PROTECTION

Vishwas Patil

Dipartimento di Informatica
Universitá degli Studi di Roma – La Sapienza
Via Salaria 113, 00198 Roma, Italy

http://www.dsi.uniroma1.it/˜patil

Protection?

- “A mechanism to control the access”

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.
  - by destroying or modifying another user’s data

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.
    - by destroying or modifying another user’s data
    - by reading or copying another user’s data without permission

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.
  - by destroying or modifying another user’s data
  - by reading or copying another user’s data without permission
  - by degrading the service another user gets (e.g., crash the system)

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.
  - by destroying or modifying another user’s data
  - by reading or copying another user’s data without permission
  - by degrading the service another user gets (e.g., crash the system)
- So, protection is necessary (convinced?)

Protection?

- “A mechanism to control the access”
- It is a general term for all the mechanisms that control the access of a program to other things in the system.
- Original Motivation: to keep one user’s malice or error from harming other users.
  - by destroying or modifying another user’s data
  - by reading or copying another user’s data without permission
  - by degrading the service another user gets (e.g., crash the system)
- So, protection is necessary (convinced?)
- Let us take an abstract approach to the subject.

Directions/Objectives

- towards enforcing rules of modular programming so that it is possible, using the protection system, to guarantee that errors in one module will not affect another one (reliability of a large system)

Directions/Objectives

- towards enforcing rules of modular programming so that it is possible, using the protection system, to guarantee that errors in one module will not affect another one (reliability of a large system)
- towards the support of proprietary programs, so that a user can buy a service in the form of a program which he can only call, but not read (a proprietary compiler)

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*
- A message consists of an identification of the sending process followed by an arbitrary amount of data

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*
- A message consists of an identification of the sending process followed by an arbitrary amount of data
- The identification is supplied by the system and therefore cannot be forged

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*
- A message consists of an identification of the sending process followed by an arbitrary amount of data
- The identification is supplied by the system and therefore cannot be forged
- processes are assigned integer names by the system in such a way that a given integer always refers to the same process; aside from this property the names have no significance

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*
- A message consists of an identification of the sending process followed by an arbitrary amount of data
- The identification is supplied by the system and therefore cannot be forged
- processes are assigned integer names by the system in such a way that a given integer always refers to the same process; aside from this property the names have no significance
- The sending process supplies the data. Any process may send messages (at its expense) to any other process

The Message System

- Message system consists of processes that share nothing and communicate with each other only by means of *messages*
- A message consists of an identification of the sending process followed by an arbitrary amount of data
- The identification is supplied by the system and therefore cannot be forged
- processes are assigned integer names by the system in such a way that a given integer always refers to the same process; aside from this property the names have no significance
- The sending process supplies the data. Any process may send messages (at its expense) to any other process
- Messages are received one at a time in the order in which they were sent

The Message System

- Within this system everything belongs to some process and cannot be accessed by any process other than its owner

The Message System

- Within this system everything belongs to some process and cannot be accessed by any process other than its owner
- Each process is therefore a single domain

The Message System

- Within this system everything belongs to some process and cannot be accessed by any process other than its owner
- Each process is therefore a single domain
- It can also be viewed as a separate machine, complete with memory, file storage, tape units, etc., and isolated by hardware from all other processes except for the message transmission system

The Message System

- Within this system everything belongs to some process and cannot be accessed by any process other than its owner
- Each process is therefore a single domain
- It can also be viewed as a separate machine, complete with memory, file storage, tape units, etc., and isolated by hardware from all other processes except for the message transmission system
- This scheme provides a logically complete protection system.

Analysis of the Message System

- Subroutine calls: Process A and B, A calls B

Analysis of the Message System

- Subroutine calls: Process A and B, A calls B
- To call B, A sends B a message specifying the parameters and then waits for B to reply

Analysis of the Message System

- Subroutine calls: Process A and B, A calls B
- To call B, A sends B a message specifying the parameters and then waits for B to reply
- To return, B replies with another message containing the value, if any, and then waits for another call

Analysis of the Message System

- Protection: CALL and RETURN

Analysis of the Message System

- Protection: CALL and RETURN
- CALL:
  - called program knows what are valid entry points and has control over its execution

Analysis of the Message System

- Protection: CALL and RETURN
- CALL:
  - called program knows what are valid entry points and has control over its execution
- RETURN:
  - The caller knows from which subroutine/process it is expecting a return value (trusted third-party – guard)

Analysis of the Message System

- Protection: CALL and RETURN
- CALL:
  - called program knows what are valid entry points and has control over its execution
- RETURN:
  - The caller knows from which subroutine/process it is expecting a return value (trusted third-party – guard)
  - spurious returns are ignored

Analysis of the Message System

- Protection: CALL and RETURN
- CALL:
  - called program knows what are valid entry points and has control over its execution
- RETURN:
  - The caller knows from which subroutine/process it is expecting a return value (trusted third-party – guard)
  - spurious returns are ignored
  - run-away processes! (bug/error/malicious subroutine)

Analysis of the Message System

- Protection: CALL and RETURN
- CALL:
  - called program knows what are valid entry points and has control over its execution
- RETURN:
  - The caller knows from which subroutine/process it is expecting a return value (trusted third-party – guard)
  - spurious returns are ignored
  - run-away processes! (bug/error/malicious subroutine)
- unauthorized domain Y tries to call B

Flaws in the Message System

- It is impossible to retain control over a runaway process, since there is no way to force a process to do anything, or to destroy it

Flaws in the Message System

- It is impossible to retain control over a runaway process, since there is no way to force a process to do anything, or to destroy it
- An elaborate system of conventions is required to get processes to cooperate

Access Matrix

- In order to provide facilities for controlling processes from outside, it is necessary to have a systematic way of controlling access to one process from others

Access Matrix

- In order to provide facilities for controlling processes from outside, it is necessary to have a systematic way of controlling access to one process from others
- In order to provide useful conventions for sharing among processes, it is necessary to have a systematic way of describing what is to be shared and of controlling access to shared things from various processes

Access Matrix: Object System

- three major components:
  - a set of objects – X
  - a set of domains – D
  - an access matrix or access function – A

Access Matrix: Object System

- Objects are the things in the system which have to be protected

Access Matrix: Object System

- Objects are the things in the system which have to be protected
- Typical objects in existing systems (1971) are processes, domains, files, segments, and terminals

Access Matrix: Object System

- Objects are the things in the system which have to be protected
- Typical objects in existing systems (1971) are processes, domains, files, segments, and terminals
- Objects have names with global validity

Access Matrix: Object System

- Objects are the things in the system which have to be protected
- Typical objects in existing systems (1971) are processes, domains, files, segments, and terminals
- Objects have names with global validity
- Object names are handed out by the protection system on demand, and their interpretation is up to the programs that operate on the objects

Access Matrix: Object System

- Objects are the things in the system which have to be protected
- Typical objects in existing systems (1971) are processes, domains, files, segments, and terminals
- Objects have names with global validity
- Object names are handed out by the protection system on demand, and their interpretation is up to the programs that operate on the objects
- In the message system, each domain was also a process and had exclusive access to its own objects and none to any others.

Access Matrix: Object System

- Objects are the things in the system which have to be protected
- Typical objects in existing systems (1971) are processes, domains, files, segments, and terminals
- Objects have names with global validity
- Object names are handed out by the protection system on demand, and their interpretation is up to the programs that operate on the objects
- In the message system, each domain was also a process and had exclusive access to its own objects and none to any others.
- This idea is now being generalized so that objects can be shared between domains

Access Matrix: Object System

- Each domain owning objects in the message system agrees by convention that it will do certain things with these objects upon demand from some other domain, provided the other domain has access according to the rules

Access Matrix: Object System

- Each domain owning objects in the message system agrees by convention that it will do certain things with these objects upon demand from some other domain, provided the other domain has access according to the rules
- Note that domains are objects, and that objects do not ’live in’, or ’belong to’ domains

Access Matrix

- The access of domains to objects is determined by the access matrix *A*

Access Matrix

- The access of domains to objects is determined by the access matrix *A*
- Its rows are labeled by domain names and its columns by object names

Access Matrix

- The access of domains to objects is determined by the access matrix A
- Its rows are labeled by domain names and its columns by object names
- Element $A_{i,j}$ specifies the access which domain i has to object j

Access Matrix

- The access of domains to objects is determined by the access matrix A
- Its rows are labeled by domain names and its columns by object names
- Element $A_{i,j}$ specifies the access which domain i has to object j
- Each element consists of a set of strings called access attributes

Access Matrix

- The access of domains to objects is determined by the access matrix A
- Its rows are labeled by domain names and its columns by object names
- Element $A_{i,j}$ specifies the access which domain i has to object j
- Each element consists of a set of strings called access attributes
- a domain has 'x' access to an object if 'x' is one of the attributes in that element of A

Access Matrix

- The access of domains to objects is determined by the access matrix A
- Its rows are labeled by domain names and its columns by object names
- Element $A_{i,j}$ specifies the access which domain i has to object j
- Each element consists of a set of strings called access attributes
- a domain has 'x' access to an object if 'x' is one of the attributes in that element of A
- Attached to each attribute is a bit called the copy flag which controls the transfer of access

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

*copy flag set

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

*copy flag set

- For example, domain 1 has ’owner’ access to file 1 as well as explicit ’read’ and ’write’ access. It has given ’read’ access to this file to domains 2 and 3

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

*copy flag set

- For example, domain 1 has ’owner’ access to file 1 as well as explicit ’read’ and ’write’ access. It has given ’read’ access to this file to domains 2 and 3
- Entries in the access matrix are made and deleted according to certain rules

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

A domain d_1 can modify the list of access attributes for domain d_2 and object x as follows

- d_1 can remove access attributes from $A_{d2,x}$ if it has 'control' access to d_2. Example: D_1 can remove attributes from rows 1 and 2

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

A domain d_1 can modify the list of access attributes for domain d_2 and object x as follows

- d_1 can copy to $A_{d2,x}$ any access attributes it has for x which have the copy flag set, and can say whether the copied attribute shall have the copy flag set or not. Example: D_1 can copy ’write’ to $A_{2,File_1}$

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

A domain d_1 can modify the list of access attributes for domain d_2 and object x as follows

- d_1 can add any access attributes to $A_{d2,x}$, with or without the copy flag, if it has 'owner' access to x. Example: D_2 can add 'write' to $A_{2,File_2}$

Access Matrix

	D_1	D_2	D_3	$File_1$	$File_2$	$Process_1$
D_1	*owner control	*owner control	*call	*owner *read *write		
D_2			call	*read	write	wakeup
D_3			owner control	read	*owner	

A domain d_1 can modify the list of access attributes for domain d_2 and object x as follows

- d_1 can remove access attributes from $A_{d2,x}$ if d_1 has 'owner' access to x, provided d_2 does not have 'protected' access to x