Process Manager Application (PMAN)

version 2.1

Peter Olin

1997-11-06

Typeset in LaTeX from SGML source using the DOCBUILDER 3.0 Document System.

Contents

Chapter 1

PMAN User's Guide

The Process Manager Application (*PMAN*) is a tool which allows software developers to inspect the state of an Erlang system, as well as tracing events in the individual processes; either locally or on remote nodes.

1.1 The Process Manager

Introduction

The process manager (PMAN) is a tool for viewing executing processes in a distributed or local Erlang system. Its main purpose is to locate erroneous code by inspecting the state of the processes and by tracing events. Bottlenecks, unread messages, and bad memory handling are some of the problems that can be solved with PMAN.

Processes may be inspected individually in a process trace window. There the user may dynamically follow the execution of a process by getting trace output for sent and received messages as well as for called functions and some other process events. Information about sorce code modules executed by the processes is also accessible. Note that PMAN has some effect on the real time behavior of a running system.

Getting Started with Pman

Start PMAN from the Toolbar, or by calling the start function from the shell as in the example below:

```
(foo@heering.du.etx.ericsson.se)2> pman:start().
<0.34.0>
```

If you wish to trace a specific process without viewing the process overview window, you can use the `pman:proc/1` function as in the example below:

```
(foo@heering.du.etx.ericsson.se)5> pman:proc(list_to_pid("<0.32.0>")).
<0.37.0>
```

`pman:proc(Process)` allows you to trace a specified process without having to go through the process overview window and its menus. `Process` can be a process identifier or the name of a registered process.

The Main Window

When starting PMAN, the Main window [page 2] is displayed.

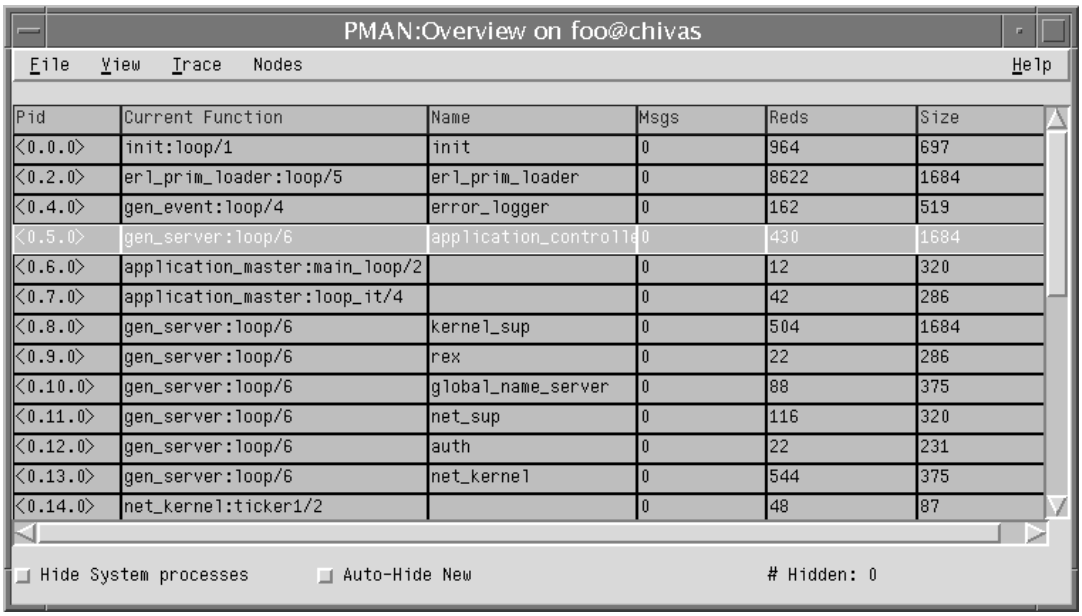

Figure 1.1: The Main Window of PMAN showing a process overview.

All currently running processes are displayed in the window. The following information is displayed for each separate process:

- the process identifier (Pid)
- the current module, function and arity (Current Function)
- the name of the process, if it is a registered process (Name)
- the number of messages in the queue (Msgs)
- the number of reductions performed (Reds)
- the size of the process, in words, calculated by adding the stack size and the heap size.(Size).

The Main Window is automatically updated every 5 seconds. There is also a Refresh function for manual updates.

In the figure illustrating the Main Window [page 2] the process <0.5.0> is highlighted, this process is selected. Some commands on the *View* and the *Trace* menus operate on a specified process, and thus requires that a process is selected. To select a process, click on it in the window. The arrow keys can be used to change selection to the process above or below the currently selected.

At the bottom of the window the following functions and information can be found:

Hide System Processes This check button controls the display of system processes. If it is selected, system processes will not be shown in the process overview, unless they are explicitly shown. The definition of system process is currently somewhat vague and ad hoc. Please experiment with the button to find out its exact effect on the display.

Auto-hide New This check button cotrols the treatment of newly created processes. If it is selected, processes created since the previous automatic or manual refresh will not be shown in the process overview.

 # Hidden This label displays the number of processes currently executing that are not shown in the process overview.

The File Menu

Default Options... This menu item opens a dialog that allows the user to set the default options for trace windows that are opened.

Save Options The options that are set using the *File->Default Options...* are saved using this menu item. The options are automatically loaded the next time PMAN is started.

The options are stored under the user's home directory, in a separate subdirectory named `.erlang_tools`, in the file `pman.opts`.

Exit Closes all windows and exits the application.

The View menu

This menu mainly contains functions for controlling what to display in the Main Window.

A process may be explicitly hidden, or explicitly shown, or neither.

Explicitly hidden processes are never shown, but can become explicitly shown by issuing any of the Show-functions in the View Menu.

Explicitly shown processes are always shown, but can become explicitly hidden by issuing any of the Hide-functions in the View Menu.

Processes that are neither explicitly hidden, or explicitly shown, are shown or hidden depending on the state of the two radio buttons at the bottom of the window.

Hide All Processes All visible processes are moved to the set of explicitly hidden processes, and as such, they are not shown.

Hide Modules... Opens a dialog the lets the user select a set of modules. The process overview will not show any processes running code from those modules.

Hide Selected Processs The selected process is moved to the set of explicitly hidden processes, and as such, it is not shown.

Module Info Opens a window showing information about the module the selected process is currently executing code from.

Refresh The process overview is updated. Normally this is automatically performed every 5 seconds.

Show All Processes All running processes are moved to the set of explicitly shown processes.

Show Processes... Opens a dialog that prompts the user to select a set of currently hidden processes to move to the explicitly shown set.

The Trace Menu

Kill Terminates the selected process by calling `exit(Pid,kill)` where `Pid` is the PID of the selected process.

Selected Process Opens a Trace Window for the selected process. Tracing will start immediately with the default trace flags set from the Main Window.

Shell Process Opens a Trace Window for the shell process on the node PMAN is running on. If the shell process dies, the opened Trace Window will find the PID of the automatically started new shell process, and continue to trace that process.

The Nodes menu

The Nodes menu contains one entry for each known node in a distributed Erlang system. This menu will not appear if the Erlang system executing PMAN is not a part of a distributed system.

By selecting a node from the Nodes menu, the process overview window will change its view, and display the processes running on that node.

> Note:
> The menu item *Trace->Shell Process* will not attempt to trace the shell process on the displayed node, it always operates on the node on which PMAN is running.

Trace Windows

A Trace Window continuously outputs trace information for a traced process. A Trace Window automatically uses the trace options set in the Main Window, but it is also possible to change the options for each Trace Window individually.

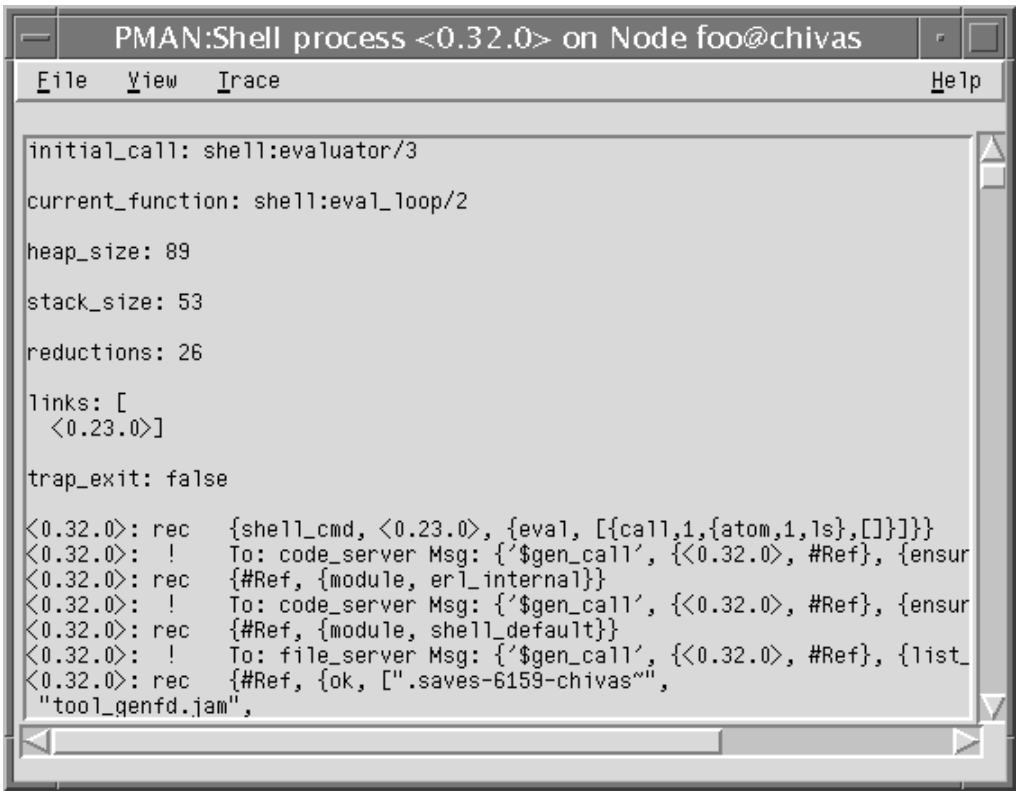

Figure 1.2: A Trace Window

There is no limit to how many Trace Windows can be open at the same time. However, notice that if more processes are traced, the performance degradation of the system will be more noticeable.

Window contents

The process window is used to display information about a specific process. The process window displays the following information, where applicable:

- initial call
- current function
- messages
- dictionary
- heap size
- stack size
- reductions
- links
- trap exits.

Also use this window to display ordered trace messages. The type of trace messages which you may want to display include function calls, BIF calls, as well as sent and received messages, etc.

First in each trace message will be the PID of the process being traced. Note that if the inheritance flags for tracing are set, the trace messages of the spawned/linked processes will be shown in the same window as the spawning/linking process.

Each trace message also has a menmonic tag to help you follow the execution of a process:

! This tag indicates that a message has been sent. Following the `To:` tag will be a PID or the name of a registered process. Next, following the `Msg:` tag will be the sent message.

rec This tag indicates that a message has been received. Following this will be the received message.

call This tag indicates either a BIF-call or a call to another function (only available for trace compiled code). Following this will be the actual call, with all the arguments.

link This tag indicates that a link between the traced process and another process has been created.

spawn This tag indicates that the traced process has spawned another process. Following this will be the PID of the spawned process.

exit This tag indicates that traced process has exited. Following this will be the EXIT reason.

The File Menu

Options... Opens a dialog that prompts the user to enter trace options for this specific Trace Window.

Save Buffer... Opens a dialog that prompts the user for a file name to save the current Trace Window contents in.

Close Stops tracing of the process, and closes the Trace WIndow.

The View Menu

Clear Buffer Clears the contents of the Trace Window.

Module Info Opens a window with module information for the module the process is currently executing code from.

The Trace Menu

All Linked Processes Opens a Trace Window for each of the processes linked to the process being traced in the current Trace Window.

Linked Process -> The Linked Process submenu has one entry for each process linked to the process being traced in the current Trace Window. Select one of the processes to open a new Trace Window for that process.

Kill Terminates the process being traced in the current Trace Window by calling `exit(Pid,kill)` where `Pid` is the PID of the traced process.

The Options Dialog

The Options Dialog allows the user to specify the amount of output, and the destination of output for traced processes.

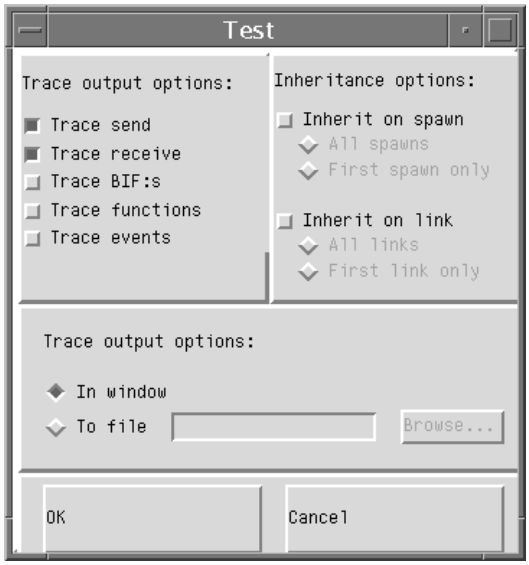

Figure 1.3: The Options Dialog

In the upper left corner of the dialog, there are check buttons for determining what to output in the Trace Window:

Trace send Select this check button if you wish to display messages sent from the process.

Trace receive Select this check button if you wish to display received messages.

Trace BIF:s Select this check button if you want to see calls to builtin functions.

Trace functions Select this check button if you want to see all calls to funtions. Note that this option will have no effect, unless the process is running code that has been compiled with the `trace` option.

Trace events Select this check button if you want to see process events, such as spawn, link and exit.

In the upper right corner of the dialog, there are options for controlling the behaviour of spawned or linked processes:

Inherit on spawn The user may select if spawned processes shall also be traced. And if so, if all spawned processes should be traced, or just the first spawned process.

If a spawned process is traced, it will get the same trace options that are set for the spawning process. And the output will be shown in the same Trace Window as that of the spawning process.

.

Inherit on link The user may select if a process that is being linked to shall be traced. And if so, if all linked processes shall be traced, or just the first one linked to.

If a linked process is traced, it will get the same trace options that are set for the linking process. And the output will be shown in the same Trace Window as that of the linked process.

In the lower part of the Options Dialog, the user may select whether the trace information shall be output to a file,or appear in the trace window.

Sending trace information to a file is much more efficient than displaying it in the Trace Window. Furthermore, if the amount of trace data is large, it will not be lost if tracing to a file. The trace information in the Trace Window has an upper limit (approx. 10,000 lines), after which the output buffer will be cleared .

PMAN Reference Manual

Short Summaries

- Erlang Module **pman** [page 10] – A graphical interface for inspecting running Erlang processes.

pman

The following functions are exported:

- `start() -> Pid`
  [page 10] Starts the process manager
- `start(LIModuleExcluded) -> Pid`
  [page 10] Starts the process manager
- `start_notimeout() -> Pid`
  [page 10] Starts the process manager
- `start_notimeout(LIModuleExcluded) -> Pid`
  [page 10] Starts the process manager
- `proc(Process) -> Pid`
  [page 11] Traces a single process
- `proc(A,B,C) -> Pid`
  [page 11] Traces a single process

pman (Module)

The Process Manager (PMAN) is a tool that provides functionality for inspecting the state of running processes in an Erlang system.

The user interface provides two main views:

- a node overview showing all running processes on the selected node
- a process trace window for following the execution of one or more selected processes.

PMAN operates either locally or in a distributed Erlang system.

Refer to the Tools chapter in the Erlang Development Environment User's Guide for a detailed description of the functionality.

Exports

```
start() -> Pid
start(LIModuleExcluded) -> Pid
```

Types:

- Pid = pid()
- LIModuleExcluded = [atom()]

`start/0` starts the process manager with the user's saved settings, if there are any.

`start/1` also starts the process manager, with the difference that it will not show any processes executing code in any of the modules listed in `LIModuleExcluded`

`start/0` and `start/1` return the Pid of PMANs main window process if the start-up succeeds within 20 seconds, otherwise it will fail with the EXIT reason {`startup_timeout, pman`}.

```
start_notimeout() -> Pid
start_notimeout(LIModuleExcluded) -> Pid
```

Types:

- Pid = pid()
- LIModuleExcluded = [atom()]

`start_notimeout/0` starts the process manager with the user's saved settings, if there are any.

`start_notimeout/1` also starts the process manager, with the difference that it will not show any processes executing code in any of the modules listed in `LIModuleExcluded`.

`start_notimeout/0` and `start_notimeout/1` return the Pid of PMANs main window process. It will hang indefinitely waiting for successful start-up.

```
proc(Process) -> Pid
proc(A,B,C) -> Pid
```

Types:

- Pid = pid()
- Process = pid() | atom()
- A,B,C = integer()

`proc/1` and `proc/3` both open a trace window for the specified process. They are convenience functions for bypassing the process overview window in the graphical user interface. `Process` can either be an atom representing a registered process, or a PID.

`proc/3` is merely a convenience function that takes three integers as arguments, representing the three parts of the PID. (To avoid having to use the BIF `list_to_pid/1`).

The functions return the PID of the process controlling the trace output window. If there is a problem they will fail with the EXIT reason `undefined`.

See Also

Pman in User's Guide.

List of Figures

Index

Modules are typed in *this way*.
Functions are typed in `this way`.