Emulating an Operating System

Aaditya Naik, Aditya Chakraborti, Aditya Pansare, Utkarsh Pant

***Abstract*— The purpose of this report is to understand the procedures and practical applications of various operating system phenomena by emulating an operating system. The operating system, called E-OS, is a minimal and lightweight operating system, built to be modular and exploit the benefits of object-oriented programming.**

I. Introduction

EMULATED OS, abbreviated to E-OS, is an operating system built from a modular approach featuring various necessities of a basic operating system. The aim of this project is to gain insight into the various requirements of an operating system while keeping the source code clean and easy to understand. While E-OS does not directly interact with the hardware of the machine and operates through a windows kernel, it still emulates the functioning of the main modules found in an operating system. E-OS follows a modular approach, which allows specific modules to be changed while still interacting with other modules by means of special function calls. This offers a layer of abstraction between any two modules.

II. A Basic Overview

E-OS is a minimalistic emulation of a few functionalities of a generic personal operating system. The user can interact with the operating system and pass several commands. A full list of these commands will be talked about in the COMMANDS section.

The architecture of E-OS consists of four main modules- The shell, which interacts with the user, a process management module, which is responsible for creating and terminating processes, a memory management module, responsible for memory allocation and deallocation and finally a file management module, responsible for various file management functionalities like the reading of a file and writing to a file.

A. The Shell

The Shell is designed as an interface for the user to interact with the operating system using easy to follow commands- a more reliable and less risky way of exploiting the operating system features. The Shell accepts inputs into a buffer that can accommodate a maximum of 100 characters, and the parses the user’s input to obtain the command given and the arguments passed. It is important to note that all arguments are space

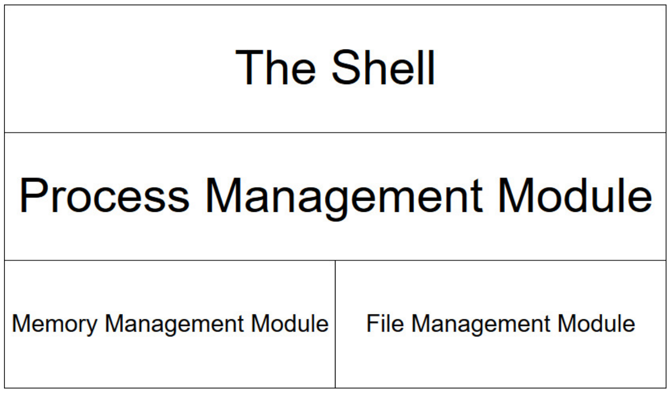


Fig. 1- The Architecture of E-OS

separated and that using space in a n3ame or a command will result in the operating system to interpret it as two arguments or one command and one argument. Escape characters are not yet supported in the E-OS shell.

The basic functioning of the shell is as follows. Once the system boots up, the shell asks for a user login. The password is hidden for security. When the user logs in successfully, the shell shows a prompt, which indicates that the shell is ready to receive a command.

There are two main types of commands the shell identifies- commands that it can perform itself, and commands that need assistance from the processing module. The commands that the shell itself can perform include basic echo commands, commands to print out the user name, basic arithmetic, etc. For commands needing file access or other resources like processes, where the shell cannot, by its own capabilities, perform the command, the shell calls a special function in the process management module, which then interacts with the necessary modules. The output of any command is displayed without a prompt on the shell console.

B. The Process Management Module

Process management in EOS is laid out like so: The Process Management module lies below the Shell, using which the user issues commands to create processes that perform specific functions. Owing to this responsibility of interacting with the File Management module, the Process Management Module is above the Memory Management Module and File Management Module.

Every time a user issues a specific command, a process is created and added to memory, and provided with a *command code* that specifies the purpose of that process. The process stays in memory until it is executed. There are many processes that have to be handled, and for that we need many scheduling algorithms. EOS uses the round-robin algorithm, which is based on the philosophy of "providing a fair, definite time for each process to execute itself, and after the time slice, next process takes over". As a process completes execution, it is removed from memory. A Process Allocation Table is maintained to track the start-address and size of every process. The contiguous allocation method has been simulated in EOS. The Process Management Module takes care of simple shell commands, read operations, writes operations and other tasks in the system. Every process is a separate entity which coexists with other processes. These entities have well defined structure.

C. The Memory Management Module

The Memory Management Module, essentially deals with various logical functions for storage, retrieval and other "optimization" services. Unlike "The Shell", this module is not directly accessible by the user and the reason for its abstraction is the security of the system. It is possible that a user might not know the logical organization of the module and may get confused to consequently hamper or alter the module components. This will result in deregulation and "crashes" of the system.

This module resides between the physical storage mechanism and the Process Management Module and alongside File Management Module, as per design architecture. Whenever something needs to be stored or retrieved, the Process Management Module creates a "process" that communicates with the Memory Management Module with a command request and supplies the parameters of the target file or data. The Memory Management Module accepts these parameters and performs the "store" or "retrieve" functions and returns a result to the "Process Management Module". Based on this result, further actions can be taken.

This module as implemented in the project has a few limitations resulting in some constraints. Here a file or a process is stored in contiguous memory locations at a stretch in the memory block. There is a distinction between "Files" and "Processes" in the implemented "Memory Space". "Memory Allocation Tables" consists of details of the stored file or process such as the name, the size and the "start block index" of the memory. These allocation tables provide the OS with the information about the "processes and files" in the memory and thus efficient access, loading or offloading is brought about in the memory. Ideally, the processor accesses or executes files or processes the data that is residing in the "Main Memory".

D. The File Management Module

The file management module of EOS manages all of the files stored in the secondary memory array. It has a set of functions via which the module interacts with files. It is the bridge between a process and the files stored in the secondary memory.

If a user wants to interact with a file, they will type the appropriate command in the shell. The shell will then create a process to interact with the file. The process will contain the name of the file which it will pass on to the file management module. The file management module will then utilize its functions to interact will the file.
Some of these functions are:

1) Create a file or write to a file

This will create a new file or overwrite a file. It will generate an ID, making sure that no two IDs are ever same. The size of the file will be calculated based on the length of the data, fetch the starting position of the file in the memory array from the Memory Management module and will take the data from the user and put it in the file.

2) Read a file

This will allow the user to read the contents of a file by specifying the file name to the shell. If the file does not exist, then an error message is returned.

3) Append a file

This function allows to add data to a previously created file. It will take the name of the file to find the file and append the new data to that file. If the file is not found, or the file does not exist, it will create a new file with the file name being the name that was specified to the shell by the user, and it will then store the new data in that file.

4) Delete a file

This function provides the functionality to delete a previously created file. Deletion will only be successful if the user has the authority to delete file. If, the file name specified does not exist, then no file would be deleted and an error message would be returned.

The files that are created, are stored in the memory array in a contiguous manner. This is an implementation of contiguous file allocation. In this, blocks of the files are stored in a sequential manner in the memory array. Since, contiguous file allocation is not an efficient user of memory, and leaves chunks of memory that is wasted, compaction is done to reduce the wasted memory.

The File Management module has a File Allocation Table (FAT) to specify and recognize the addresses of each file. It will contain the name, the starting location and the size of a file. Every time a file is created, a new entry in the FAT table is assigned. This way any file stored in the memory can be located.

The File Management module also provides the functionality of providing access to processes to read the FAT table and find the file location.

III. COMMANDS

This is a handy list of commands to try out on the command line.

A. exit

Exits the shell, shutting down the OS. Takes no arguments.

B. echo

Echoes your arguments back at you.

C. *who*

Tells you who you are. Takes no arguments.

D. *help*

Displays all inbuilt commands and their description. Takes no arguments.

E. *cow*

Shows an ASCII cow. Takes no arguments.

F. *add*

Adds the arguments together.

G. *sub*

Subtracts argument 2 from argument 1. Takes 0, 1 or 2 arguments.

H. *mult*

Multiplies all arguments.

I. *div*

Divides argument 1 by argument 2. Takes 0, 1 or 2 arguments.

J. *addusr*

Adds a new user to the system.

K. *remusr*

Removes a user. Takes one argument.

L. *chpass*

Changes a user's password. Takes no arguments.

M. *login*

Takes you to the login screen.

N. *roundrobin*

Processes the arguments in a round robin fashion. Takes multiple processes as arguments, each process being represented by 3 arguments- the name, the size and the burst time.

O. *lf*

Lists files. Takes no arguments.

P. *fat*

Displays the file allocation table. Takes no arguments.

Q. *read*

Displays the contents of the file passed. Takes 1 argument.

R. *write*

Writes a file to memory. Takes multiple arguments, the first being the file name, and the rest being the contents.

S. *append*

Appends the contents to a file in memory. Takes multiple arguments, the first being the file name, and the rest being the contents.

T. *rm*

Remove a file from memory. Takes one argument.

U. *addprc*

Adds a process to volatile memory until specifically removed. Takes 2 arguments, the process name and size.

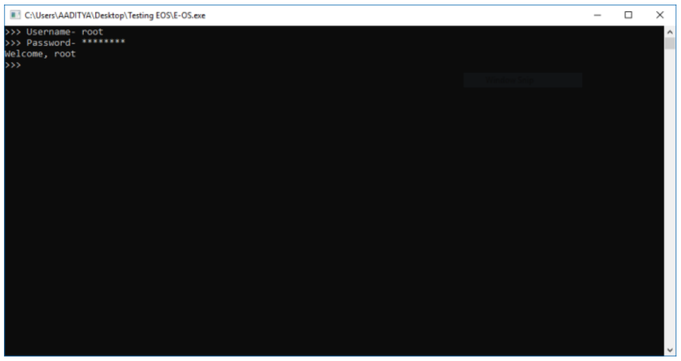
V. *delprc*

Removes process from volatile memory. Takes 1 argument.

W. *pat*

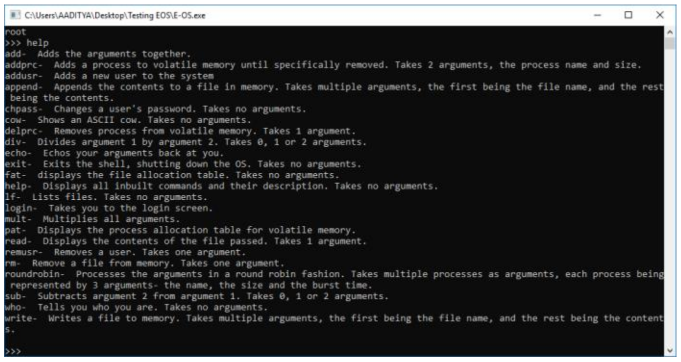
Displays the process allocation table for volatile memory.

IV. SCREENSHOTS


C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

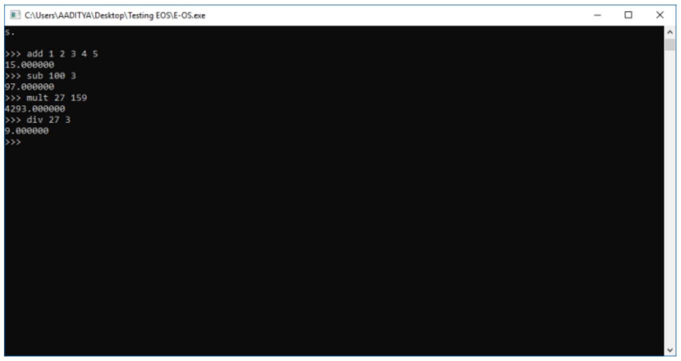
```
>>> Username- root
>>> Password- ********
Welcome, root
>>>
```

Fig. 2- Logging into the system


C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

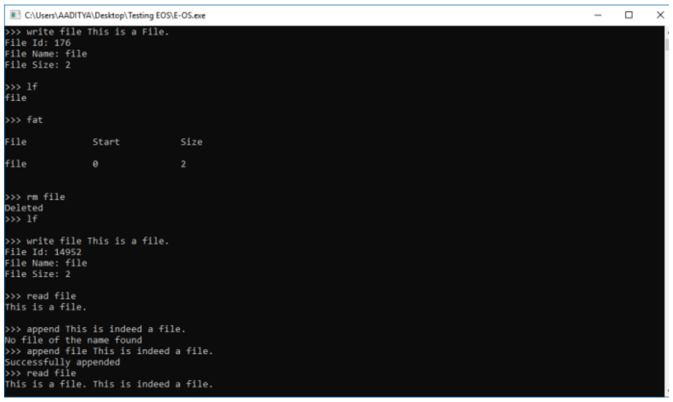
```
root
>>> help
add-  Adds the arguments together.
addprc-  Adds a process to volatile memory until specifically removed. Takes 2 arguments, the process name and size.
addusr-  Adds a new user to the system
append-  Appends the contents to a file in memory. Takes multiple arguments, the first being the file name, and the rest
 being the contents.
chpass-  Changes a user's password. Takes no arguments.
cow-  Shows an ASCII cow. Takes no arguments.
delprc-  Removes process from volatile memory. Takes 1 argument.
div-  Divides argument 1 by argument 2. Takes 0, 1 or 2 arguments.
echo-  Echos your arguments back at you.
exit-  Exits the shell, shutting down the OS. Takes no arguments.
fat-  displays the file allocation table. Takes no arguments.
help-  Displays all inbuilt commands and their description. Takes no arguments.
lf-  Lists files. Takes no arguments.
login-  Takes you to the login screen.
mult-  Multiplies all arguments.
pat-  Displays the process allocation table for volatile memory.
read-  Displays the contents of the file passed. Takes 1 argument.
remusr-  Removes a user. Takes one argument.
rm-  Remove a file from memory. Takes one argument.
roundrobin-  Processes the arguments in a round robin fashion. Takes multiple processes as arguments, each process being
 represented by 3 arguments- the name, the size and the burst time.
sub-  Subtracts argument 2 from argument 1. Takes 0, 1 or 2 arguments.
who-  Tells you who you are. Takes no arguments.
write-  Writes a file to memory. Takes multiple arguments, the first being the file name, and the rest being the content
s.

>>>
```

Fig. 3- Basic commands


C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

```
s.

>>> add 1 2 3 4 5
15.000000
>>> sub 100 3
97.000000
>>> mult 27 159
4293.000000
>>> div 27 3
9.000000
>>>
```

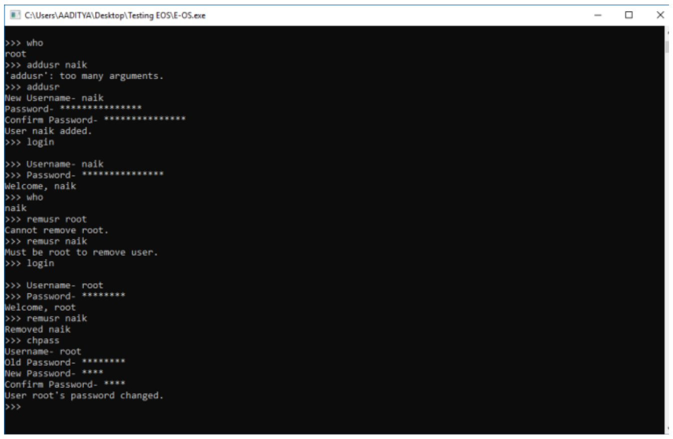
Fig. 4- Basic arithmetic on the shell



C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

```
>>> write file This is a File.
File Id: 176
File Name: file
File Size: 2

>>> lf
file

>>> fat

File            Start           Size

file            0               2


>>> rm file
Deleted
>>> lf

>>> write file This is a file.
File Id: 14952
File Name: file
File Size: 2

>>> read file
This is a file.

>>> append This is indeed a file.
No file of the name found
>>> append file This is indeed a file.
Successfully appended
>>> read file
This is a file. This is indeed a file.
```

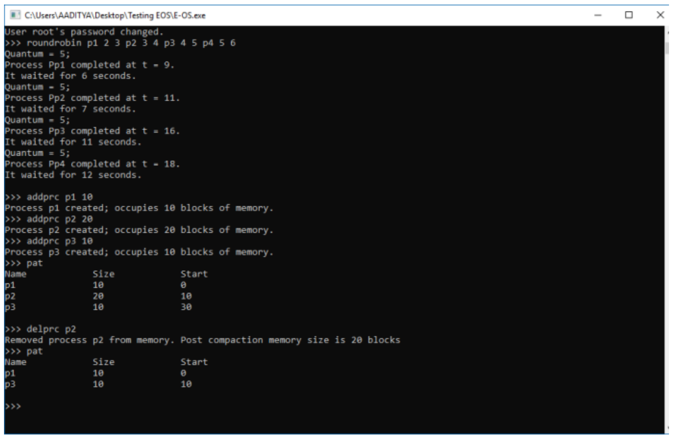
Fig. 5- File making and editing commands



C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

```
>>> who
root
>>> addusr naik
'addusr': too many arguments.
>>> addusr
New Username- naik
Password- ***************
Confirm Password- ***************
User naik added.
>>> login

>>> Username- naik
>>> Password- ***************
Welcome, naik
>>> who
naik
>>> remusr root
Cannot remove root.
>>> remusr naik
Must be root to remove user.
>>> login

>>> Username- root
>>> Password- ********
Welcome, root
>>> remusr naik
Removed naik
>>> chpass
Username- root
Old Password- ********
New Password- ****
Confirm Password- ****
User root's password changed.
>>>
```

Fig. 6- Modifying users and passwords



C:\Users\AADITYA\Desktop\Testing EOS\E-OS.exe

```
User root's password changed.
>>> roundrobin p1 2 3 p2 3 4 p3 4 5 p4 5 6
Quantum = 5;
Process Pp1 completed at t = 9.
It waited for 6 seconds.
Quantum = 5;
Process Pp2 completed at t = 11.
It waited for 7 seconds.
Quantum = 5;
Process Pp3 completed at t = 16.
It waited for 11 seconds.
Quantum = 5;
Process Pp4 completed at t = 18.
It waited for 12 seconds.

>>> addprc p1 10
Process p1 created; occupies 10 blocks of memory.
>>> addprc p2 20
Process p2 created; occupies 20 blocks of memory.
>>> addprc p3 10
Process p3 created; occupies 10 blocks of memory.
>>> pat
Name            Size            Start
p1              10              0
p2              20              10
p3              10              30

>>> delprc p2
Removed process p2 from memory. Post compaction memory size is 20 blocks
>>> pat
Name            Size            Start
p1              10              0
p3              10              10

>>>
```

Fig. 7- Process management commands

V. Contributions

A. The Shell- Aaditya Naik

B. Process Management Module- Utkarsh Pant

C. Memory Management Module- Aditya Pansare

D. File Management Module- Aditya Chakraborti

VI. Conclusion

A downscaled replication of a few functionalities and components was achieved using simple, primitive programming concepts on an object-oriented C++ platform. This project has its own limitations owing to excessive programming necessity, proper conceptual distinction and showcasing of various conditional branches on various actions. Although, all these could be brought down to discussion and certain parameters can be assumed to consequently give us a valid small-scale model of an "Operating System". Apart from the generic implementation, we could test our creativity in a few areas which helped us understand the base concept and create an attitude "to tackle- come what may".

Acknowledgment

We, the development team of EOS are delighted over the completion of the project and have thoroughly enjoyed working on the same. We would like to thank Dr. Dhirendra Mishra (MPTSME, NMIMS) who assigned us this project as a part of our evaluation structure. We observed that there were limitations regarding the implementation of the idea as conceptualized, but we were happy to move past them and try to figure out alternative and analogous solutions around them and make the project run. We, definitely, have gained a life-long learning experience and have realized how teamwork is essential even in small projects and to perfect an idea that is conceived.

References and Footnotes

References

[1] Andrew S. Tanenbaum & Herbert Bos, *"Modern Operating Systems"*,4th edition, Pearson Education, Inc., Upper Saddle River, New Jersey, 07458.

[2] William Stallings, "*Operating Systems: Internals & Design Principles"*, 7th ed., Pearson Education, Inc., Upper Saddle River, New Jersey, 07458