

CS 518

Virtual Memory and Sharing in Multics

Randy Wang
Fall, 2002
Princeton University

Outline

- Background
- Details of dynamic linking and fine-grained sharing
- Postscript

Outline

- **Background**
- Details of dynamic linking and fine-grained sharing
- Postscript

The “2nd System Effect”

- 1st system:
 - terrified of failure
 - simplified to its bare bone
 - successful beyond its intended life-span
- 2nd system:
 - hugely ambitious
 - usually conceived by academics
 - many good ideas
 - a little ahead of its time
 - doomed to fail
- 3rd system:
 - pick and choose the essence
 - usually made by good hackers
 - emphasize elegance and utility over performance
 - become widely adopted
- 4th systems:
 - maturation

The Multics Lineage

- 1st system: CTSS
- 2nd system: Multics
- 3rd system: Unix
- 4th system: BSD
- (This is a common pattern that repeats in many systems)

CTSS (1959 - 1965, MIT)

- Compatible Time Sharing System (CTSS)
 - Transition from batch processing to time sharing
 - Work on-line, and store info on-line
- Hardware
 - Off-the-shelf hardware
 - 4 consoles
 - 2 tape drives per user
- Software
 - interactive debugging (!)
 - editors
 - shells
 - No protection among users
- By early 60s, CTSS is showing its limitations
 - More memory, disks (drums)
 - More users: up to 32!
 - Want sharing!

Multics (1963 - 1970, MIT)

- Joint effort among MIT, Bell Labs, and GE
- Redesign everything!
 - Customized hardware
 - New programming languages
 - New operating system
- Success or failure?
 - The system never really worked
 - But probably the single most influential operating system
 - Explored virtually every single aspect of OS (with the exception of networking)

“Extreme Research”

- One hallmark of many influential work:
 - **Take a simple good idea and push to its logical extreme!**
 - Learn a lot
 - And step back to make it practical
- What is the extreme idea in this paper?
 - **“Extreme sharing”**
 - Sharing of anything (code and data)
 - Sharing at very fine grain
 - Sharing dynamically
 - (This is an idea that keeps getting re-derived time and time again)

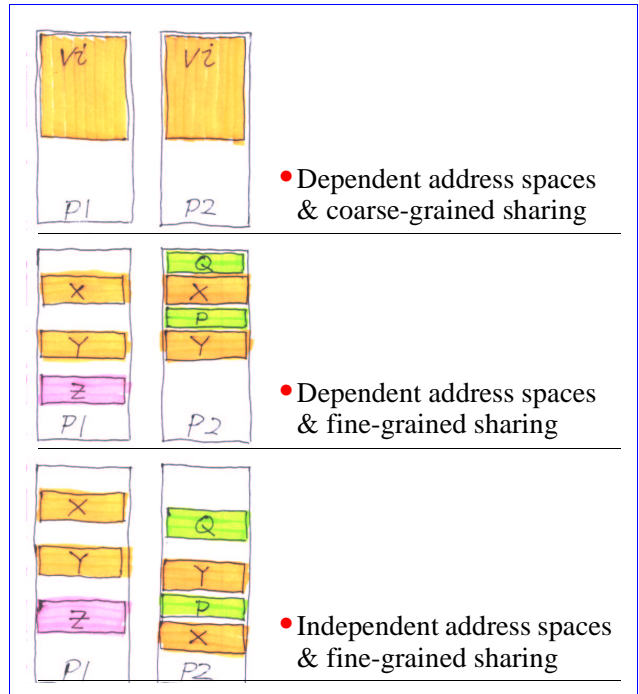
4 Key Ideas

- Naming and addressing
 - combine virtual memory and file system
 - a file name + offset in file = segment # + offset in segment
- Dynamic linking
 - recompile anything without relinking everything
- Fine grained sharing
 - share procedures instead of entire programs
- Autonomy
 - independent address spaces
 - (making virtual addresses really independent in the presence of fine grained sharing is hard!)

CS518

8

Randy Wang



CS518

9

Randy Wang

Outline

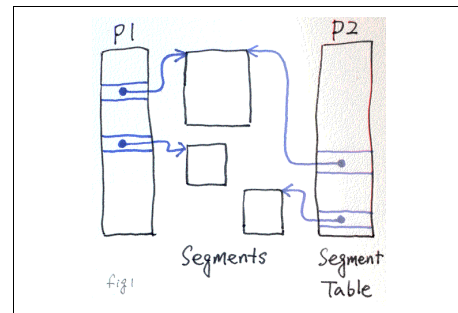
- Background
- **Details of dynamic linking and fine-grained sharing**
- Postscript

CS518

10

Randy Wang

Basics



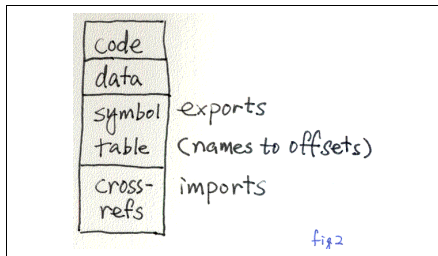
- Process = address space \sim segment table
- Generalized (virtual) address = segment index + offset
- (In reality, there is an additional level of paging but it's not important for this paper)

CS518

11

Randy Wang

Problem 1



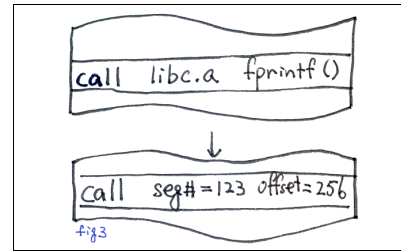
- Review: what's in an object file?
- A static linker resolves addresses statically
- Dynamic linking resolves these addresses at run time
- Problem 1: how does dynamic linking work?

CS518

12

Randy Wang

Dynamic Linking



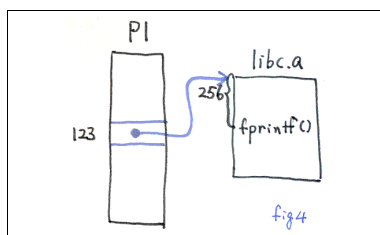
- Cross-refs are initially symbolic names
- Link trap on 1st reference
- Dynamic linker overwrites symbolic address with general address
- Return and retry instruction

CS518

13

Randy Wang

“Making Known” Segments And the Dynamic Linker



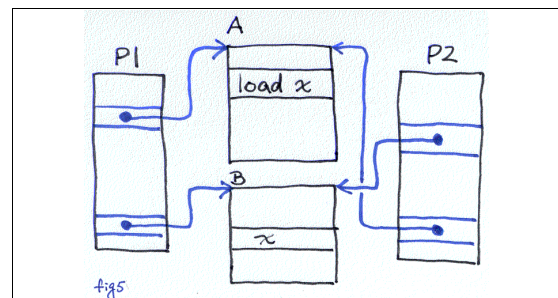
- Is the referenced file in the segment table?
- If not, “make known” the segment
 - The OS maps the file (like mmap())
- Find the symbol in symbol table of the referenced segment

CS518

14

Randy Wang

Problem 2



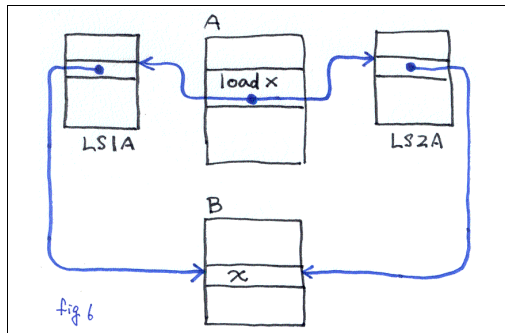
- Virtual addresses of same data (x, for example) are different for different processes
 - Because of independence of address spaces
- But the code segment (load x, for example) are shared by different processes
- So can't overwrite the addresses in the code segment
- Can only overwrite process-specific state!

CS518

15

Randy Wang

Linkage Section



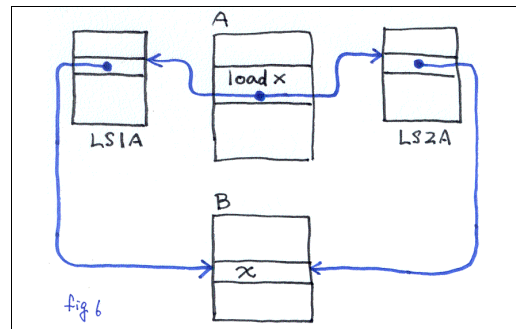
- A linkage section per segment per process
- In a linkage section, one entry for every item imported from other segments

CS518

16

Randy Wang

Sequence of Events (for referencing “external data”)



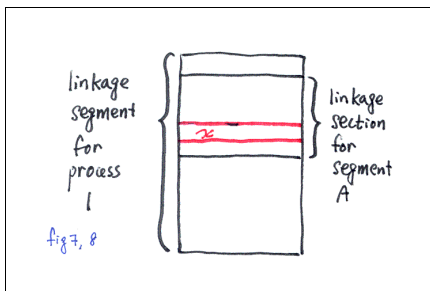
- Link trap for data reference
- Map referenced segment if necessary (make known)
- Modify address in linkage section

CS518

17

Randy Wang

Linkage Segment



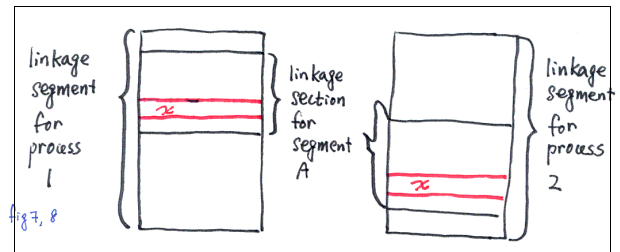
- Linkage segment: all linkage sections of a process grouped into one segment

CS518

18

Randy Wang

Problem 3



For different processes:

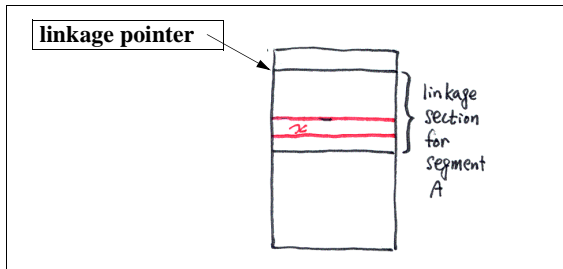
- The layout of each linkage section for the same segment is the same
- But the relative ordering of the linkage sections within a linkage segment is different
- So how do I reference an entry in the linkage section?

CS518

19

Randy Wang

Linkage Pointer



- For each process,
- For each currently executing code segment,
- Put the beginning address of the corresponding linkage section in a hardware register
- All references to the linkage section are relative to the linkage pointer

CS518

20

Randy Wang

Problem 4

When procedure X in segment P calls procedure Y in segment Q,

- Where does the new linkage section for Q come from?
- How do I change the linkage pointer?
 - Linkage pointer contains a segment number
 - So it can't be in P or Q
 - It has to be part of the per-process state

CS518

21

Randy Wang

Basic Ideas of the Final Solution (the buck stops here!)

When procedure X in segment P calls procedure Y in segment Q,

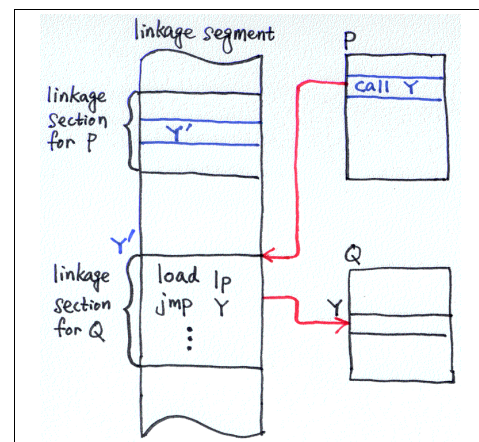
- Where does the new linkage section for Q come from?
 - When Q is referenced for the first time, instantiate a new linkage section for procedure Q
- How do I change the linkage pointer?
 - Put the code that changes the linkage pointer in the callee's (Q's) linkage section

CS518

22

Randy Wang

Detailed Sequence of Events (for procedure calls)



When procedure X in segment P calls procedure Y in segment Q,

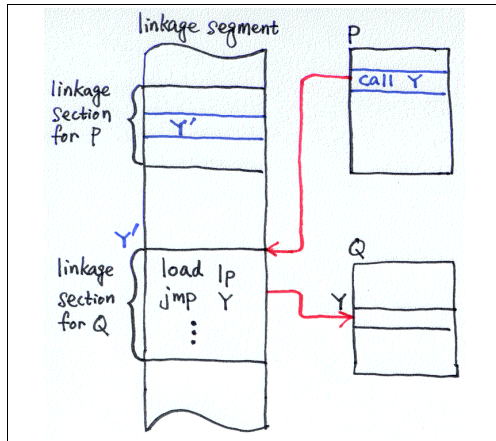
- Link trap when referencing a symbolic form of Y
- Map code segment of Q (making it known)
- Instantiate a linkage section for Q
- Resume execution

CS518

23

Randy Wang

Instantiating the Linkage Section



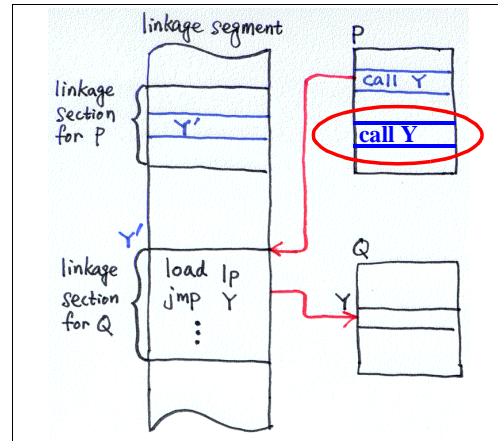
- Copy external references from Q's object file
- Manufacture two instructions for each exported procedure
 - One to load the linkage pointer
 - One to jump to procedure Y

CS518

24

Randy Wang

Question 1



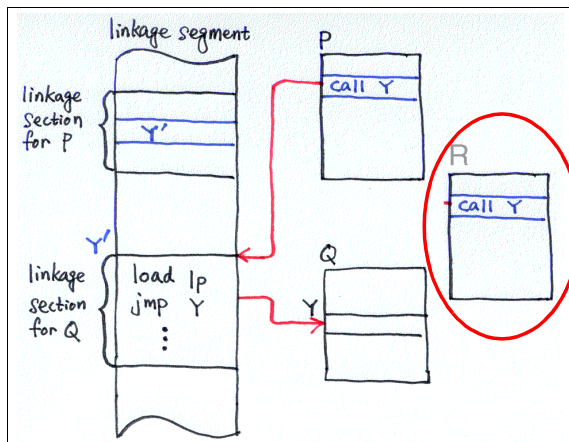
- What happens when there is a second call to Y in P?

CS518

25

Randy Wang

Question 2



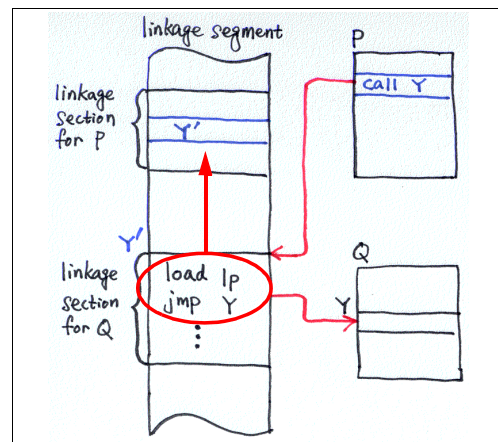
- What happens when there's another segment R that calls Y also?

CS518

26

Randy Wang

Question 3



- Can I move the "prolog code" into caller's linkage section?

CS518

27

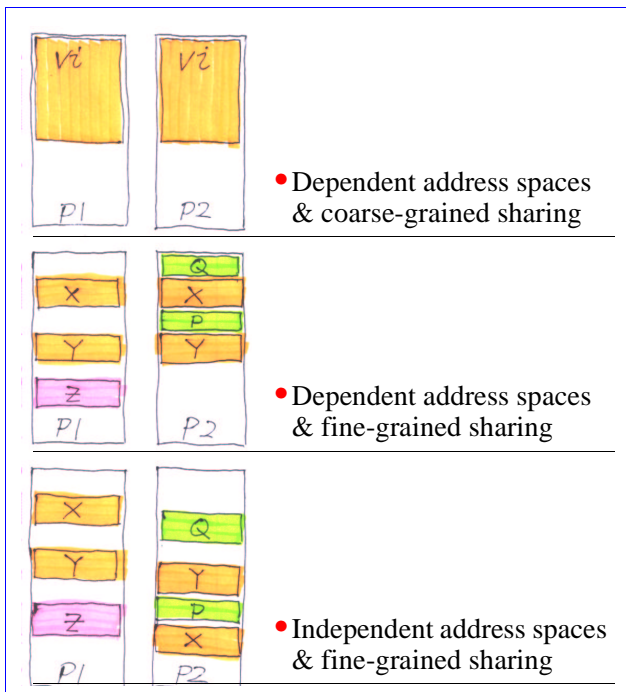
Randy Wang

Outline

- Background
- ~~Details of dynamic linking and fine-grained sharing~~
- **Postscript**

High Level Question: How Did We Get into This Mess?

- Naming and addressing
- Fine grained sharing
- Dynamic linking
- Autonomy (independent address spaces)



Postscript

- Post-traumatic syndrome for the next 20 years in terms of dynamic linking and fine grained sharing
 - Unix
 - Pilot
- Until the empire strikes back in the 80s
 - MIT X-windows
 - Megabytes of X toolkits cause Unix workstations to thrash
 - Need shared libraries
- Similar mechanisms are now standard in all major operating systems

Project Idea