viztop – Intuitive Visualization of Remote Real-Time Monitoring of Linux Processes

Zoran Constantinescu
Dept. of Computer Science and Information Technology
UPG University of Ploiesti
Ploiesti, Romania
zoran@unde.ro

Monica Vladoiu
Dept. of Computer Science and Information Technology
UPG University of Ploiesti
Ploiesti, Romania
monica@unde.ro

***Abstract*—We introduce our tool for intuitive visualization of remote monitoring of processes' dynamic in a running Linux operating system, in real-time. It provides for overcoming the limitations of text-based process monitoring commands or tools available in current Linux distributions. It can display, via a web browser, the processes existing at any given time, in a running operating system, using a graph of interconnected nodes (processes, threads). Visual cues are used for representing different information, a large amount being available to the user, in a much easier way to understand than in text based tools. As processes are created, terminated, or they change state, the nodes in the process graph are added, removed, or change shape, color, size, line type, etc. The relations between processes are also shown. There is a strong correlation between the visual elements and the features of monitored processes. This tool provides both the big picture with regard to resources' usage in a computer system and plenty of useful details for improving system administration and performance. It can be also a valuable educational tool for learners, helping them to understand the dynamics of processes in operating systems.**

Keywords—visualization of processes' dynamic in operating systems; remote, real time monitoring; operating system education

I. INTRODUCTION

Visualization is an important asset to be used in instruction and education, in general. It can significantly contribute to increasing of both the efficiency and efficacy of the educational processes. Computer Science education, in particular, can benefit incredibly for incorporating visualization tools, both generic and custom made, in day to day teaching and learning activities [1-4].

Literature on this subject is abundant and shows using of visualization in various aspects of CS education, such as algorithm animations [2], theoretical Computer Science [3], security [5], key concepts in learning programming [6], computer architecture and organization [7], and many more.

In our department, we coordinate two study programs in Computer Science (bachelor and master). Finding ways to improve students instruction is a constant concern, and our work presented here subscribes to this goal. We aim to provide for improving understanding of dynamic of processes in Operating Systems (OS) by means of an in-house developed application, called **viztop**. It allows intuitive visualization of remote monitoring of the dynamic of processes in a running Linux operating system, in real-time. **Viztop** provides for overcoming the limitations of the text-based process monitoring commands or tools available in current Linux distributions. Thus, it can display, from anywhere in the Internet, via a web browser, the processes that exist at any given time in a running operating system using a graph of interconnected nodes (both processes and threads). Visual cues are used for representing different information about these processes and threads, such as shapes, colors, size, text, lines, etc. These cues allow a large amount of information to be shown to the user, in a much easier way to understand, when compared to classic text-based tools from Linux (ps, top, etc.). As processes are created, terminated, or they change state, the nodes in the process graph are added, removed, or change shape, color, size, line type, and so on. The relations between processes are also shown. There is a strong correlation between the visual elements and the characteristics of the monitored processes. This tool can be valuable for students and other learners, helping them to easily understand the dynamics of processes in OS, in real time. Nevertheless, the knowledge obtained can be used to improve system administration and performance, because it offers both the big picture, with regard to resources' usage in a computer system, and, at the same time, plenty of useful details.

The main contributions of this work are as follows: a comparison between similar tools based on a list o specific criteria (from which resulted the **viztop**'s requirements), and the viztop tool itself, which goes significantly further other similar tools, with its dual benefits for OS education and administration. The next section includes the related work, both in the literature and in practice, while the third one presents the main aspects related to the development and use of **viztop**. The last section is dedicated to the conclusions and future work.

II. RELATED WORK

The work related closely with ours is rather scarce, up to our knowledge, this being the main reason we have started the current endeavor, in the first place. Nevertheless, intuitive and comprehensive visualization of processes, connexions, traffic, load, resource usage in computer systems and networks, in general, is a hot topic in both the literature and the practice. Both aspects will be addressed further on.

A. Related work in the literature

We present first related work, in which the authors have been concerned with going further than just using existing visualization tools for illustrating specific aspects of computer systems and networks, either by developing new tools or by enhancing existing ones.

In [8-10], the task of representing flows in networks is approached, considering the multifold challenges, i.e. different flows typically traverse the same edges, flows may split and join again along their routes, and some flows may even go so far as to traverse the same nodes and edges several times. The authors acknowledge the limitations of traditional 2D visualization. For example, they tend to exhibit readability problems, in case of many flows traversing a single edge. To address these shortcomings, they propose a novel 2.5D metaphor that, instead of drawing flows as parallel curves on the plane, uses the third dimension to stack the flows one above the other. Their approach is implemented in a tool, the latest version being BGPlay3D that is a web application that allows change visualization in BGP routes associated with an Internet number resource (IP prefix or origin AS). It provides a graphical illustration of the links across all AS paths between the BGP collection points and the target resource(s). Without it, understanding the AS level topology from data repositories would be really difficult. A BGP hijacking simulation is also provided [9].

In [11], the authors show how graphs can be used as an effective modeling tool in computer security. They survey a large variety of fundamental security and privacy issues (network traffic monitoring, intrusion detection, vulnerability analysis, forensic analysis, authentication, access control, privacy compliance, and trust negotiation) addressed in their related work by several classic graph drawing techniques, such as force directed, layered drawing, bipartite drawing, treemap, circular, and 3D, and implemented in visualization prototypes.

Using process graphs for illustrating visualization exploration is shown in [12]. Visualization exploration is an iterative process of setting parameters, rendering, and evaluating results. A case study on how visualization graphs were used to improve a network visualization tool (the OASCBrowser [13]) is also presented. The OASCBrowser is a tool for visually detecting anomalies in internet routing information. It displays different types of changes to ownership of autonomous systems (called OASC events), which are labeled with different colors. It allows browsing through recorded dates with different types of AS changes highlighted. Anomalies are found by visually searching the dates for unusual patterns. The authors show that, trough this analysis, redundant exploration has been quickly identified and eliminated.

Rivet is a visualization system for studying complex computer systems [14]. Its main design goal has been *to support the rapid development of interactive visualizations capable of visualizing large data sets* because it is a well-known fact that computer systems analysis and visualization is an unpredictable and iterative process. The main capabilities are its *support for many data sources, interactivity, composition, and user-defined data transformations*. Several case studies of computer systems visualizations generated within Rivet, including on parallel systems, superscalar processors, and mobile network usage, are available as well.

Other related work is concerned with using visualization to improve teaching and learning of operating systems. We present further here some papers on this subject.

A very interesting original research is shown in [15]. First, the authors acknowledge that the major challenge in teaching operating systems is *the complex, intangible, and nondeterministic nature of an actual computer system containing many cores operating in parallel*. Under the umbrella of constructivism, the students participating in their study are invited to visualize the effect of running their own programs at OS level. Participants can thus observe the duration of underlying system calls and the actual scheduling performed by the operating system that is otherwise hidden.

The authors also performed various experiments to make the students aware on the impact of design choices on performance. Before this study, simulators or programming a small scale OS have been used to that outcome. However, the time dimension of an actual system lacked. Using visualization tools (control flow view being the main one) has allowed *construction of an abstract mental model of the system, including the time dimension*. The main shortcoming of their approach is that *students have been overwhelmed by specific technical details related to usability of visualization*. In our view, this may be due to the fact that they used many tools at once (such as Eclipse or LTTng), or even more instances of the same tool (Eclipse as IDE and for tracing).

In [16], a comparison of using MLFQ and CAMERA visualization tools in OS classes is provided, as well as an evaluation of student performance in those classes. The learner engagement with visual technology in this study at the "viewing" (passive form of engagement), "responding" (answering to questions about the visualization), and "changing" (modifying the visualization) levels are compared. Their results show that *learning improves as the level of student engagement with visualization technology increases*.

B. Related work in practice – a discussion

This section includes a discussion on UNIX/Linux tools used for system administration. More, a comparison between similar tools, based on a set of specific criteria (that have been the requirements for **viztop**) is also shown in Table 1.

General UNIX/Linux tools for statistics. One of the most used tool for showing processes in Linux is **ps**, which displays a snapshot of the current processes with detailed information about them (process id, cpu usage, memory size, state, owner, tty, etc.). There are plenty of tools giving details of different resources used in the system: **netstat** for network connections, routing tables, interface statistics, masquerade connections, **ss** for dumping socket statistics, TCP and state information, **vmstat** for reporting information about processes, memory, paging, block IO, traps, disks and cpu activity, **vnstat** for network traffic monitoring of each interface, **iostat** command for monitoring system input/output device loading by observing the time the devices are active in relation to their average transfer rates, **dstat** for generating system resource

statistics allowing the view of system resources instantly, and **ifstat** for network interface statistics, showing the difference between the last and the current call. All these tools are part of the operating system itself or come as open-source applications. They are text-based and used for fast console inspection, usually giving a snapshot of the system at the execution time, and some of them offer also a simple periodic display with updates of the parameters [17].

UNIX/Linux tools for real-time monitoring of system parameters. The classic tool used is **top** that provides a dynamic real-time view of a running system. It can display summary system information as well as a list of processes or threads currently being managed by the kernel. Over the time, alternatives to this tool emerged: **atop**, **htop**, **vtop**, etc. There are also similar tools for dynamic monitoring of I/O requests (**iotop, iptraf**) or process based network traffic (**nethogs**), network connections on each interface (**iftop**), or for monitoring specific parameters for applications, e.g. databases – **mytop** for MariaDB server performance, or **innotop** for MySQL/MariaDB InnoDB transactions monitoring. There are also tools for combined monitoring of more parameters like **glances** (processes, I/O requests, networking) or tools capable also of collecting performance data like **nmon** [17].

RRDtool is an open source integrable tool for handling time series data such as network bandwidth, temperatures, or cpu load, where the data is stored in a circular buffer-based database with a fixed storage footprint over time. It can log and graph different parameters in time [18].

Cacti is an open source, web-based, complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. It includes a fully distributed and fault tolerant data collection framework, advanced template-based automation features for devices, graphs, and trees. It can be used not only as a performance management tool, but also for fault management, log management, device discovery, router configuration backup, network mapping, and NetFlow data collection and display. Similar tools are **Observium** (open source and commercial) and **LibreNMS** (open source), as low-maintenance auto-discovery network monitoring platforms supporting also a large variety of devices. All these tools offer only limited information about the processes in each of the monitored systems, and only as high level load or number of processes, without any detailed data about each of the running processes, inter-relations, or their dynamics. Additional scripts or plugins can be written to monitor specific applications [19-21].

PRTG is a network monitoring solution for a central information point for all applications and services in a network. It continuously monitors the performance of applications by means of specialized software sensors, which reports inconsistencies in the normal execution when a service is unavailable, or over a specified threshold. PRTG has a lot of in-built tools such as packet sniffing software, jFlow/sFlow monitoring, firewall and IP monitoring, and even a network discovery and diagnostic tool to automatically locate network devices and optimize their troubleshooting [22-23].

Netdata is a system for distributed real-time performance and health monitoring. It allows collecting real-time metrics, such as CPU usage, disk activity, bandwidth usage, website visits, etc., and displays them on-the-fly, in live, easy-to-interpret, charts on web based dashboards, allowing the user to obtain an overview of what is happening in their system or application. The application plugin can break down system resource usage to processes, users and user groups, and can iterate through the whole process tree, collecting resource usage information for every configured running process, then present it in charts, and track them through time [24].

Grafana is an open source visualization and analytics platform that unifies data sets into an interactive diagnostic workspace. It is built on a plug-in architecture that allows interaction with the underlying data sources without creating data copies. Grafana provides charts, graphs and alerts in a modern web interface [25].

EtherApe is a graphical network monitor for Unix with an X-Windows user interface. It features link layer, IP and TCP modes, and displays network activity graphically. Hosts and links change in real time in size with traffic, while protocols are displayed color coded. It can filter traffic to be shown, and can also read packets from a file as well as live from the network [26]. A similar tool is **Kibana**, a data navigation and visualization application that is used with the open-source ElasticSearch, and as part of the ELK (ElastiSearch, Logstash, Kibana) stack. Kibana is the method through which users control their ElasticStack, as well as visualize the data coming from it, usually by means of analyzing systems logs that list and describe system events [27]. Visualization is interactive using histograms, line graphs, pie charts, sunbursts, time series, etc. Both tools offer a modern web interface.

Nagios is one of the oldest network monitoring tools. It has a free version, Nagios Core, and a payed one, Nagios XI. Nagios monitors the network for problems caused by overloaded data links or network connections, as well as monitoring routers, switches and more, allowing users to gain real-time visibility into the status of each network device. The information is presented by means of a web server [28].

Solarwinds SAM (Server and Application Monitor) is a server monitoring and reporting tool, providing server, application, virtualization, and infrastructure monitoring capabilities, through a single web console. It provides custom collections of templates, application monitors, and alerts, to intelligently monitor application status and issues, monitor different application types including application servers, authentication servers, database servers, etc. [29].

Further, a comparison between these tools is provided. The criteria used to compare their features are rooted in the services they provide (Table 1): (1) basic, text-based process monitoring, (2) intuitive representation of information, using rich visual cues, (3) representing the processes' dynamic and the relations between them (4), in real-time (5), (6) a bird's eye view and a close range one (7), potential for educational use (8) and cost (9). These have also been the requirements taken into account in **viztop** development because, as it can be seen, none of them checks all the boxes.

	processes monitoring	visual	dynamics	relations	real-time	bird's eye	details	education	free
Linux tools for statistics	yes	limited (color)	snapshot	no	no	partially	yes	limited	free
Linux tools for monitoring	yes load, count	limited (color)	periodic updates	partially	yes	partially	yes	yes	free
RRDtool	load, count	limited	time plots	no	no	no	no	not built-in	free
Cacti, Observium, LibreNMS	load, count	web	time plots	no	no	yes	no	not built-in	free
PRTG	load, count	web	time plots gauge	no	yes	yes	no	not built-in	paid
Netdata	possible	web	time plots gauge	no	yes	yes	partial	not built-in	free
Grafana, Kibana	load, count	web	time plots gauge	no	yes	yes	no	not built-in	free
EtherApe	no	GUI	graph	no	yes	yes	network only	yes	free
Nagios Core,XI	possible	web	time plots gauge	no	yes	yes	partial	not built-in	free paid
Solarwinds SAM	unable to asses	web	time plots gauge	no	yes	yes	partial	not built-in	paid

Table 1. Comparison of the main tools for UNIX/Linux monitoring

III. VIZTOP – THE APPLICATION DEVELOPMENT AND USE

The main goal of the **viztop** application is to provide for intuitive visualization of remote monitoring of the dynamic of processes in a running Linux operating system, in real-time. It has two components, a *backend* application for collection of the information about running processes in the system, and a *frontend* for the visualization of this information.

The **viztop backend** consists of an application written in standard C for Linux, which periodically scans the */proc filesystem* for all the processes in the system. It stores in memory information about all these processes, their states, and different data about resource usage, open files, network connections, etc. The backend contains also a *websocket* used for sending updated data to the frontend. For optimization purposes and to reduce the volume of transferred data, the backend only sends relevant updates about the processes, e.g. new processes, ended processes, changes or information of a process's state. Data is also compressed, to further reduce the data transferred and to allow higher update rates to the user. The backend allows multiple frontend connections from different users, and it will only send relevant data to each user.

The **viztop frontend** consists of a single web application written in JavaScript. It allows displaying the processes and information about them in a visual interface by using a graph of nodes and edges with annotations. Each node represents a process in the system, while edges are used for representing their dependencies (parent-child). The frontend receives updated data about the processes from the backend using the websocket. The user can visualize all this using a standard web browser. Different users can connect to the backend over the Internet from different locations. The **viztop** visualization interface will be presented in details further on (Fig. 1).

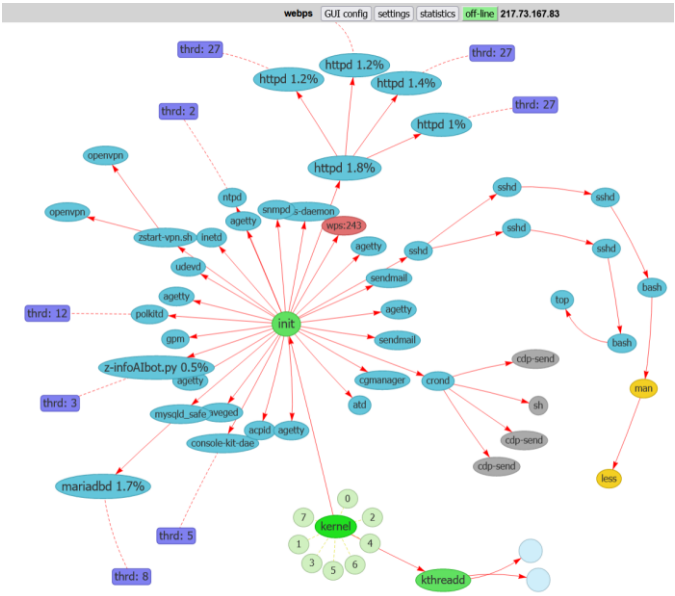

Fig. 1. User interface for process visualization

Each process from the system is represented as a separate node in the graph, which is connected to other nodes. As new processes are created, new nodes will appear dynamically in the process graph. If existing processes are ended, the corresponding nodes will be deactivated and they will fade away after a few seconds, allowing the user to get a feeling of the process dynamics in the system.

Each node of the graph (a process) has a color associated with it (Fig.2), which gives the user a clue of its state. For instance, green nodes are kernel processes (kernel, kthreadd, and the init/systemd startup), red nodes are processes in a run state (R), yellow ones are in stop mode (T), gray ones are zombie (Z), while blue nodes are in a sleep state (S).

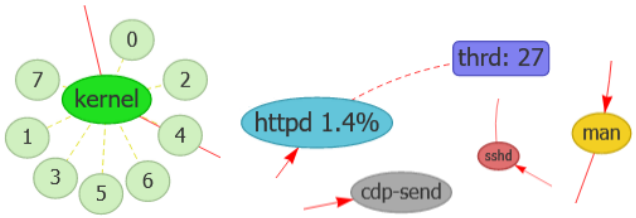

Fig. 2. Processes in different states

. The kernel node has attached a number of nodes from 0 to the maximum number of processors in the system (virtual cpus). Nodes for processes that contain multiple threads are colored dark violet and include the number of threads.

Nodes in the visual graph are connected with each other with red arrows, if there is a parent-child relationship between the corresponding processes (Fig.3). In this image, we have the *init* process created after the boot, which in turn starts a shell script (*zstart-vpn.sh*) that starts two *openvpn* processes. When processes are terminated, their connections in the graph are also removed together with the associated nodes.

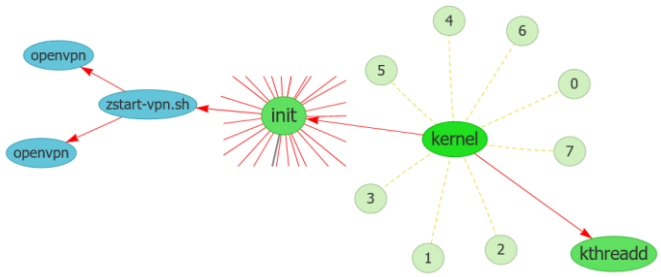

Fig. 3. Child and parent processes

In the visual graph, each node's size is directly related to the memory used by the corresponding process (Fig.4). In the depicted example, the *mariadb* process is using 43% of the total memory in the system, *java* 17.1%, *netdata* 9.4%, etc. This gives the user a quick overview of each process's memory usage. For the nodes using more than a specified minimum memory percentage (eg. 1%), this value is also shown in the node's description. The size of each node is dynamically adjusted in time according to the memory usage.

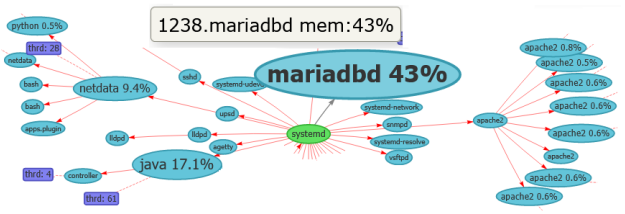

Fig. 4. Processes with different memory usage

Fig. 5 shows an example of how processes are created and destroyed, as the user executes command line instructions. Thus, a *bash* shell is created by the *sshd* process for a remote user connection, then the user executes the *su* command, and a new *bash* shell is created. The user then executes a *man* command, which then calls the *tbl*, *nroff*, *man*, and *less* commands, then the *man* process is stopped (T state – yellow color) and a *python3* script is executed, which in turn starts three other *php* commands, and so on. This is an intuitive view of how the dynamics of processes in a Linux operating system happens, and how, by executing different commands in the system, new processes are created or destroyed.

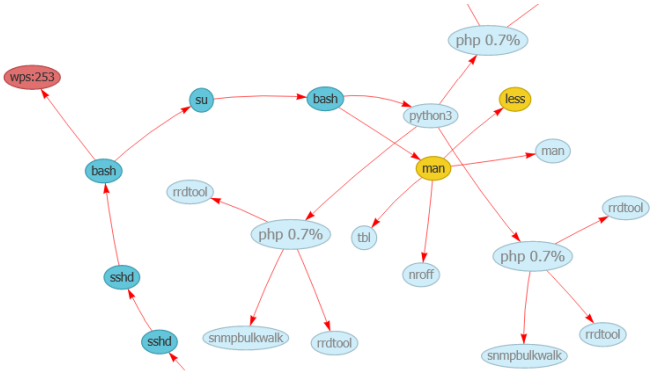

Fig. 5. Dependencies between processes

Another example is presented in Fig.6, where the *init* system process starts an Apache web server, which consists of a master *httpd* process that creates four more *httpd* child processes, each of these containing 27 threads. In time, the dynamics of the Apache web server can be monitored, as requests from the web clients create new threads, new *httpd* processes, or old ones are destroyed. Fig.7 shows how the graph of process nodes can evolve over time, as new processes and connections are created or deleted.

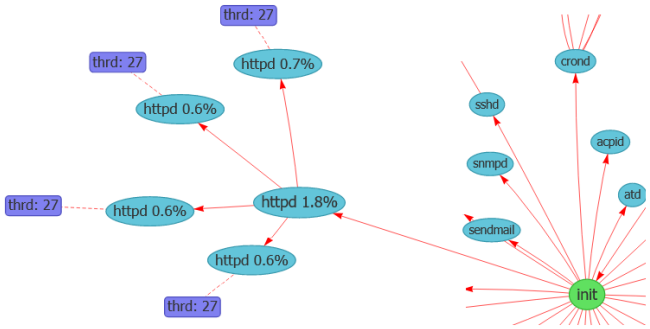

Fig. 6. Startup httpd server started by init

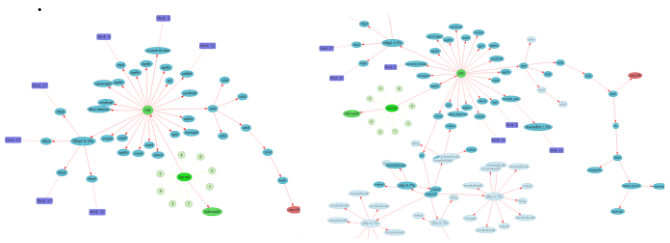

Fig. 7. Dynamics of processes in time

Another feature offered concerns the network sockets available on each process, which are depicted as violet nodes, including both the IP address and connection port number. In Fig. 8, we exemplify with how a *sshd* server manages network connections. The master *sshd* process is listening on port 22 (LISTEN 0.0.0.0:22), then each new *ssh* connection on the server creates a new *sshd* process, with an established TCP connection on local port 22 with the remote client. Here, two of the child *sshd* processes allow X-Forwarding for the clients, by allowing listening on ports 6010 and 6011.

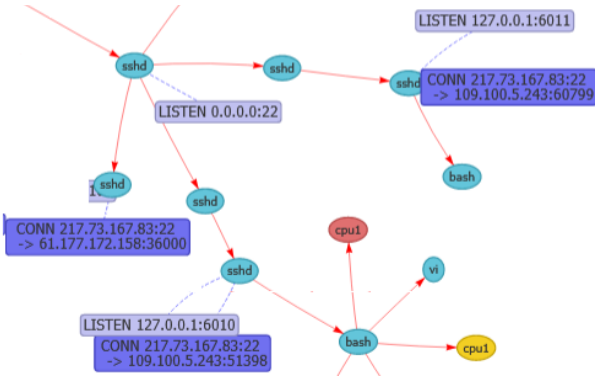

Fig. 8. Network connections for some processes

IV. CONCLUSIIONS AND FUTURE WORK

In this paper, we introduced a tool developed in house aiming at improving the understanding of dynamic of processes in operating systems. The main services provided have been established after comparing the main tools used for process monitoring in Unix/Linux, and taking into account how these can be enriched, in order to provide extra potential for both OS education and administration. We rooted our decisions about this in our long experience (50+ years combined) in OS instruction and Unix/Linux administration, in particular, and in CS education, in general.

In our current view, further work is twofold. First, we intend to use **viztop** next semester to perform some pedagogical research together with our students, with regard to its benefits in teaching and learning about processes in operating systems. Second, development of further OS visualization counterparts would be useful, such as memory management, resource allocation, file system change, etc..

References

[1] E. Fouh, M. Akbar, and C. A. Shaffer, "The Role of Visualization in Computer Science Education", Computers in the Schools, 29:1-2, pp. 95-117, DOI: 10.1080/07380569.2012.651422, 2012.

[2] D. Schweitzer, W. Brown, "Interactive visualization for the active learning classroom", SIGCSE '07 - 38th SIGCSE symposium on CS education, pp. 208–212, https://doi.org/10.1145/1227310.1227384, 2007

[3] T. Naps, S. Cooper, B. Koldehofe, C. Leska et al., "Evaluating the educational impact of visualization", ACM SIGCSE Bulletin, 35(4), pp. 124–136, https://doi.org/10.1145/960492.960540, 2003.

[4] D. Schweitzer, W. Brown, "Using visualization to teach security", Journal of Computing Sciences in Colleges, 24(5), pp. 143–150, https://dl.acm.org/doi/abs/10.5555/1516595.1516626, 2009.

[5] F. W. B. Li, C. Watson, "Game-based concept visualization for learning programming", MTDL '11 - 3rd int'l ACM workshop on Multimedia technologies for distance learning, pp. 37–42, https://doi.org/10.1145/2072598.2072607, 2011.

[6] D. Chudá, "Visualization in education of theoretical computer science", CompSysTech '07 - 2007 int'l conference on computer systems and technologies, 84, pp. 1–6, https://doi.org/10.1145/1330598.1330687, 2007.

[7] G. R.Garay, A. Tchernykh, A. Yu. Drozdov, S. N.Garichev et al., "Visualization of VHDL-based simulations as a pedagogical tool for supporting computer science education", Journal of Computational Science, 36, September 2019, 100652, https://doi.org/10.1016/j.jocs.2017.04.004, 2019

[8] M. Candela, P. Angelini, L. Antonetti Clarucci, M. Patrignani, M. Rimondini, and R. Sepe, "BGPlay3D: Exploiting the Ribbon Representation to Show the Evolution of Interdomain Routing", in S. Wismath, A. Wolff, (Eds.), 21st International Symposium on Graph Drawing (GD '13), Springer-Verlag, Lecture Notes in Computer Science, 2013, Poster, p 526.

[9] M. Candela, "Real-time BGP Visualisation with BGPlay", https://labs.ripe.net/author/massimo_candela/real-time-bgp-visualisation-with-bgplay/

[10] M. Candela, "Adaptive and responsive web-oriented visualization of evolving data: The interdomain routing case", Master's thesis, Roma Tre University, 2012.

[11] R. Tamassia, B. Palazzi, and C. Papamanthou, "Graph drawing for security visualization", in Tollis, I. G., Patrignani, M. (Eds.), Graph drawing 2008 (LNCS 5417, pp. 2–13). Berlin Heidelberg, Springer-Verlag, 2009.

[12] T. J. Jankun-Kelly, "Using Visualization Process Graphs to Improve Visualization Exploration", in Freire J., Koop D., Moreau L. (eds) Provenance and Annotation of Data and Processes. IPAW 2008. Lecture Notes in Computer Science, vol 5272. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-89965-5_10, 2008.

[13] S.T. Teoh, K.L. Ma, F. Wu, X. Zhao, "Case study: Interactive visualization for internet security", in 13th IEEE Conference on Visualization (Vis 2002), pp. 505–508, 2002.

[14] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum and P. Hanrahan, "Rivet: A flexible computer systems visualization environment", Computer Graphics 34(1), February 2000, pp. 68–73, 2000.

[15] F. Giraldeau, M. R. Dagenais, H. Boucheneb, "Teaching Operating Systems Concepts with Execution Visualization", 2014 American Society for Engineering Education Annual Conference & Exposition, Indianapolis, Indiana, pp. 1-15, 10.18260/1-2—23101, 2014

[16] X. Yuan, B. Pioro, R. Archer, Y. Li, "Teaching Operating Systems Using Visualization: A Comparative Study", Iskander M. (eds) Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8739-4_102, 2008.

[17] Linux man pages, https://www.kernel.org/doc/man-pages/, 2021.

[18] RRD tool - logging and graphing, https://oss.oetiker.ch/rrdtool, 2021.

[19] Cacti – the complete RRD-tool based graphing solution, https://cacti.net

[20] Observium – network monitoring platform, https://observium.org, 2021

[21] LibreNMS - network monitoring system, https://librenms.org, 2021

[22] Paessler PRTG – network monitor, https://www.paessler.com/prtg,2021

[23] D. Zobel, "Monitoring applications and services with network monitoring, white paper, https://hlassets.paessler.com/common/files/pdf/whitepaper/application-monitoring_en.pdf, 2013

[24] Netdata – infrastructure monitor - https://netdata.cloud, 2021.

[25] Grafana - open observability platform, https://grafana.com, 2021.

[26] EtherApe - network monitor, https://etherape.sourceforge.io, 2021

[27] Kibana – visualization tool, https://www.elastic.co/kibana/, 2021

[28] Nagios – network monitor, https://nagios.org, 2021.

[29] Solar winds – server and app monitor, https://solarwinds.com/sam