xps: Dynamic Tree Watching under X

Rocky Bernstein – Breakaway Solutions

ABSTRACT

The xps program dynamically displays the Unix processes as a tree or forest in an X Window, the roots on the left and the leaf processes (those with no children) on the right. The status of each process running, sleeping, stopped, etc., can be indicated by different colors. Different users can appear as different colors too.

Process selection can be made per user, all users, or through a regular-expression pattern.

In contrast to the terminal-based pstree or tree-widget based programs, the tree display uses diagonal lines, and effort is made to effectively use the full 2-dimensional area of the screen by balancing levels and centering the children of a node between their parent. A goal of the program is to give an idea of what's going on graphically as things may be constantly changing. Therefore the display algorithm tries to keep processes close to their parents to reduce the amount of scrolling to see localized process creation and destruction. Some effort is also given to make sure that the tree layout doesn't get wildly reorganized when there are small or localized changes. This makes it easier for the eye to pick up and recognize the changes over a potentially large display area.

We describe here criteria for tree animations such as this one and how the xps layout algorithm works.

There are some other miscellaneous features of xps. One can select viewing the processes by a single user, a regular expression for users, by all users, and perhaps show kernel processes. One can click on a process to get more information (via ps or a user-specified program) about that process, send a signal, or set the process priority, assuming you have the permission to do so.

Since programs of this ilk can consume a bit of CPU on their own, some effort has been made to turn off the update process when the program is iconified or not visible for some other reason such as being obscured by another window. Some attention has been paid to make algorithm display fairly fast in most situations, although it has to be admitted that this comes sometimes at the expense of a nicer layout.

What Is xps? Why xps?

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. Some examples include:

- Learning more about what is going on in configuration and installation processes; make or GNU configure, for example
- Tracking down zombie creation
- Understanding the processes for a user or contained in a process group; these generally cluster into subtrees
- Helping distinguish a process via its lineage. For example one may have many bash sessions running. Some may be spawned from sshd, some from telnet, some via emacs or an xinit session.

To glean what's going on, many systems administrators use ps or top or a top variant, like gtop. The ps program can list the process id and its parent process id. However it is sometimes useful to display the parent-child relation graphically.

Considerations for Doing a Tree Layout for xps: Tree Animations

xps uses a custom tree-layout algorithm for positions nodes and draws lines between them using low-level Xlib calls. Probably each time a new graphics library came out I considered ditching the custom tree-layout routine with a generic tree package. However to date, I've not found an acceptable one.

Tree widgets are often used for browsing file systems or menu systems. Generally these objects are fairly static. There is generally a way to collapse or expand a branch in the tree; this makes sense when information doesn't change all that much. But in the context of xps, sometimes the new or changed information is precisely what you want to see. So instead, in xps, symbolic or filtering rules are used to focus the display, rather than by zooming in or out from a user-selected tree branch. In particular, one can select the area of interest by a regular expression which is matched against the process-owner name.

In trees with menus or files, the display is pretty much static, and therefore the time users spent to customize the display may be cost effective; it may not be as important to spend time initially making connections between tree objects immediately visually distinct. Convention or experience seems to indicate that it is acceptable to use a Manhattan-metric line connection style, that is, where tree lines either go horizontally or vertically. See Figure 1 for a tree layout using Manhattan-metric lines.

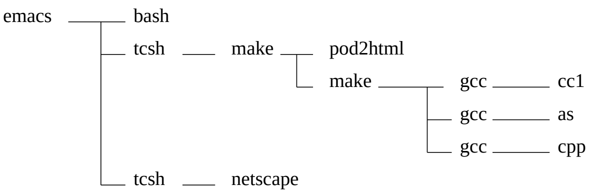

Figure 1: Lopsidedness and space-wasting layout using a Manhattan-metric tree widget.

The tree-layout requirements of xps are perhaps a bit more difficult to satisfy. In fact xps might be better thought of as an animation rather than a static charting program, and I am not aware of much literature on doing animations with trees or forests.

For an analogy, consider the differences between viewing a still-life painting and an animated cartoon. The cells of an animated cartoon do not have to be as perfectly rendered as a painting. Instead it is more important in an animation to make the *series* of cells relate to convey a story or action.

Below we gives a simple example of the kinds of problems faced.

In a world where things don't change, like a figure in a book, most people will find it most it pleasing for graphs with one or two children to look as in the left-hand part of Figure 2 rather than the left-hand part of Figure 3.

Figure 2: Aesthetic tree layout when things don't change.

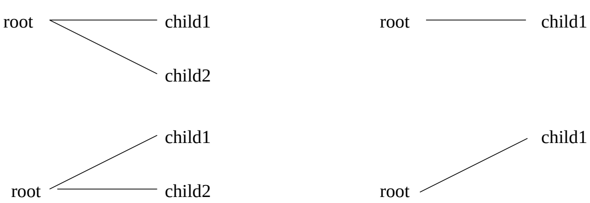

Figure 3: Possibly better tree layouts when trees shift between one another.

However, now consider the case where things are constantly changing. Suppose we want to chart what is going on when we have a process that forks a process, child1, and then forks a number of other processes sequentially. For a moment there will be child2; it dies, and then suppose a moment later a new processes, child3, is spawned. Suppose that now runs for a short period of time and dies; so we are back to the single child1; then a new process child4 starts up and so on.

In this scenario, if you render the one or two child processes as in Figure 2, you will see a bit of flicker. This is distracting and does not assist comprehending what's really going on: that a new process is being started and finished while nothing is happening to child1. If instead the graphs were rendered as in Figure 3, the display is more pleasing to the eye *in animation*; it is less spastic and more comprehensible even though the left-hand or right-hand side in isolation may not be the most esthetic rendering of the graph.

In sum, hysteresis of layout can help visualization.

Now consider the differences between a Manhattan-metric layout of the variety one often sees with a tree widget, and one that uses diagonal lines in an artificial but not uncommon situation as is often seen in xps.

Notice in Figure 1, how the root of the tree, emacs, is very far removed from its bottom-most node. This is not a problem with the Manhattan-metric connection style per se. However almost all most tree widget programs do this kind of layout; to set the position of a node, the layout algorithm can be extremely simple since it doesn't have to consider the positions of the children.

In addition to the lopsidedness, there's a bit of space that is wasted between make and netscape that doesn't appear in Figure 4. Furthermore, the blank area to the right of bash and pod2html is reclaimed reducing the overall dimensions of the graph. However, the more-compact layout is at not at the expense of readability.

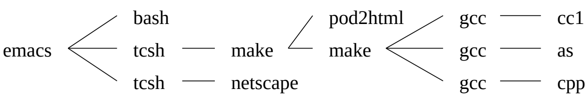

Figure 4: Layout. Nodes are more centered around their children; diagonal lines improve readability. Layout makes better use of available space and therefore dimensions are reduced.

Finally, compare Figure 1 and Figure 4 to see how the use of diagonal lines helps readability.

Experience has suggested these display criteria for a tree-animation program such as xps:

- The layout needs to be fast – it is performed every second
- The layout should be compact
- The layout should have hysteresis – reduce flickering which is annoying to watch and makes finding important changes hard to spot
- The layout should be ''pretty'' on the display so users can understand the relationships quickly

We now go into some of these needs in more detail.

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 should be pretty much linear. Currently one pass from root to the leaves is made. I've considered making a backward pass as well – 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 the status of each process (e.g., whether it is running or sleeping or stopped), and if this alone changes some redisplay of the node color (not position) is needed. If the display becomes iconified or fully obscured, no layout is computed.

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 also generally takes more time to compute and draw.) On the other hand, when needed, I find diagonal lines are easier to follow than Manhattan-metric lines.

Similarly, I've found that lining things up horizontally and vertically helps.

Layout Heuristics

The key to a program like this is its tree-forest display algorithm. In the previous section we described desirable qualities; in this section we'll give a rough idea of the actual heuristics used. In the next section we go over how this is implemented and the time complexity of the layout algorithm.

Trees tend to be narrow at the root and get bushy as one moves towards the leaves. Therefore the approach used in `xps` is put nodes into fixed levels (which run horizontally) as we move from the roots to the leaves (left to right here).

The algorithm we use pretty much makes one pass to keep things fast. We keep track of the maximum number of nodes at a given level as we move from roots to leaves. As the breadth increases, the gap between levels decreases up to a minimum threshold. When things get too tight, the overall dimensions of the graph increases; in display, the tree breadth is shown in the vertical direction.

Figure 5 gives an example of how levels are redivided as we move deeper in a tree.

Putting things in levels increases the likelihood that parent-child nodes will line up and that they will stay lined up over time. Actually, although it might be useful to have `xps` take into account old positions of node as discussed in a previous section, right now it doesn't; it is just an artifact of the way the layout occurs that this tends to happen.

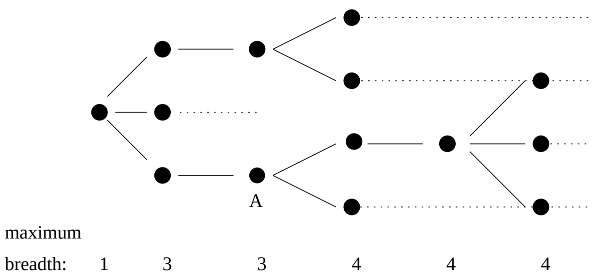

Figure 5: Tree layout along maximum-breath levels. Dotted lines show the current levels that are in effect for layout at subsequent levels. The node labeled A has rank 2 and virtual rank 3.

After creating the levels for a given depth, the dotted lines in Figure 5, `xps` next positions the node for that depth. To this end, the program keeps track of its level position or *virtual rank* in that depth, and compares it to the virtual rank of its parent. The virtual rank differs from the number of nodes placed so far (or *rank*), when we've decided to leave a level slot empty so as to position a child closer to its parent. In Figure 5, the node marked "A" has a virtual rank of 3, while its rank is 2. This is because we've moved the node down to its level slot 3 so it will line up with its parent.

Note that since the number of levels only increases, and when this happens inter-level space between level generally decreases, if the virtual rank of child is less than or equal to the virtual rank of its parent, it can be positioned no further down than its parent.

When setting the position of a node, `xps` tries to position the virtual rank of the middle child close to or less than the virtual rank of the parent, but not so much that the overall breadth is increased.

Figure 6 shows what goes on here. In the upper-left diagram we show the virtual ranks and before readjustment; to the right of that, after the readjustment at the new depth. In the bottom-left we see what happens when the new level is attached to a node further down; since centering the children would increase the overall breadth and dimensions of the tree we do not center around the middle child. The bottom-right graph show a heuristic not employed by `xps` but might be if a backward pass were made: readjusting the parent.

There is one layout problem that should be mentioned and is shown in Figure 7.

We see here that in drawing lines from the end of a node, nodes with a short name can sometimes cross over a neighboring node name. Currently `xps` does not try to prevent this from happening. We have experimented with doing a layout as in Figure 8 without much success. Most of the time, though, the lines do not go through process labels.

Layout Algorithm implementation

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

The roots are then sorted by uid and pid. There is generally a small number of roots, often one. At depths deeper than the root, nodes are first arranged by attachment to a parent.

Within the children of a node, sorting is done by a bubble sort. Many people with computer science training cringe when they hear this, because they have been taught a bubble sort takes $O(n^2)$ time in the worst case, while there are many sorting algorithms that take $O(n \log n)$ time in the worst or average case. However, a bubble sort has an advantage our most sorting algorithms: when used on *almost* sorted data, the running time is linear.

In the context of xps, the processes are often pretty much sorted by process id. The bubble-sort code we use does a check to stop early when the items are fully sorted. Also, it should be noted that the number of items to be sorted is often relatively small, since the sort is performed only on children of a node. A bubble sort generally has less overhead than most

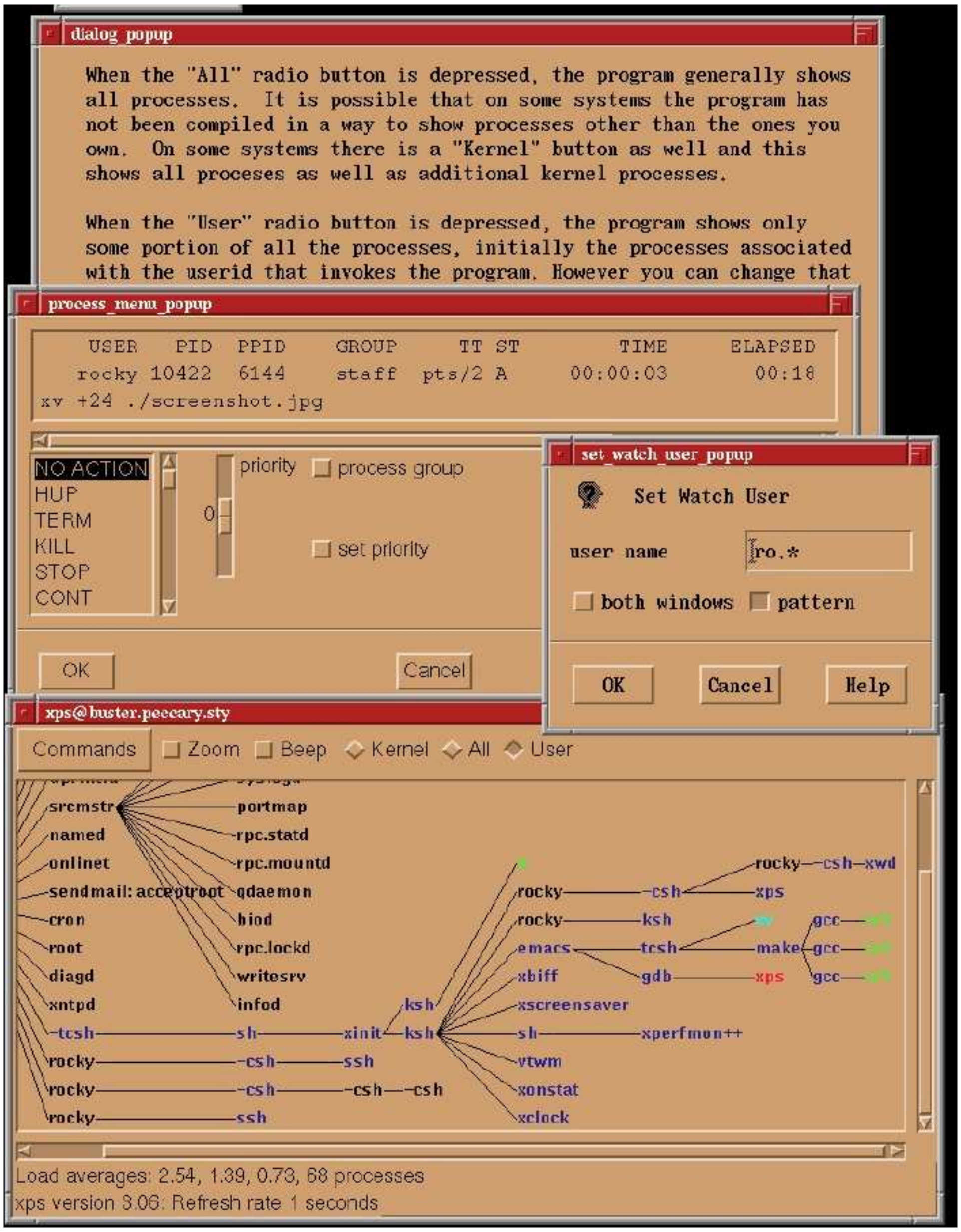

Figure 9: Aesthetic tree layout when things don't change.

other sorting programs. Still, a check on the number of children should be made and a sort like quicksort could be used if the number of children exceeds some threshold like 40 (which is in my experience rare).

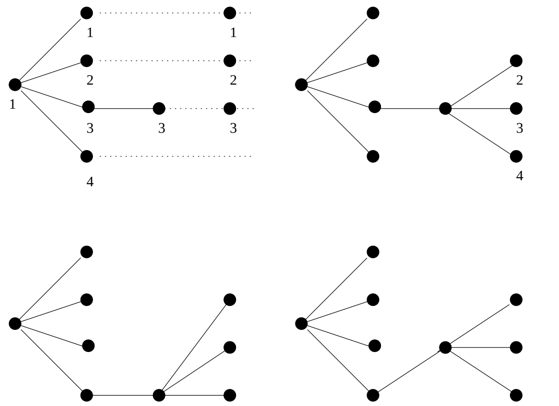

Figure 6: Example showing positioning heuristic.

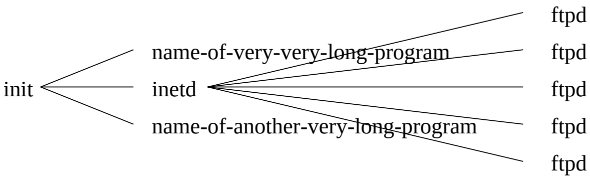

Figure 7: Crossover problems. This can be avoided by starting the vertex after inetd further to the right. Also note that following these lines would be easier if the difference in slopes is increased.

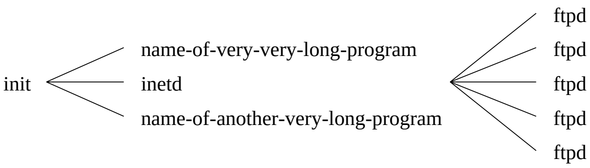

Figure 8: Crossover problems addressed by moving the drawing point further out. Increased sloped of lines may also increase readability. Compare with Figure 7.

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 a radix sort could be done on the levels. Then we'd have $O(n + pc^2)$.

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

Other Niceties of xps

xps does a sort by user id and by process id within that. This tends to group related processes together and the process id arranges things by age. It has been suggested that if it is desirable to sort precisely by age, that should be done instead of sorting by process id.

xps has the ability to filter out processes by user id 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.

Finally, xps has the ability to point and kill. (The interface hasn't been developed to the point of arcade games, but still it might give the systems administrator a heightened feeling of power; in contrast to mass kill programs like skill, it can be satisfying to see the process die before your very eyes.)

A screenshot from the program is given in Figure 9.

Future?

Just about everything could be improved. The distribution contains an extensive list of things to do. Above were some suggestions for how tree layout might be improved.

The thing I would most like to see is xps ported to the KDE qt and 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 process id 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.

Currently, the program contains coordinates of nodes before its position gets updated. However nothing else related to the history of the layout is saved, such as how long a node has been in the same position. It might be interesting to experiment with algorithms which make incremental improvements over time using say local heuristics. Analogous is the splay-tree algorithm which tends to make a tree balanced over time by making small localized changes as nodes are accessed.

Acknowledgements

Derik Lieber wrote the original version of this program.

Many people have read and made helpful suggestions on this paper including Stuart Frankel, Ph.D., George MacDonald, and Mike Welles.

Related work

pstree by Werner Almesberger found on Linux distributions shows a static Manhattan-metric process tree relation using character-oriented graphics.

Another program named pstree, by Lars Christensen, does about the same thing and runs on other versions of Unix. Find it at ftp://ftp.thp.Uni-Duisburg.DE/pub/source/ .

I. Herman, G. Melançon, M. S. Marshall ''Graph Visualisation and Navigation in Information Visualisation'' can be consulted for a survey on graph visualization and navigation techniques, used in information visualization. For example it cites the classic paper on tree layout, E. M. Reingold and J. S. Tilford, ''Tidier Drawing of Trees,'' IEEE Transactions on Software Engineering, SE-7(2), pp. 223-228, (1981).

However a number of subsequent papers including those found in the survey given above question the tree heuristics used in this paper. For xps, I have not found this paper all that useful, although it is an interesting read. See the survey at http://www.cwi.nl/InfoVisu/Survey/StarGraphVisuInInfoVis.html .

Finally George MacDonald's treeps program is similar. Find a home page at http://www.slip.net/gmd/tps/treeps.htm .

Availability

See the project's home page at http://www.netwinder.org/rocky/xps-home . The program is available from ftp://netwinder.org/users/r/rocky/xps.tar.gz .

Author Information

Rocky ''Falling squirrel'' Bernstein left the University of Maryland with two bachelor's degrees and then attended the Stevens Institute of Technology where he earned a master's degree. Rocky has worked in a number of institutions, such as the City University of New York, IBM Research, NASA Goddard Institute for Space Studies, The Associated Press (AP), and – currently – at ''Breakaway Solutions.'' His e-mail address is rocky@panix.com .