# CrazyOS: A Case for Toy Operating Systems

Praneet Kapoor
*Electrical and Electronics Engineering*
*Dr. Akhilesh Das Gupta Institute of Technology and Management*
New Delhi, India
kapoorpraneet2619@gmail.com

Rachit Jain
*Electrical and Electronics Engineering*
*Dr. Akhilesh Das Gupta Institute of Technology and Management*
New Delhi, India
rachitjain24@gmail.com

Shristi Sharma
*Electrical and Electronics Engineering*
*Dr. Akhilesh Das Gupta Institute of Technology and Management*
New Delhi, India
shristisharma1808@gmail.com

Sumit Gaur
*Electrical and Electronics Engineering*
*Dr. Akhilesh Das Gupta Institute of Technology and Management*
New Delhi, India
sumit.gaur1999@gmail.com

Md. Saquib Faraz**
*Electrical and Electronics Engineering*
*Dr. Akhilesh Das Gupta Institute of Technology and Management*
New Delhi, India
mdsaquibfaraz@gmail.com

***Abstract*—Technical education of undergraduate students should ensure that they are provided with practical knowledge which makes them fit for the ever changing industry. One of the ways of accomplishing this is by attempting to include project based learning alongside regular coursework. This project is a step towards this direction. We have presented a single-user, single-tasking, real mode operating system. The operating system comes with two applications, a shell and a line editor. A library of subroutines has also been created which includes custom implementation of getchar, putchar, printf, strlen, strcmp, strcpy and atoi in x86 assembly language. With only 1690 lines of code, the source code of the operating system can be easily understood by students and then used as a sandbox for learning about the design of the 8086 microprocessor and the original IBM PC.**

***Index Terms*—Real mode, Operating system, QEMU, NASM, BIOS, Make, Makefile, shell, line editor, 8086, IBM PC.**

## I. Introduction

### A. Motivation

Engineering is a practical profession. A good engineer should be able to use fundamental principles to solve problems or derive practical results. The ability to do so requires experience, which is often termed as "*getting hands dirty*". It is of primary importance that an engineering student should have their hands sufficiently dirty in the profession they wish to pursue. A student interested in pursuing a career in silicon industry should have considerable experience in designing systems on an FPGA board. If he[1] has only memorized the important principles from some standard book, that student will have no value in the industry. It is because of this reason that academicians and professors all around the globe try to the best of their knowledge to keep the curriculum in sync with the advances that are being made in the industry. Despite all their efforts, there will always remain a few deficiencies in the roadmap thus produced. Education system of a few countries and the curriculum followed in a few top-tier institutions is considered as the gold-standard for technical education by other countries and institutions. An educated discussion about the about the state of engineering education in different countries across the globe will deviate us from the subject matter of this report. Therefore, we restrict our discussion to the state of technical education in India. This is appropriate as all the authors of this paper are of Indian nationality and have first hand experience of the education system.

In India, institutes offering technical education need to get certified by the All India Council for Technical Education (AICTE). AICTE has produced a model curriculum for technical education. Engineering Institutions can either follow this curriculum or develop their own using the model curriculum as a template. As the project presented in this paper is related to computer architecture and operating systems, we will take the example the course on Computer Architecture offered to Electrical Engineering undergrads [5]. There are six modules in this course. Module 1 tells about the differences between CISC and RISC, data types, system buses, and multi-bus organization. Module 2 then jumps to memory organization, followed by Module 3 discussing I/O devices, PCI and PCI Express bus. Module 4 then discusses x86 architecture, and its addressing modes and instruction. Module 5 discusses instruction level pipelining (ILP) and module 6 discusses other architectures like MIPS. Three flaws can be found in this course:

1) *Undirected and overloaded course structure*: The course, and other similar courses, can be considered to be well designed theoretically. However, they are overloaded with theoretical concepts. It would be better to just focus on the architecture of single computer, preferably IBM PC. The course can then describe the 8086 microprocessor and later describe ILP, and paging. Then the architecture of the IBM PC can be dealt with in detail.
2) *Impractical course structure*: Continuing with the ex-

[1]"He" should be read as "he or she" throughout this paper.

ample provided previously, most of the courses dealing with microprocessors start of with 8085 microprocessor. Though it is still used in legacy systems in the industry, no new projects are developed using it. Students would be better served by teaching them about ARM or RISC-V or x86 architectures as most of the computing systems today are built around them. Furthermore, it must be noted that the practical assignments given to students are trivial. Examples include writing subroutines to multiply numbers, or to move strings in 8086's assembly language. A more practical exercise would be to write subroutines to print a string on the screen or to separate words out from a string.

3) *Obsolete tools*: Most of the tools, be it teaching resources or software tools, used in Indian schools and colleges are obsolete. Instead of following texts like *The 8086 Family User's Manual* [1], students and teachers alike follow notes and presentations which don't talk about the practical side of various concepts like segmentation, effective address calculation and string operations, to name a few. Courses on C programming and data structures still require students to develop and run code in Turbo C environment. This is ridiculous because Turbo C's compiler does not follow the standard implementation of C and C++. In the course on microprocessors, codes are run on expensive trainer kits or on emulators, like emu8086, which is not the industry standard for developing 8086 assembly code.

Disgruntled by the approach taken in teaching one of his favourite subjects, one of the authors of this paper (Kapoor), decided to write his own 16-bit operating system from scratch. He considered this to be fun challenge which would allow him to tinker with the hardware and understand the architecture of 8086 microprocessor and the original IBM PC. His objective at first was to develop a UNIX like operating system. However, as UNIX and its derivatives (Linux, MINIX [10], BSD, etc.) are complex systems and developing them would require ample amount of free time, it was decided that operating system so designed would be a simple one consisting of just a primitive shell and line editor. Basic commands can be executed from shell. Students can develop the their own programs using the libraries provided by the author, assemble using NASM, create a virtual disk and run it in real mode on QEMU's emulation of i386 based IBM PC. Due to such unusual design of the operating system, it was named *CrazyOS* by its creator.

### B. Literature Survey

To develop concrete objectives, a brief of review of histor of computing systems was made. History of computers involve studying their evolution. Just like a study of human history consists of the study of various civilizations and generations, a study of computer history involves the study of various generations of computers. Beginning of a new generation are marked with the usage of newer, better, and reliable technology in comparison to the previous generation. The technology most used for this classification is the hardware. By using hardware as the basis for classification, most authors divide computer history into five generations, starting in 1951 with the launch of UNIVAC I [4][8]. By using programming languages and operating systems as the basis for classification, we arrive at similar results [10]. Table I shows various generations of computers and the operating system used in those generations. After reviewing the history of computing systems, it was decided that the operating system will be developed in the spirit of MS-DOS v1.25 whose source code has been made public by Microsoft [9].

### C. Objectives

The objective of this project was to develop a simple single-user, single-tasking, real mode operating system from scratch for educational purposes. This objective was achieved by completing the following goals:

1) To develop a bootloader from scratch.
2) To develop a set of libraries containing certain standard functions found in C's `stdio.h` and `string.h` libraries. This would assist the user in building disk applications for the system.
3) To develop a shell which enables the user to issue basic commands to interact with system components like disks, APM, etc.
4) To develop file system should be simple enough to understand, implement and use with programs like the line editor. For these reasons, the file system uses contiguous sector allocation [10].
5) To develop a build environment which allows the user to write and build their own disk applications, and run them using the operating system.

## II. Implementation of CrazyOS

### A. Set-up of the Development Environment

The development environment of CrazyOS consists of five software tools: code editor, assembler, emulator, build automation software, and source version control software.

Initially, the author of the project (Kapoor) used gVim for writing and editing source code. However, as the size of the project grew, the author found himself to be unable to work productively with gVim. For this reason, a switch to VScode was made.

NASM was chosen over GNU AS because of its support for Intel syntax [3]. Most technical institutions use Intel syntax during course on 8086 processor. Using NASM meant that students learning about x86 architecture won't have to learn a new syntax (AT&T syntax) of writing assembly code.

QEMU was used for emulating an i386 based IBM PC [6]. GNU Make was used for automatically building binaries, running the emulator, and removing the binaries. Git was used for tracking the files in this project.

### B. Structure of the Project Directory

CrazyOS started out as a project to teach its author about the 8086 microprocessor and the IBM PC. The author expects to write similar toy operating systems for computers using

TABLE I: Generations of Computers on the Basis of their Hardware

| Generation | Year | Hardware | Operating System |
|---|---|---|---|
| 1st Generation | 1951-1960 | Vacuum tubes | Non-existent |
| 2nd Generation | 1959-1965 | Transistors | Batch Processing |
| 3rd Generation | 1964-present | Integrated circuits | Multiprogramming, Time-sharing, UNIX |
| 4th Generation | 1971-present | VLSI, Microprocessors | Disk Operating System (DOS), Linux distros, Windows NT |
| 5th Generation | Present-future | Parallel processing architecture | |

i386, x86-64, ARM and RISC-V processors. These ambitions are reflected in the commit history and by the structure of the parent directory. Assuming the reader clones the project from Github into a directory named `CrazyOS`, the directory tree that will be obtained is shown in Figure 1. *depth = 0* indicates the number of files and subfolders that are being shown in the source tree.

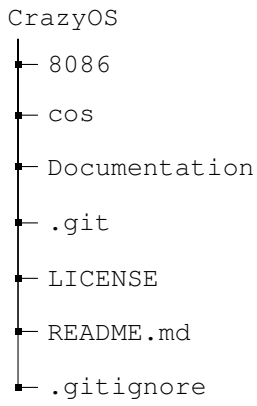


Fig. 1: Parent directory tree (depth = 0)

As has been mentioned before, the directory `../8086`[1] contains the source files for the project. The tree of this directory is given in Figure 2.

### C. Bootloader and Kernel Header

The bootloader was the first program written in this project. It is contained in the `../8086/boot` directory. It is a very simple bootloader which uses standard BIOS interrupt no. 13 to load the operating system from disk into the main memory. The bootloader loads 32 sectors from the first disk at address 1000:0000. It then performs an intersegment call to `kernel_entry` subroutine present in `../8086/kernel/kernel.asm`. If `kernel.asm` forms the head of the operating system, then `kerne_entry` subroutine would be the scalp and `kernel_main` would be the cranium of the operating system. The bootloader performs an

[1] `..` is used to refer to the parent directory, in this case `CrazyOS`. In fact, `CrazyOS` is itself a part of some directory and therefore it can be written as `../CrazyOS`.

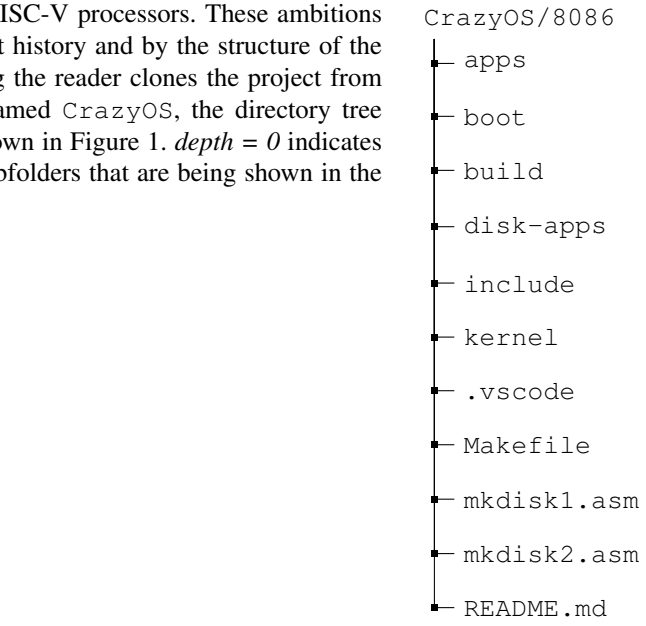


Fig. 2: Directory tree for `CrazyOS/8086` (depth = 0)

intersegment call to `kernel_entry` subroutine which has been loaded, along with the rest of the operating system, in block 1. `kernel_entry` subroutine is responsible for initializing `cs`, `ds`, `es`, and `ss`.

```
kernel_entry:
        mov ax, cs
        mov ds, ax
        mov es, ax
        mov ax, 0x9000
        mov ss, ax
        mov bp, 0x9ffe
        mov sp, bp
```

`kernel_main` calls the `main` subroutine in `../apps/shell/shell.asm`. This launches the shell from where the user can execute various commands.

### D. Library Functions

In CrazyOS, library functions are placed in `../8086/include` folder. Source tree of this directory is shown in Figure 3. These functions use BIOS calls to access hardware facilities. In fact, the whole operating system is heavily dependent on BIOS interrupt calls for

device management. These BIOS calls can be thought of as CrazyOS' system calls. These interrupts were studied and used by following *Ralph Brown's Interrupt List* [7]. The subroutines present in the library are analogous to functions present in the standard C library such as `printf`, `scanf`, `atoi`, etc., which a programmer often uses in their application to get the job done. Subroutines dealing with printing text on screen and taking input from the keyboard are in `../include/ttyio` directory. Subroutines performing string operations such as `strlen` (for finding length of the string), `strcmp` (for comparing two strings), and `atoi` (for converting a number in string format into a positive integer) are present in `../include/string/string.asm` file.
`..include/string` directory has another assembly source file besides `string.asm`. It is `lexer.asm` which is responsible for breaking a string of characters into separate "words" using some separator. The function used for performing this primitive lexical-analysis is `cmd_lexer`. It *Tokenizes* the string that is being pointed at by `si` using the character stored in `al` as a separator. It generates its own personal copy of the string which is stored in local label `.COPY`. This subroutine is capable of generating a maximum of 5 tokens: one command token and four argument tokens. Command token is stored in global label `com` and argument tokens are stored in `arg1`, `arg2`, `arg3`, and `4` which are also global labels.
Subroutines which allow the user to read and write to disks are in `../include/disk` directory. `../include/apm` contains subroutines for using Advanced Power Management (APM) for managing power of the device (or in this case, an emulator).
Time and date is obtained from CMOS memory on RTC on the motherboard using the subroutines present in `../include/cmos` directory. The subroutines in this directory are also used to produce pseudo-random numbers by `srng` subroutine in `../include/random/srng.asm`. These pseudo-random numbers are generated by the multiplication of seconds count in CMOS with a 16-bit prime number and then addition of the resulting product with a different 16-bit prime number.
Finally, subroutines present in `../include/sound` directory are used to produce sounds by feeding the piezoelectric speaker on the motherboard with a square wave of chosen frequency. This square wave is generated by 8254 Programmable Interrupt Timer (PIT) chip [2].

```
CrazyOS/8086/include
├── apm
│   ├── apm_com.asm
│   └── apm.asm
├── cmos
│   ├── cmos_com.asm
│   └── cmos.asm
├── disk
│   ├── disk_com.asm
│   └── disk.asm
├── random
│   └── srng.asm
├── sound
│   └── sound.asm
├── string
│   ├── lexer.asm
│   ├── README.txt
│   └── string.asm
└── ttyio
    ├── legacyprint.asm
    ├── printf.asm
    ├── README.txt
    ├── stdtty.asm
    └── ttyio.asm
```

Fig. 3: Directory tree for `CrazyOS/8086/include` (depth = 1)

### *E. Shell*

The shell is considered as the *mother of all programs*. It is one of the first programs which runs in most operating systems. It provides the user an environment where he can issue commands and run other programs [11]. CrazyOS' shell is implemented by the source code present in `../apps/shell` directory. This directory contains two source files: `shell.asm` and `cmdlist.asm`. There is only a single subroutine present in `shell.asm`, `main`. This subroutine firstly prints the prompt, `>`, using `printf`. Then it calls `getline` subroutine which takes input from the keyboard until the enter key gets pressed. Then using a blank space as the token separator, `lexer` splits the line into command word and arguments which are then executed.
The commands which are supported by the shell are present in `cmdlist.asm`. `com` string is compared with the strings present in this file one by one until a match is found. If no match has been found then an error message is printed on the screen.

### *F. File System and the Line Editor*

CrazyOS uses contiguous file allocation which stores files in consecutive blocks of memory [10]. This makes the design

of the operating system even simpler. To read or write a file to a disk, the user just needs to read consecutive blocks till the end of the file has been reached. End of text files made using editor provided with CrazyOS is indicated by a byte having the value 0xaa.
This simple file system is used by the line editor whose source code is present in `../apps/edit` directory. The line editor is started by typing `ed SEG:OFF` in the shell. Here, `SEG` is the 16-bit segment base address and `OFF` is the 16-bit offset from the base address at which the contents of file will be stored. Due to the use of segments, the size of the file cannot exceed 64 kiB or about 819 80-character wide lines. Also, each file is terminated with a magic number, 0xaa. This hex-number maps to *á* in code page 437. As in this simple project we are not using accented letters, therefore, we do not have to worry about the loss of characters which might happen to exist in the file buffer after 0xaa.
Once the editor starts, the user is told about the number of characters that have been found in the file buffer. Like `ed`, `edit` allows the user to enter a single line at a time. The line so entered should not exceed 80 character limit. If the first character of the line is an asterisk, than the subsequent characters are considered as editor commands and are not stored in the file buffer. The file can be saved permanently by using `disk write` command to write the buffer to the disk.

### G. Disk Applications: ../disk-apps

As we have discussed before, applications which come on a separate disk are called disk applications in this project. Source code for disk applications are saved at `../disk-apps` directory. At present, the whole project has only a single disk application, namely `soundnlight.asm`. This assembly file contains code which reads a single byte from the kernel (present in block 1), and

1) displays a pixel having a color corresponding to the value of the byte that has been read using INT 10,0C BIOS call.
2) generates a note corresponding the value of the byte that has just been read using `sound_tiny` subroutine present in `../include/sound/sound.asm`.

Then the pointer `si` is incremented and the next byte of the kernel is read. The screen is assumed to have a size of $640 \times 480$ pixels. Other available screen resolutions are $640 \times 350$ and $320 \times 200$ pixels.

## III. Results & Discussion

### A. Cloning & Running CrazyOS

Since its inception, CrazyOS's version control has been done using Git and the repository has been hosted on Github. To download the source files of the project on a local machine, the user must first install Git and generate SSH keys and link them with his account on Github. Once these preliminary tasks have been done, the user has to type the following command in a terminal to clone CrazyOS:

```
$ git clone git@github.com:PraneetKapoor2619/CrazyOS.git
```

The $ should be ignored; it is just a standard way of telling readers that they should type the text ahead in a terminal.
Once the project has been cloned, the reader has to make sure that the version of CrazyOS that he uses has the short commit ID 1a17fa0. For this, the following command has to be entered in the terminal:

```
$ git reset --hard 1a17fa0
```

To move to the project directory, the following command is executed:

```
$ cd CrazyOS/8086/
```

The reader can use the latest version of the operating system but it might not be stable. The reader should refer to commit messages if he wishes to the latest version of CrazyOS.
Once in the project directory, the user must create a build directory by the following command:

```
$ mkdir build
```

The binary files and the disk images are built using `make build` command. To run the operating system, type

```
$ make run
```

To exit QEMU without powering off the emulator from CrazyOS's shell, just press `Ctrl + Alt + Q`. `build` directory is cleaned by using `make clean` command. To remove `asm.bin` and `disk2.bin` in the `build` directory, the following commands should be used:

```
$ rm build/asm.bin build/disk2.bin
```

### B. Using the Shell

Upon running the emulation, the user will be greeted with a startup screen showing a text message, and current time and date, as shown in Figure 4).


Fig. 4: CrazyOS startup screen

User can enter commands into the shell to get their job done. An example is shown in Figure 5, where `time`, `date`, and `srng` commands have been used.


Fig. 5: Using `time`, `date`, and `srng` commands

### C. Bugs

The project has only three known bugs which is astonishing because it has been completely written in assembly language. These bugs are:

1) `power level` command is not able to show the charge in the battery of the machine on which the emulator is running..
2) Executing an invalid command string in the line editor insert that command string except for the asterisk.
3) System might become unstable if an intersegment call is performed.

### D. Discussion

CrazyOS is comparable to disk operating systems (DOS) like CP/M and MS-DOS v1.25. Both MS-DOS v1.25 and CrazyOS have many similarities, few of which are:

1) Both operating systems work in real mode.
2) The command-prompt/shell of both of these operating system has commands built into the source code of the shell itself.
3) Both operating systems make use of BIOS calls to access the hardware.
4) Both of these operating system are meant for a single user only.
5) Neither CrazyOS nor MS-DOS v1.25 has the ability to multitask between different software.

However, unlike MS-DOS, CrazyOS

1) does not have any system calls which makes it heavily dependent on the BIOS calls provided by the OEM.
2) does not support running applications having binary size greater than 64 kiB. Returning to shell from smaller applications after an intersegment call makes the shell unstable.
3) does not use proper file allocation table (FAT). Therefore, disk space is not utilized efficiently.

All modern day operating systems, like Linux kernel based distros and Windows 10, are far superior than DOS. They are one of the most complex pieces of software in the world which provide programmers and everyday users with hardware support, system calls, and file systems among many other things which allow them to use the resources of the machine efficiently and get their job done. With millions of lines of codes, even tech-gurus can only master a few integral components of the operating system in their lifetime. Clearly, the simplicity of CrazyOS comes at the cost of functionality and makes it a toy operating system which can be used for educational purposes. Porting CrazyOS for x86-64 and ARM based processors while implementing a few fundamental system calls and a file system will tremendously improve its usefulness.

## IV. Conclusion & Future Scope

### A. Conclusion

We have successfully implemented a single-user, single-tasking, real-mode operating system named CrazyOS. The operating system comes with two applications: a shell and a line editor. Both of these applications have been demonstrated to work successfully. Various subroutines responsible for handling devices and for processing user commands form the library of the operating system. These library functions can be used to develop various disk applications which can then be run from the shell, as has been successfully demonstrated by running `soundnlight.asm` program.

### B. Future Scope

The total lines of code which are there in the project directory can be computed with the help of `cloc` utility. It is revealed that there are in total 1718 lines of assembly code. If one excludes the number of lines of code in `soundnlight.asm` program, then the total lines of code for the project comes out to be 1690. Both of these numbers show that the operating system is small enough to be read, understood, used, and played by a student over the course of their undergraduate study. If it is incorporated correctly in courses related to computer architecture and x86 microprocessor family, this operating system can prove to be useful for disseminating practical knowledge of the aforementioned project.

The project can be further expanded in the following ways:

1) As an educational exercise, versions of CrazyOS working on computers using i386, x86-64, ARM and RISC-V processors can be developed.
2) Documentation related to each version of CrazyOS can be published online. This will help in increasing the number of people who learn from it.
3) A separate operating system can be developed for Raspberry Pi line of single board computers (SBC). These are easily and cheaply available all around the world. When playing with a toy operating system on a cheap SBC, the user will not have to worry about bricking their main computer. Furthermore, as RPis are increasingly being used in embedded systems, a lightweight operating system will prove to be a boon for embedded systems developers.

We would conclude this paper by quoting the words of Linus Torvalds[11] and Terry Davis This operating system has been developed because it was a fun challenge. Unlike Linux

or Windows, it is not a professional operating system. In comparison to them, it is just a dirt-bike using which the user can go on accomplishing fun little challenges of their own liking.

## ACKNOWLEDGMENT

We would like to thank our family and friends for supporting us during tough times.

## REFERENCES

[1] *The 8086 Family Users Manual*. Intel Corporation, 1979.

[2] *8254 Programmable Interval Timer*. Intel Corporation, 1993.

[3] *The NASM Documentation*. The NASM development team, 2015. URL https://www.nasm.us/xdoc/2.15.05/html/nasmdoc0.html.

[4] Ionescu Andreea. Categories and Generations of Computers. *European Journal of Computer Science and Information Technology*, 3(1):15–42, 2015.

[5] Prof. Anupam Basu et al. *Model Curriculum for Undergraduate Degree Courses in Engineering & Technology*. AICTE, 2018.

[6] Fabrice Bellard. Qemu, a fast and portable dynamic translator. 41(46):10–5555, 2005.

[7] Ralph Brown and Marc Perkel. *Ralph Brown's Interrupt List: Indexed HTML Version - Release 61*. URL https://www.ctyme.com/rbrown.htm.

[8] Linda Weiser Friedman. From Babbage to Babel and Beyond: A Brief History of Programming Languages. *Computer Languages*, 17(1):1–17, 1992.

[9] Tim Paterson. MS-DOS v1.25. https://github.com/microsoft/MS-DOS/tree/master/v1.25, 2018.

[10] Andrew S. Tannenbaum and Albert S. Woodhull. *Operating Systems: Design and Implementation*. 2003.

[11] Linus Torvalds and David Diamond. *Just for fun: The story of an accidental revolutionary*. Harper Audio, 2001.