Operating System Support for Easy Development of Distributed File Systems

Kenichi Kourai* Shigeru Chiba**
Takashi Masuda*

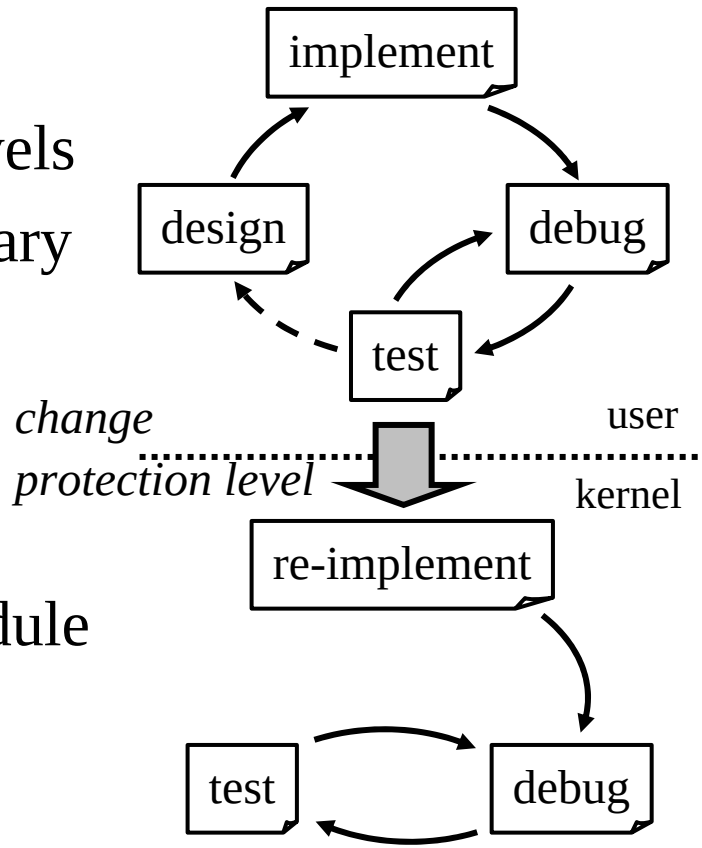
*University of Tokyo **University of Tsukuba

Developing a new distributed file system (DFS)

- Implementing a file system in the kernel is difficult.
- 2-step development is typical.
  - Uses two different protection levels

  Step 1. Implement a user-level library
    - Good for prototyping
      - Emulates a file system
    - Debug is easy.
      - The protection level is high.

  Step 2. Re-implement a kernel module
    - A file system runs faster.
      - No overheads due to protection.



Problem of the 2-step development

- Programmers must write a file system *twice*.
  - An API and a data structure are different.
    - In the 1st step, they write an emulation library.
    - In the 2nd step, they write a real file system.
- Debug is difficult in the 2nd step.
  - Programmers must debug a file system without a protection mechanism.

Multi-level protection

- Enables to change protection levels without modifying the source code of a file system
  - *Write once*, but *run at multiple protection levels*
- Why multi-level?
  - Programmers can choose a trade-off between performance and protection.
    - Programmers can select an appropriate protection level depending on the stability of a file system.

Developing a new DFS with multi-level protection

Phase 1. Implement a file system

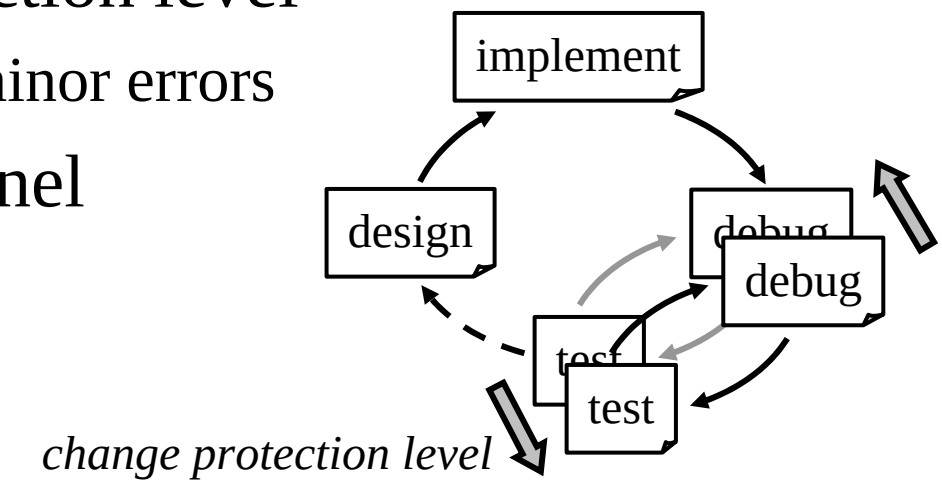
- Debug it at the highest protection level

Phase 2. Lower the protection level

- Run it faster and find minor errors

Phase 3. Run it in the kernel

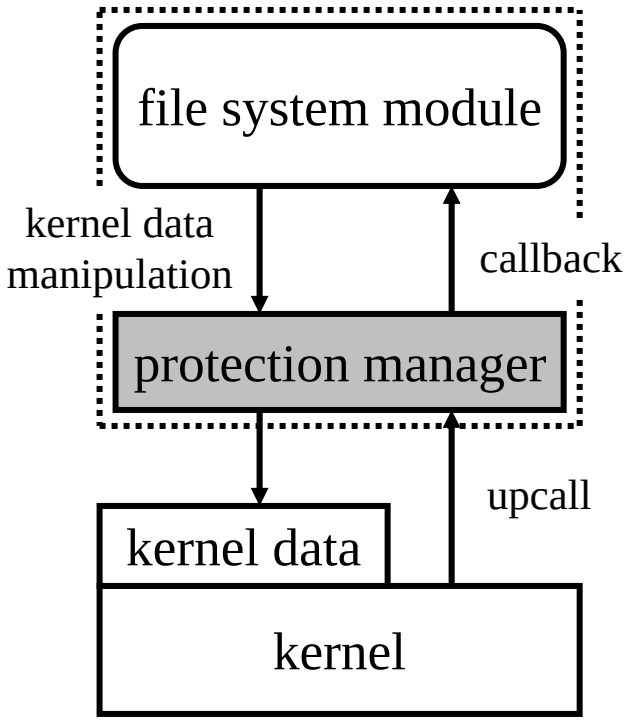
- No protection



No need to modify a file system between these phases

The CAPELA operating system

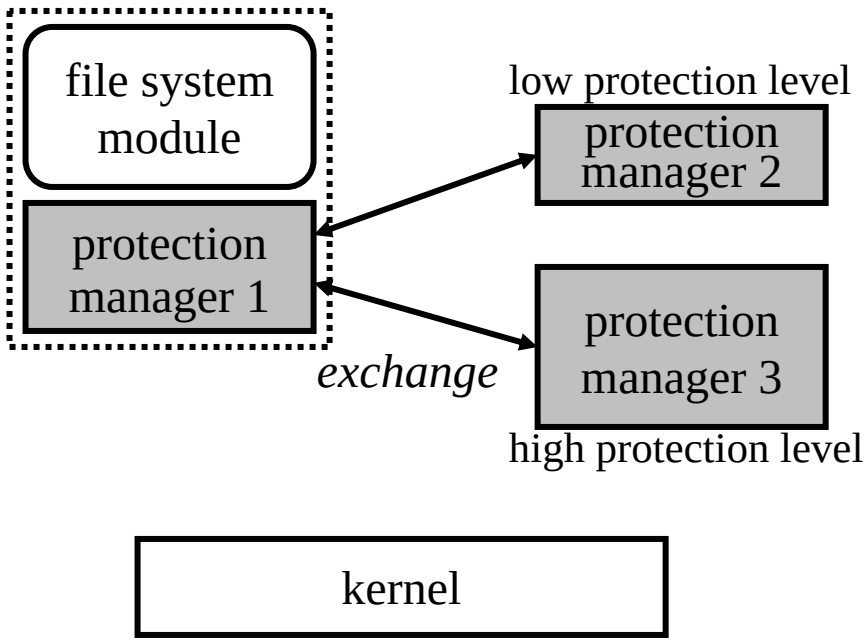
- CAPELA provides protection managers for the multi-level protection.
- A protection manager is a gateway for accessing the kernel data.
  - Protects the kernel data at a predefined level
  - Notifies a file system module of upcalls from the kernel
  - A library linked with a file system module



Changing protection levels

- CAPELA provides multiple protection managers.
  - Each manager provides a different protection level.

1. Exchange protection managers
   - Relink with a module
2. Restart a file system module

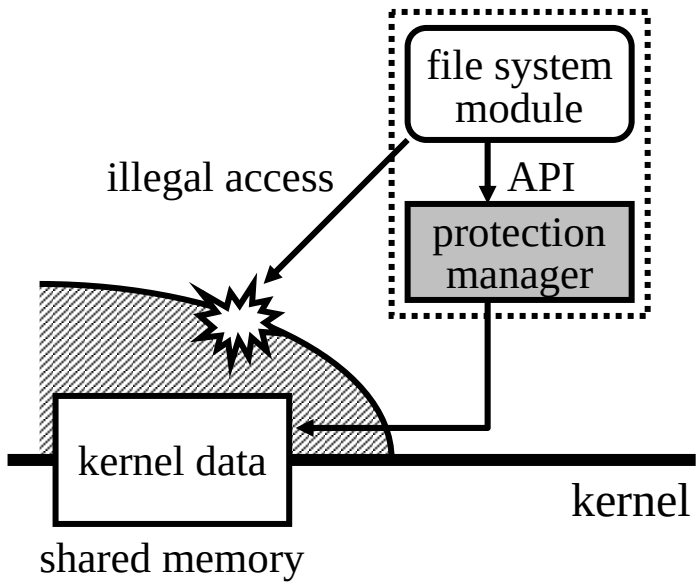


Common APIs

- All protection managers provide the same APIs so that modifying source code is not needed.
  - API for manipulating the kernel data
  - API for registering callback functions
- These APIs encapsulate differences among protection managers.

Implementation of multiple protection levels

- The protection managers use different combinations of protection techniques.
  - Illegal memory accesses
    - Switching address spaces
    - Memory protection
    - Replicating/checking data
    - Logging for recovery
  - Deadlocks
    - Wait-for-graph check



Experiments

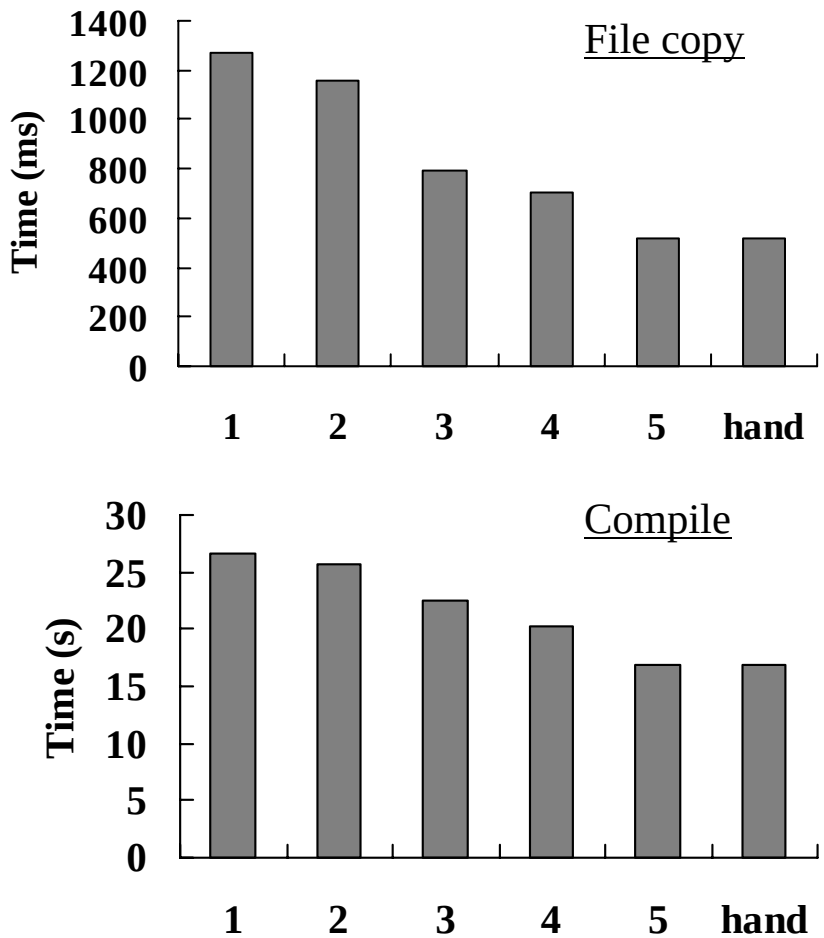
- We implemented NFS on CAPELA.
- We measured performance at five protection levels & a hand-crafted version.
  - copy a 64KB file on NFS
  - compile a program on NFS

	1	2	3	4	5
Memory protection	○	△			
Data replication/check	○	○	○		
Address space switch	○	○	○	○	
Logging	○	○	○	○	
Deadlock check	○	○	○	○	

SPARCstation 5 & PC(6x86/133MHz) connected with 10Mbps network

Results

- Performance is improved if a protection level is lowered.
- Overheads of the maximum protection level are acceptable for debugging. (120%, 60%)
- Overheads of the minimum protection level are negligible. (0.1%)



Related work

- Mach[Accetta et al.86]
  - A module is a user process involving high overheads.
- SPIN[Bershad et al.95]
  - Downloads a protected module into the kernel
  - The protection level cannot be changed.
- Chorus[Rozier et al.88]
  - A module runs at either the user level or the kernel level.
  - The protection level cannot be changed while debugging.

Conclusion and future work

Conclusion

- We proposed the multi-level protection.
  - Write once
  - Run at multiple protection levels
- Performance overheads are acceptable (at the highest level) and negligible (at the lowest level).

Future work

- We will apply this mechanism to a network subsystem and other subsystems.