Multics

Background
- ancestor of unix
- very sophisticated monolithic OS with many innovations
- paid attention to security (ring, access control list, …)
- unique virtual memory mgmt
- first hierarchical file system

Virtual memory
general benefit:
  make programming easier
  improve portability
  use large virtual address space on small physical memory
  memory allocation does not need to to be contiguous
specific benefit
 help code/data sharing
 allow calling procedures with only symbolic names (no need to recompile when the library changes)
      (modern technique: dynamic linking)

Basic segment idea in Virtual memory mgmt
 3 segments (code, stack, heap); each segment's base and bound information is kept in register
 problem: a segment could be too big to fit into physical memory; contiguous physical memory allocation required within segment

basic paging
 page table
 problem: a page is not a natural logical unit as a segment

multics:
 2^14 segments (per process); 2^18 per seg

 How to support segment look up for so many segments?
  Can we use registers? no
  Use segment (descriptor) table
  Is segment descriptor table per process or global? per process
      Same segment could have different segment ID in different processes
      Same segment could be associated with different access privilege in different processes
 What is in each descriptor in the table?
      (1) The location information of the segment
      (2) The access control information
 It is Operating System who fills the segment descriptor table

General Address
segID, word number

Instruction addressing mode
register (segment id, word-number), offset (external)
procedure base register , offset (internal)

special: indirect addressing mode
like pointer, the address specified in the instruction is M. The final data D is stored at location N.
N can be find at memory location M.
more levels of indirection is possible.
————

how to share data across segments?

Suppose procedure P (in segment x) tries to use variable D in a different segment.
what are the challenges?
When P refers to D in its machine code,
can P directly uses the offset of D in D's segment?
no. Because that would force P to recompile everything D's segment changes.
Symbolic name of D is used. Operating System will check the symbol table in D's segment to figure out the offset of D within its segment at run time.
can P directly uses the segment ID of D's segment?
no. Statically (when P is compiled), no one knows what is the segment ID of D's set.
In fact, in different processes, D's segment ID is likely different.
Symbolic name of D's segment (a path) is used.

The symbolic pair <D's segment>|<D> is put in the linkage segment of P.
The first time, execution traps into OS. OS uses the path to find D's segment, check the access control list to make sure the current process can indeed read/write/execute the D, OS loads D's segment into physical memory, OS fills the segment descriptor table, obtain the segment ID; OS checks D's segment's symbol table to find the offset of D; OS modifies the run-time linkage segment image to replace symbolic names/paths with segment id and offset.
Future execution will not trap into OS.

The process of OS checking the access control list of a segment, and add that segment's descriptor into a process' segment table is called "make the segment known to a process".

————
how to share code

the trick (or unique challenge): how to update the linkage register when control flow changes from one segment to another. Jump to the linkage segment, instead of the

callee code, first, load the program counter into linkage register.

—
impact: the power of indirection!
====
nucleus
  where microkernel is used
  can child process' address space overlap?