Operētajsistēmu inženierija

Multics

Kursu vada Leo Seļāvo

MULTICS

- 1964.g.: CTSS - Compatible Time-Sharing System (Project MAC at MIT)
- 1965.g.-1969.g.: MULTICS
- 1969.g.: UNIX

- MULTICS = Multiplexed Information and Computing Service

- Origin of many ideas in today's OSes
- Motivated UNIX design (often in opposition)

Mērķi:

- Convenient remote terminal access
- A view of continuous operation
- A wide range of capacity to allow growth or contraction without either system or user reorganization
- A reliable file system that users trust their only copy of programs and data
- Sufficient control of access to allow selective sharing of information
- The ability to structure hierarchically both the logical storage of information as well as the administration of the system
- The capability of serving large and small users without inefficiency to either
- The ability to support different programming environments and human interfaces within a single system
- The flexibility and generality of system organization required for evolution through successive waves of technological improvements and the inevitable growth of user expectations

Multics inženierija

- Galveno ideju izstrāde
- Detalizēta programmu moduļu specifikācija
- 3000 lapas - Multics System programmers' Manual
- Realizācija PL/1 prog. Valodā
- Moduļu implementācija un integrācija un testēšana
- Izrādījās, ka paredzamā izpilde būtiski atšķirās no reālās => vajadzēja papildus iterācijas izstrādei

Pieredze

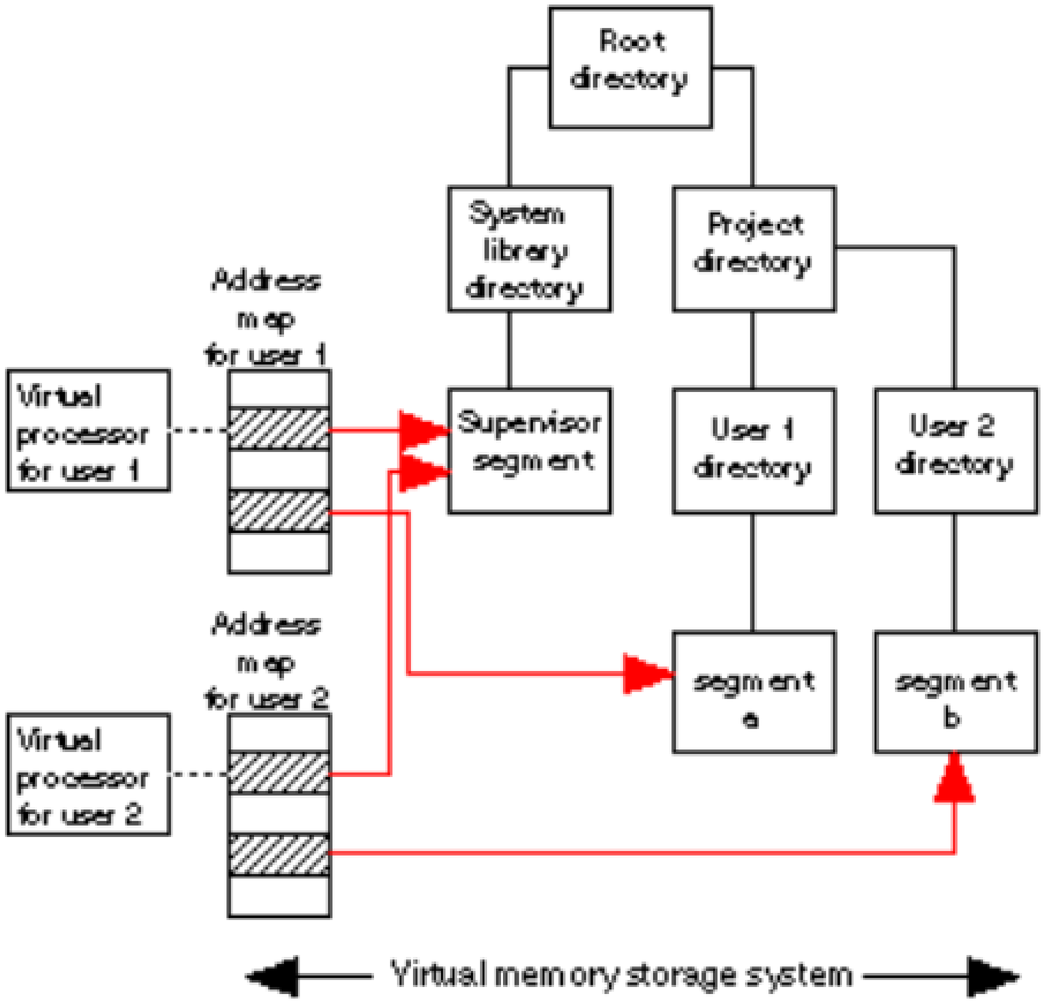
- specifications representing less-important features were found to be introducing much of the complexity
- the initial choice of modularity and interfacing between modules was sometimes awkward
- the most important property of algorithms is simplicity rather than special mechanisms for unusual cases

Idejas

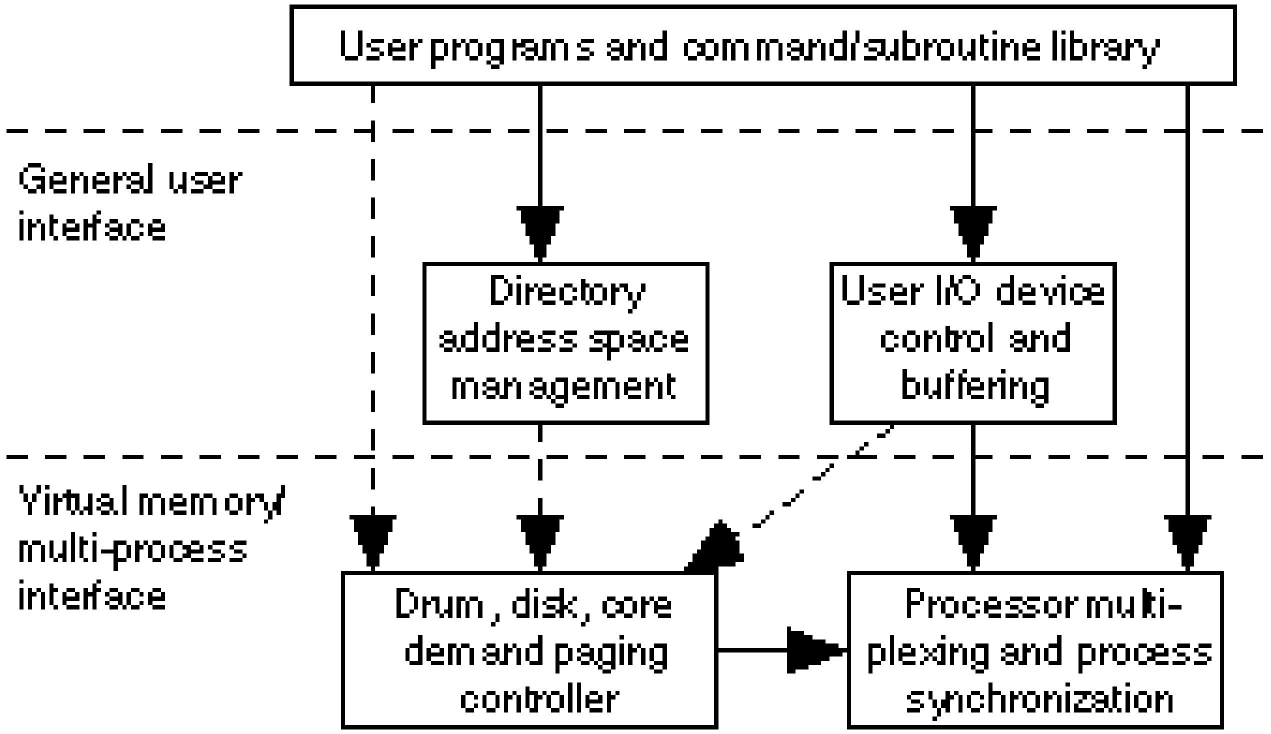
- Modular division of responsibility
- Dynamic reconfiguration
- Automatically managed multilevel memory
- Protection of programs and data
- System programming language

Faili atmiņā

The entire storage hierarchy may be mapped into individual user process address spaces (see arrows) as if contained in primary memory. Illustrated are the sharing of a supervisor segment by user 1 and user 2 and private access to segment a and segment b. The necessary primary storage is simulated by a demand paging technique which moves information between the real primary memory and secondary storage.



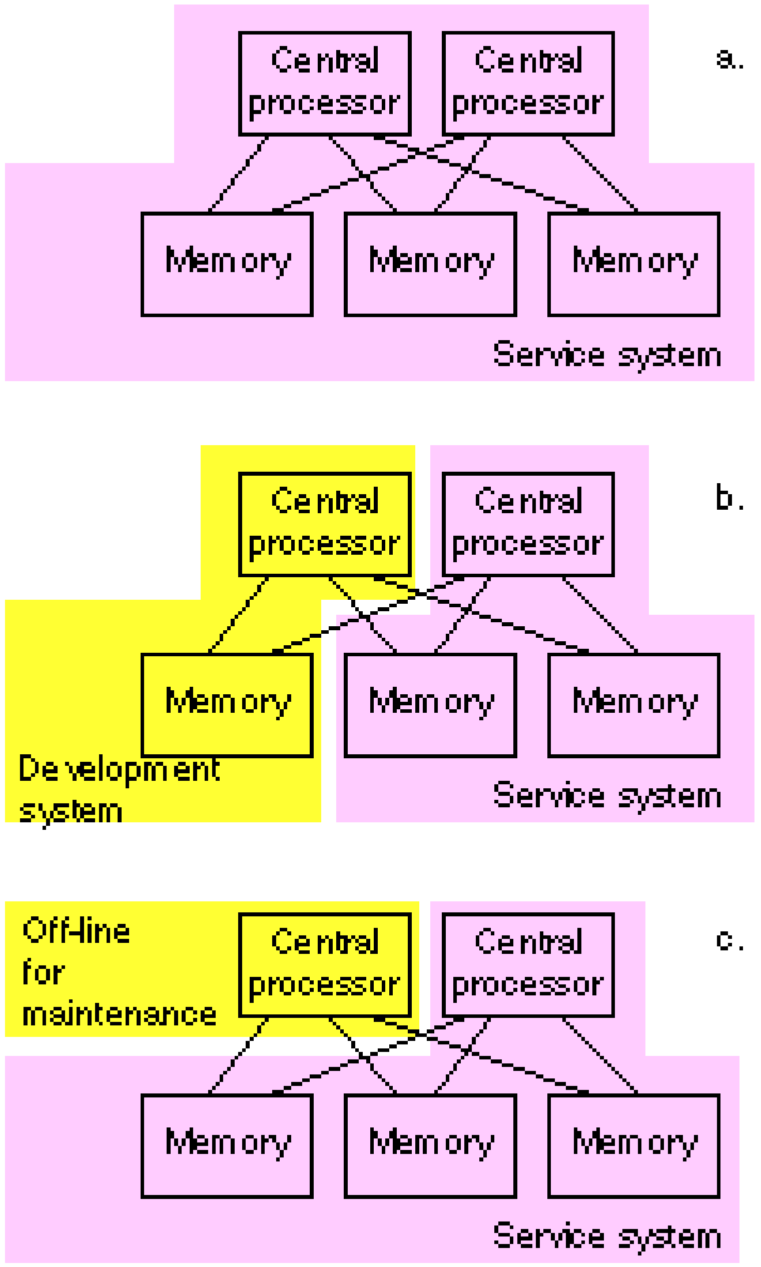
Modularitāte


User programs and command/subroutine library
General user interface
Directory address space management
User I/O device control and buffering
Virtual memory/ multi-process interface
Drum, disk, core demand paging controller
Processor multi-plexing and process synchronization

Major lines of modular division in Multics. Solid lines indicate calls for services. Dotted lines indicate implicit use of the virtual memory.

Dinamiskā rekonfigurācija

Dynamic reconfiguration permits switching among the three typical operating configurations shown here, without currently logged-in users being aware that a change has taken place.



Hierarhiska atmiņas arhitektūra

Automatically managed multilevel memory

- A strategy to treat core memory, drum, and disk as a three-level system has been proposed, including a "least-recently-used" algorithm for moving information from drum to disk
- A scheme to permit experimentation with predictive paging algorithms
- A series of measurements was made to establish the effectiveness of a small hardware associative memory used to hold recently accessed page descriptors.
- A set of models, both analytic and simulation, was constructed to try to understand program behavior in a virtual memory.

Aizsardzība: “Rings of Protection”

- A hardware architecture which implements the mechanism was proposed. Subroutine calls from one protection ring to another use exactly the same mechanisms as do subroutine calls among procedures within a protection area.
- As an experiment in the feasibility of a multilayered supervisor, several supervisor procedures which required protection, but not all supervisor privileges, were moved into a ring of protection intermediate between the users and the main supervisor.

Sistēmas programmēšanas valoda

- The transition from an early PL/I subset compiler to a newer compiler which handles almost the entire language was completed. The **significance** of the transition is the demonstration that **it is not necessary to narrow one's sights to a "simple" subset language** for system programming.
- Yet, the time required to implement a full PL/I compiler is still too great for many situations in which the compiler implementation cannot be started far enough in advance of system coding. For this reason, there is considerable interest in defining a smaller language which is easily compilable, yet retains the features most important for system implementation.
- Roughly, of the 1500 system modules, about 250 were written in machine language. Most of the machine language modules represent data bases or small subroutines which execute a single privileged instruction.

UNIX vs Multics

- UNIX was less ambitious (e.g. no unified mem/FS)
- UNIX hardware was small
- just a few programmers, all in the same room
- evolved rather than pre-planned
- quickly self-hosted, so they got experience earlier

What did UNIX inherit from MULTICS?

- a shell at user level (not built into kernel)
- a single hierarchical file system, with subdirectories
- controlled sharing of files
- written in high level language, self-hosted development

What did UNIX reject from MULTICS?

- files look like memory
  - instead, unifying idea is file descriptor and read()/write()
  - memory is a totally separate resource
- dynamic linking
  - instead, static linking at compile time, every binary had copy of libraries
- segments and sharing
  - instead, single linear address space per process, like xv6
  - (but shared libraries brought these back, just for efficiency, in 1980s)
- Hierarchical rings of protection
  - simpler user/kernel
  - for subsystems, setuid, then client/server and IPC