Instructor Notes

Daley, R.C., and Dennis, J.B.
Virtual Memory, Processes, and Sharing in MULTICS
Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 306-312.

1. MULTICS (Multiplexed Information and Computing Service) made fundamental design decision so that the system could effectively serve the computing needs of a large community of users with diverse interests. What were three of their objectives?

- Late 1960s, Highly influential (UNIX)
- Did everything - no new ideas left

---

1) Large, machine-indepedent virtual memory
 → no user burden

2) Programming generality → only need to know symbolic names
 → further references provided only when/if needed

 → Don't link before running → dynamic linking

3) Permit sharing of data + procedures (less memory needed)
 → Memory sharing is very fine grained

2. Start with some standard definitions. What is a process? What is a segment? How many segments can be in an address space? What are the two types of segments? What operations can be performed on each type of segment? How does a file relate to a segment? How are symbolic names mapped to segments?

Process: One-to-one correspondence with its own <u>address space</u>

Segment: Logically distinct unit of info
- length + access privelege [attributes]
- grow + shrink independently of others

How many? 2^{14} segments in an address space

2 Types: data - no instruction fetch
procedure - no writing - "pure"

<u>Files</u> + segments can be the same
- Take a symbolic name (e.g. "/user/lib/libc.so") and lookup in directory structure, get segment which can be placed in address space

Where is "segment number" found?

in address space

3. Addressing Background. Each word is identified by a generalized address <segment number, word number>. ~~How~~ is the generalized address formed for fetching an instruction? ~~Other~~ addressing modes enable referencing information relative to the current procedure base register or other registers, or indirect addressing (designated by "`its`" in the address mode field).

Register used for segment number depends on context

- descriptor base register (DBR)
- procedure base register (PBR)
- (argument pointer)
- (base pointer)
- linkage pointer
- (stack pointer)

Instruction Fetch:

PBR	PC

set to segment of current procedure ~~segm~~

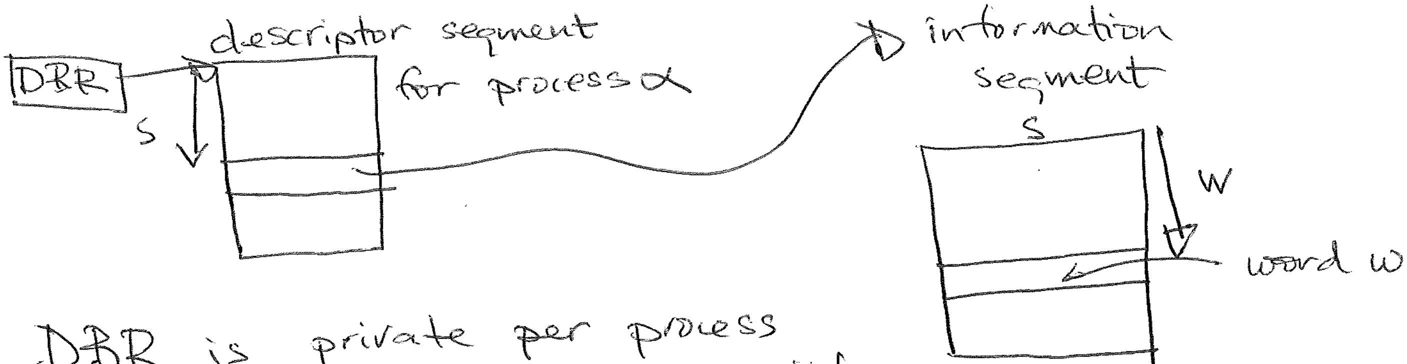
- different for every process
- addressing is all done relative to PBR

→ change PBR on call + return

4. Let's investigate how MULTICS goes from a generalized address <s, w> to a physical address. How is the segment number s used? How is DBR (Descriptor Base Register) set? How is the word w used? How is s allocated within a process? How is s allocated across different processes?

Once have "s", load <s,w>

Where does segment s live in physical memory?



DBR is private per process
- switch on context switch

How is "s" allocated? (within a process?)

- start @ 0 and increment
  (keep contiguous)

Across processes?

- No relationship
- Same {symbolic segment} may be referenced w/ <u>different</u> segment numbers

5. Why is paging needed in addition to this segmentation? Assume a 1024 word page. How is paging implemented on top of this? How many memory references are needed? How can this be sped up?

- Descriptor segments could be large

  2^{14} segments | process

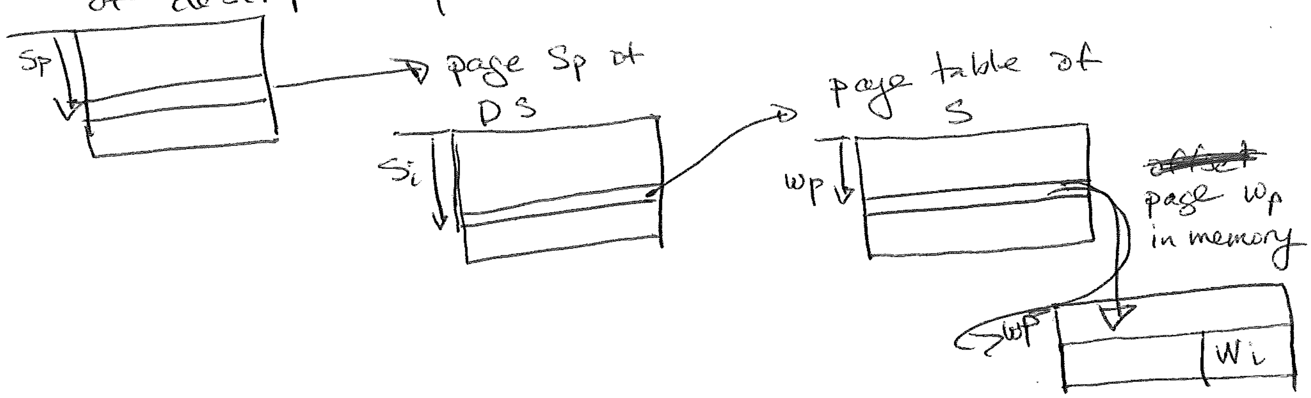
- Information segments could be large

  2^{18} words | segment

1024 word | page → 2^{10} words → 10 bits for address

4	10	8	10
S_p	S_i	W_p	W_i

DBR → page table of descriptor segment



4 look-ups from <s, w> generalized address

Need TLB to optimize

6. Why is dynamic linking so useful? What are the 3 requirements? A) What does it mean for a segment to be "pure"? Why are pure segments needed? What is the system implication of having pure ~~segments~~? B) What is the problem of using symbolic names? What will be the solution?

if "changes" are ~~needed~~

→ sharing: borrow procedures while saving space
get most recent version

1) Pure - content cannot be modified
- Needed due to sharing
- Use indirection to record changes (process specific)

2) Ref w/ symbolic name (no prior arrangement) (don't know order)
- Slow to use (lookup w/ paths)
- Shortcut — put in indirection location too

3) Procedure segments are invariant to compilation of others
→ Use symbolic names, no assumed orderings

7. What does it mean to "make a segment known"? How is a segment referenced after it is known? Why is it useful to do this?

- Translating symbolic name to local seg. #

Each process will have unique S_α
(remember: in different orders)

Done Lazily on demand

Take symbolic path name, directory search,
pass protection checks

Just use segment #, s, afterwards

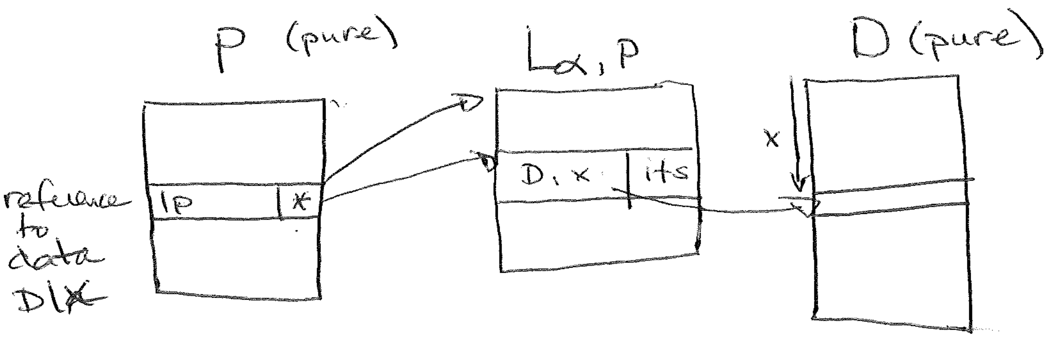
More efficient

merge?

data

8. Consider the case where procedure P in process α accesses <D>|[x] (both D and x are symbolic). How can we make segment D known without changing the contents of P (which must be pure)? Why must every process have a different "`linkage section`" for procedure P? (And, of course, the linkage section is a segment...)

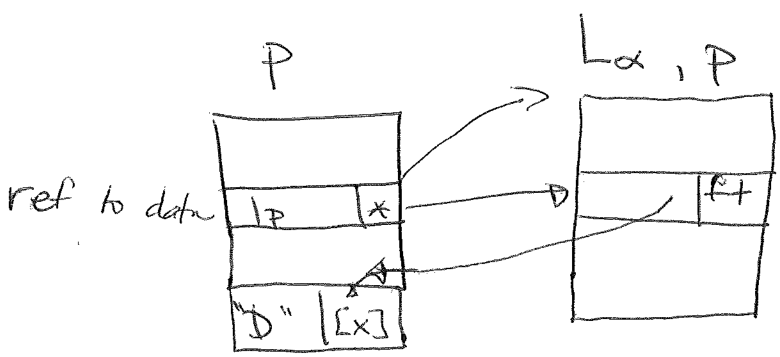
- Indirection



linkage section is different for every process because D will have different segment #

9. What does the `link data` look like before the link is established for process α? What happens during the trap? What is the `symbol table`? ~~What does the link data look like after the trap?~~ After this, how many references by generalized address are needed to access $D_\alpha|x$? How does one find the link data for the currently running procedure segment?

Before:



Trap: do segment path lookup
allocate segment in Descriptor Segment

Symbol Table: Contains offsets ~~for between~~
symbol word names → word number in segment

Subsequent access: Faster - 2 generalized address lookups

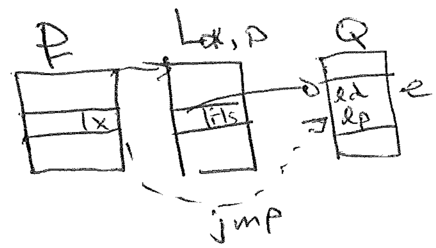
Link data for Lα,P? In register - lp - linkage pointer

10. Consider the case where P is calling Q|e. How does MULTICS set up the `linkage pointer, lp`, for Q? Remember that lp cannot be stored in Q (pure) and is different for every process. How can the `Procedure Base Register (PBR)` be used to set lp??

P calls Q|e, must change to lp of Q.

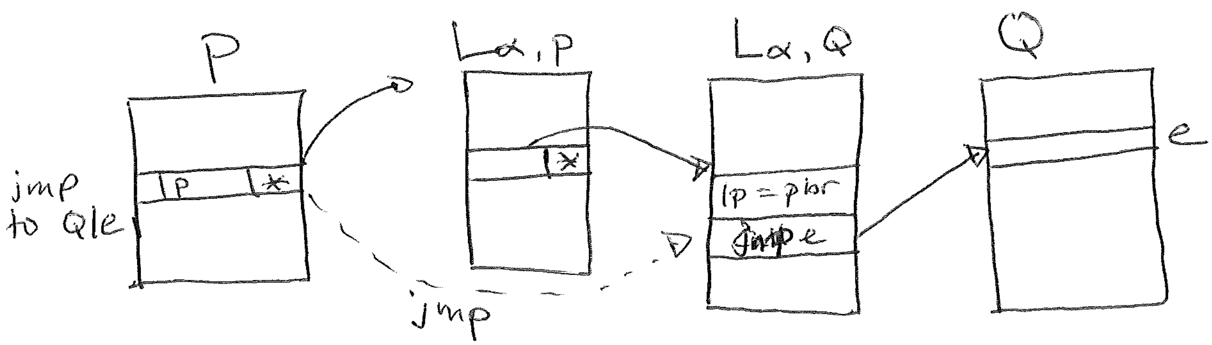
Use indirection, jmp indirection

Can't



because lp is different for every process

Hint: Instr. to set lp to PBR (wherever you are executing)



- Every entry point in procedure segment has 2 extra instructions
- Associate code for loading Q with Q

11. Conclusions. What concepts did MULTICS push to the extreme?
Can you anticipate any problems for MULTICS?

- Sharing
- Segments
- Indirection

Complex (esp. compared to other OSes in era)
- Early for its time + slow hardware
- Lots of cpu overhead