all cases, please report the architecture (SPARC, Intel, PPC), the ProCtool version, and the Solaris version. Then, if possible, please include a specific set of steps needed to reproduce the error.

If the problem isn't easy to reproduce, and if either of the programs dump core, then please include the core file in the bug report (or at least include a traceback from dbx or adb).

If the display program (proctool) is dying without leaving a core file, you can invoke it under a debugger and ask for a traceback when it quits. This could also be done with the monitor program (pmon), but is more difficult since you would have to attach to the running program. For pmon problems, try setting the environment variable PROCTOOL_DEBUG and re-running. This will generally cause the program to dump information about what went wrong and what it was doing. One other useful debugging technique is to run ProCtool under "truss" and include the last few hundred lines of output in the bug report (ie. "truss -f proctool > /tmp/bugreport 2>&1").

23.0 Futures

Many of the features provided are very coarse and will be refined in the future, depending on user reaction and interest. So if you sort of like it,but want to see something else in there, please send us mail. Some possible extensions envisioned by us are:

- Distributed ProCtool. As mentioned above, most of the work for a distributed system is done since there is a separation between the front-end and the data accumulator (pmon). By choosing another communication method, such as RPC or Tooltalk a truly distributed system is possible. Possible extension - monitoring multiple systems.
- More multi-thread information. Current breaks down a process by LWP only. Future versions may show user-level thread information.
- Multi-thread ProCtool itself. A lot of functionality like data gathering can be done in parallel.
- Extend Monitor capabilities. The list of variables are limited to per process parameters such as UID, RSS, etc. Future versions may include system wide parameters as load average, swap space available, etc.
- Extend Monitor expressions. Possibilities are to have 'higher-order' predicates such as NumberOfProcesses(UID == 0) which would return the number of processes meeting the logical expression within the predicate.
- Truss (trace). The ability to turn on and off tracing on a running process.
- Ability to edit new dispadmin time-sharing and real-time tables.
- SunOS 4.x. version? We do not have plans to backport to SunOS 4.x. For the right price, we can have a 4.x version for you.

Appendix A Monitor Expression Syntax

<expr> **::==** *<expr_list>*

<expr_list> **::==** *<subexpr>* **[** *<logical op> <expr_list>* **]**

| *<paren_list list>* **[** *<logical op> <expr_list>* **]**

<paren_list> **::==** *'(' <expr_list> ')'*

<subexpr> **::==** *<valvar> <relational op> <valvar>*

<valvar> **::==** *<value>* **|** *<varname>*

<logical op> **::==** *AND* | *OR*

<relational op> **::==** == **|** *!=* **|** < **|** > **|** >= **|** =<

<value> **::==** *<integer value>* **|** *<string>* **|** *<float value>*

<logical ops> have a higher priority than *<relational ops>*

Appendix B Process Area Variables

- PID, UID, PPID, PGID, SID. Process, user, group, parent, parent group, and session ID's for the process.
- CPU%. The CPU utilization as maintained by the kernel (also reported by "/usr/ucb/ps"). It is a running average, so the value is more "damped" than the CPU% value that proctool previously reported (which was computed on a sample-by-sample basis).
- MEM%. The percent of system memory used by this process.
- SIZE, RSS. The total size and the resident set size of the process (in KBytes)
- STACK, HEAP. The size of the process stack and heap (in KBytes).
- CPU. A count of the "cpu ticks" which is provided for compatibility with the output of "ps".
- TIME. The sum of the "user" and "system" times for this process (in seconds).
- CTIME. The sum of the "user" and "system" times for reaped children (in seconds).
- SDATE, STIME. The date and time of process creation (in YY/MM/DD and HH/MM/SS format to enable the standard dictionary sort to accurately sort the date and time).
- ST. Process state
  - O On processor - the currently running process. On a uniprocessor, this will always be the monitor process, "pmon", since it is obviously running while it is probing the system.
  - R Runnable. The process is in the run queue and is ready to be dispatched.
  - Z Zombie. The process has terminated and its parent process is not waiting for it.
  - T Stopped. The process is in the stopped state. Contrary to the "ps" man page, it is not necessarily being traced.
  - S Sleeping. The process is waiting for an event to complete.
  - I Idle. The process is being created.
  - X SXBRK state. The process is waiting for more primary memory.
- CLS. Process Class. This may take on at least the following values.
  - TS Time Share Class (ie. most user application processes)
  - IA Interactive Class (ie. processes run under the OpenLook window manager)
  - SYS System Class (ie. scheduler, pageout daemon)
  - RT Real Time Class (ie. real-time data acquisition process)
- PRI. The priority of the process (see "man priocntl" for a description of priorities).
- NICE. The "nice" value for a process (see "man -s 2 nice" for a description).
- TTY. Generally, this will be the number of the pseudo tty device associated with the process, however, it will be the string "co" for the console device and "?" if there is no tty for the process.
- ADDR. The physical memory address where the process is loaded.

- WCHAN. The "wait channel" for a process. If the process is sleeping, this is the address (in the kernel) that it is waiting on. Currently, there are no mnemonics associated with each address (like there was in the SunOS 4.X version of "ps"), however, you can at least determine if multiple processes are waiting on the same resource. If the process is not sleeping, this will be 0.
- FLAGS. Process flags. See /usr/include/sys/procfs.h for a more complete listing, but some of the flags of interest are included below:
  - 0x0001 PR_STOPPED The LWP is stopped.
  - 0x0008 PR_ASLEEP The LWP is sleeping in a system call.
  - 0x0010 PR_FORK Inherit-on-fork is in effect.
  - 0x0020 PR_RLC Run-on-last-close is in effect.
  - 0x0100 PR_ISSYS System Process.
  - 0x0200 PR_STEP The LWP has a single-step directive in effect.
  - 0x0400 PR_KLC Kill-on-last-close is in effect.
  - 0x1000 PR_PCOMPAT Ptrace compatibility mode is in effect
- CPU#. Processor Number. This is the processor id which last ran this process.
- MSGS/S. Messages per second. The sum of messages sent and received per second.
- IO/S. Characters read and written per second.
- LWP. This is an ever-increasing number which corresponds to the highest LWP index created for this process. It is **not** a count of the current number of active LWP's because it is not decremented when an LWP exits.
- MPF/S. Minor page faults (those not requiring disk I/O) per second.
- PF/S. Major page faults (those requiring disk I/O) per second.
- CS/S. Context switches per second.
- USER. User name.
- CMD. The name of the process.
- CMDLINE. The name of the process and all of the invocation arguments.