IPS User's Guide

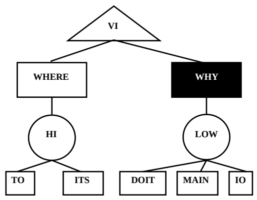
Version 5.0 (October 7, 1992)

Jeff Hollingsworth

Bruce Irvin

Barton P. Miller

University of Wisconsin-Madison



-- --

--

Release Notes (5.0)

This new major release of IPS includes the following additions:

- Trace browsing (Section 3.2)
- Multiple program support (Section 4)
- Comparison mode (Section 4.3)
- Support for SPARC and CRAY-YMP architectures
- Interactive help windows (Appendix 2)
- Manual phase selection (Section 3.1)
- Support for automatic window stacking (Section 3.10.3)
- Open visualization interface (Reference Guide) and modules (Appendix 1)

1. Introduction

IPS[2] is an interactive, trace driven performance measurement system for parallel and distributed programs. It runs on DECstation, Sun, Vax, Sequent Symmetry, and Cray Y-MP computers (or a heterogeneous combination of these computers). IPS provides a large amount of performance data about the execution of a parallel program, and this information is organized hierarchically so that access to it is easy and intuitive.

Performance data is organized hierarchically from the whole application down to particular procedures inside the application. This model guides the user in finding performance bottle necks in their applications. Multiple applications (or multiple runs of the same application) can be measured in a single IPS session. Many performance metrics are available from IPS and most of these metrics can be presented in simple tables, profiles, or in graphs plotting the metric over time. These metrics include elapsed CPU time, synchronization blocking time (for both messages and busy waiting), I/O rates, procedure calls, and message rates. IPS also includes high-level analyses, such as Critical Path analysis [3], to help the user more quickly find performance bottlenecks.

IPS has open interfaces to allow easy support of new sources of performance data and new visualization modules. External performance data can include data collected from the operating system, network, and hardware. The external data facility is designed so that it is a simple task for users to add new metrics and modify old ones. Similarly, it is a simple task to add new types of visualizations.

There are two parts to IPS, an X Window based user interface called the master, and a slave analyst. The slave analyst, which runs on the same machine as the application, is responsible for processing statistics from the traces collected during program execution.

Instrumenting a program to run with IPS requires no changes to source code. All that is required is that the program be re-compiled using a special compiler flag.

The next section provides a step-by-step tutorial on how to compile, run and query performance metrics using IPS. This section introduces the basic features of IPS. Section 3 describes advanced features of IPS, such as phase analysis, window environment and user interface control, defining new metrics based on existing ones, and dynamic trace enabling. Appendix 1 lists manual pages for external visualization and data collection modules that are provided with the distribution. Appendix 2 lists the error messages that you might encounter during an IPS session and an explanation of each message.

An Reference Manual, Installation Guide, and a Porting Guide are also provided as part of the IPS software distribution. We would appreciate any comments about the document. Please send comments and bug reports to: ips@cs.wisc.edu.

2. Tutorial

This section describes how to compile an application for IPS profiling, how to run it, and how to query performance metrics after it has run. It traces a small distributed program from compilation through execution to querying of metrics.

2.1. Compiling an Application

Before an application can be run using IPS, it must be first compiled and linked using the IPS libraries. For C programs this involves using the version of cc located in `/usr/public/ips` and supplying the **-pi** (**-Wc,-pi** if your are using DYNIX/PTX) option. For example if we wished to compile an application consisting of two programs `test2a.c`, and `test2b.c`, the following commands would be issued.

```
% /usr/public/ips/cc -pi test2a.c -o test2a
% /usr/public/ips/cc -pi test2b.c -o test2b
```

Warnings: Application programs should exit with a status of 0 if they completed normally. In addition, the current version of IPS requires that programs using the Sequent parallel programming library call `m_kill_procs()` before exiting. If this routine is not called, the application program can hang in an infinite loop. Also, Sequent spin locks **MUST** be initialized by calling `s_init_lock`, calling `s_unlock` is **not** sufficient.

2.2. Describing Your Application to IPS

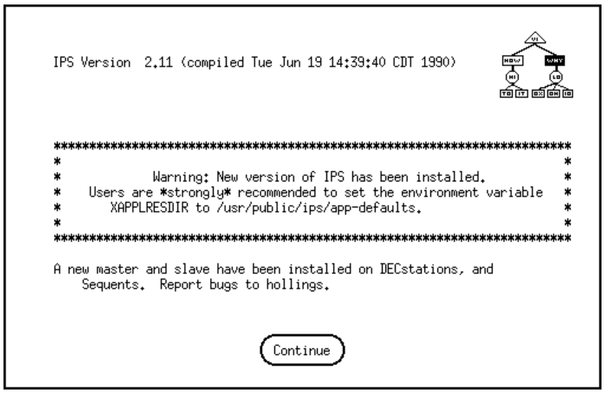
Once an application has been compiled, IPS can be used to run the program and view the performance data for the program. To start IPS, enter the command ips†. This creates two windows on the screen. The upper window is the *tree window*, and the lower one is the *message of the day (motd) window*. The motd window indicates which version of IPS is being run, and when it was compiled. In addition, there may be several lines of messages with information about the current release of IPS or its configuration on your machine. To dispose of this window, press any mouse button inside the **continue** box at the bottom of the window. (IPS won't let you continue until this is done.) A sample motd window appears in Figure 2.1.

2.2.1. Editing

IPS programs consist of multiple processes running on one or more machines. Before an application can be run using IPS, the structure of the application must be described. This is done via a graphical tree editor. The nodes in the tree specify the machines on which the application is to be run, and the command(s) that should be invoked on each machine. Application descriptions can also be saved in files for future executions. Figure 2.2 shows the IPS program tree and its main menu. The root node of the tree contains the name of the application (defaults to prog). The next level in the tree defines the machine(s) to run the application.

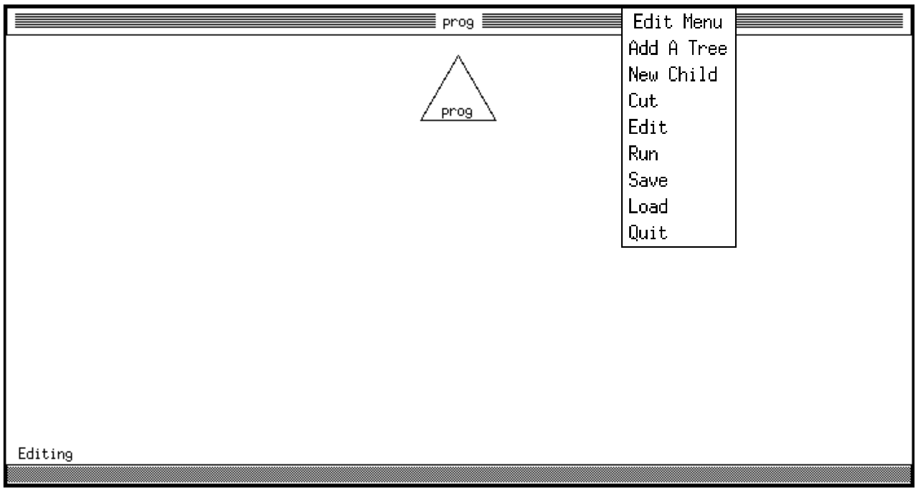
To add a machine, select the root node in the tree (Press the middle mouse button while the cursor is over the root node), and then select the "**New Child**" item in the program tree banner menu. (Banner menus appear when the left mouse button is pressed while over the striped banner at the top of all IPS windows). When "**New Child**" is selected, a property sheet similar to Figure

† On most UW machines ips is located in /var/scratch/bin.{machine type}/ips/ips.



Sample IPS message of the day window.

Figure 2.1.

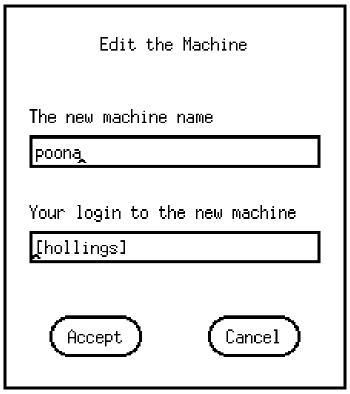


The program tree window as it appears when IPS starts

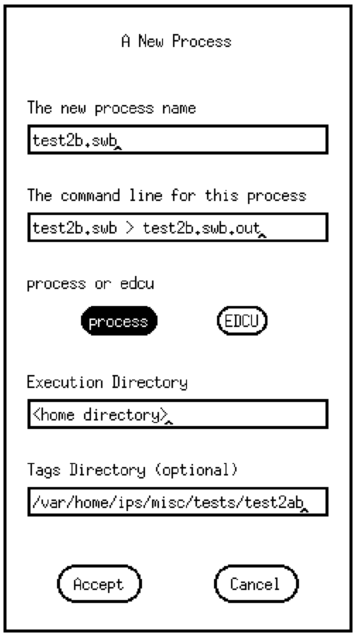
Figure 2.2.

2.3 will appear. The top box inside this sheet specifies a machine which will run one or more of the application processes. The host name should be a valid form of the name as might be used in a *rlogin* command to that host. In addition, there must be an entry in the `.rhosts` file on this machine for the host running the IPS master (even if they are the same machine). The next box specifies the login name to use on the remote host. If this field is left blank, the login name is assumed to be the same name as the account running the IPS master. Once the machine specification is complete, pressing any mouse button in the **Accept** box will add the machine node to the tree. The process should be repeated for each machine that will be used by the application.

Once the machines have been specified, processes to run on each machine can be specified. Select a machine node in the tree, and then select the "**New Child**" option in the banner menu. This causes a property sheet menu similar to Figure 2.4 to appear below the program tree. The first box in this sheet defines a name for the new process. This name identifies the process in the tree and is often the same as the command to run. The next box is used to specify the command line required to run a UNIX executable file for the process. This command will be run from the home directory of the account on the named remote machine. It is a normal UNIX shell command and can contain I/O redirection and command line arguments. The next item in the sheet is a set of buttons that indicates if the node is an application process or an external data collector process (external data collectors are described in section 4). The fourth box specifies the execution directory for the process. This is a directory on the remote machine where the application should execute. If no value is specified, the user's home directory on the remote machine will be used. The `Tags Directory` box is optional and defines the directory location of a *tags* file for the application. A tags file may be used by IPS or by EVUs to locate the application's source files. Pressing any mouse button in the **Accept** box causes the new process to be added to the tree. This step can be repeated for each initial process that is run on any of the machines. Figure 2.5 shows a tree for our application.



A machine level property sheet.
Figure 2.3.



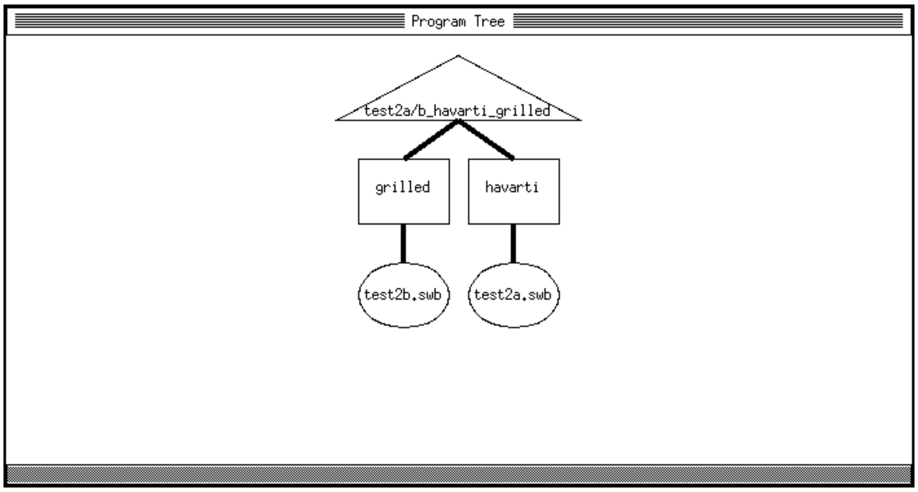
A process level property sheet.
Figure 2.4.

2.2.2. Running the Application

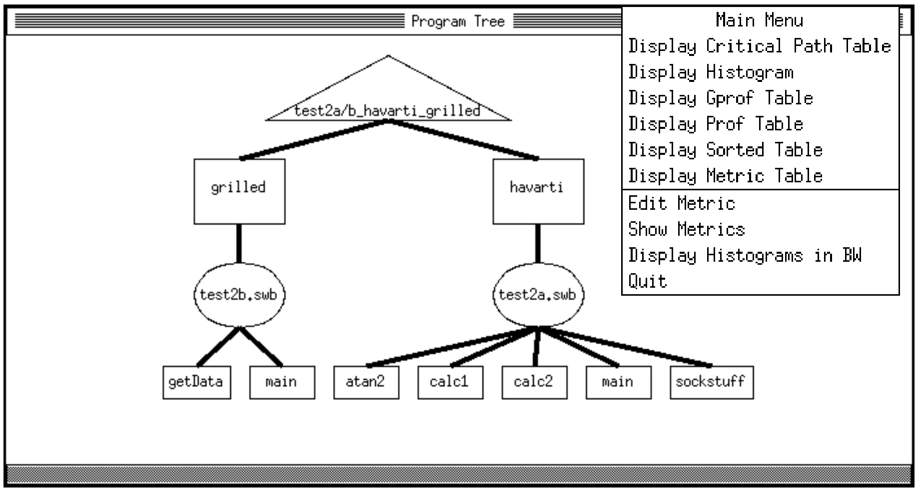
Once the program tree has been specified, the application is run using the "**Run**" command from the banner menu. While the application is running, the program tree window is grayed and mouse events are ignored. When the application finishes (and the slave analyst has finished collecting statistics), the tree will contain a new level (see Figure 2.6). If additional processes were created during execution (there were none in our example), new process nodes would appear. Also, nodes appear for each procedure that was executed. Only procedures that were called are listed, and they are sorted in alphabetical order. This is **not** a call graph so all procedures are immediate children of the process.

2.3. Displaying Metrics

At this point the tree might be quite bushy due to the addition of many procedure level nodes. IPS provides several ways to manage this complexity. The scroll bar at the bottom of the window is used to adjust which section of the tree is displayed. Mouse buttons can also be used to control the display. Pressing the left mouse button while over a tree node hides all of the



A typical IPS program tree before running the application.
Figure 2.5.



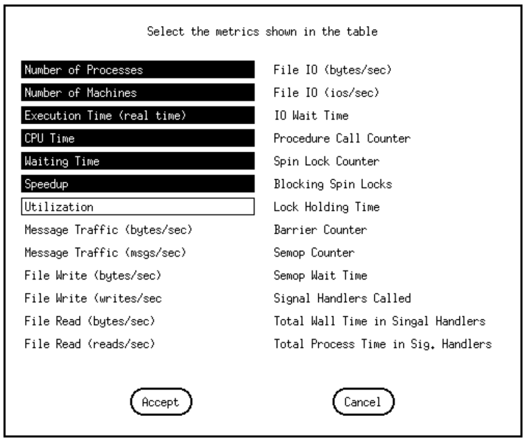
Program tree after the application has run.
Figure 2.6.

children (and descendents) of that node. Pressing the right mouse button while over a tree node displays all of the children of that node (but not their descendents). Pressing the letter "h" while the mouse is over a node hides the node and all of its descendents. There are also ways to hide only the "uninteresting" nodes (based on how many resources they used). These techniques are described in section 3.3.

Several displays can be created at this point. The program tree banner menu contains options for each type. Creating a display is a two step process. First select the program tree node by pressing the middle mouse button while over that node. The second step is to select the desired display from one of the menus. Two menus are available: the **Main** menu, which you can pop up by pressing the *left* mouse button in the program tree banner, and the **EVU** menu, which you can pop up using the *middle* mouse button. The Project menu has entries for each of the built-in IPS displays. The EVU menu contains entries for the External Visualization Units (EVUs). EVUs are discussed in section 2.3.6.

Tabular statistics for any node in the tree can be created using the "**Display Metric Table**" menu option. Sorted tabular data can be displayed by using the "**Display Sorted Table**" menu option. Graphs of metrics vs. time are created by selecting the "**Display Histogram**" menu item. *Prof* style profile tables can be displayed by selecting the "**Display Prof Table**" option. *Gprof*[1] style profile tables can be displayed by selecting the "**Display Gprof Table**" option. Critical path tables are created using the "**Display Critical Path Table**" option.

2.3.1. Tables of Metrics



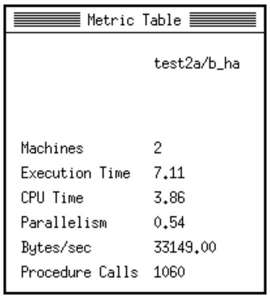
Tabular data property sheet showing six metrics selected.
Figure 2.7.

If the "**Display Metric Table**" option is selected, a new property sheet will appear in the lower left part of the screen (see Figure 2.7). This sheet is used to select which metrics will appear in the table. Any number of metrics may be selected. To select a metric, press any mouse button while over the named item. Once all metrics have been selected, pressing **Accept** will create the table (see Figure 2.8). Pressing the **Cancel** button will destroy the sheet and no table will be displayed.

Once the initial table has been built, the table's banner menu is used to modify the information displayed in the table or to destroy the table. The "**Add Selected Node**" menu item adds the currently selected node from the program tree to the table (see Figure 2.9). The "**Add Children**" menu item adds all child nodes (visible or hidden) of the currently selected node to the table. Adding a large number of procedure-level items to the table might take a few seconds. A more selective way to add nodes (columns) to the table is the "**Add Displayed Children**" menu item. This option adds only those child nodes that are currently displayed in the tree. Although nodes may not be visible (due to a bushy tree), they might still be displayed, and will be added when the visible children option is used. To remove nodes (columns) from the table, use the "**Delete Node**" menu item from the table's banner menu. This removes the currently selected program tree item from the table.

2.3.2. Sorted Tables

If the "**Display Sorted Table**" option is selected, a new property sheet will appear in the lower left part of the screen (see Figure 2.7). This sheet is used to select a single metric which will be displayed sorted by value for all nodes at the currently selected level of the program tree. To select a metric, press any mouse button while over the named item. Once all metrics have

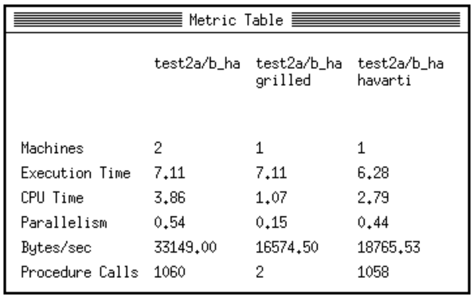


Metric Table

	test2a/b_ha
Machines	2
Execution Time	7.11
CPU Time	3.86
Parallelism	0.54
Bytes/sec	33149.00
Procedure Calls	1060

A sample metric table with only a program level node.

Figure 2.8.



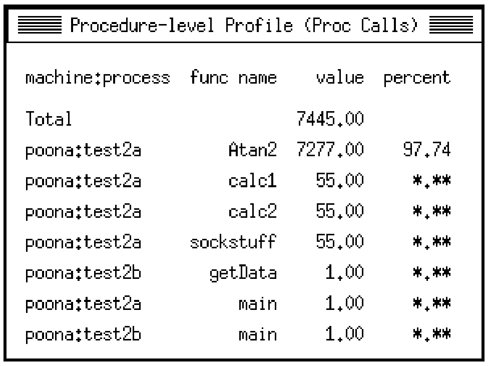
Metric Table

	test2a/b_ha	test2a/b_ha grilled	test2a/b_ha havarti
Machines	2	1	1
Execution Time	7.11	7.11	6.28
CPU Time	3.86	1.07	2.79
Parallelism	0.54	0.15	0.44
Bytes/sec	33149.00	16574.50	18765.53
Procedure Calls	1060	2	1058

A metric table with both program and machine level nodes.

Figure 2.9.

been selected, press **Accept** to create the table (see Figure 2.8). Figure 2.10 shows a sample sorted table.



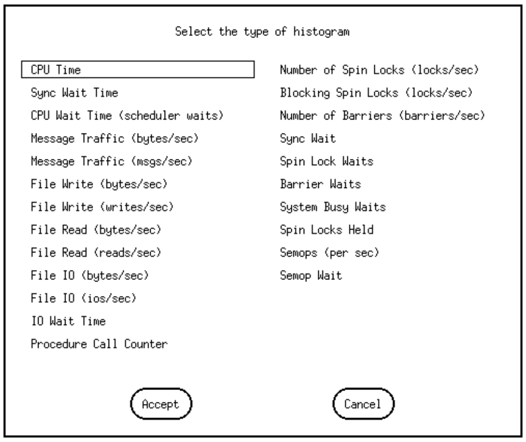
Procedure-level Profile (Proc Calls)

machine:process	func name	value	percent
Total		7445.00	
poona:test2a	Atan2	7277.00	97.74
poona:test2a	calc1	55.00	*.**
poona:test2a	calc2	55.00	*.**
poona:test2a	sockstuff	55.00	*.**
poona:test2b	getData	1.00	*.**
poona:test2a	main	1.00	*.**
poona:test2b	main	1.00	*.**

A Sorted Table showing the number of times each procedure was called.

Figure 2.10.

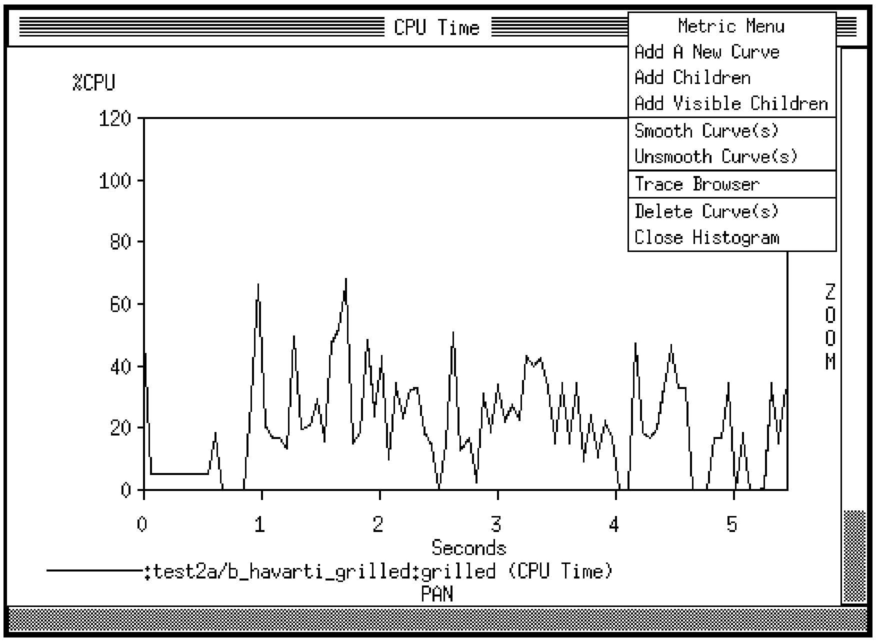
2.3.3. Histograms



Histogram property sheet with CPU Time metric selected.

Figure 2.11.

If the "**Display Histogram**" option is selected, a new property sheet similar to Figure 2.11 will appear in the lower left corner of the screen. This sheet is used to select which metric will be initially graphed. Only one item may be selected. Pressing the **Accept** button at the bottom of the sheet causes the histogram for the currently selected tree node to be built and displayed (see Figure 2.12).



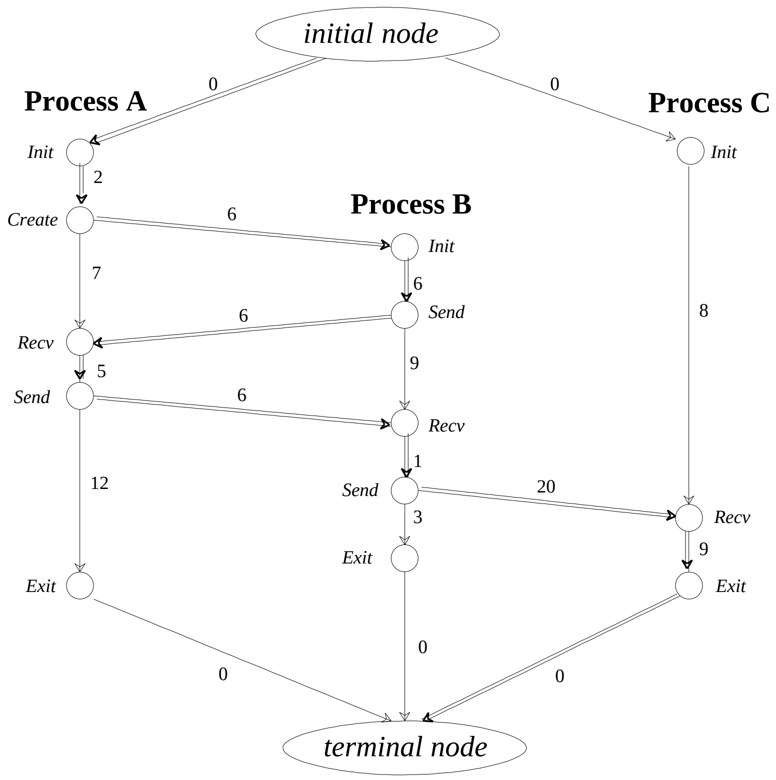
Sample CPU time histogram.
Figure 2.12.

To take a more detailed look at a particular interval of time, the zoom bar along the right side of the window is used. As you zoom in, the pan bar at the bottom of the window changes to indicate the percentage of the the total histogram displayed. This pan bar can also be used to change what time interval is currently visible.

The histogram banner menu contains many options to change displays. To add new curves to the histogram use the "**Add A New Curve**" option. This option causes the metric property sheet, shown in Figure 2.7, to re-appear. Any number of metrics may be added to the histogram. The program tree node selected at the time the **Accept** button is pressed is the one that will be used in creating the histogram. Curves may be deleted from the histogram by selecting the curve, and then choosing the "**Delete Curve(s)**" option on the banner menu. Curves are selected by pressing any mouse button while over the legend at bottom of the histogram. The other menu options are for advance features described in Section 3.2.

2.3.4. Critical Path Profile

Critical path is a unique feature for locating the sections of code in a parallel program that caused it to run slowly. Interactions between programs (messages passing, and locks) form a set of dependencies between processes in a program. Critical path analysis[3] involves constructing a directed acyclic graph of these dependencies (see Figure 2.13). Each arc in the graph is assigned a weight proportional to the amount of time consumed between the two points on the arc. For CPU time arcs, the length of the arc is time the program ran. For message arcs it is the time



A sample Program Activity Graph (PAG) for a three process application. The numbers along the arcs indicating arc weights. The thick arcs are on the critical path through the graph.

Figure 2.13.

required to send the message between processes. The longest time-weighted path though the graph is the critical path. At each level in the program tree, IPS can display the items that form the critical path, and their contribution to it.

To display the critical path for a particular level in the tree, select any node at that level, and choose the "**Display Critical Path**" item from the banner menu. This will create a critical path table similar to Figure 2.14 and Figure 2.15. The percentages shown in the right column of the critical path table indicate the percent of time that each item contributed to the critical path. Asterisks appear in the time column for items that contribute more than zero but less than one percent to the critical path. The purpose of this information is to guide the user in deciding what functions to modify to speed up the program's execution. Since it is possible that the second longest path through the graph may be just a bit shorter than the longest, reducing a component along the critical path might not result in a corresponding improvement in the application's run-time.

IPS provides a what-if mechanism to gauge the effectiveness of improving the performance of a function (or other critical path item). Any combination of items in the critical path may be selected, and their times set to zero. The critical path of the program is then recalculated without these items. To select an item for zeroing press any mouse button while over the item. Once all desired items have been selected, select the banner menu item "**Zero Selected Item(s)**" from *any* critical path window. This creates a "Context" table (see Figure 2.16). The first item in the table is the global context which is the context in which no items on the critical path are zeroed. Each

Program-level Critical Path

statistic	time	%time
TOTAL LENGTH	2.76	
CPU time	2.75	99.78
Inter-mach Msg	0.01	*.**

Program level critical path table.
Figure 2.14.

Procedure-level Critical Path

machine:process	func name	time	%time
TOTAL LENGTH		2.75	
havarti:test2a.swb	calc2	2.20	80.00
havarti:test2a.swb	sockstuff	0.22	8.00
havarti:test2a.swb	atan2	0.17	6.18
havarti:test2a.swb	main	0.08	2.91
havarti:test2a.swb	calc1	0.04	1.45
grilled:test2b.swb	getData	0.04	1.45

Procedure level critical path table.
Figure 2.15.

of the other contexts consists of one or more zeroed items. To compute the critical path for one of these contexts, select the context by pressing any mouse button while over it. Next select the level in the program tree, and the "**Find Critical Path**" banner menu item (from the program tree banner). This creates a critical path table for the selected context (see Figure 2.17). To destroy a critical path table use the "**Close Critical Path**" banner menu item in the desired window.

Context Table

Context	Zeroed Items	Phase
1	Global Context	*
2	havarti:test2a.swb (atan2)	*
	havarti:test2a.swb (calc2)	*

A CP context table with two items zeroed out.
Figure 2.16.

Procedure-level Critical Path (Context 2)

machine:process	func name	time	%time
TOTAL LENGTH		0.97	
grilled:test2b.swb	getData	0.90	92.78
havarti:test2a.swb	main	0.06	6.19
havarti:test2a.swb	sockstuff	0.01	1.03

A procedure level Critical Path for context 2.
Figure 2.17.

2.3.5. Standard Profiles

The banner menu can be used to display profile tables containing information about the total contribution that an item makes to the runtime of the program. Profile tables are similar to the tables created by the standard UNIX profiling tool *Prof*. Like critical path tables, profile tables can be created for each level in the program tree, and the level is specified by selecting any tree node at the desired level. Figure 2.18 and Figure 2.19 show sample profile tables for the program and function level respectively.

2.3.6. Gprof Profiles

Finally, the program tree banner menu can be used to display gprof profiles. These profiles show more detailed information than the prof tables and are similar to the tables created by the standard UNIX profiling tool *Gprof*. Figure 2.20 shows a process level gprof table. This table is obtained by selecting a process node from the program tree. Figure 2.21 shows a gprof table for a specific procedure. This table can be obtained either by selecting a procedure node from the program tree, or by highlighting a procedure in an existing gprof table.

Program-level Profile

statistic	time	%time
CPU time	3.68	62.37
CPU Sync Wait	2.22	37.63

Program level profile table.
Figure 2.18.

Procedure-level Profile

machine:process	func name	time	%time
havarti:test2a.swb	calc2	2.20	37.29
grilled:test2b.swb	Sync Wait (main)	1.20	20.34
grilled:test2b.swb	Sync Wait (getData)	1.00	16.95
grilled:test2b.swb	getData	0.91	15.42
havarti:test2a.swb	sockstuff	0.22	3.73
havarti:test2a.swb	atan2	0.17	2.88
havarti:test2a.swb	main	0.07	1.19
grilled:test2b.swb	main	0.06	1.02
havarti:test2a.swb	calc1	0.04	*.**
havarti:test2a.swb	Sync Wait (main)	0.02	*.**

Procedure level profile table.
Figure 2.19.

Program level - Gprof

parents <cycle> name children <cycle>	%time	self	descendents	called/tota called+self called/tota
main	100.00	0.00	3.68	0
a2		0.46	0.00	1/2
b2		0.23	0.34	3/11
c2		0.12	0.00	1/9
e2		0.11	0.00	1/2
--------------------	-----	----	-----------	-----------
<cycle 1 as a whole>	66.03	2.43	0.00	21
b2		1.19	0.00	10/11
c2		1.02	0.00	9/9
d2		0.22	0.00	2/2
--------------------	-----	----	-----------	-----------
d2 <1>				1
c2 <1>				7
main		0.23	0.34	3/11
b2 <total>	41.58	1.19	0.34	11
b2 <1>		1.19	0.00	10
a2		0.34	0.00	1/2
c2 <1>				8

Program level gprof table.

Figure 2.20.

Procedure level - Gprof

parents <cycle> name children <cycle>	%time	self	descendents	called/total called+self called/total
d2 <1>				1
c2 <1>				7
main		0.23	0.34	3/11
b2 <total>	41.58	1.19	0.34	11
b2 <1>		1.19	0.00	10
a2		0.34	0.00	1/2
c2 <1>				8
d2 <1>				2
cycle 1				

Procedure level gprof table.

Figure 2.21.

Each row in a gprof table corresponds to a procedure. A gprof table for a target procedure first lists parents (those procedures that call the target procedure), followed by the target procedure itself. If the target procedure is involved in a cycle[†], a row is then inserted giving more detailed information about the procedure's cyclic activity. Finally, the children (procedures called by the target) are listed. A process level gprof table simply displays the above information for each of its procedures.

The *name* column lists procedure names (indenting parents and children). If the procedure is involved in a cycle, the cycle number is given in brackets. The *%time* column shows the total time spent in this procedure and its children. The remaining fields have different meanings for parents, target, and children. For parents, the *self* column shows the amount of time spent in the

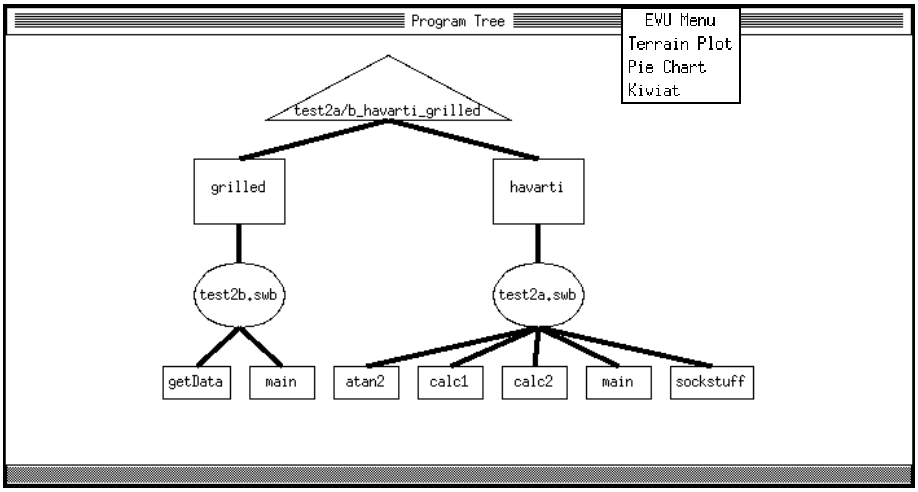
† A cycle occurs when one procedure calls another and at some point during that execution of the second procedure, the first procedure is called again.

target on behalf of the parent. For the target procedure, the *self* column shows the total amount of time spent it in the procedure. For children, the *self* column shows the amount of time spent it the child on behalf of the target. The *descendents* column for parents shows the amount of time spent in children of the target on behalf of the parent. For the target procedure, the *descendents* column shows the total amount of time spent in children of the procedure. For each child, the *descendents* column shows the amount of time spent in children of this child on behalf of the target. The *called* field for parents gives the number of calls to the target. If the parent and target are not involved in the same cycle, the total number times the target is called is shown under *total*. The total number of calls to the target is given in the target procedure's *called* field. However, if the procedure calls itself, these calls are totaled separately in the *self* field. The number of calls to each of the target's children is given in the child's *called* field. A child's *total* is the total number of times the child is called.

IPS's gprof gives more accurate results than the standard gprof because it uses trace-based computation as opposed to sample-based computation.

2.3.7. External Visualization Units

External Visualization Units are user programs that display performance information gathered by IPS-2. An External Visualization Unit is started the same way as a built-in display: select a tree node and then select the name of the desired EVU from the EVU menu. A typical EVU menu is shown in Figure 2.22.



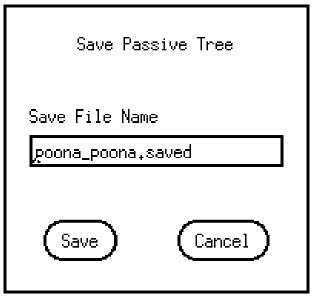
The EVU Menu.
Figure 2.22.

For information on how to write an EVU or how to tell IPS to include an EVU in the EVU menu, see section 5.

2.4. Leaving IPS

When the **Quit** item is selected from the banner menu, a property sheet similar to Figure 2.23 appears on the screen. This is to confirm that IPS should be terminated, and to permit the saving of the traces collected during the application's execution. If the **Save State** button is pressed, then the trace logs will not be deleted when IPS terminates. When an execution trace is saved, a new ips file is created containing the information required to re-process the trace logs. A property sheet similar to Figure 2.24 will appear to specify the name of this file. The section on passive mode explains how to use the save file to view previously collected data.

The Quit Confirmation Property Sheet.
Figure 2.23.

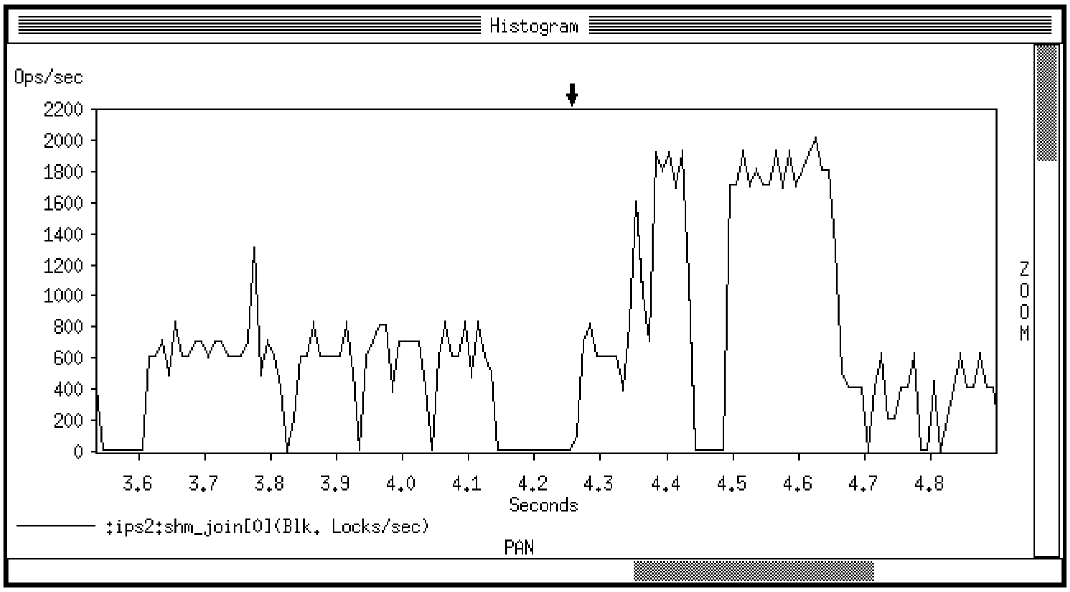


The Save file property Sheet.
Figure 2.24.

3. Advanced Features

This section describes additional features IPS provides for examining performance bottle necks in parallel programs, and customizing the way the master displays information. It assumes knowledge of the information in the tutorial section.

3.1. Phase Analysis



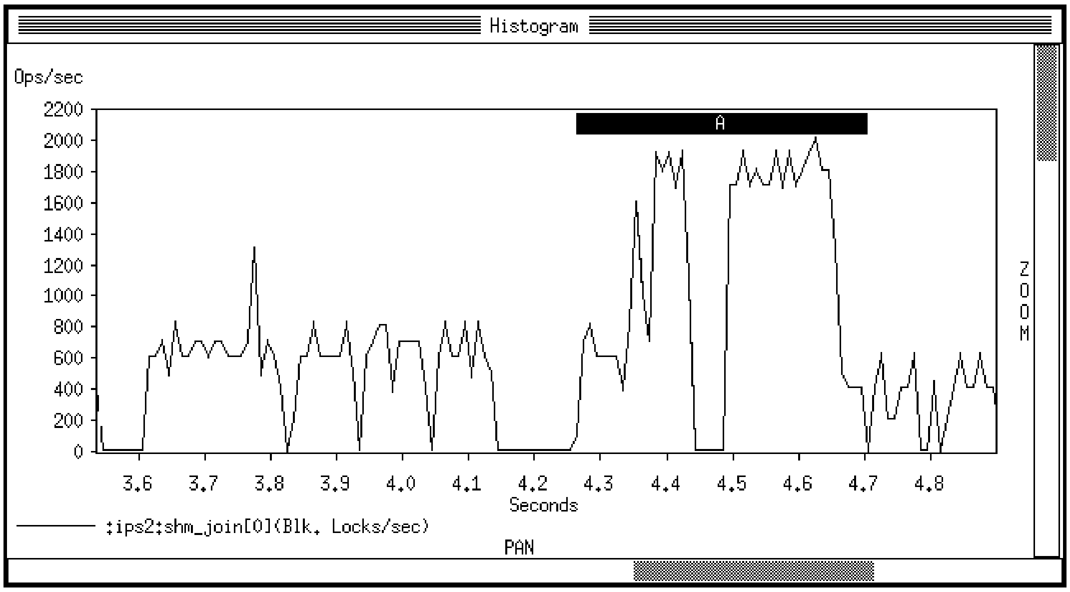
Histogram with single curve. The Arrow near the top of the histogram shows the start of a phase being defined.

Figure 3.1.

Phase analysis is a feature that IPS provides for examining the performance during an interval of a program's execution. A phase corresponds to a particular sub-part of a program's execution. For example, a program might consist of three phases: initialization, computation, and aggregation of results. Once a phase has been defined, any of the IPS metrics may be recomputed for only the phase interval.

To select a phase, press the left mouse button while the cursor is over one of the vertical lines. This will define one end of the phase. An arrow will appear at the top of the selected phase boundary (see Figure 3.1). If the first end of a phase has been selected and you decide not to define the phase, pressing the middle mouse button will abort the definition of the phase. To finish selecting a phase, press the left mouse button again. This creates a phase. The phase is indicated on the histogram by a horizontal bar (see Figure 3.2). In addition, a new tabular window similar to Figure 3.3 is created. The first entry in the table is the implicitly defined phase "*", which is the entire execution of the program. Each additional line corresponds to an explicitly defined phase, and the interval of time that it covers.

Once a phase has been defined, any of the other IPS tables may be created with information specific to that phase. To create a table for a specific phase, select the phase by pressing any mouse button while the cursor is over the desired item in the phase table. Next, select an item from the program tree banner menu. For example, figure Figure 3.4 shows a process level critical



Histogram with a single phase "A" defined.

Figure 3.2.

path table for the phase "A". This information is **not** a recalculation of the critical path for the interval, but instead shows the part of the overall critical path that occurred during the phase. The total length for a phase is the length of the critical path during that phase. The percentages in the third column of the table are computed using the total length shown in the first row as the denominator.

Phase Table

Phase	Start	End
*	0.00	5.95
A	4.26	4.70
B	3.54	4.26
C	4.70	4.90

A Phase table showing 3 user defined phases.

Figure 3.3.

Process-level Critical Path - Phase A

statistic	time	%time
TOTAL LENGTH	1.79	
CPU: topaz:shm_join #0	0.96	53.63
CPU: topaz:shm_join #1	0.30	16.76
CPU: topaz:shm_join #3	0.27	15.08
CPU: topaz:shm_join #2	0.23	12.85
Sync Wait: topaz:shm_join #0	0.02	1.12
Sync Wait: topaz:shm_join #1	0.01	*.**
Fork: topaz:shm_join #0->topaz:shm_join #1	0.00	*.**

A Process Level Critical Path Table for a single Phase of execution.

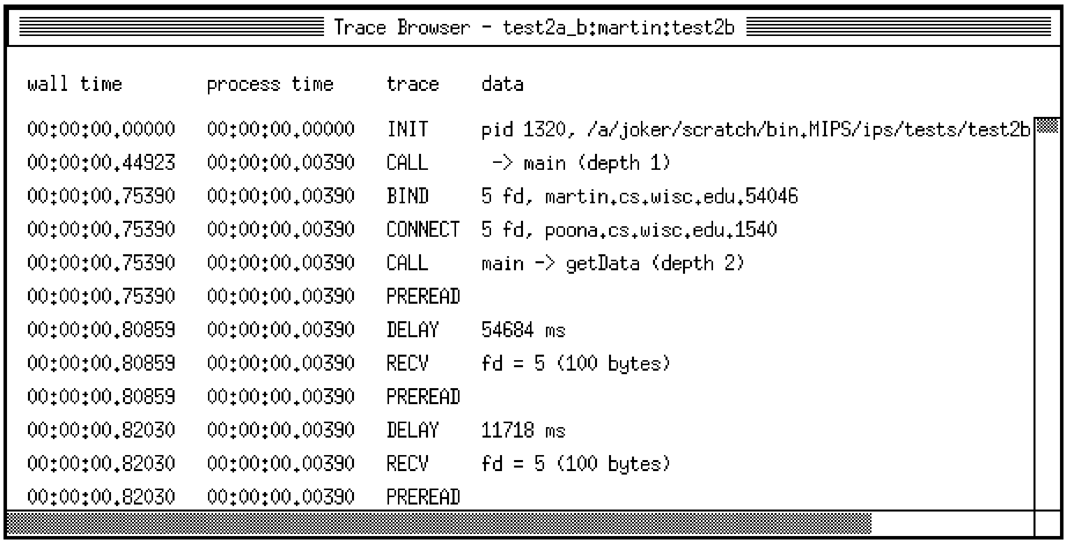
Figure 3.4.

3.2. Trace Browser

The traces created by ips2 can be viewed through trace browser windows. To view the trace file of a particular process, select that process node in the program tree window. To select which section of that process's trace file you wish to view, move the cursor over the desired area on the histogram window and press the right mouse button. This causes a trace hair to appear. The trace hair can be dragged to the desired location. After the trace hair is positioned selecting the **Trace Browser** option from the histogram banner menu will bring up a trace browser window.

Traces are displayed one per line. Each trace record contains a wall time, a (productive) process time, the trace record type, and record type specific data. Times are displayed in <hours>:<minutes>:<seconds> since the start of the application.

The contents of a trace browser window can be changed by two methods. First the user can add to the top or bottom of the display by selecting **Prepend Traces** or **Append Traces** respectively. In both cases half the traces displayed are removed and an equal number added. A second way in which to change the contents of the display is to select **Bind To Trace Hair** from the trace browser menu. This causes repositioning of the trace hair in the histogram window will update the trace browser. Reselecting **Bind To Trace Hair** will "unbind" the trace browser window from the trace hair. Figure 3.5.


Trace Browser - test2a_b:martin:test2b

wall time	process time	trace	data
00:00:00.00000	00:00:00.00000	INIT	pid 1320, /a/joker/scratch/bin.MIPS/ips/tests/test2b
00:00:00.44923	00:00:00.00390	CALL	-> main (depth 1)
00:00:00.75390	00:00:00.00390	BIND	5 fd, martin.cs.wisc.edu.54046
00:00:00.75390	00:00:00.00390	CONNECT	5 fd, poona.cs.wisc.edu.1540
00:00:00.75390	00:00:00.00390	CALL	main -> getData (depth 2)
00:00:00.75390	00:00:00.00390	PREREAD	
00:00:00.80859	00:00:00.00390	DELAY	54684 ms
00:00:00.80859	00:00:00.00390	RECV	fd = 5 (100 bytes)
00:00:00.80859	00:00:00.00390	PREREAD	
00:00:00.82030	00:00:00.00390	DELAY	11718 ms
00:00:00.82030	00:00:00.00390	RECV	fd = 5 (100 bytes)
00:00:00.82030	00:00:00.00390	PREREAD	

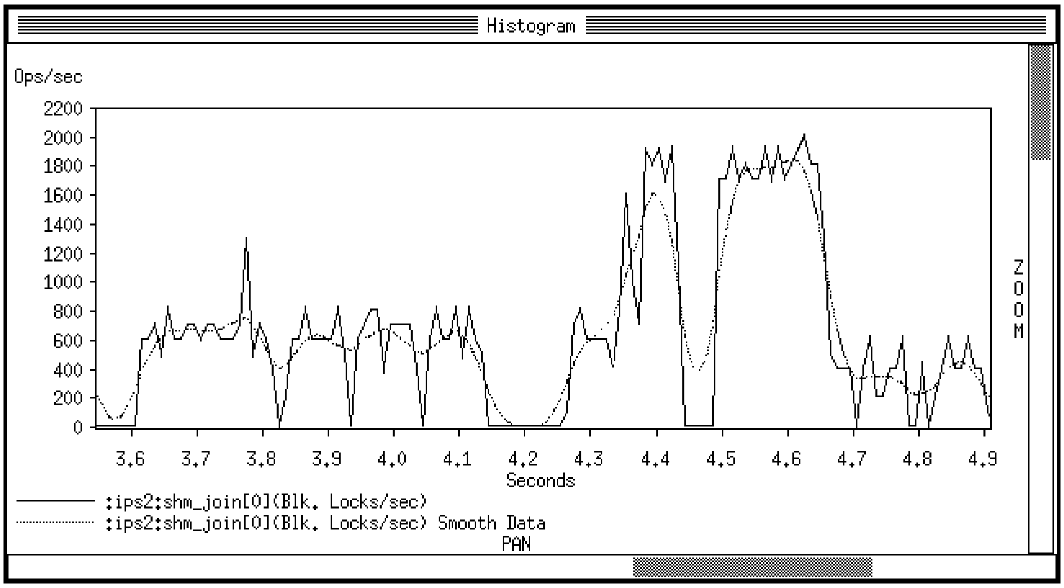
A Trace Browser Window.
Figure 3.5.

3.3. Smooth Data

One additional curve can be displayed from the histogram banner menu. The **Show Smooth Data** menu item creates a "smoothed" curve for each curve currently displayed on the histogram. This curve shows the effect of applying a low pass filter to the data, which removes spikes from the curve. A sample of this type of curve is shown in Figure 3.6.

3.4. Selective Node Hiding

This section describes automated techniques for hiding "uninteresting" nodes in the program tree window, based on resources used. Either the percent of the critical path or the percent of overall execution time may be used as the criteria to prune the nodes from tree. To prune the tree based on critical path, type an entry of the form *nn*`c` in the program tree window, where *nn* indicates the minimum percentage contribution a node must make to the critical path for it to be shown. Similarly, *nn*`t` only shows nodes that constitute at least *nn* percent of the total runtime of the program. If the cursor is over a node in the tree, then only nodes at that level in the tree and higher that meet the selection criteria are displayed. If the cursor is not over any node, only



A Histogram showing smoothed data.

Figure 3.6.

function level nodes are affected. Using the automated pruning techniques doesn't affect your ability to manually display or hide nodes in the tree. Any of the commands described in section 2.3 may still be used.

3.5. Listing Metrics and Units

To see a list of currently defined metrics and units, choose **Show Metrics** from the program tree banner menu. IPS-2 will present a list of metrics and units, giving their names and titles. The names are used as identifiers when editing metrics and building derived metrics, whereas the titles are used in menus and displays of the metrics. For each metric, the list indicates whether the metric is built-in (base), external, or derived. You may cut and paste names from the list when defining derived metrics. To cut a name from the list, position the mouse pointer over the name and double click the left mouse button. To paste the name, position the mouse over a property sheet text entry box and single click the middle mouse button.

3.6. Derived Metrics

IPS histogram displays (section 2.3.2) show how metrics vary over time. Internally, a metric's histogram is represented as an array of values (called buckets) with each value corresponding to an interval of time during the application program's execution. Derived metrics allow an IPS user to combine histograms with arithmetic formulas to create new histograms. A user may combine built-in, external, or other derived metrics to form new derived metrics and may view derived metrics in the same displays as built-in and external metrics. This section describes how to create derived metrics.

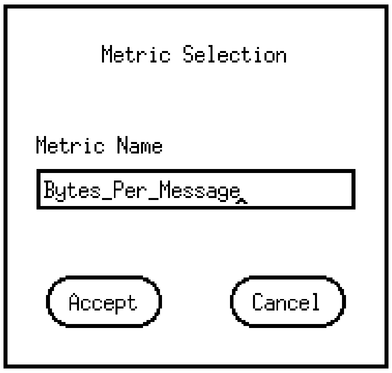
To create a derived metric, choose **Edit a Metric** from the program tree banner menu. A property sheet like the one in Figure 3.7 will appear. Enter the name of your new metric. The name must begin with a letter and contain only alphanumeric characters, underscores, and periods. Also, the name must not conflict with any existing metric.

A property sheet like the one in Figure 3.8 will now allow you to pick attributes for your new metric. The **Formula** attribute must be filled in, but the other attributes are optional. A metric formula consists of terms separated by the arithmetic operators `*,+,-,` and `/,` and it may specify precedence with parentheses.

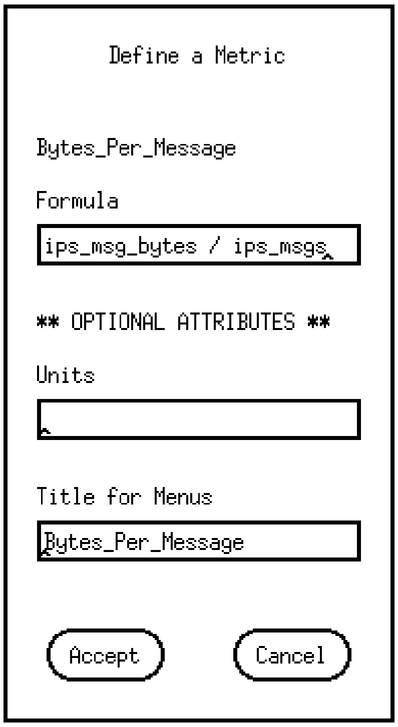
The terms of a formula may be scalar floating point values or vectors of floating point values. A histogram, indicated by a metric name, is one type of vector. Other vectors may be built up by combining vectors and scalars with arithmetic operators. All vector arithmetic is calculated on a bucket by bucket basis. There are also several functions that may be applied to a vector to produce a scalar. These include the **min**, **max**, **mean**, **sum**, and **stddev** functions.

Note that some of the base metrics may only appear as values in tables (i.e. they have no histogram representation). When these *tableonly* metrics are used in metric formulas they are considered to be scalar values. All other metrics are considered to be vectors and their histograms are used when computing derived metrics.

Figure 3.8 shows a metric formula that computes the number of bytes per message. The metric is computed by dividing the number of bytes by the number of messages bucket by bucket. The result is a new histogram containing these ratios.



Selecting a metric to edit
Figure 3.7.



Defining a derived metric
Figure 3.8.

Some base and external metrics have **scalefactor** attributes that cause their histograms to be scaled just prior to appearing in histogram displays. The scale factor is multiplied to each bucket of the histogram. For example, each bucket of the histogram for the metric `ips_cpu_time` contains the number of process time seconds used during the bucket interval. This representation is convenient for data collection and for calculating total process time, but is unintuitive if displayed directly in time graphs. To get a more intuitive histogram for display, `ips_cpu_time` has a **scalefactor** attribute of `100/bucketwidth`. By scaling each bucket by `100/bucketwidth` (`bucketwidth` is a pre-defined constant that is equal to the length of time represented by one bucket in a histogram) the master obtains a histogram that represents `ips_cpu_time` as percentage values.

Normally, it is illegal to combine terms in a derived metric formula if the terms have different scale factors. For example, a formula like this:

```
(ips_cpu_time / ips_processes)
```

is illegal because the metric `ips_cpu_time` has a **scalefactor** attribute of `100/bucketwidth` whereas `ips_processes` has no **scalefactor** attribute. If you want to derive a histogram containing the ratios of CPU time per process, you could instead create a metric with this formula:

```
(scale(ips_cpu_time) / ips_processes).
```

The **scale** function transforms `ips_cpu_time` into the intuitive form seen in histogram displays, allowing it to be combined with `ips_processes` in the derived metric formula.

The **Units** attribute is used to label the y-axis on histogram displays. You may use an existing unit by entering its name (Section 3.4 explains how to list unit and metric names). If no existing unit fits your new metric, then you may enter a new unit label -- you will be asked if you really want to define a new unit. If you leave the **Units** attribute blank, the master will examine the **Formula** attribute and attempt to choose a reasonable unit. If the formula is an additive combination of terms that have the same unit, then the master will use that unit for the new metric. Otherwise, the **Units** attribute will default to `ips_blank_units`.

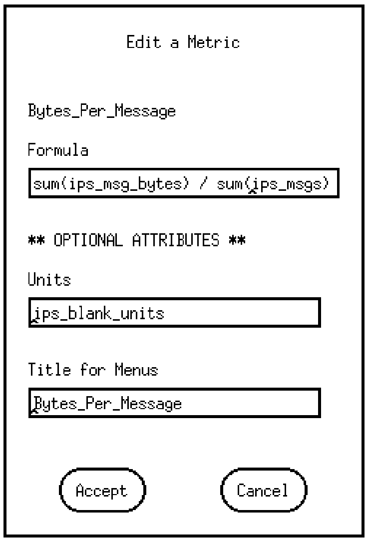
The **Title for Menus** attribute is used in menus and data displays. The title may be any string that does not include a double quotation character. You may want to set the **Title for Menus** attribute if you want a descriptive title for your metric.

3.7. Editing Metrics

If you wish to edit an existing metric, choose **Edit a Metric** from the program tree banner menu and enter the name of the metric you want to edit. Figure 3.9 shows the property sheet for editing metrics. We have changed the metric formula to calculate the total number of bytes sent in messages divided by the total number of messages. The result, a scalar, will be duplicated in all of the buckets in the resulting histogram.

3.8. Passive Mode

IPS normally runs an application and then collects the performance statistics. However, there are applications that can not be run from within IPS. IPS includes an option to run the application alone (with profiling linked in), and have it produce a trace log. IPS can then be run and will use the information collected from the execution of the program. To enable passive mode select the program level node in the program tree, and edit it. A property sheet similar to Figure 3.10 will appear. Press the **Passive** button, and then the **Accept** button. Next, select a process level node, and then select the **Edit** option from the banner menu. A property sheet similar to Figure 3.11 will appear. Enter the desired process identifier. Repeat this step for each



Editing a metric

Figure 3.9.

initial process in your application. It is not necessary to create process nodes for forked processes. IPS creates these nodes from information in the trace logs. Once all of the process ids have been entered, select the **Run** option from the banner menu. IPS then locates the trace files, and loads them. Once it finishes reading the trace logs, the program tree will contain function level nodes. After the trace logs have been processed, using IPS-2 in passive mode is identical to using it in active mode.

3.9. Controlling the size of trace logs

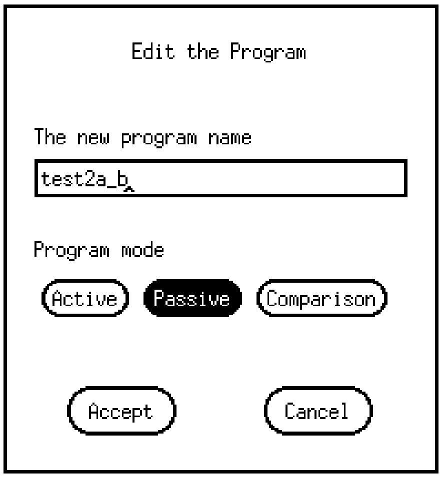
Depending on the number of functions called, and amount of synchronization in a program, IPS-2 can produce large amounts of trace information. There are several ways to minimize the amount of trace produced. A special compiler flag (`-pi0`) is provided that will turn off function level trace data for selected source files. All other IPS-2 tracing is still done, and the time consumed by the untraced function is "charged" to the calling function. If the calling function is also not traced, then the first function (working up the call stack) that is traced will receive the time.

3.10. Customizing Your IPS Environment

IPS includes a large number of options to customize the way in which it works. These options control the configuration and display of IPS. User's may set environment variables, change X resources, and specify window stacking translations to alter their environments. This section describes each settable option and explains how to change options to customize the IPS environment.

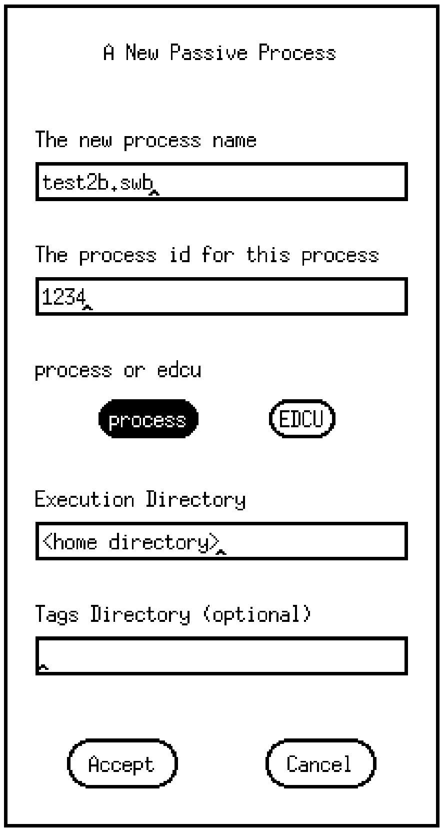
3.10.1. Environment Variables

Figure 3.12 shows the IPS environment variables. The `IMONDIR` variable specifies the directory in which the application trace files should be placed; the default is the user's home directory. `IPSDIR` specifies the directory in which the application should be run; the default is the user's home directory. `IPS_MAX_BUFFER` specifies the maximum amount of memory (in bytes) that the slave will use to buffer the trace log; the default is one megabyte.



Enabling Passive mode.

Figure 3.10.



A process level property sheet for passive mode.

Figure 3.11.

Variable	Function
`IMONDIR`	Directory to put trace files in
`IPSDIR`	Directory to run Application in
`IPS_MAX_BUFFER`	Maximum size of trace in slave memory at once
`IPS_TBSIZE`	Maximum size of trace buffer in application
`DISPLAY`	Which display should IPS Master Use
`XAPPLRESDIR`	Location of the IPS Application Defaults File

Environment Variables

Figure 3.12.

`IPS_TBSIZE` specifies the amount of memory reserved by the instrumentation library to store traces before flushing them to file; the default is 64 kilobytes. `DISPLAY` indicates the display on which the user interface should appear. `XAPPLRESDIR` specifies the directory containing the IPS X application resource file.

Any of the environment variables may be set in the user's `.cshrc` file or with the **setenv** command. The variables that control the behavior of the slave and instrumentation library must be set on the machines that run the user's application (and therefore the IPS slave).

3.10.2. X Resources

IPS uses many X resources, and many of them can be separately specified for each type of window. For example, the background of the title bar for critical path and profile tables can be made different colors. In X, resources are specified in a hierarchy from application, to widget type, to class, to subclass. The classes are listed in Figure 3.13. Most classes correspond to one of the types of displays that IPS can create. Subclasses correspond to particular parts of a display (e.g. the title or a scroll bar). The subclasses used in IPS are listed in Figure 3.14. For example, the title bar background color on the tree window is specified as `ips*tree*titleBar*background`.

Class	Display
`ips`	any IPS display
`ips*Tabular`	any tabular display
`ips*profTable`	CPU time profile tables
`ips*gprofTable`	CPU time gprof tables
`ips*cpTable`	Critical path profile tables
`ips*phaseTable`	Phase table
`ips*metricTable`	Metric table
`ips*zeroTable`	Critical path Zero Table
`ips*tree`	Program tree
`ips*traceBrowser`	Trace Browser tables
`ips*hist`	Histogram Display

IPS Widget classes
Figure 3.13.

Subclass	Use	Relevant Display
`data`	actual data being displayed	any
`graph`	data area (including. scales)	hist
`horizontal`	horizontal scroll bar	any
`titleBar`	display title	any
`vertical`	vertical scroll bar	table
`zoom`	zoom bar	hist

IPS Widget Subclasses
Figure 3.14.

The resources used by IPS are listed in Figure 3.15. `Accuracy` specifies the number of digits to the right of the decimal place to display in data tables. `Background` and `fore-ground` control the colors to use for the windows. `BorderWidth` specifies the number of pixels wide the border around a window show be. `Font` defines the name of the font to use in IPS. Although non-mono space fonts are permitted (and mostly supported), they have not been fully integrated into IPS. `MaxHeight` sets the limit (in pixels) on how tall a data table can get before scroll bars will be added. This is useful to limit the height of tables that contain lots of information, such as function level critical path tables or trace browser tables. `maxItem-sPerCol` controls the number of items in a column for a property sheet. If the property sheet has more than `maxItemsPerCol` items, a new column is created. `MaxWidth` limits the maximum width of data tables similar to the way `maxHeight` limits the height of tables.

`Place` is used to control whether IPS will automatically specify the initial *x* and *y* locations for newly created displays. By default IPS will try to place new displays in a "convenient" place. If this option is disabled control of window placement is passed to the window manager (which normally asks the user about placement). `placePopUps` controls the placement of pop-up displays such as property sheets. `placeWindows` controls the placement of all other windows. The `slave` resource indicates which UNIX executable file should be used as the slave analyst. `ColorLineStyles` and `BWlineStyles` are used to specify the color and style of the lines used in histograms. `ColorLineStyles` is only used on color displays. `BWlineStyles` is used on mono screens, and color screens when the banner menu item **Display Histograms in BW** item is selected. It is a list of style and color pairs. The number of items specified determines the number of different types of lines a histogram can show. The format is *<color name>:<style>*. Color name can be any named color for you machine. In addition, the pseudo color `foreground` makes the line the same color as the foreground of the histogram. The line style field is either the keyword *solid* for solid lines, or a comma separated pair of integers indicating the number of foreground pixels and the number of background pixels. For example, if we were using a color display and wanted the lines to be solid red, green, blue, and white the line resource would be:

`red:solid;green:solid;blue:solid;white:solid`.

Resource	Function	Default
`BWlineStyles`	types of lines for Histograms (Mono)	machine dependent
`ColorLineStyles`	types of lines for Histograms (Color)	machine dependent
`accuracy`	num of digits of accuracy to use	2
`background`	color of background	white
`borderWidth`	# of pixels to use in border	1
`font`	font to use for Text	system dependent
`foreground`	color of foreground	black
`maxHeight`	max height of tabular windows	screen height
`maxItemsPerCol`	max # of items per col. in a property sheet	13
`maxWidth`	max width of tabular windows	screen width
`motd`	file name for the IPS message of the day	/usr/public/ips/motd
`place`	place a particular class of windows	inherited from placeWindows
`placePopUps`	don't ask about property sheet placement	on
`placeWindows`	don't ask about window placement.	on
`slave`	which slave to run	/usr/public/ips/slave
`sysctl`	location of the file defining base metrics	/usr/public/ips/sysctl

X Resources
Figure 3.15.

3.10.3. Window Stacking

IPS manages screen real estate by arranging windows into stacks. You can use standard window manager routines to move, raise, iconify, and resize windows. However, IPS also provides built-in action routines to manage window stacks. The action routines allow each individual stack of windows to be managed separately.

The window stacking action routines are shown in Figure 3.16. *ShuffleUp()* causes the window on top of the stack to be placed on the bottom, and all the other windows are moved up one position. *ShuffleDn()* causes the window on the bottom to be placed on the top, and all the other

Routine	Action
ShuffleUp	Move all windows up one position
ShuffleDn	Move all windows down one position
StackTop	Move window to top of stack
RestackOne	Put window back into stack
RestackAll	Put all windows back into stack

Window stacking action routines

Figure 3.16.

windows are moved down one position. Neither ShuffleUp() nor ShuffleDn() will affect any window that has been moved away from the stack. *StackTop()* moves a particular window to the top of the stack. *RestackOne()* replaces a moved window back into the stack. *RestackAll()* replaces to the stack all windows that have been moved out of the stack.

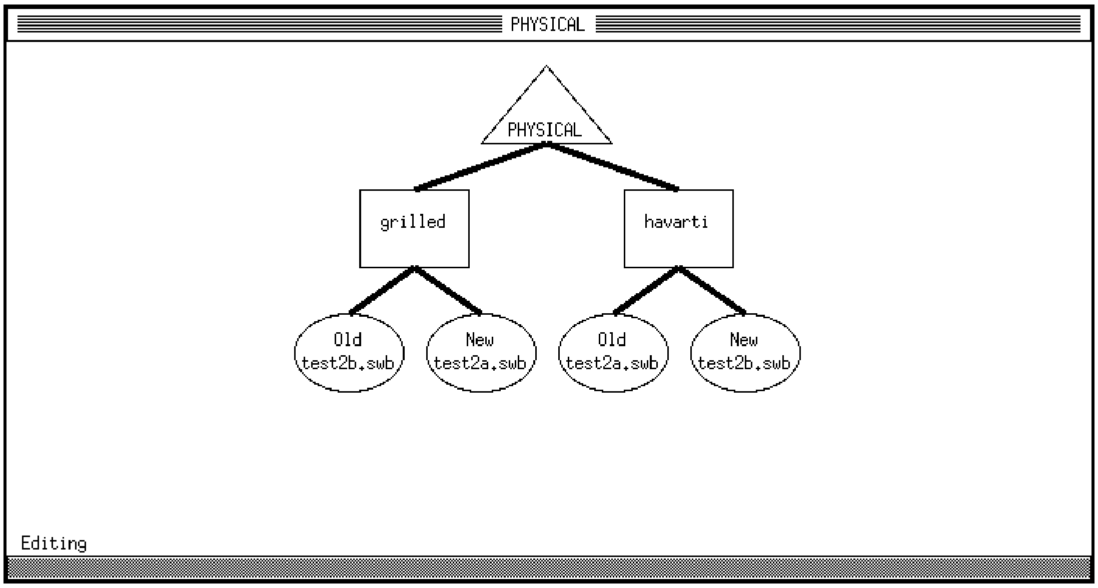
IPS statically defines which windows will be placed in which stack, but you can customize your X environment to specify which X event sequences will trigger which action routines. You simply need to replace the translation tables for the three widget classes that can be stacked: `ips*Tabular*titleBar`, `ips.tree.titleBar`, and `ips.hist.titleBar`. Default translations for these classes are defined in the X application resource file in the **XAPPLRESDIR**, and you can define replacement translations in your `.Xdefaults` file.

4. Multiple Program Trees

IPS allows users to analyze multiple programs in a single measurement session. This feature is useful for workload studies and allows multiple versions of the same program to be compared at once. To study multiple programs, the user creates multiple program trees. This section describes how to create multiple program trees, how to compare multiple programs in tables and graphs, and how to compare multiple versions of the same program.

4.1. Creating multiple program trees

To create additional program trees, choose **Add a Tree** from the program tree menu. IPS will prompt for the name and mode of the new tree. If you wish to run a multiple program workload, you should choose **Active** for the mode of the new tree. If you are replaying a multiple program run, then choose **Passive** mode. If you are comparing an old run with a new run of the same program, then you should choose **Comparison** mode. Comparison mode is further explained in Section 5.2.



Sample physical tree. The two logical trees (Old and New) are not shown.

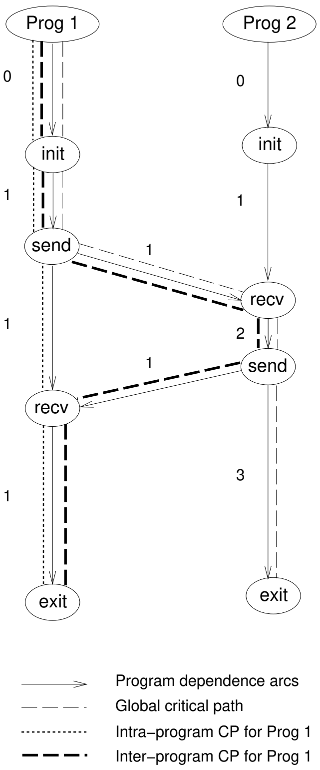
Figure 4.1.

Whenever you create multiple trees in IPS, an additional tree is automatically created for you. This tree is distinguished from other trees by the name **PHYSICAL**. Other trees are called *logical* trees. The physical tree is an alternative representation of the logical trees -- it contains all of the machines and processes defined in all of the logical trees. In the physical tree, a machine node will only appear once even though that machine might be used by more than one program. Process nodes in the physical tree indicate the program to which they belong. Because the physical view does not contain any information that is not already in one of the logical trees, you may not add or delete nodes in the physical tree. Figure 4.1 shows a sample physical tree.

As you create additional logical trees, fill them out with machine nodes and process nodes as normal. As you edit logical trees, IPS updates the physical tree automatically. When you are ready to run the programs (or replay passive mode traces), choose **Run** from any of the program tree banner menus. IPS will return after all of the traces (and external histograms) are processed.

4.2. Analyzing multiple programs

All of the standard features of IPS work with multiple program trees. In particular you can select a node from any tree and select the type of display you wish to see. By selecting a node in a logical tree you will create displays of information about only the program contained in the logical tree. By selecting nodes in the physical tree you can create displays that include data from all of the programs.



Sample PAG showing three types of critical path -- note that the processes are in two different programs

Figure 4.2.

If your programs have dependencies between them (e.g. they exchange messages) then paths in the program activity graph may cross from one program to another. In this case there are three types of critical path displays that can be examined. *Global* critical path analysis considers paths through all of the programs. To see a global critical path, choose nodes in the physical tree and select **Display Critical Path Table** from the banner menu. The second type of analysis, called *intra-program* critical path, considers only those paths that are completely contained within a particular program. In this case, inter-program arcs of the PAG are ignored. To display an *intra-program* critical path, select nodes in the logical tree for the program that you want to isolate. The third type of critical path analysis, called *inter-program* critical path, considers only those paths that begin and end in a particular program. Intermediate arcs of the inter-application critical path, however, may be in any of the programs. Inter-program critical path displays are not yet available in IPS. Figure 4.2 shows the differences between global, intra-program, and inter-program critical paths for a particular set of programs.

4.3. Comparison Mode

Comparison mode is similar to passive mode in that old traces are replayed for analysis. However, IPS will automatically shift the timeframe of all comparison mode programs so that histogram curves align with those of the active and passive mode programs. In effect, it appears that all of the programs ran at the same time. All of the other features of IPS operate normally with comparison mode applications.

To study multiple versions of a single program, create an IPS program tree for each version of the program. All but one of the trees should be in comparison mode. The remaining tree can be in active or passive mode. After the programs have been run and the traces have been collected, you can include data from all of the versions of your program in the same displays.

Appendix 1: Manual Pages

The following manual pages have been included: *editor(1)*, *iocollect*, *itags(1)*, *monEDCU(1)*, *terrain(1)*, and *vmcollect(1)*.

NAME

editor – IPS-2 Editor EVU

SYNOPSIS

Editor (select Editor in the EVU Menu)

DESCRIPTION

editor is an External Visualization Unit for IPS-2 to see the source code of the traced program. It provides the user the possibility to select a procedure level node of the tree in the IPS-2 main window to get an xterm window running vi(1) with the cursor at the beginning of the selected procedure. The information required to locate the tags file (machine, login name, directory) are taken from the IPS master. If the Tags Directory field has not been specified by the user during the edit session or in the description file, editor pops up a property sheet similar to IPS to get appropriate information. If you expect the editor as well with the main() function, you should use the itags(1) tool to fix your tags file.

SEE ALSO

vi(1), itags(1), ctags(1)

BUGS

Various. The user is expected to run either a csh or a tcsh on the local and remote machine. The path for xterm and vi on the remote machine has to be properly set. The .rhosts on the remote machine must contain an entry for the local host. The remote machine must be enabled by xhost to pop up a window on the local machine. The DISPLAY environment variable is specified trough hostname(1), which works only well in the same Internet domain. The itags(1) program is an inconvenient workaround. Errors are reported to standard error. There is no integrated error management for EVUs yet. Does not detect absence of tags file.

AUTHOR

Joerg Micheel - UW Madison, IPS-2 Project

NAME

iocollect – IPS EDCU interface to "iostat" data

SYNOPSIS

iocollect [**–ct**] [**–usni**] [**–IO**] [*interval*]

DESCRIPTION

iocollect is a simple EDCU front-end to the **iostat** Unix utility that reports on IO activity. **iocollect** iteratively collects and processes data on CPU utilisation and terminal IO activity. The data collected is fed into histograms and can be viewed via the IPS interface. Running **iocollect** by itself is not very interesting.

OPTIONS

activity class options default to nothing. Therefore one of more of the command line options must be specified if the EDCU is to be of any use.

–c Report the percentage of time the system has spent in user mode, in user mode running low priority processes, see in system mode, and idling. This implies independently specifying the four CPU sub-classes.

–t Report the number of characters read and written to terminals. This option implies independently specifying the two terminal IO sub-classes.

–inus Report the percentage CPU time spent by the system in (1) idle (2) nice (3) user and (4) system mode rspy. These sub-class options provide more control over the metrics collected than the CPU class option.

–IO Report on the rate of character (1) input and (2) output activity to and from all terminals in the system rspy.

interval Collect and dump data once each *interval* seconds. Default is 1 second. Currently, this interval is restricted to an integer (hence 1 second is the minimum). Negative values here will lead to no data collection at all.

FILES

/dev/kmem

/vmunix

SEE ALSO

vmstat (8), vmcollect (1), iostat (8).

NAME

itags – fix tags file for IPS-2 Editor EVU

SYNOPSIS

itags

DESCRIPTION

is a utility to fix the special treatment of main() functions by ctags(1). It creates an appropriate entry in the tags file. You should run itags once after creating the tags file with ctags. No arguments are required unless you choose a different name for the tags file. itags complains about more than one main function entry in the tags file.

SEE ALSO

ctags(1), editor(1EVU), IPS-2 Users Guide

BUGS

No known ones.

AUTHOR

Joerg Micheel - UW Madison, IPS-2 Project

NAME

monEDCU – kernel statistics EDCU

SYNTAX

monEDCU [*interval*] [**–perCPU]**

DESCRIPTION

The *monEDCU* EDCU collects statistics on processes, memory, and the file system. The statistics are gathered by reading kernel memory, so the user must have read privileges for /dev/kmem. If an *interval* argument is specified, then *monEDCU* will collect statistics every *interval* seconds. Otherwise, *monEDCU* will collect statistics once per second. If the *perCPU* switch is specified, then the per CPU metrics are also collected (multi-processor systems only).

METRICS COLLECTED

System Time (system_time) -- Time the system spent in system mode.

User Time (user_time) -- Time the system spent in user mode.

Total Time (total_time) -- Sum of system_time and user_time.

Number of Procs (num_proc) -- Number of processes in the system (occupied process table entries).

Number of Running Procs (num_running) -- Number of processes currently running.

Number of Runable Procs (num_runable) -- Number of processes that are runnable and waiting on the run queue to be dispatched.

Processes Fast Waiting (num_fastwait) -- Number of processes that are waiting on disk I/O, page I/O, or a kernel resource.

Processes Sleeping (num_sleeping) -- Number of processes that are sleeping.

Processes Swapped Out (num_swapped) -- Number of processes that are swapped to disk.

System Call Rate (syscall_rate) -- Number of system calls per second.

Context Switch Rate (cs_rate) -- Number of context switches per second.

Interrupt Rate (intr_rate) -- Number of interrupts per second.

Trap Rate (trap_rate) -- Number of traps per second.

Forks (num_forks) -- Number of forks per second.

Vforks (num_vforks) -- Number of vforks per second.

Execs (num_execs) -- Number of execs per second.

Packets In From Network (pkts_in) -- Number of packets received per second.

Packets Out To Network (pkts_out) -- Number of packets transmitted per second.

Disk Operations (disk_ops) -- Number of disk transfers per second.

Disk I/O (Kbytes) (disk_bytes) -- Number of kilobytes transferred per second.

Tty Characters Input (tty_in) -- Number of characters received on TTY lines per second.

Tty Characters Output (tty_out) -- Number of characters transmitted on TTY lines per second.

Page Faults (vm_faults) -- Number of page faults per second.

Page Reclaims (vm_rec) -- Number of page reclaims from any free list (minor faults) per second.

Dirty Pages Reclaimed (vm_dirty_rec) -- Number of page reclaims from the dirty list per second.

Zero Pages Created (vm_zero_pages) --

Pages In Transit (vm_page_intrans) --

Code Page Ins (vm_code_in) --

Pages Paged In (vm_pgs_pg_in) -- Number of pages paged in per second.

Page Ins (vm_pg_ins) -- Number of page ins (major faults) per second.

Page Outs (vm_pg_outs) -- Number of page outs per second.

Pages Paged Out (vm_pgs_pg_out) -- Number of pages paged out per second.

Swap Ins (vm_swp_ins) -- Number of processes swapped in per second.

Pages Swapped In (vm_swp_pgs_in) -- Number of pages swapped in per second.

Swap Out (vm_swp_outs) -- Number of processes swapped out per second.

Pages Swapped Out (vm_swp_pgs_out) -- Number of pages swapped out per second.

VM Deficit (vm_deficit) --

Free Memory (vm_free) -- Free memory in Kbytes.

Dirty Memory (vm_dirty) --

Total VM Of All Processes (vm_total) -- Sum of virtual sizes of all processes.

Real Memory (vm_real) -- Sum of resident sizes of all processes.

Active Virtual Memory (vm_avm) -- Sum of virtual sizes of all processes that have executed in the last 20 seconds.

Disk Blocks Read (fs_blks_read) -- Number of actual disk reads initiated by the file system.

Disk Blocks Written (fs_blks_written) -- Number of actual disk writes initiated by the file system.

Disk Cache Read Hits (fs_read_hit) -- Percentage of file system read requests that were satisfied from the buffer cache.

Disk Cache Write Hits (fs_write_hit) -- Percentage of file system write requests that did not result in an actual disk write.

File Locks In Use (lock_used) -- Number of record/file locks currently in use.

Pct. Of Avail. File Locks Used (lock_pct_used) -- The percentage of available record/file locks currently in use.

Locked Files (lock_num_locked) -- Number of file headers currently in use. This may also be considered the number of locked files.

PER-CPU METRICS

Note: the perCPU switch must be enabled for these metrics.

CPU Time Processor #<id> (cpu_time_<id>) -- User time for processor *id*.

BUGS

Some of the statistics are not really gathered by the kernel, and their values are zero. It is impossible to tell which statistics are actually zero and which are just not gathered.

SEE ALSO

edcu(3), vmcollect(1), histogram(5), iostat(1), monitor(1), vmstat(1)

AUTHOR

Jeff Hollingsworth - UW-Madison, IPS-2 Project

NAME

terrain − Produce terrain plots from IPS data.

SYNOPSIS

terrain [–iconic] [–display host:display] [+rv] [-rv] [–bg color] [–fg color] [–fn font] [–name string] [–title string] [–xrm]

DESCRIPTION

Terrain is an **EVU** (External Visual Unit) of **IPS** which provides a three dimensional performance data visualization for **IPS**. It draws a terrain plot according to the performance data of a pre-selected node's children. It also allows users to manipulate the terrain plot in different ways, such as viewing from different angles and smoothing with mean or median.

Creating a terrain plot is a three step process. First select a program tree node which has more than one child nodes in **IPS Program Tree** by pressing the middle mouse button while over that node. The second step is to pop up the **EVU** menu by pressing the middle mouse button in the program tree banner and selecting the **Terrain Plot** option in the menu. Then a **Property Sheet** will appear on the screen. The third step is to select a metric to be plotted from the **Property Sheet**. After a metric is selected, a terrain plot for the children of the currently selected tree node will be built and displayed.

OPTIONS

–iconic This option cause the terrain window to iconize when **terrain** starts up.

+rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–rv This option turns off reverse video.

–display host:display
This option specifies the X server to contact.

–bg color This option specifies the color to use for the background of the window. The default is white.

–fg color This option specifies the color to use for the foreground of the window. The default is black.

–fn font This option specifies the font to be used for displaying normal text.

–name string This option specifies the icon name for **terrain**'s windows.

–title string This option specifies the title for **terrain**'s windows.

–xrm resourcestring
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

MAIN WINDOW

The main window consists of a banner, two scroll bars, two rotation indicators, and the main viewer where the terrain is plotted.

The two scroll bars are used to control the rotation of the terrain. The vertical scroll bar at the right end of the window is used to control the vertical rotation of the terrain. The angle of rotation is relative to viewing the terrain from top and ranges from 0 to 180 degrees. The horizontal scroll bar at the bottom of the window is used to control the horizontal rotation of the terrain. The angle of rotation ranges from 0 to 360 degree and is relative to the terrain's x-z plain which is defined by the time axis and the metric axis.

There are three ways to rotate the terrain using the scroll bars. Pointing the mouse cursor at a particular position in either scroll bar and pressing the middle mouse button once will cause the terrain to immediately rotate to the angle corresponding to the mouse cursor position; Pressing the left or right button when the mouse cursor is in the vertical scroll bar will cause the terrain to rotate 10 degrees upward and downward respectively, and pressing the left or right button when the mouse cursor is in the horizontal scroll bar will cause the terrain to rotate 20 degrees towards left and right respectively; Pressing the middle mouse button and then moving the mouse cursor without releasing the mouse button will cause the terrain to rotate continuously according to the movement of the mouse. At the same time, the original detailed color terrain will be replaced by a black and white outline version for speed. Once the mouse button is released the rotation will stop and the original terrain will appear at an angle corresponding to the final position of the mouse cursor.

There are two rotation indicators between the banner and the main viewer. They show the current angle of rotation of the terrain in both vertical and horizontal dimensions.

POPUP MENU

The main menu can be invoked by pressing the left mouse button while the mouse cursor is in the banner. An option can be selected by releasing the mouse button while the mouse cursor is over the option. The following are the options currently available:

Display All Children
Show a terrain which includes all children of the selected node in the program tree.

Display Visible Children Only (default)
Show a terrain which includes only the visible children of the selected node in the program tree. Note that hiding a child in the program tree alone will not update the terrain. This option needs to be selected in order to update the terrain.

Show Raw Terrain (default)
Show the raw terrain without any smoothing effects.

Show Smooth Terrain
Show the cooked terrain which is smoothed by either mean or median.

Use Mean When Smoothing (default)
Use mean as the algorithm for smoothing.

Use Median When Smoothing
Use median as the algorithm for smoothing.

Quit Terrain Cause **terrain** to exit.

FEATURES

Once **terrain** is started, the selected node will be memorized and will not be affected by other node selections in the program tree. Although this prevents users from viewing different nodes within one **terrain** session, this enables users to run multiple sessions simultaneously with different nodes.

LIMITATIONS

It is difficult to specify an exact angle of rotation as the scroll bars are the only way to specify angles.

The degree of smoothing is currently hard-wired. An adjustable smoothing effect will be more desirable.

Number of points along the **time** axis is currently constrained by the size of the window and cannot be adjusted.

The selected node must have at least two children in order to produce a terrain plot.

Terrain must be invoked through **IPS EVU Menu**. It will print an error message and abort if it is invoked in any other way.

BUGS

If the selected node does not have enough children to produce a terrain or no node is selected when invoking **Terrain Plot** from the **EVU Menu**, **terrain** will print an error message to stdout and abort.

If the "**Display Visible Children Only**" option is selected from **Terrain Menu** and the selected node does not enough visible children to produce a terrain, a terrain with all children will be displayed and no error message will be printed.

Data transfer from **IPS** to **terrain** will freeze the IPS main window. This normally happens after metric selection and can last for a minute.

AUTHOR

Chi-Ting Lam - UW-Madison, IPS-2 project.

ACKNOWLEDGEMENTS

Terrain is derived from **GNUPlot 2.0**. The following is a list of the authors of **GNUPlot**.

Thomas Williams, Colin Kelley - original gnuplot
Russell Lang, Dave Kotz, John Campbell - gnuplot 2.0 additions
Gershon Elber - gnuplot 2.0 3-D extensions
Chris Peterson (MIT) - original Xlib gnuplot support
Dana Chee (Bellcore) - mods to original support for gnuplot 2.0
Arthur Smith (Cornell) - graphical-label-widget idea (xplot)
Hendri Hondorp (Univ of Twente, Netherlands) - Motif xgnuplot
Ed Kubaitis (UIUC) - X11 terminal driver for gnuplot 2.0

NAME

vmcollect – ips virtual memory statistics collector

SYNTAX

vmcollect [*interval*] [–**switch] ...**

DESCRIPTION

The *vmcollect* EDCU collects statistics for processes, virtual memory, disk, trap, and cpu activity by filtering output from the system utility, *vmstat.*

If an *interval* argument is specified, then vmcollect will collect statistics every *interval* seconds. Otherwise, *vmcollect* will collect statistics once per second.

Vmcollect monitors system statistics according to its command line *switches.* The switches are as follows:

all all known statistics

r number of processes in run queue

b number of processes blocked for resources (i/o, paging, and so on)

w processes that are runnable or short sleeper (< 20 seconds) but swapped

avm number of active virtual pages

fre number of pages in the free list.

di number of dirty pages. (Sequent only)

re page reclaims (simulating reference bits) per second

rd dirty page reclaims per second (Sequent only)

at pages attached (found in free list not swapdev or filesystem) per second (DECstations only)

pi pages paged in per second.

po pages paged out per second.

fr pages freed per second (DECstations only)

de anticipated short term memory shortfall (pages/sec)

sr pages scanned by clock algorithm per second (DECstations only)

di disk operations per second. This is normally reported separately for each disk.

in (non clock) device interrupts per second

syc system calls per second

cs cpu context switches per second

us user time

syt system time

id idle time

SEE ALSO

edcu(3), histogram(5), iostat(1)

Appendix 2: Error Messages

This is a **partial** list of error messages in the IPS-2 system. We are still working on compiling this list. Each error contains the line that would be produced by IPS-2 in its error window, and a short description of the error.

unimplemented function (100)

This error indicates an incompatibility between the Slave Analyst and the User Interface. This will most likely occur when the User Interface has been upgraded to a new release and the Slave Analyst has not.

no such host (101)

An internal programming error in the Master Analyst has caused it to request data for a nonexistent host.

no such process (103)

An internal programming error in the Master Analyst has caused it to request data for a nonexistent process.

no such function (104)

An internal programming error in the Master Analyst has caused it to request data for a nonexistent procedure.

connection lost (105)

The Master Analyst lost its connection to one or more of the slave analysts. This can occur due to network failures, hosts failures, or the Slave Analyst dying.

couldn't find trace file (106)

The Slave Analyst could not find the trace file for the named process. This can happen in passive mode if the trace file was deleted. Trace files consist of two files per process (imon.<pid> and imonaux<pid>). It can also happen in active mode if the application program did not terminate normally. (For example the application could have dumped core).

trace file is not in current format, recompile application (107)

The Slave Analyst is expecting the trace logs to be in one format and the application program has produced traces in another format. This is most likely to occur when the Slave Analyst has been upgraded to a new version, and either the instrumentation library has not been upgraded or the application program has not been recompiled.

Unable to process trace log (108)

The application program was aborted during its execution, and the resulting trace logs did not contain enough information to process.

couldn't create application process (109)

The Slave Analyst was unable to fork a new process for an EDCU or an application process. This can happen if on the machine running the Slave Analyst, the per user process limit has been reached, the total process limit for the machine has been reached, or there is insufficient swap space for the new process.

couldn't start application (110)

The Slave Analyst was unable to start the application process or EDCU. This could result from a bad file name for the process, or file permission problems for the application process.

malloc failed (111)

The Slave Analyst ran out of memory. This could be due to a lack of swap space or the swap space could be full due to other programs running on the system. For parallel Slave Analysts (Sequent Symmetry) data is not put into the swap partition, but is put into shared memory which swaps from the user file system. Normally this swapping area is in the /tmp partition, but this can be controlled by setting the TMPPATH environment variable.

EDCU error (112)

An error was encountered in processing the named EDCU's trace data.

no such metric (113)

The Master Analyst requested a metric that the Slave Analyst doesn't know about. This represents a serious programming problem in the Master Analyst. We would appreciate being informed if this error occurs.

invalid directory (114)

The supplied execdir or IMONDIR directory does not exist, or you do not have permission to access it.

<other message> (115)

An undefined error message. The contents of the error message should be self explanatory.

Slave Internal Consistency Failure. (200)

The data in the Slave Analyst is inconsistent. This error indicates a programming error in the Slave. We would appreciate being informed if this error occurs. Please write down the file and line number to help us track down the problem.

Network Communication Error in Backward Slack (201)

An operation during inter-process communication failed. This is a serious programming error in the slave.

Could not start slave analyst on <hostname> (202)

An error occurred when trying to start the slave analyst on the machine <hostname>. This can result from the host running the user interface not being in the .rhosts file for the host you are trying to start the slave analyst on. It also can result if the wrong name was entered for the host (For example you might need to use the fully qualified Internet Domain Name for the host). It can also occur if the user name supplied for the host is incorrect.

Master Internal Consistency Failure. (203)

A programming error in the master analyst was detected. Would we appreciate hearing if this error occurs. Please write down the file and line number so we can track down the problem.

Could not open system control file (204)

Master could not open system control file.

Could not open %s (205)

When parsing system control file (sysctl), the master encountered an error. In particular, there was an error while running the preprocessor over the file.

NEWIPSMETRIC: Too many metrics (206)

An internal programming error caused the master to attempt to allocate too many metrics.

NEWIPSMETRIC: REALLOC failed!! (207)

The master could not allocate any more space for metrics. The master probably ran out of memory.

FreeBin: Attempt to free NULL Bin array! (208)

An internal programming error caused the master to attempt to free a NULL histogram.

FreeBin: Negative Reference count for Bin Array! (209)

An internal programming error caused the master to attempt to free a histogram that had already been freed.

Error with metric ... (210)

The master found an inconsistency with the indicated metric. The error message should indicate the nature of the inconsistency.

Illegal Frame passed to Change Frame Menu (211)

An internal programming error caused the master to try to update a menu of an unknown histogram frame.

Cannot handle rogue metric (212)

The master could not place an external histogram in the program tree because the metric is not a program level metric. This happens when an EDCU collects a metric for a machine that is different from the one it ran on.

Cannot handle rogue metric (213)

The master could not place an external histogram in the program tree because the metric's tree location does not match the current program tree. This is usually caused by a bug in the External Data Collector.

Internal Error (Indicator in RelHist) (214)

An internal error was encountered in trying to resolve a metric name for a derived metric formula.

UnOpEval: Unknown unary op (215)

An internal error caused the master to try to resolve a derived metric formula with an unknown unary operator.

UnOpEval: Unknown op type (216)

Internal Error: Master tried to resolve a derived metric formula by applying a unary operator to an unknown operand.

DMetEval: (Bad node_kind in Dmet) (217)

Master cannot evaluate derived metric formula.

BuildTable - RequestInput failed (218)

Internal error detected in master when a property sheet failed.

Cannot define any more metrics (219)

You may not define any more derived metrics in this session. To define more metrics, save the session and restart it in passive mode.

Error in NewFrame (220)

An internal error caused the master to fail to find any unused histogram frames.

Error in NewHist (221)

An internal error caused the master to fail to add a curve to a histogram frame.

Illegal Frame passed to Destroy Frame (222)

An internal error caused the master to fail to destroy a histogram frame.

Bad argument passed to AddHistogram (223)

Internal Error. The bad argument could have been non-existent frame or NULL data.

Bad arg. passed to DeleteHistogram (224)

Internal Error. The bad argument could have been non-existent frame or a non-existent curve in the frame.

Bad argument passed to MakeFuzzyVisible (225)

Internal Error. The bad argument could have been non-existent frame or a non-existent curve in the frame.

Bad argument given to MakeFuzzyInvisible (226)

Internal Error. The bad argument could have been non-existent frame or a non-existent curve in the frame.

Could not send process (227)

The master could not send a process to a particular slave. Probably due to some inconsistency between master and slave revisions.

Could not send EDCU (228)

The master could not send a process to a particular slave. Probably due to some inconsistency between master and slave revisions.

Running the application (229)

Master has detected that the slave has failed while running the application program.

Error on line %d of file '%s'. (%s) (230)

Parsing error: The master encountered an error while parsing a script file.

NEWUNIT: Too many units (231)

Cannot allocate space for more units.

fcntl SETF ASYNC (232)

A fcntl call failed when the master tried to enable asynchronous processing of X events.

fcntl clear ASYNC (233)

A fcntl call failed when the master tried to disable asynchronous processing of X events.

Error: run.c Grey tile not large enough. (234)

Master encountered a problem while trying to grey the background of the tree window.

could not create a grey Pixmap (235)

Master encountered a problem while trying to grey the background of the tree window.

malloc failed (236)

Master probably ran out of memory.

realloc failed (237)

Master probably ran out of memory.

Master/Slave Analyst Version Incompatibility (238)

The version of the slave analyst being started by the master analyst is not compatible with the current master analyst. Check that the newest version of the Slave Analyst is being used.

GetGeometry failed (239)

IPS was unable to get the geometry of a new tree window.

Unknown rid identifier (240)

Master got an unknown routine identifier from an evu. Either somebody tried to play with VISI_FD (3), or there is an EVU library inconsistency between libevu.a and usri/evu*.c Check that you use the most recent ips version and evu library.

Unknown depth identifier (241)

During search trough the internal ips tree a node with an unknown TreeNode->depth identifier was found. This is a serious internal error in the master and should never happen. Consult the ips maintainer.

Unknown Trace Browser Identifier (242)

Trace browser received a trace record of unknown type from slave.

Error reading from slave on <host>: <error> (243)

Something is wrong with the slave analyst on the remote host. This error can result from the remote host crashing or if the slave analyst or sysctl file is not correctly installed.

Error poll/select lied: (244)

This is an internal consistency failure. Please contact the person who maintains IPS-2 for your site.

PinfoMatch(): unknown case (245)

An internal error in ips caused an unknown case to happen. Please dump core and report this condition to the person who maintains ips.

CpathMaster: registrar got bad msg type (246)

An internal error in ips caused an unknown case to happen. Please dump core and report this condition to the person who maintains ips.

Consistency Error in New Phase (247)

We could not find a free slot in the phase table. This is an internal error not the case of reaching a constant limit of ips. Please report this to the person in charge of ips at your site.

Error in NewGPhase (248)

We could not find a free slot in the phase table. This is an internal error not the case of reaching a constant limit of ips. Please report this to the person in charge of ips at your site.

Slack Support is not Compiled in (249)

The user interface is not compiled to support slack analysis. You need to re-compile the master with -DSLACK.

SHOWNODE: Unknown node: '%s' (250)

There is a confusion with the name of a machine node. Try using the short form of the machine in the name field. For example: asiago instead of asiago.cs.wisc.edu

Could not execute '%s' (251)

Could not run the preprocessor over the script file. Check to make sure cpp is installed on your system.

Could not open script file '%s' (252)

The system cannot find the script file (.ips file). Check to make sure the script file exists and restart IPS.

Could not open %s (253)

Could not find the file produced by the preprocessor. This usually means that something in your script file caused the preprocessor (cpp) to crash.

Phase restricted gprof tables are temporarily unavailable. (254)

There are some bugs in phase restricted gprof tables. These are being fixed, and such tables should be available soon. If you feel you need these tables before they are debugged, send mail to ips@cs.wisc.edu.

Slave returned a bad tabval (255)

The master requested a batch of table values from the slave, and one of the table values was invalid. Continuing from this error will allow the master to continue processing the rest of the valid table values. Frequent occurrence of this error message indicates a programming error in the slave.

References

1. S. L. Graham, P. B. Kessler and M. K. McKusick, "gprof: a Call Graph Execution Profiler", *Proc. of SIGPLAN Notices'82 Symposium on Compiler Construction*, 1982, pp. 417-420.
2. B. P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, S. Lim and T. Torzewski, "IPS-2: The Second Generation of a Parallel Program Measurement System", *IEEE Transactions on Parallel and Distributed Systems 1*, 2 (April 1990), pp. 206-217.
3. C. Yang and B. P. Miller, "Critical Path Analysis for the Execution of Parallel and Distributed Programs", *8th Int'l Conf. on Distributed Computing Systems*, June 1988, pp. 366-375.

Index

Table of Contents