**Fastcomcorp Research**

White Paper

# Multics OS

Public Version

## Introduction

Multics also known as, the Multiplexed Information and Computing Service, was an operating system developed led by MIT, General Electric, and Bell Labs. It was developed on the GE 645 computer and the Honeywell Series 60. It was a pioneering research project with some of the brightest minds in academia. Multics was the predecessor of UNIX and all known operating systems. We came across Multics white papers and our research group was shocked what we discovered. This white paper is a brief description of some of our discussed topics and what we learned about on this operating system.

At Fastcomcorp Research we are brainstorming about the development of an operating system or platform to interconnect all computers into one unified exchange system. We did a study on Windows, Linux, OpenBSD, and UNIX. Then we came across Multics the predecessor of UNIX and the Multics operating system its source code. We came to find out that Multics was the first OS to provide a hierarchical file system and ACL's (Access Control Lists).

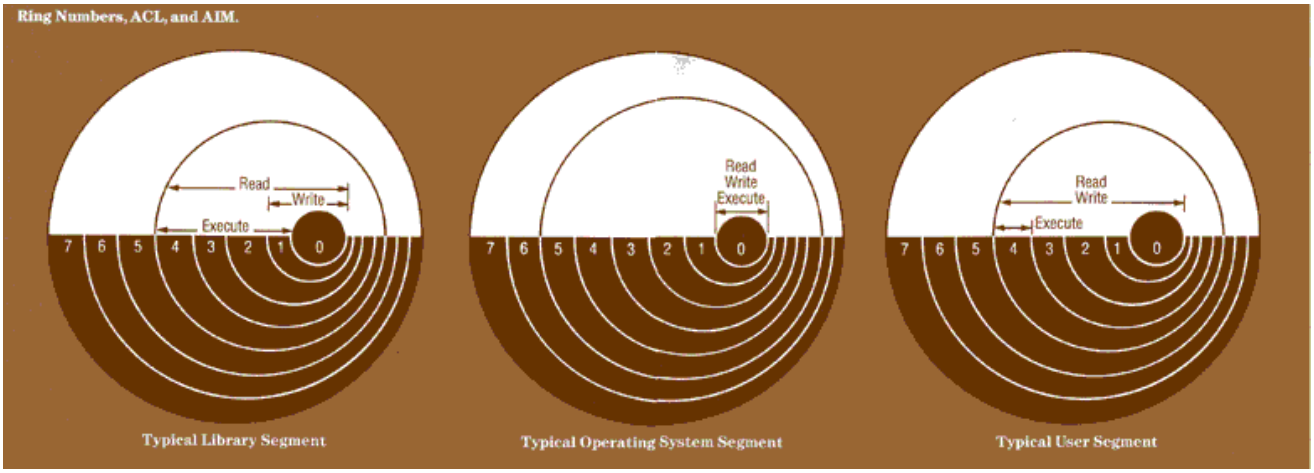
During the time when we found Multics, we were already having discussions on Konrad Zuse's work. Having discussions on his book he wrote, Calculating Space.

```
Translated for Massachusetts Institute of Technology, Proj. MAC, by:
Aztec School of Languages, Inc.
Research Translation Division (164)
Maynard, Massachusetts and McLean, Virginia
```

On the cover page in the bottom we noticed the name of the Project MAC. Project MAC is the name of the MIT project where Multics was born.

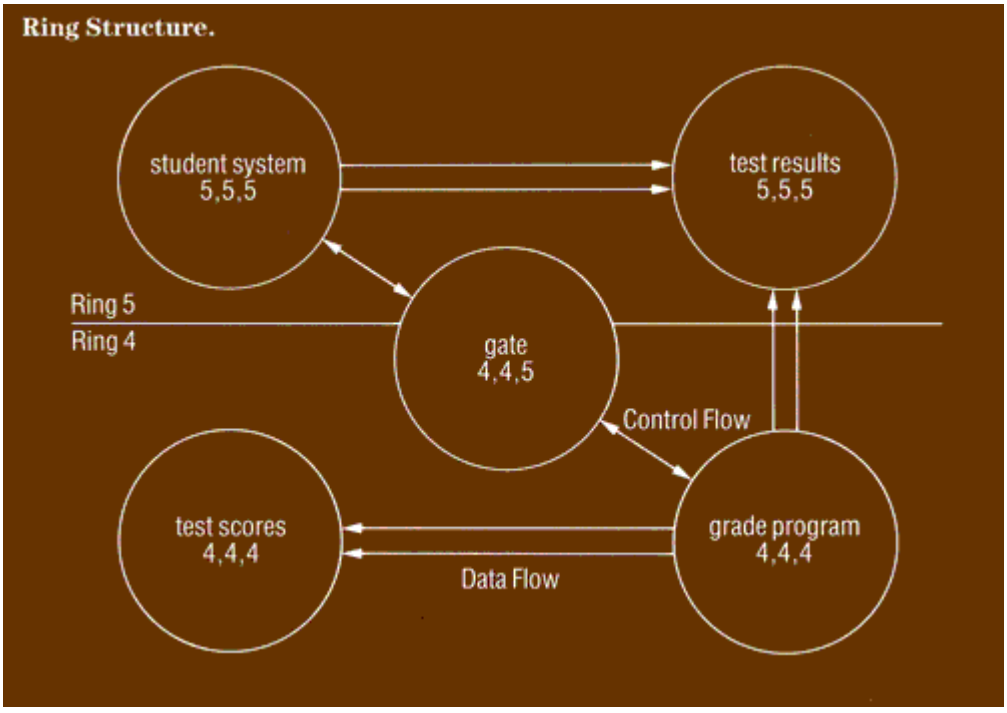
## Operating Systems Ring Security

One of the first things we noticed as we read about Multics was the security ring mechanism that was built in. Multics comes with 8 ring security and separated into three segments which are the operating system segment, library segment, and the user segment.

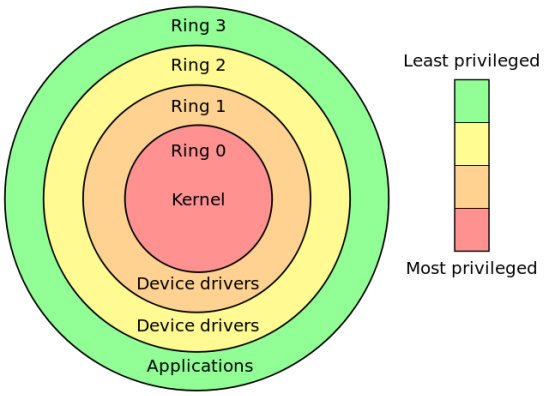


In addition to the protection of the Multics operating system, the ring mechanism is also utilized to protect user subsystems. *(This is an example was found in multicians.org)*

For example, a teacher could restrict his students to ring 5 by asking a system administrator to allow users on the teacher's project to log in only in ring 5. He might then write a gate segment with ring numbers [4,4,5] and an ACL granting execute access to all users on his project, and a gradebook segment with ring numbers [4,4,4] and an ACL granting write access to all users on his project. When the students finished homework problems in a segment in ring 5, they could call the teacher's gate into ring 4. The gate segment would examine the student's work, store a grade on behalf of the student in the gradebook segment, and return to the student in ring 5. Because the students would have access to the gradebook segment only through the gate, they would not be able to examine or modify the grades. The teacher, who could log on in ring 4, however, could modify the grades.

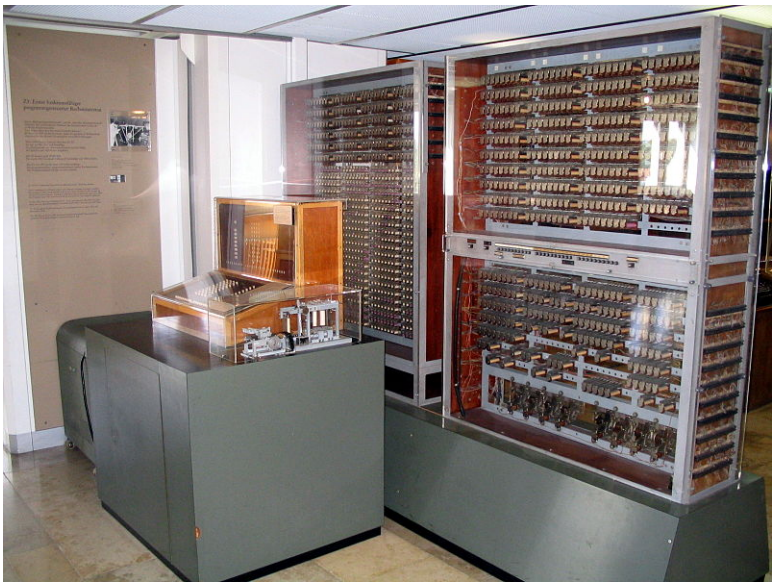



**Today's modern kernels use only utilize one to three rings of protection.**



## Z3 Computer

We studied some parts and the history of Konrad Zuse's Z3 computer which is known for being the first program-controlled computer at 5hz to get into the mindset of the Multics project. The Z3 had the ability to perform arithmetic exception handling. There were things did not understand, but we did our best.





## Foundation of Multics OS - PL/I

PL/I stands for Programing Language One. PL/I and Cobol came from the same lab work with similar technologies. We asked ourselves as to why did the Multics development team not use Cobol? We came to find out it was because of the security PL/I provides. PL/I uses character strings that are either of fixed length, or of a variable length, but the maximum length is always specified. PL/I can also accurately allocate the correct amount of memory required for any of its compiled code. This setup greatly reduced the likelihood of a buffer overflow. Which occurs when an operating system fails to allocate the correct amount of system memory for the code when it is executing it.

So the guys who developed UNIX were part of the Multics project, they wrote it in the C programing language and assembly, but mainly C. So one of us wondered since C requires the searching for a null byte to determine string length and PL/I can accurately allocate the correct amount of memory. Why didn't the UNIX development team not use PL/I as its base? Could PL/I not work with C? Then again their main objective was for the operating system to be a convenient platform for programmers developing software. We came to the conclusion after researching that little did the UNIX team know by doing their setup allowed for anyone to write a malicious code down the road to exploit this weakness to access resources beyond what the operating system had permitted, which is a common point of attack in today's operating systems written in the C language.

## Multics OS Login and Password Storage

Multics security features also included enciphered passwords, a login audit trail, and software maintenance procedures. The system passwords in Multics were never stored in clear text. Instead, when a user entered their login password, the password was enciphered, then compared to the password stored on the system in the same cipher.

We looked up what was enciphered because we have never heard of that terminology.

**Encipher**: To effectively hide written information behind seemingly useless jargon

We came to find out that an enciphered password is to prevent the passwords stored in the system from being easily learned if an attacker reads the file where they are stored. An enciphered password is generated by some mathematical algorithm that is usually a one-way function. By utilizing a one-way function means that one can only use it to get the enciphered results, but cannot run the same algorithm on the enciphered result to get the original password.

## AIM (Access Isolation Mechanism)

The access isolation mechanism built in on the Multics OS was a security feature that enforced the classification of information and the authorization of the users. Each object in the system had a classification level and categories, and each user of the system had an authorization level and categories they could and could not access.

## Access Control List (ACL)

Multics was the first OS to implement ACL. ACL was developed for the Multics OS as a mechanism to provide system-wide administrative control over the access of processes, directories, and over the propagation of access. It was embedded together with the Multics eight ring system.

## Virtual Memory

We came to find out that Multics was the first operating system to utilize virtual memory. It had a segmented, paged virtual address space that was offered on its eight levels of privilege (rings). It supported a tree-structure hierarchy of directories, up to an arbitrary limit of 16 levels. The file names and directory names were full ASCII and up to 32 characters long.

# Multics Eight Ring Privileges

**Ring 0** is used by the supervisor. By design, none of the code in ring 0 comes from the file system hierarchy. The supervisor is segmented and a special "segment loader" that reads the kernel segments off of the boot tape into main memory. A few of these kernel segments were visible thru the file system, and those that were had to be "patched" into it. After initialization, the supervisor mounted the file system and made it available to users.

**Ring 1** holds "extensions" to the supervisor; vendor-supplied code that is less privileged than the kernel, but more privileged than all users. Ring 1 is within the "security kernel" and is therefore able to avoid the restrictions of the Access Isolation Mechanism (AIM).

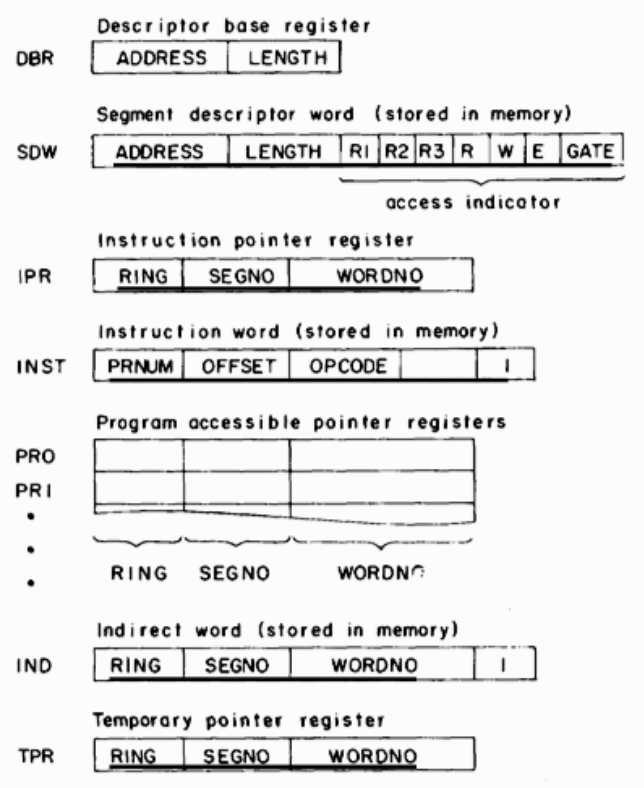
**Rings 2 and 3** are available to run code that is less privileged than the supervisor, but more privileged, and protected from, the normal users. Rings 2 (and higher) are outside the security kernel and cannot escape AIM. (These rings have been used for database managers, TCP/IP drivers, etc. Some of these products also have ring 1 components, making them truly "3-ring circuses"). By convention, ring 2 is used for vendor-supplied applications and ring 3 is available for user-supplied applications.

**Ring 4** is where typical user programs run in.

**Rings 5** is available to run less-privileged user programs.

**Rings 6 and 7** have almost no direct access to system functions; they must call user-supplied code in inner rings to accomplish any useful work.

*The Schematics Description of Relevant Storage Formats and Processor Registers*



## Why UNIX Became More Popular Than Multics?

This was one of our biggest questions. By digging around the internet. We came to find out from the comments of UNIX developers who left the Multics project that the software was buggy and complex that this was not the case. According to a statement from F.J Corbato, J.H. Saltzer, and C. T. Clingen mentioned that there were no system failures traced to the software. It was mainly hardware, operator error, and some software bugs introduced during the course of development.

Multics was in use for over 30 years and it generated allot of money for Honeywell. Multics was a commercial product and Honeywell did not publish its source code until not long ago. Also Multics required specialized software which we would love to get our hands on to run the source code we found.

To conclude the question. Multics was engineered to run on big expensive mainframes. UNIX when it was written, it was freely shared with the community and ran on modest hardware. As time progressed there were many contributions to UNIX that put in a growth curve. Eventually UNIX became an enterprise product. From UNIX now we have MAC OS and Linux built from UNIX. Without Multics there would be no UNIX.

## Application of This Research

This research inspired the group to deeply look into studying mathematical and computing system. Especially into Boolean algebra and logic gates. Also into seeing if there was a geometry connection in computing.

## Fastcomcorp Research Team

Chris Reihsmann

Chad Pierce

Eric Gough

Francisco Pinochet

Kimlong Loung

Michael Thompson

Ryan Sema