Abstract

The display criteria and algorithm of a program to run dynamic updates of a process tree/forest are described.

The program is available from:
`ftp://netwinder.org/users/r/rocky/xps.tar.gz`
home page:
`http://www.netwinder.org/~rocky/xps-home`

To glean what's going on in a computer, many systems administrators use `ps` or `top` or a `top` variant, like `gtop`. The `ps` program can list the processe id and its parent process id. However it is sometimes useful to list the processes in tree order. For example, there are a number of front ends or GUIs that perform various underlying commands, and sometimes one wants to see what's going on behind the scenes. For example, I've found it interesting to watch GNU `configure` run; sometimes seeing the relationship of who spawned whom gives a better idea of what is going on and may help better identify a process. Or one might want to find the shell run from `sshd` rather than, say, an `emacs` shell or an `xterm` process.

`xps` uses a custom tree-layout algorithm which positions nodes and draws lines between them using low-level Xlib calls. Probably each time a new graphics library comes out I considered using a tree package. To date, I've not found an acceptable tree display package, perhaps because the requirements needed by `xps` are a bit difficult to satisfy. In fact `xps` might be better thought of as an animation rather than a static charting program. For an analogy, consider the differences between viewing an MPEG (and its display protocol) and a JPEG (and its display protocol).

Here are the display criteria for `xps`:

- the layout needs to be fast — it is performed every second
- the layout should be compact
- the layout should have hysteresis — reduce flickering
- the layout should be "pretty" on the display

Fast layout:

In designing any monitoring program, such as `xps`, a cardinal rule should be: don't trash the system you are trying to monitor.

Because layout needs to be fast and work for a large number of nodes, positioning decisions have to be pretty much linear. Currently one pass from root to the leaves is made. I've considered making a backward pass as well—from leaves to roots.

The program has a number of hacks to avoid layout recalculation when it is not needed. It checks to see if processes have changed; if not, nothing is changed. However this is tricky since `xps` shows process status and if this changes *some* redisplay, albeit limited, is needed. If the display is iconified or if its display window is obscured, no layout is computed.

Compact, hysteresis, pretty layout:

Experience has shown that it is preferable to draw straight lines for parent/child connections rather than diagonal lines since diagonal lines take longer to draw and can appear jagged unless antialiasing is used. (Antialiasing generally takes more time to compute and draw.) On the other hand, when needed I find diagonal lines are easier to follow than a Manhattan metric S-shaped line which uses only horizontal of vertical lines.

Similarly, I've found that lining things up horizontally and vertically helps. So the program stores the maximum breadth at any depth in the forest and arranges other levels below that to fit into "slots" in this spacing. This also helps compacting the overall tree.

Other niceties

In addition, `xps` does a sort by userid and pid within that. This tends to group related processes together and the pid arranges processes, especially sibling processes roughly by age.

`xps` has the ability to filter out processes by userid or userid-regular expression. Each user is assigned a different color. Reducing the amount of display has a two-fold benefit; it not only unclutters the display but it also reduces the amount of time needed for display.

Layout

The program needs to do a topological sort. A depth-first search is done to assign depth number.

Within levels, sorting is done via a bubble sort. I believe this is okay because the processes are often are pretty much sorted by process id and this is roughly how we want them to be arranged. The bubble sort code has a check to stop early when the items are fully sorted. The amount of bubble sorting is relatively small, since it is only on children of a node. The sorting to arrange things into levels by parent is $O(npc^2)$ for n nodes, p non-leaf nodes, and where c is the maximum number of children a parent has. This is a rough analysis and looks unpromising, but in practice is probably pretty good. I suppose radix sort could be done on the levels. Then we'd have $O(n + pc^2)$.

Still someone might want to do timings with various algorithms. The program is not linear. Faster tree-layout and arranging algorithms would be helpful in handling larger trees.

Future?

Just about everything could be improved. The distribution contains an extensive list of things to do.

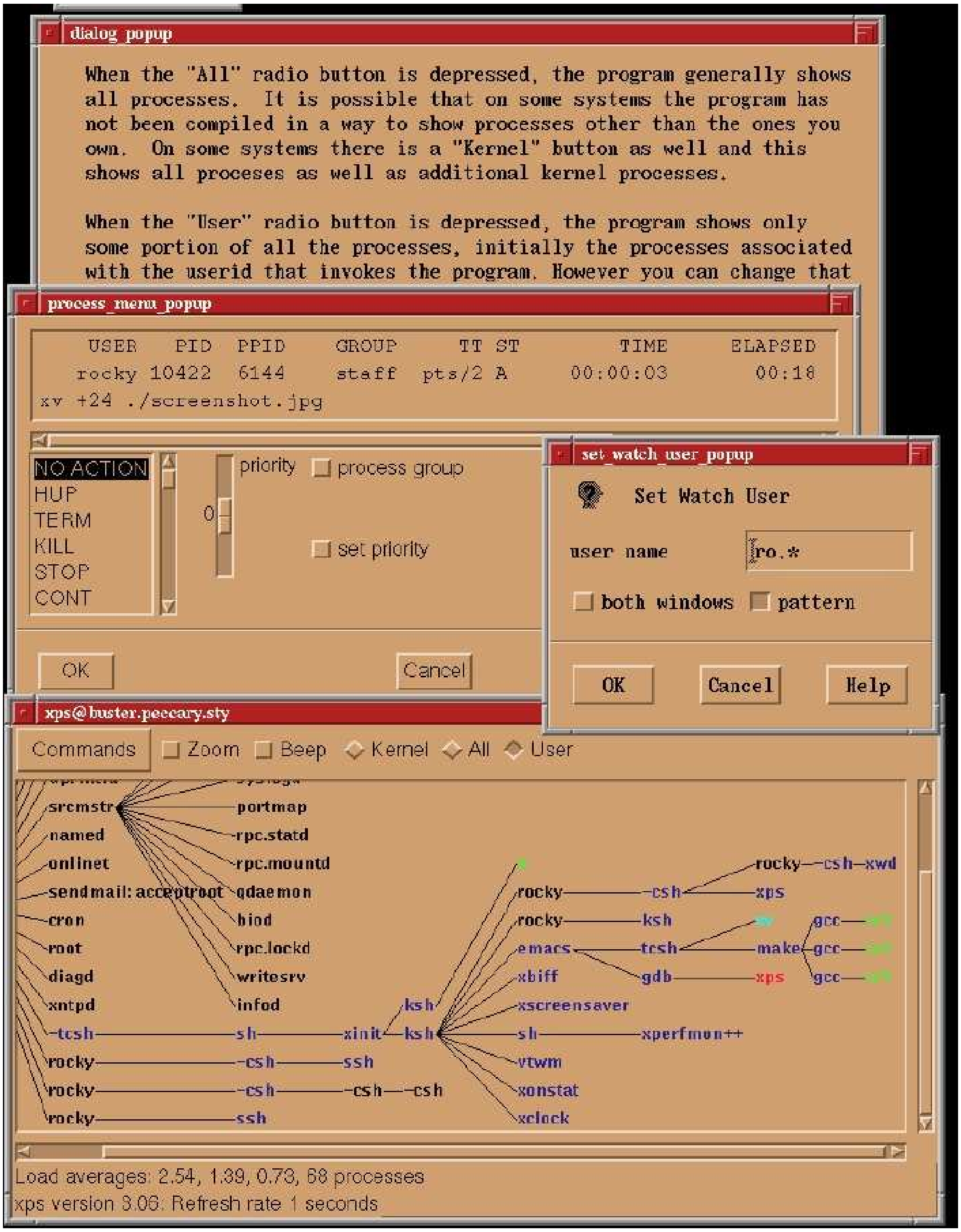
The thing I would most like to see is this ported to the GNOME `gtk+` and `glib` libraries. Any volunteers?

Display is done by drawing on a canvas. This is primitive. Process names are not widget labels, just X strings drawn on a canvas. Figuring out the process under the mouse and showing which process is selected is a bit low-level and not tool-like. Currently, a horizontal and a vertical linear search is done to find the pid under the mouse. Binary search might be faster.

Alternatively, if a full-fledged widget for the nodes of a tree were used, then X (or an X-toolkit) would worry about when the widget is selected; presumably it uses an algorithm at least as good as binary search. Making the selected node look, well, *selected* would then be done by the toolkit.

It would be nice to benchmark the various toolkit approaches for efficiency.

One might experiment to compare speed and intuitiveness of using a canned tree widget versus a hacked layout customized for this application. Personally, I prefer the tree layout, but the code is not toolkit idiomatic.


dialog_popup
When the "All" radio button is depressed, the program generally shows all processes. It is possible that on some systems the program has not been compiled in a way to show processes other than the ones you own. On some systems there is a "Kernel" button as well and this shows all proceses as well as additional kernel processes.
When the "User" radio button is depressed, the program shows only some portion of all the processes, initially the processes associated with the userid that invokes the program. However you can change that
process_menu_popup
USER PID PPID GROUP TT ST TIME ELAPSED
rocky 10422 6144 staff pts/2 A 00:00:03 00:18
xv +24 ./screenshot.jpg
NO ACTION
HUP
TERM
KILL
STOP
CONT
priority
0
process group
set priority
OK
Cancel
set_watch_user_popup
Set Watch User
user name
ro.*
both windows
pattern
OK
Cancel
Help
xps@buster.peccary.sty
Commands
Zoom
Beep
Kernel
All
User
srcmstr
named
inetd
sendmail: accept
cron
root
diagd
xntpd
-tcsh
rocky
rocky
rocky
portmap
rpc.statd
rpc.mountd
qdaemon
biod
rpc.lockd
writesrv
infod
sh
-csh
-csh
ssh
xinit
ksh
ksh
ssh
-csh
-csh
rocky
rocky
emacs
xbiff
xscreensaver
sh
vtwm
xconstat
xclock
-csh
ksh
tcsh
gdb
xperfmon++
rocky
-csh
xwd
xps
make
xps
gcc
gcc
gcc
Load averages: 2.54, 1.39, 0.73, 68 processes
xps version 3.06. Refresh rate 1 seconds