# 第 2 讲：OS Architecture & Structure

## 第二节：History

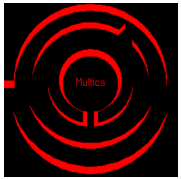
陈渝

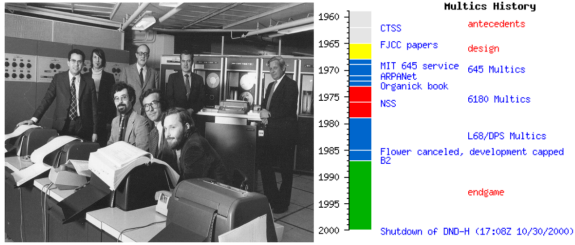
清华大学计算机系

*yuchen@tsinghua.edu.cn*

2020 年 2 月 23 日

# History




MULTiplexed Information and Computing Service (MULTICS)

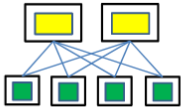
- Multics is a timesharing OS begun in 1964 and used until 2000.
- Joint project between MIT, Bell Labs, and GE
- Primary usage was with a mainframe and multiple terminals.

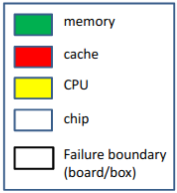
# Multics onGE645

GE645 computer
Symmetric multiprocessor



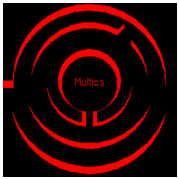

Communication was by using "mailboxes" in the memory modules and corresponding interrupts (asynchronous).

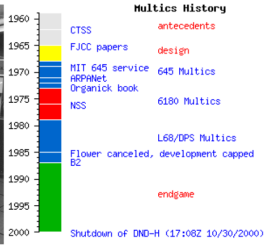





- Reliable interconnect
- No caches
- Single level of shared memory
  - Uniform memory access (UMA)
- Online reconfiguration of the hardware
  - Regularly partitioned into 2 separate systems for testing and development and then recombined
- Slow!

# History



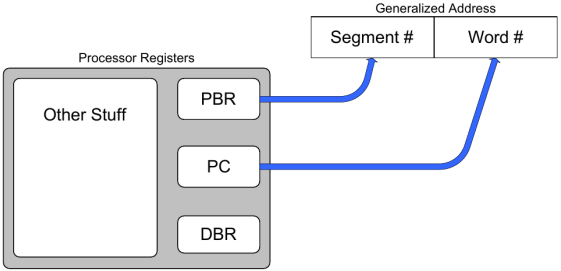




Design Goals&Features of multics(1964) ?

- Segmented|Paging based Virtual memory
- First hierarchical file system
- High-level language implementation (IBM's PL/I)
- Shared memory multiprocessor
- Dynamic linking and function call by name
- Security and rings

# History

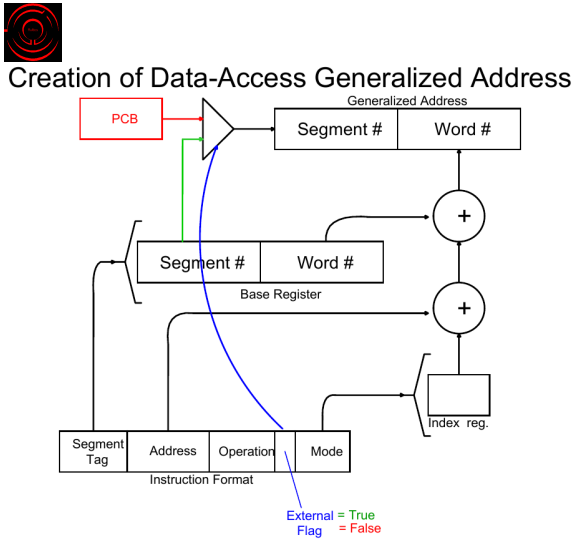
## Creation of the Instruction Fetch Generalized Address




## Addressing

- Multics uses a Generalized Address
- It is calculated differently depending on if the CPU is attempting to read an instruction or data

# History

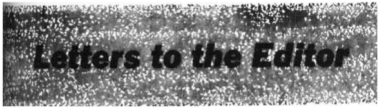



## Addressing

- Multics uses a Generalized Address
- It is calculated differently depending on if the CPU is attempting to read an instruction or data

# History

**Edsger Dijkstra considers *go to* harmful.**




**Go To Statement Considered Harmful**

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
*CR* Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More

dynamic proj
call of the p
we can chara
textual indic
dynamic dep
Let us nov
or **repeat** *A*
superfluous,
recursive pr
clude them:

THE Structure of the “THE” - Multiprogramming System

- May, 1968
- Response to a call for papers on timely research and development efforts
- Six person team
- 3 Guiding Principles
  - Select an ambitious project
  - Select a machine with a good basic design (EL X8)
  - Experience != Wisdom

# History

## SIGOPS Hall of Fame summary

"The first paper to suggest that an operating system be built in a structured way. That structure was a series of layers, each a virtual machine that introduced abstractions built using the functionality of lower layers. The paper stimulated a great deal of subsequent work in building operating systems as structured systems."

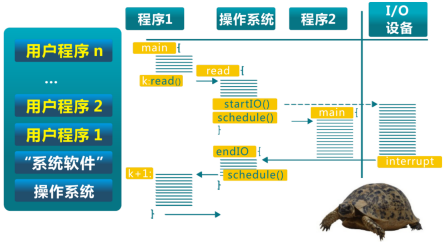
- Contributions
  - System Hierarchy
  - Storage Allocation
  - Processor Allocation
  - Semaphores

# History

THE Structure of the “THE” - Multiprogramming System

- Goal
  - Quick turn-around for short term programs
  - Economic peripheral use
  - Economic backing store and processor use
  - Flexibility as a general purpose computer

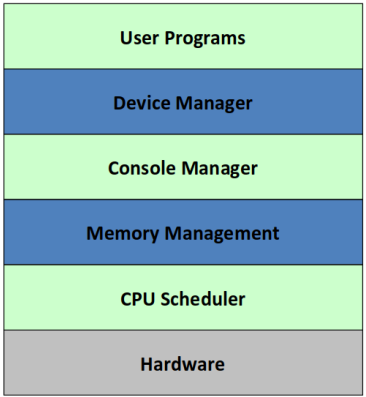
# History




THE Structure of the “THE” - Multiprogramming System

- Challenge : Complex softwares v.s. puny hardware
- Challenge : Multiprogramming

# History




THE Structure of the “THE” - Multiprogramming System

- Solution : Layered Structure
  - Virtualized I/O streams
  - Private console
  - One huge/unlimited memory space
  - One processor, manage interrupt

# History

```
begin semaphore mutex; mutex := 1;
        parbegin
            begin L1:
            P(mutex);
              critcal section 1;
            V(mutex);
            remainder of cycle 1;
            go to L1 end;

            begin L2:
            P(mutex);
              critical section 2;
            V(mutex);
            remainder of cycle 2;
            go to L2 end
parend end
```

The Structure of the "THE" - Multiprogramming System

- Solution : Semaphore
  - THE OS requires synchronization

# History

```
begin semaphore mutex; mutex := 1;
        parbegin
            begin L1:
            P(mutex);
              critcal section 1;
            V(mutex);
            remainder of cycle 1;
            go to L1 end;

            begin L2:
            P(mutex);
              critical section 2;
            V(mutex);
            remainder of cycle 2;
            go to L2 end
parend end
```

The Structure of the "THE" - Multiprogramming System

- Future : verification
  - Testing was done from the bottom level up, Each level was exhaustively tested prior to the next. But Testing is not enough.

# History

```
begin semaphore mutex; mutex := 1;
        parbegin
            begin L1:
            P(mutex);
              critcal section 1;
            V(mutex);
            remainder of cycle 1;
            go to L1 end;

            begin L2:
            P(mutex);
              critical section 2;
            V(mutex);
            remainder of cycle 2;
            go to L2 end
parend end
```

Is THE a good system at that time?

Yes

- Performance
  - 20% slower than single machine
  - Short turn-around time (latency) for short jobs
  - Benefit from the multi-programming design choice
- Programmability
  - virtual BIG memory
  - Benefit from the VM design choice
- Reliability