# A Framework for Visual Modular Design of Educational Operating System

Naeem Al-Oudat
*Communications and Computer Engineering Department, Tafila Technical University, Jordan*
*naeemodat@ttu.edu.jo*

***Abstract*— Operating systems are a vital part in most computing systems. However, learning basic concepts of operating systems are hard for normal students although they are necessary and important. State of the art in teaching operating systems depends on studying existing open source operating systems like Linux, hacking teaching operating systems like Xv6, or using simulators. Difficulties of learning still there in these methods, since they require a great deal of system programming techniques. In this paper, we propose a novel direction in learning operating systems that is solely dependent on visually building the operating system. By using this method, we mitigated the complexity of going into system programming details. The development platform consists of key building blocks that a user can drag and drop into a working panel to build his own operating system. Then, the user can compile and run his own-built system on a virtual machine or any Intel architecture hardware. This paper provides the framework's key points for building an operating system by modules. It also discussed a simple prototype as a proof of the concept.**

***Index Terms*—Educational Framework; GTK+ GUI; Modular Design; Teaching Operating System; Visual Programming.**

## I. Introduction

The operating system is an important part of any computing system no matter how complex it is. Therefore, teaching operating systems concepts to computer engineering and computer science students is necessary. Many researchers have tried to make operating systems learning as simple as possible through two main tracks. The first track is using existing open source and free operating systems or designing customized operating systems to let the students learn their internals [1]-[11]. Internals learning is conducted by modifying or rewriting some parts of the system. The second track is mainly dependent on simulations on either virtual machines or using some carefully designed standalone applications [12]-[17]. The common goal for all of these efforts is teaching the learners basic concepts of operating systems.

Any operating system must have all or some of the following basic components:

- Booting component. This part is responsible for loading the first/basic stage of the kernel.
- Kernel component. The core or the inner part of the operating system, where (in some operating system architectures) the bulk of the operating system is residing in it.
- Processes manager. Scheduling of the processes loaded on the system is the main task of this component.
- Memory manager. Utilizing the RAM and its extensions in an efficient way is the role of this component.
- File system. The main job of this component is to abstract the way of dealing with data and storing it in a permanent media as a hard disk.

Designing an environment where learners can work and design the above basic components of an operating system without getting into the complicated details is an urgent need in today's university classes of software systems. This simplicity should not make the whole process as a simulation/emulation-like design. A good option would be using pre-programmed components. The learner can select components and connect them to build and test a customized operating system. Each component has several versions, one can choose from them. Further, anyone who has good programming abilities can open any component and modify it, as he wants.

In this paper, we particularly concentrate on making operating system design simple and at the same time ready to run on an actual hardware. This is possible when using the idea of visual programming. Where the learner drags and drops blocks onto a workspace. Each block represents a ready to run source code when properly connected to other blocks. As such, a broader band of learners can benefit from the proposed framework. To make the design as simple as possible, we propose a visual drag and drop user interface environment. However, the designed operating system is ready to run on an actual computer architecture.

The rest of this paper is organized as follows. Framework system architecture along with its details are given in Section II. Section III discusses some of the characteristics of our prototype for the proposed framework. In Section IV, some related work is given. The paper is concluded and some future work directions are given in Section V.

## II. System Architecture

The proposed framework consists of three main stages, Figure 1; operating system construction stage, operating system building stage and test stage.

### A. *Construction Stage*

In this stage, the learner uses the operating system construction interface for visually building an operating system from the basic building blocks that are provided. Figure 2 shows an example for an operating system construction field.

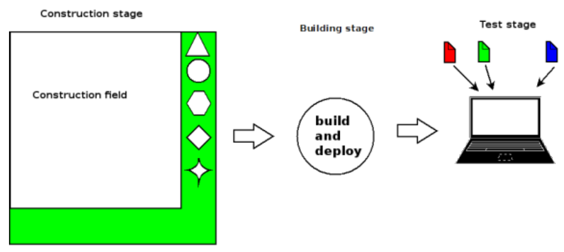


Figure 1: System architecture

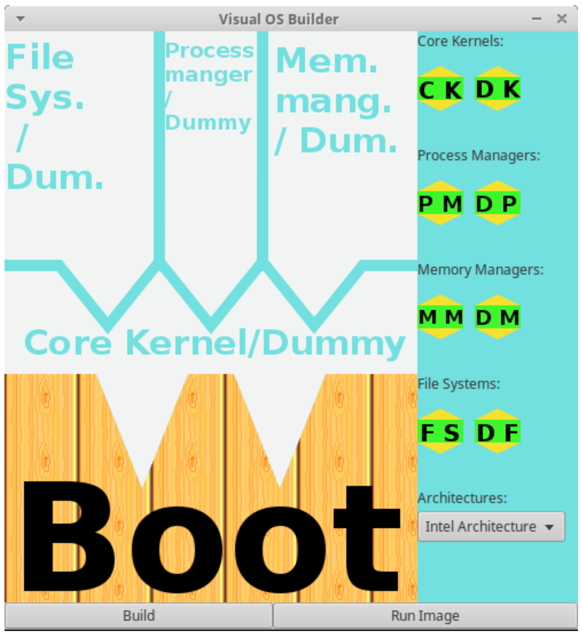


Figure 2: Construction field

Several components are available for the designer to choose from them. Further, any student/learner can open any component and modify the code therein to try whatever functionality she/he wants.

Some of the basic components that can be included come in four groups; core kernels, process managers, memory managers and file systems group. In each group, we propose two icons to be included, as shown in Figure 2, on the right pane; main icon and a dummy icon. When choosing the main icon in any group, a drop-down list of several options will appear. The learner can choose from these options. One of the available options is a custom choice, where the user can modify the code directly. The other icon is a dummy icon. The dummy icon is based on the group. For example, the dummy icon in core kernel group displays a message of user choice when the operating system is running.

One important notice is that a user can choose what architecture he is intended to use. Each architecture has its own low-level code that is different from other architectures. In our prototype, we assumed Intel Architecture.

## B. *Building Stage*

The build and deploy part is used as a second stage where the design is complete. If there is no error in the design, the building stage is successful. Then a designed OS image is deployed into a test stage as a virtual machine image or burned/dumped in any removable media to run on a real hardware.

This stage includes clicking on two buttons one after the other. The first button is the build button, where the modules dropped (during design stage) on the workspace are known. By clicking on build button, a compilation process is started for the source code. Then when this is successful, the image runs on a virtual machine by clicking "Run Image" button.

## C. *Test Stage*

To test the performance of the built OS, a set of experiments should be carried out on this OS and some performance measures are recorded for further investigations by the learner. These experiments can be preloaded on the image or can be loaded during run time. The details of these experiments are reserved for future work.

## III. System Prototype

We have designed a simple prototype for the described framework. This prototype is depicted in Figure 2. The design in this prototype is based on a monolithic kernel operating system architecture [18].

The user interface consists of two panes. The right-hand pane contains a set of icons groups: core kernel, process managers, memory managers and file system group. Each group has two icons, a main and a dummy icon. The main icon is intended to give the user a set of choices. However, we reserve this part for future work. For the kernel group, dummy icon gives the learner an ability to enter a message, which is shown on screen once the image is booted. The main icon in kernel group defines the low-level codes to be used in modules that come on top of the kernel. Main kernel module starts other modules on top as well. Once the module icon crosses the border, it turns to its final shape (see Figure 3).

For other groups the main and dummy icons do the same job as follows:

- Process managers. This set uses RR (Round Robin) scheduling algorithm to switch between two tasks. Each of the tasks runs forever and print some message on the screen, as shown in Figure 4.
- Memory managers. This set does basic and necessary functionalities like preparing the memory as a set of pages and allocating memory for the built-in tasks.
- File system. This group does not have any role in this prototype since all the necessary code is loaded into the memory and run from there.

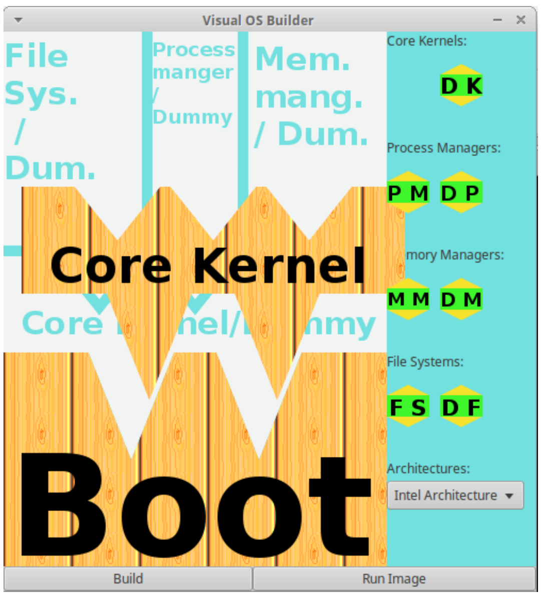
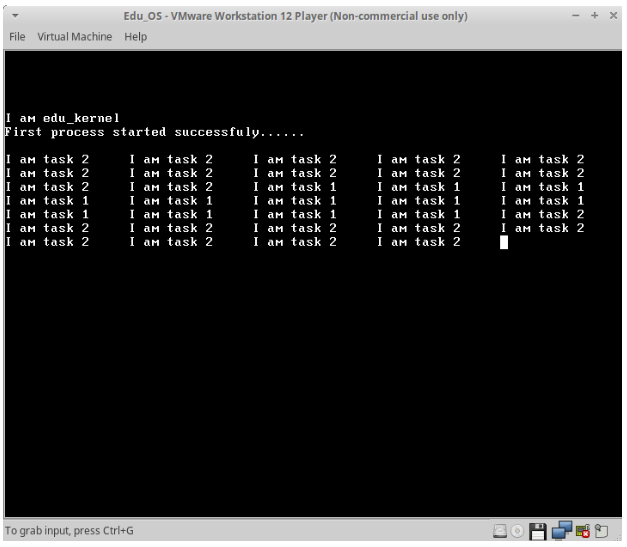


Figure 3: Core kernel block


Edu_OS - VMware Workstation 12 Player (Non-commercial use only)
File
Virtual Machine
Help

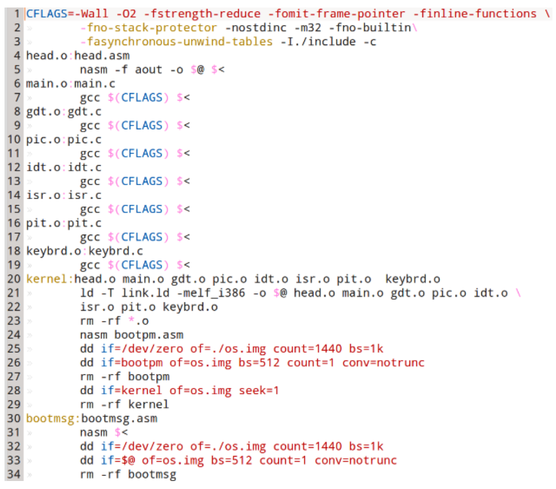
```
I am edu_kernel
First process started successfuly......

I am task 2      I am task 2      I am task 2      I am task 2      I am task 2
I am task 2      I am task 2      I am task 2      I am task 2      I am task 2
I am task 2      I am task 2      I am task 1      I am task 1      I am task 1
I am task 1      I am task 1      I am task 1      I am task 1      I am task 1
I am task 1      I am task 1      I am task 1      I am task 1      I am task 2
I am task 2      I am task 2      I am task 2      I am task 2      I am task 2
I am task 2      I am task 2      I am task 2      I am task 2      █
```

To grab input, press Ctrl+G

Figure 4: Task switching between two tasks. One print on screen “I am task 2” and the other prints “I am task 1”

The designer should follow a logical building process. Where he cannot add any block above the dummy block. For example, when a dummy kernel block is added the user cannot add process manager block or any other block. He can remove the dummy kernel, add the core kernel block then add the above blocks.

When finishing the design, a user can click on the build button. By clicking the build button, the application starts to include the needed file names in a Makefiles file (Figure 5), then make utility can starts the compilation and copying (to appropriate place) process. The copying (deployment) method used here is writing an image to a disk. "dd" utility is used to write the created image to a file that represents a floppy disk. Once these steps completed successfully, a user can click on "Run Image" button. Then the configured VMWare player, [20], runs the image of the built operating system.



```
CFLAGS=-Wall -O2 -fstrength-reduce -fomit-frame-pointer -finline-functions \
        -fno-stack-protector -nostdinc -m32 -fno-builtin\
        -fasynchronous-unwind-tables -I./include -c
head.o:head.asm
        nasm -f aout -o $@ $<
main.o:main.c
        gcc $(CFLAGS) $<
gdt.o:gdt.c
        gcc $(CFLAGS) $<
pic.o:pic.c
        gcc $(CFLAGS) $<
idt.o:idt.c
        gcc $(CFLAGS) $<
isr.o:isr.c
        gcc $(CFLAGS) $<
pit.o:pit.c
        gcc $(CFLAGS) $<
keybrd.o:keybrd.c
        gcc $(CFLAGS) $<
kernel:head.o main.o gdt.o pic.o idt.o isr.o pit.o  keybrd.o
        ld -T link.ld -melf_i386 -o $@ head.o main.o gdt.o pic.o idt.o \
        isr.o pit.o keybrd.o
        rm -rf *.o
        nasm bootpm.asm
        dd if=/dev/zero of=./os.img count=1440 bs=1k
        dd if=bootpm of=os.img bs=512 count=1 conv=notrunc
        rm -rf bootpm
        dd if=kernel of=os.img seek=1
        rm -rf kernel
bootmsg:bootmsg.asm
        nasm $<
        dd if=/dev/zero of=./os.img count=1440 bs=1k
        dd if=$@ of=os.img bs=512 count=1 conv=notrunc
        rm -rf bootmsg
```

Figure 5: Makefile file that is used to generate and deploy the OS image

## IV. RELATED WORK

To hide the complexity of real-world operating systems from new comers to the field, researchers proposed educational operating systems that concentrate on learning the implementation of basic concepts. The authors of [13] proposed a simulator to ease teaching operating systems concepts. [21] introduces an educational operating system that is built upon larc architecture that is used in computer architecture course. This operating system cannot be used on real world systems.

Tiny educational operating systems is proposed in [10] to help learners of OS courses. Ruth et. al. [22] proposed a new implementation of Embedded Xinu on Qemu virtual machine for operating systems course. Authors in [23], proposed an operating system that provides undergraduate students a platform for studying the multi-core systems. Tyrone Nicholas and Jerzy A. Barchans [24] described a distributed educational operating system (TOS) that is based on java. Felix and Juan-Carlos proposed webgeneOS [25] where students send their system programs via a network to run on the actual operating system then they receive back the results. BabyOS was presented in [26] as a compact operating system that can be used also for embedded systems besides helping learners of operating systems. In [14], the authors present a graphical simulator for teaching the operating system. Soetrisno Cahya, in [27], presented another simulator to help students in understanding the basics of operating systems. Fan Yile in [28], provides another virtual environment to run many operating systems for educational purposes. In [29], a complete user level operating system is proposed that can be used for teaching operating systems. In [30], the authors proposed MOS, another tiny operating system for students to play with.

None of the researchers succeeded in hiding the complexity of programming while building a real-life operating system.

The idea of visual development of programs and designs is not new. It is out there for developer and learners in many areas. Some examples are included here: Alice [31], TouchDevelop [32] (programming environment by Microsoft), Scratch [19] (coding tool by MIT for kids), Studio [33] (Studio for game creation by YoYo Games), and LabVIEW [34] (for engineers and scientists) to mention a small list. We adopted their idea of design easiness for the new comers to the field of operating systems.

## V. CONCLUSIONS

In this paper, a novel method for teaching and building operating system course was proposed. It is based on using small configurable code blocks. Code blocks are represented as visual modules, which can be added to each other in a controlled way to build a functional OS image. The power of this method is hiding the complexity of involving learners in programming (system-level programming).

The proposed system is in its building phase. It needs feedback from learners. As a future work, next operating systems courses in Tafila Technical University will be taught based on this design. Subsequently, feedback notes from teachers, students and teaching assistants will be analyzed and further modifications to the system will be conducted to reflect the needs of all parties. In addition, as a future work, this design can be on a server, where users can connect to and build their own customized OS then receive the built image for testing and experimenting.

## ACKNOWLEDGMENT

This work is totally supported by Scientific Research and Graduate Studies Deanship, Tafila Technical University, Jordan, under the grant number 2016/4. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the university.

## REFERENCES

[1] Nieh, Jason, and Chris Vaill. "Experiences teaching operating systems using virtual platforms and Linux." *ACM SIGCSE Bulletin*. Vol. 37. No. 1. ACM, 2005.

[2] Laadan, Oren, Jason Nieh, and Nicolas Viennot. "Structured Linux kernel projects for teaching operating systems concepts." *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011.

[3] Schmidt, Alexander, Andreas Polze, and Dave Probert. "Teaching operating systems: windows kernel projects." *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010.

[4] Hess, Rob, and Paul Paulson. "Linux kernel projects for an undergraduate operating systems course." *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010.

[5] Yodaiken, Victor. "Cheap operating systems research and teaching with Linux." *disponível on-line em http://luz. cs. nmt. edul-rtlinux* (1996).

[6] Román Otero, Rafael, and Alex A. Aravind. "MiniOS: An Instructional Platform for Teaching Operating Systems Projects." *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 2015.

[7] Andrus, Jeremy, and Jason Nieh. "Teaching operating systems using android." *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 2012.

[8] Nieh, Jason, and Chris Vaill. "Experiences teaching operating systems using virtual platforms and Linux." *ACM SIGOPS Operating Systems Review* 40.2 (2006): 100-104.

[9] Ayers, D., P. Smith, and P. Ashton. *SunOS Minix: a tool for use in Operating System Laboratories*. Department of Computer Science, University of Canterbury, 1993.

[10] Qu, Bo, and Zhaozhi Wu. "Design and Implementation of Tiny Educational OS." *Recent Advances in Computer Science and Information Engineering*. Springer Berlin Heidelberg, 2012. 437-442.

[11] Cox, Russ, M. Frans Kaashoek, and Robert Morris. "Xv6, a simple Unix-like teaching operating system." 2013-09-05]. http://pdos. csail. mit. edu/6.828/2012/xv6. html (2011).

[12] Cohen, Shimon. "Teaching Operating Systems Scheduling." *Proceedings of Informing Science and IT Education Conference*. 2010.

[13] Mustafa, Besim. "YASS: a system simulator for operating system and computer architecture teaching and learning." *European Journal of Science and Mathematics Education* 1.1 (2013): 34-42.

[14] Maia, Luiz Paulo, Francis Berenger Machado, and Ageu C. Pacheco Jr. "A constructivist framework for operating systems education: a pedagogic proposal using the SOsim." *ACM SIGCSE Bulletin* 37.3 (2005): 218-222.

[15] Jones, David, and Andrew Newman. "RCOS. java: a simulated operating system with animations." *Teaching package 1* (2001).

[16] Maia, Luiz Paulo, and A. C. Pacheco. "A simulator supporting lectures on operating systems." *Frontiers in Education, 2003. FIE 2003 33rd Annual*. Vol. 2. IEEE, 2003.

[17] Koh, Jeong-Gook. "CPUSim: A Simulator supporting the education of CPU Scheduling Algorithms." *Journal of the Korea Institute of Information and Communication Engineering* 16.4 (2012): 835-842.

[18] Tanenbaum, Andrew S., and Herbert Bos. Modern operating systems. Prentice Hall Press, 2014.

[19] Resnick, Mitchel, et al. "Scratch: programming for all." Communications of the ACM 52.11 (2009): 60-67.

[20] Workstation, VMware, and VMWare Player. "VMware Inc." *Palo Alto, California, USA* (2002).

[21] Corliss, Marc L., and Marcela Melara. VIREOS: an integrated, bottomup, educational operating systems project with FPGA support. Proceedings of the 42nd ACM technical symposium on Computer science education. ACM, 2011.

[22] Ruth, Paul, and Dennis Brylow. An Experimental Nexos Laboratory Using Virtual Xinu. Frontiers in Education Conference (FIE), 2011. IEEE, 2011.

[23] Ziwisky, Michael, Kyle Persohn, and Dennis Brylow. A down-to-earth educational operating system for up-in-the-cloud many-core architectures. ACM Transactions on Computing Education (TOCE) 13.1 (2013): 4.

[24] Nicholas, Tyrone, and Jerzy A. Barchanski. TOS: an educational distributed operating system in Java. ACM SIGCSE Bulletin. Vol. 33. No. 1. ACM, 2001.

[25] Buenda, Felix, and Juan-Carlos Cano. Webgene: A Generative and Web-Based Learning Architecture to Teach Operating Systems in Undergraduate Courses. IEEE Transactions on Education 49.4 (2006): 464-473.

[26] Liu, Haifeng, Xianglan Chen, and Yuchang Gong. BabyOS: a fresh start. ACM SIGCSE Bulletin 39.1 (2007): 566-570.

[27] Cahya, Soetrisno. Designing Operating System Simulator: A Learning Tool. Computer Modeling and Simulation, 2009. UKSIM'09. 11th International Conference on. IEEE, 2009.

[28] Yile, Fan. Utilizing the Virtualization Technology in Computer Operating System Teaching. Measuring Technology and Mechatronics Automation (ICMTMA), 2016 Eighth International Conference on. IEEE, 2016.

[29] Atkin, Benjamin, and Emin Gn Sirer. PortOS: an educational operating system for the Post-PC environment. ACM SIGCSE Bulletin. Vol. 34. No. 1. ACM, 2002.

[30] Li, Hongwei, et al. Construction of the practical teaching system on operating systems course. Education Technology and COMPUTER Science (ETCS), 2010 Second International Workshop on. Vol. 1. IEEE, 2010.

[31] Cooper, Stephen, Wanda Dann, and Randy Pausch. "Alice: a 3-D tool for introductory programming concepts." Journal of Computing Sciences in Colleges. Vol. 15. No. 5. Consortium for Computing Sciences in Colleges, 2000.

[32] Horspool, R. Nigel, and Nikolai Tillmann. "TouchDevelop: Programming on the Go. The Expert's Voice. Apress, 2013."

[33] Rohde, Michael. GameMaker: Studio for Dummies. John Wiley & Sons, 2014.

[34] Wells, Lisa K., and Jeffrey Travis. LabVIEW for everyone: graphical programming made even easier. Prentice-Hall, Inc., 1996.