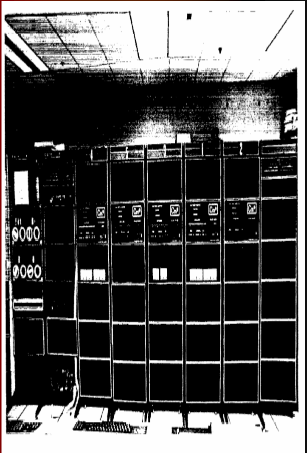
HYDRA – The kernel of a Multiprocessor Operating System by Wulf etc.

(Presentation By Alex Kachurin and Mohamed Saad Laassel)

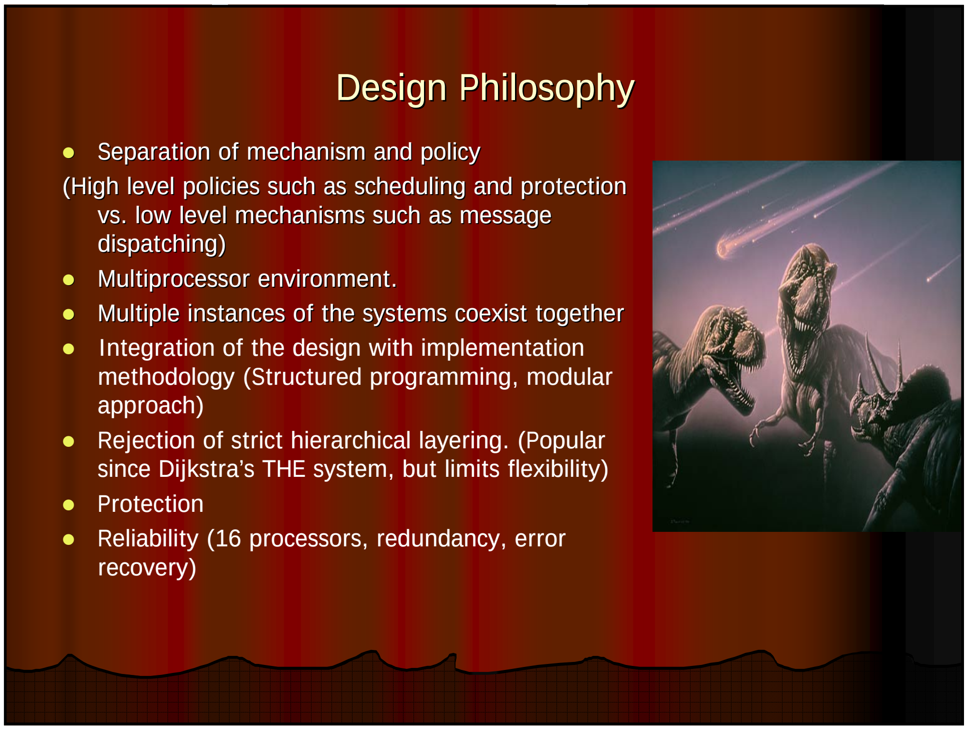
- Introduction To Hydra
- Design Philosophy
- Overview Of The Hydra Environment
- The Protection Mechanism
- Path Names And Walk Right
- Systems And Subsystems
- An Example (Bibliography System)
- Conclusion
- References

Introduction to Hydra

- Designed in early 70's, Carnegie-Melon University
- Not an OS by itself, but rather a kernel base for a collection of OS'es. (e.g Linux kernel vs. Debian Linux)
- Runs on C.mmp, a multiprocessor constructed at Carnegie-Melon University (up to 16 processors, up to 32 MB of memory, crossbar switch architecture vs. bus architecture)
- The goal is "to exploit and explore the potential inherent in a multiprocessor computer system".
- Provide an environment for effective utilization of the hardware resources
- To facilitate the construction of such environment



The Cm* computer. (Courtesy Dr. Zary Segall.)

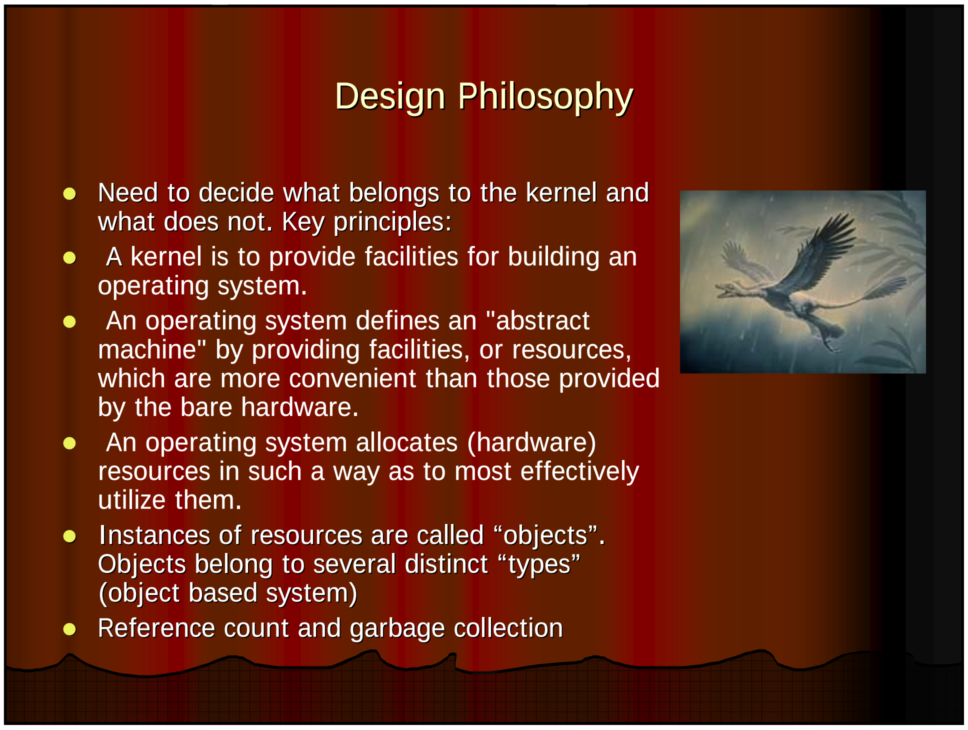


Design Philosophy

- Separation of mechanism and policy

(High level policies such as scheduling and protection vs. low level mechanisms such as message dispatching)

- Multiprocessor environment.
- Multiple instances of the systems coexist together
- Integration of the design with implementation methodology (Structured programming, modular approach)
- Rejection of strict hierarchical layering. (Popular since Dijkstra’s THE system, but limits flexibility)
- Protection
- Reliability (16 processors, redundancy, error recovery)

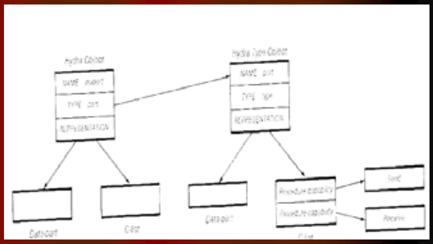


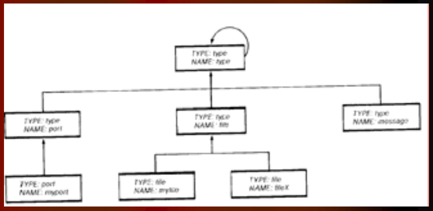
Design Philosophy

- Need to decide what belongs to the kernel and what does not. Key principles:
- A kernel is to provide facilities for building an operating system.
- An operating system defines an "abstract machine" by providing facilities, or resources, which are more convenient than those provided by the bare hardware.
- An operating system allocates (hardware) resources in such a way as to most effectively utilize them.
- Instances of resources are called “objects”. Objects belong to several distinct “types” (object based system)
- Reference count and garbage collection

Overview of Hydra Environment

- Key terms: Procedures, Local Namespaces (LNS), Processes and Capabilities
- Hydra procedures support protection facilities via templates (formal parameters list)
- Hydra procedures are reenterant and potentially recursive
- LNS is a record of execution environment at the time of invocation
- LNS is dynamic and gets generated each time a procedure is invoked, based on capabilities
- LNS gets erased when a procedure finishes
- Capability is a reference to an object along with a collection of access rights to this object.
- Capabilities are manipulated by the kernel so they can't be forged.
- Capabilities are stored in C-lists





Overview of Hydra Environment

- Hydra Process is the smallest entity that can be scheduled for execution
- Processes are represented as stacks of LNS, representing the

cumulative state of a single sequential task.

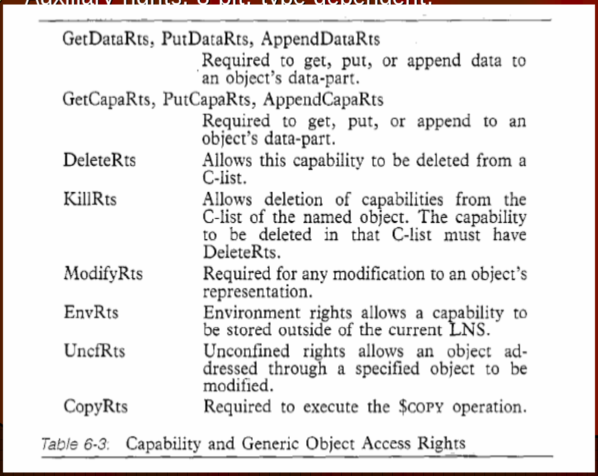
- Synchronization primitives (Semaphores, Locks, Mutexes)

The Protection Mechanism

- Protection vs. Security: protection is a mechanism, security is a policy
- Protection is procedure-based as opposed to process-based
- Procedure itself is an object so it has capabilities list (caller independent capabilities). Caller independent capabilities are inherited from the called process.
- Procedures have templates (formal parameters list). Access rights of the actual parameters are checked at the time of invocation.
- Callee (the called procedure) has more freedom than the calling procedure.
- Kernel provides CALL and RETURN mechanisms to instantiate a procedure or return to calling procedure.

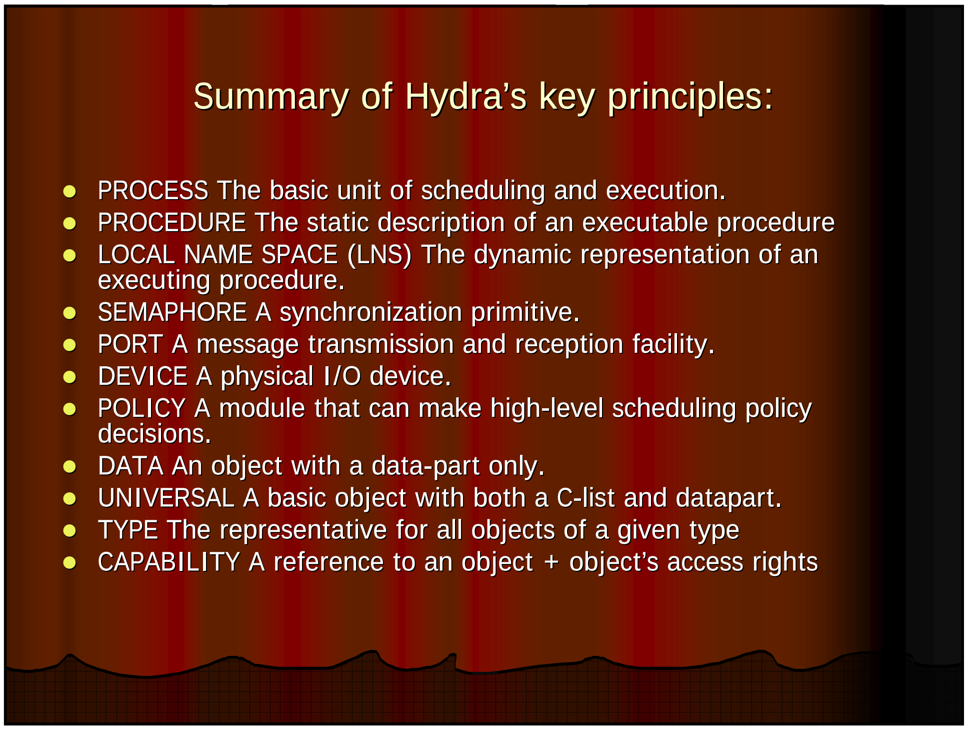
The Protection Mechanism

- Hydra Access Rights: Generic rights, 16-bit, type independent.
- Auxillary rights, 8-bit, type dependent.



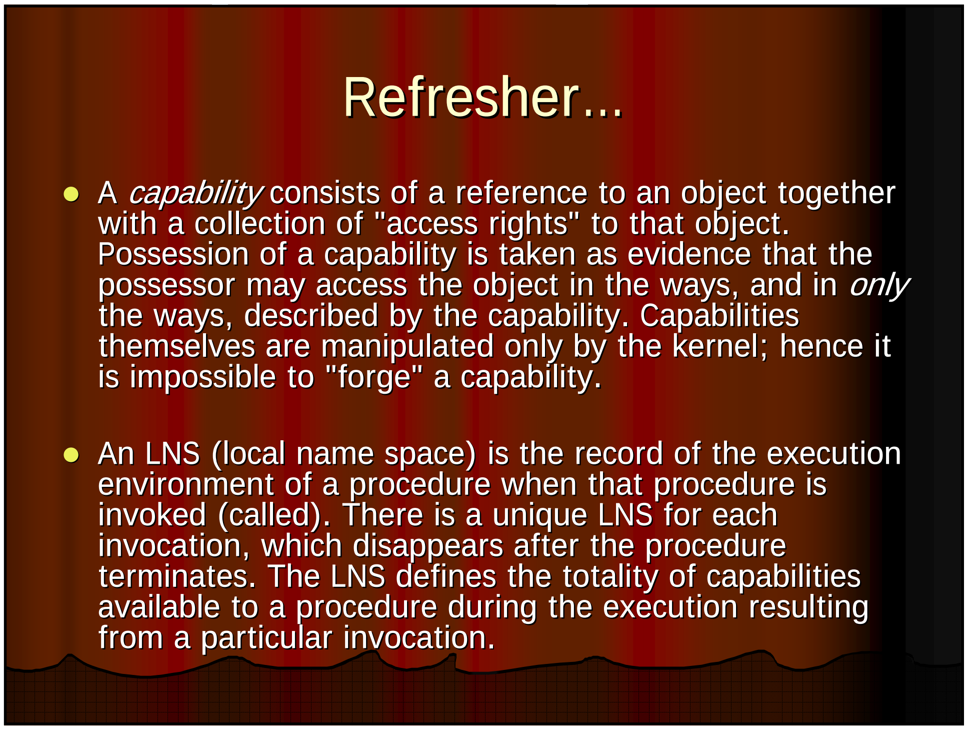
GetDataRts, PutDataRts, AppendDataRts	Required to get, put, or append data to an object's data-part.
GetCapaRts, PutCapaRts, AppendCapaRts	Required to get, put, or append to an object's data-part.
DeleteRts	Allows this capability to be deleted from a C-list.
KillRts	Allows deletion of capabilities from the C-list of the named object. The capability to be deleted in that C-list must have DeleteRts.
ModifyRts	Required for any modification to an object's representation.
EnvRts	Environment rights allows a capability to be stored outside of the current LNS.
UncfRts	Unconfined rights allows an object addressed through a specified object to be modified.
CopyRts	Required to execute the $COPY operation.

Table 6-3: Capability and Generic Object Access Rights



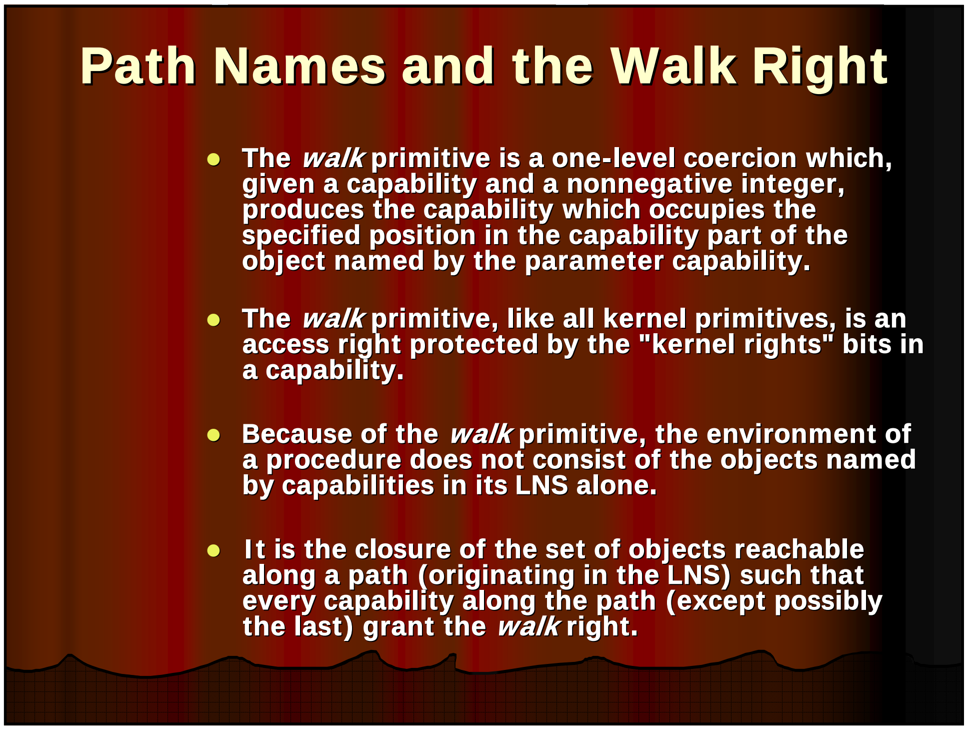
Summary of Hydra's key principles:

- PROCESS The basic unit of scheduling and execution.
- PROCEDURE The static description of an executable procedure
- LOCAL NAME SPACE (LNS) The dynamic representation of an executing procedure.
- SEMAPHORE A synchronization primitive.
- PORT A message transmission and reception facility.
- DEVICE A physical I/O device.
- POLICY A module that can make high-level scheduling policy decisions.
- DATA An object with a data-part only.
- UNIVERSAL A basic object with both a C-list and datapart.
- TYPE The representative for all objects of a given type
- CAPABILITY A reference to an object + object's access rights



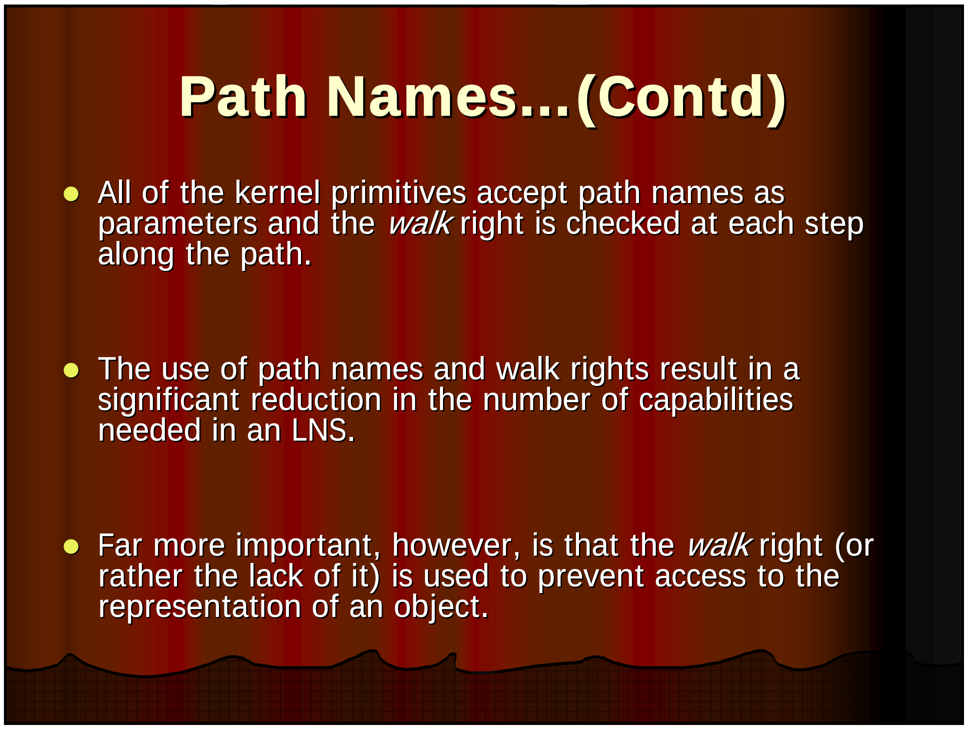
Refresher...

- A *capability* consists of a reference to an object together with a collection of "access rights" to that object. Possession of a capability is taken as evidence that the possessor may access the object in the ways, and in *only* the ways, described by the capability. Capabilities themselves are manipulated only by the kernel; hence it is impossible to "forge" a capability.
- An LNS (local name space) is the record of the execution environment of a procedure when that procedure is invoked (called). There is a unique LNS for each invocation, which disappears after the procedure terminates. The LNS defines the totality of capabilities available to a procedure during the execution resulting from a particular invocation.



Path Names and the Walk Right

- The *walk* primitive is a one-level coercion which, given a capability and a nonnegative integer, produces the capability which occupies the specified position in the capability part of the object named by the parameter capability.
- The *walk* primitive, like all kernel primitives, is an access right protected by the "kernel rights" bits in a capability.
- Because of the *walk* primitive, the environment of a procedure does not consist of the objects named by capabilities in its LNS alone.
- It is the closure of the set of objects reachable along a path (originating in the LNS) such that every capability along the path (except possibly the last) grant the *walk* right.



Path Names...(Contd)

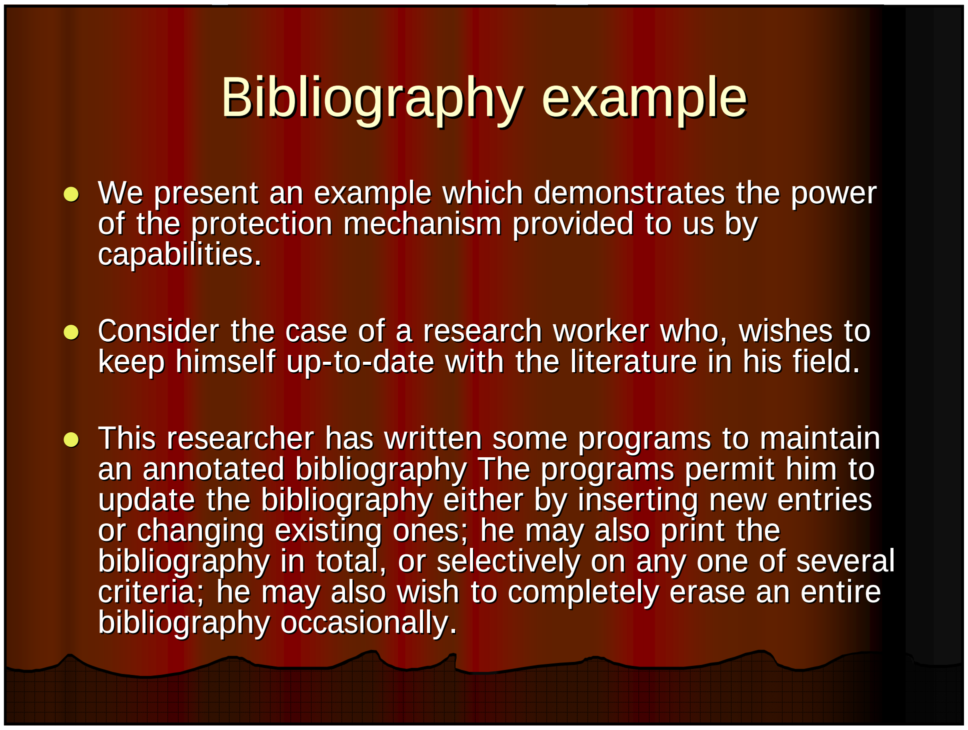
- All of the kernel primitives accept path names as parameters and the *walk* right is checked at each step along the path.
- The use of path names and walk rights result in a significant reduction in the number of capabilities needed in an LNS.
- Far more important, however, is that the *walk* right (or rather the lack of it) is used to prevent access to the representation of an object.

Systems and Subsystems

- In the HYDRA context a user environment consists of a collection of resources (objects) of various types and procedures which operate on them.
- The environment in which one user operates may or may not be the same as that for another user, it may be totally different, or may partially overlap.

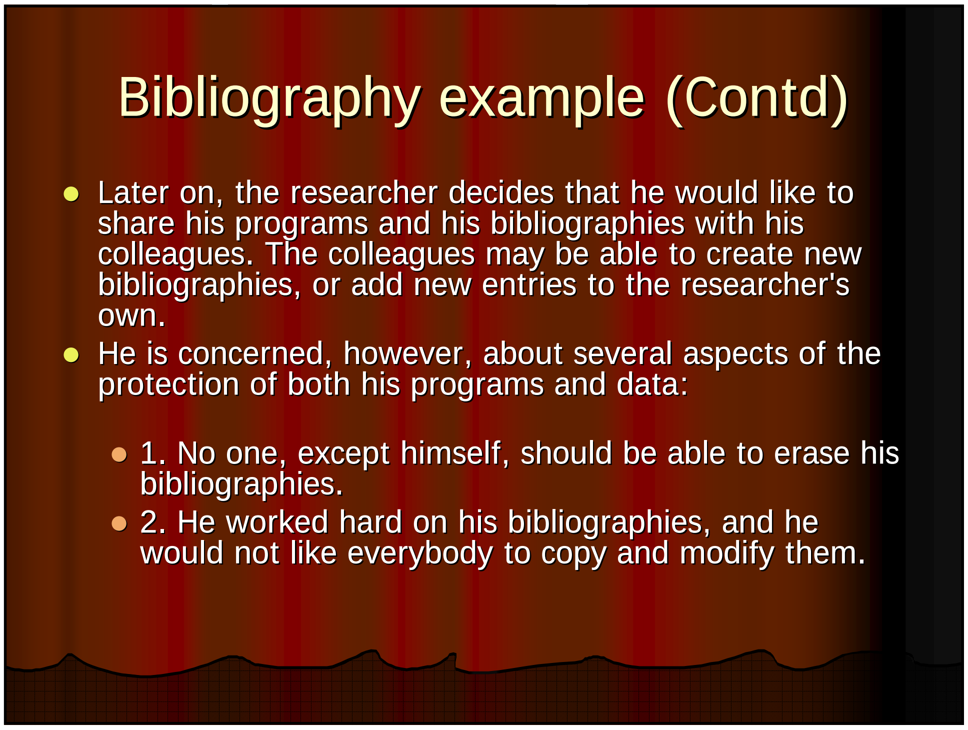
New object types

- First, an instance of an object may be created by invoking a kernel primitive *create* and passing to it a capability referencing the representative of type of object one wishes to create.
- Then, invoking *create* with a capability referencing the distinguished object named TYPE will create the representative of a new type class.
- Subsequent calls on *create* passing capabilities referencing this new type representative will create instances of the new class of objects.



Bibliography example

- We present an example which demonstrates the power of the protection mechanism provided to us by capabilities.
- Consider the case of a research worker who, wishes to keep himself up-to-date with the literature in his field.
- This researcher has written some programs to maintain an annotated bibliography The programs permit him to update the bibliography either by inserting new entries or changing existing ones; he may also print the bibliography in total, or selectively on any one of several criteria; he may also wish to completely erase an entire bibliography occasionally.



Bibliography example (Contd)

- Later on, the researcher decides that he would like to share his programs and his bibliographies with his colleagues. The colleagues may be able to create new bibliographies, or add new entries to the researcher's own.
- He is concerned, however, about several aspects of the protection of both his programs and data:
  - 1. No one, except himself, should be able to erase his bibliographies.
  - 2. He worked hard on his bibliographies, and he would not like everybody to copy and modify them.

The Implementation in HYDRA

- A bibliography is a new type of virtual resource. Therefore, we would create a new object type; call it BIBLIO.
- We create new procedures which are applicable to bibliography objects, for example:
  - U(β) Update
  - P(β) Print
  - PWOA(β) Print Without Annotations
  - E(β) Erase
- In each of these, β must be a capability which references a bibliography object.

Fig. 1. Bibliography example.

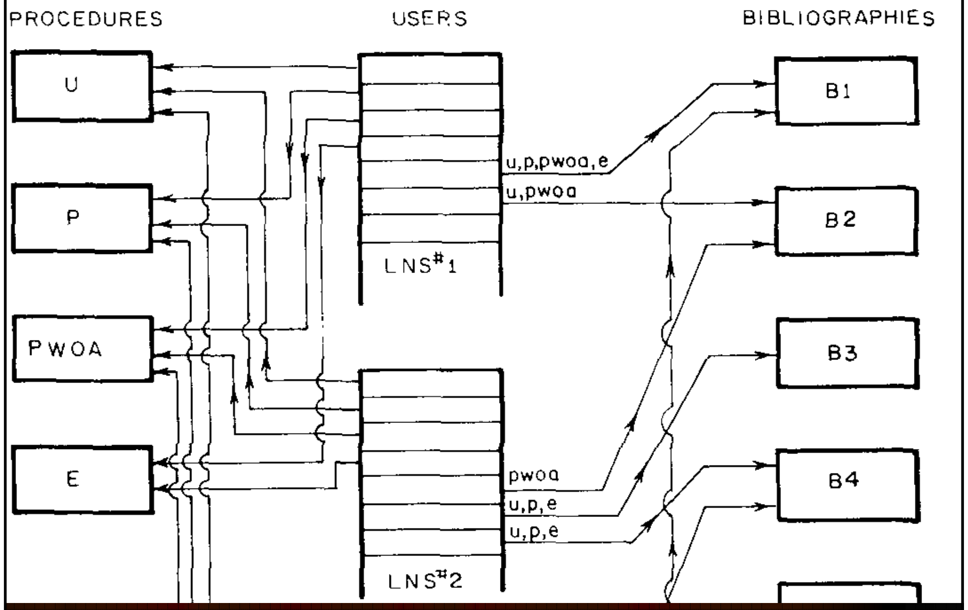
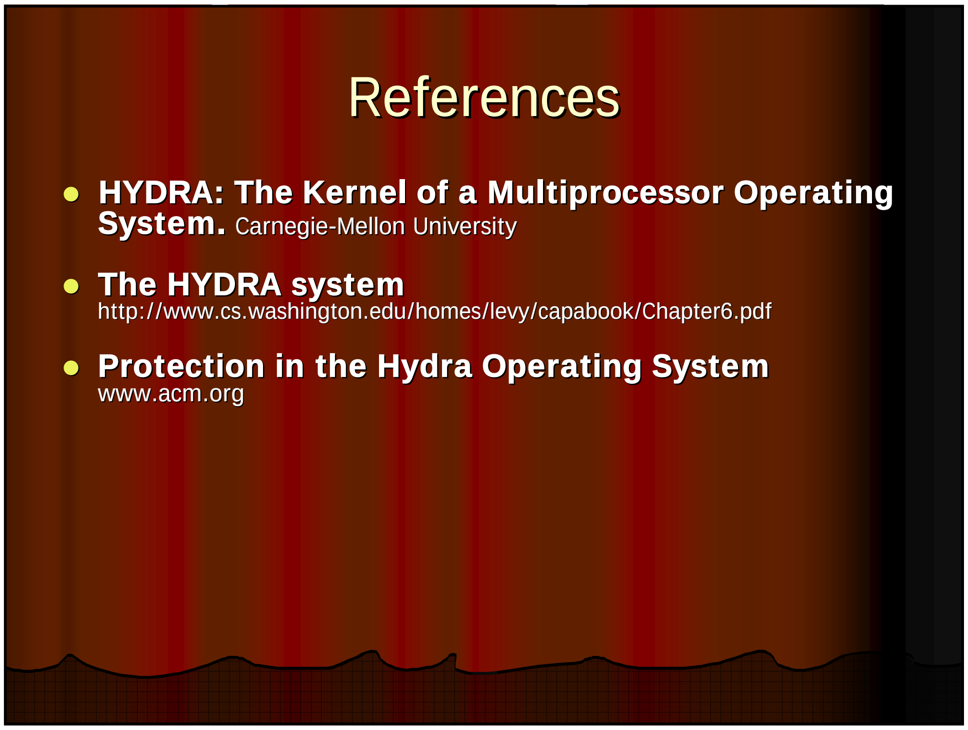


Diagram explanation

- User #1 may access all of the procedures U, P, PWOA, and E. He may also access bibliography objects B1 and B2. He may perform any of the operations U, P, PWOA, and E on B1, but he may only perform U and PWOA on B2.
- User #2 may also access all of the procedures and, in addition, may access three bibliography objects: B2, B3, and B4. He may only perform PWOA on B2, but may perform U, P, or E on B3 and B4.
- Assume that user# 2, did not have access to procedure E, the right to perform E on B3 and B4 is useless, since he does not have a reference to capability E.

Conclusion

- The HYDRA O.S, was designed with one ultimate goal in mind: to be the "kernel" base for a collection of operating systems designed to exploit and explore the potential of a multiprocessor computer system.
- This goal was realized through the introduction of a generalized notion of "resource," both physical and virtual, called an "object.“
- Mechanisms are presented for dealing with objects, including the creation of new types, specification of new operations applicable to a given type, sharing, and protection of any reference to a given object against improper application of any of the operations defined with respect to that type of object.
- The mechanisms provide a coherent basis for extension of the system in two directions: the introduction of new facilities, and the creation of highly secure systems.



References

- **HYDRA: The Kernel of a Multiprocessor Operating System.** Carnegie-Mellon University
- **The HYDRA system**
  http://www.cs.washington.edu/homes/levy/capabook/Chapter6.pdf
- **Protection in the Hydra Operating System**
  www.acm.org