# Protection by Operating System

(Group Project)

Available: October 11, 2022

Due: October 27, 2022

This project is worth 10% of the overall weight for this course.

### Learning Outcomes and Objectives

|  |
| --- |
| After successful completion of this assignment, students will be able to:   1. Understand and simulate an operating system’s (OS’s) protection mechanism. 2. Use an access matrix, access control lists, and capability lists to provide protection. 3. Use and improve their critical thinking, problem solving, and software development skills to transform given project specifications into an implementation using threads. 4. Write robust code and test their solution to ensure resilience against improper inputs. 5. Communicate with the instructor and teaching assistants to seek a greater understanding and clarification on topics they need support with. |

### Choice of Implementation

|  |
| --- |
| This project can be implemented in either standalone Java or POSIX threads (pthreads) with C/C++. Both options are supported with documentation provided on Moodle. This includes some commented source code, library tutorials, and supportive videos. This project specification is written with reference to both implementation choices, and relevant hints are provided, where applicable. |

### Student Outcomes Assessed for ABET Accreditation

| Student Outcomes (SOs) Assessed |
| --- |
| SO 1. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions. |
| SO 5. Function effectively as a member or leader of a team engaged in activities appropriate to the program’s discipline. |
| SO 6. Be proficient in more than one programming language on more than one computing platform. |

### Detailed Instructions

This project will help you understand user access and security through implementation of the following tasks:

1. Implementation with an Access Matrix
2. Implementation with an Access List of Objects
3. Implementation with Capability Lists for Domains
4. Command Line
5. Report

This document will give a detailed specification of the above tasks. In addition, it will provide design and implementation hints as preparation for further work. The focus of this assignment will be implementing user access control schemes.

## Task 1: Implementation with an Access Matrix

You will be devising and implementing an access matrix-based solution to provide protection to objects as well as guard against unauthorized domain switches.

You will be implementing an access matrix, of size N x (M + N) which accounts for storing access rights for N number of domains (each modeled by a thread) against M number of objects (each modeled as a character buffer) plus the (N-1) other domains to allow or disallow switching (ignoring switching into its own domain). For each run, your program should dynamically generate N and M, and populate the access matrix randomly as well. That is, randomly generate N and M to exist in the range [3,7] similar to past assignments. An example of this matrix is shown below:

| Object / Domain | F1 | F2 | F3 | D1 | D2 | D3 |
| --- | --- | --- | --- | --- | --- | --- |
| D1 | R |  | R/W | - |  | allow |
| D2 |  | W |  | allow | - |  |
| D3 | R | R |  | allow |  | - |

Each entry of the access matrix stores the rights a domain has on an object. During the program’s run, at random times, the users (i.e., threads that represent each domain) attempt to access different objects. To simplify this, either attempt to change the object (write) or print the buffer contents (read). Each request should be followed by yielding the process [3,7] times. If access to an object is allowed, claim the semaphore and yield. If disallowed, yield, then continue. Such access request from the users is to be handled by an arbitrator function, which examines the access matrix against the user’s rights to access requested object, and then either grants or rejects access to the requested object.

The users may also wish to switch domains at times (i.e., acquire the access rights of another domain or user, as allowed by the access matrix). Whenever a user wishes to change domain, its request is handled by the arbitrator function, which check the access matrix to see if requested switch would be permissible. If so, permission is granted. If not, permission to switch is denied. Make sure to have some struct in place to switch permissions of the thread, updating its read/write permissions respectively.

As this project is a simulation of an access matrix based protection, you are requested to print output statements denoting events such as creation and display of access matrix, request for an object by a user/process/domain/thread, checking of request by arbitrator function, permission allowed or denied, thread is yielding X times, domain switching requests, and whether domain switching was allowed.

In each run, your program should generate at least 5 requests from each thread. Generate a random number X in the range of [0, M+N] to correspond to a column in the access matrix. If X <= M, generate another number [0,1] to either read/write from object M. If M < X <= M+N, then attempt to switch to domain (M+N) – X (if it is the current domain, regenerate X). Note that semaphores/locks must be used to access objects properly. More specifically, use synchronization to protect access to the objects to disallow one user from reading an object being written to, or writing to an object begin read, etc. If a user is granted access to an object, it must yield [3,7] times before releasing the semaphore. After each request, the thread must yield to allow other threads to proceed first.

See Task 4 for how to run this program in your chosen implementation.

### Report

As part of your report, answer the following question about Task 1:

1. Is there any chance of deadlock in this simulation? What changes could cause deadlocks?

## Task 2: Implementation with Access Lists for Objects

Instead of using an access matrix, as you did in Task 1, you will be using access lists for objects and the arbitration function will be using such lists to determine legality of access and domain switches. All other specifications for this task are the same as in Task 1 above.

The access list should be of size M + N. Each domain entry should correspond to whether or not a user can switch domains. Each object will know what operations are allowed upon it (read, write, read/write). Note that Access Lists focus primarily on object-based access.

In each run, your program should generate at least 5 requests from each thread with similar randomness to the previous task. Note that semaphores/locks must be used to access objects properly. Again, protect each object with a semaphore, and if a thread is granted access to an object, yield [3,7] times before releasing it. At the end of the request, yield again to let another user in.

See Task 4 for how to run this program in your chosen implementation.

### Report

As part of your report, answer the following questions about Task 2:

1. How does this task compare with Task 1? Which is easier to implement?

## Task 3: Implementation with Capability Lists for Domains

Instead of using an access list, as you did in Task 2, you will be using capability lists for domains, and the arbitration function will be using such lists to determine legality of access and domain switches. All other specifications for this task are the same as in both Task 1 and Task 2.

A capability list is domain based and should be of size N (number of domains). Each entry in the list corresponds to the objects that the domain has access to. This can be more general than the previous tasks; you have the option to only specify whether a domain has access to the object or not (allowing it to perform all operations), or continue allowing specific R, W, and R/W rights.

In each run, your program should generate at least 5 random requests from each thread. Note that semaphores/locks must be used to access objects properly. Again, protect each object with a semaphore, and if a thread is granted access to an object, yield [3,7] times before releasing it. At the end of the request, yield again to let another user in.

See Task 4 for how to run this program in your chosen implementation.

### Report

As part of your report, answer the following questions about Task 3:

1. What disadvantages do capability lists introduce given a more domain-based approach?

## Task 4: Command Line

For this task, you are required to add a command line argument to your program that will select which task to run. Invalid or incomplete values should result in an error message and should not cause a segfault or otherwise crash your program.

The list of valid command line arguments is as follows:

| Argument | Task |
| --- | --- |
| -S 1 | Task 1: Access Matrix |
| -S 2 | Task 2: Access List for Objects |
| -S 3 | Task 3: Capability List for Domains |

If you are running your program directly from the command line, add the argument to the end of the command you use to run the program (as in ./main.exe -S 1).

For Java in IntelliJ, you can pass one or more arguments to your program using the Program Arguments field under Run > Edit Configurations. For example, for task 1, you would type “-S 1” here.

## Task 5: Report

Please submit a detailed report describing your design and implementation. In addition to the questions listed under each task, the report should answer the following question:

1. What is the importance of protection in a multi-user system? What could happen if all users were granted equal permission?

Your report should be in .pdf, .txt, .doc, or .odt format. Other formats are acceptable, but you run the risk of the TA being unable to open or read it. Such reports will receive 0 points with no opportunity for resubmission.

Clearly include the name and ULID of all group members in your report. The questions in the report should be arranged by their associated task, then numbered. There is no minimum length, although insufficient detail in your answers will result in a penalty.

## Submission

Throughout your code, use comments to indicate which sections of code have been edited by you. This will help us better find and grade your work. For example:

//Begin code changes by <your name here>.

...

//End code changes by <your name here>.

**For C/C++**, use the following directory structure, then compress to a .zip or .tar.gz archive.

project02\_<ULID>

│

└───project02\_<ULID>\_report.(pdf|txt|doc|odt)

│

└───code

│

└───main.c

│

└───*If needed:* <sources>.(c|cpp), <sources>.h, <subfolders>

Here, <sources>.(c|cpp), <sources>.h, and <subfolders> refer to any extra source files, headers, or subfolders your program needs. Name the source file with your main() function (where program execution will start) main.c or main.cpp.

You do not need to include any object files (of type .o), link libraries, or compiled executables in your submission. Please include the command used to compile your project in your report, as the TA may not use the same type of machine or specific development environment that you do.

**For Java**, use the following directory structure, then compress to a .zip or .tar.gz archive.

project02\_<ULID>

│

└───project02\_<ULID>\_report.(pdf|txt|doc|odt)

│

└───code

│

└───.idea

│

└───code.iml

│

└───src

│

└───com

│

└───company

│

└───Main.java

│

└───*If needed:* <sources>.java, <subfolders>

The names of the folders labeled here <com> and <company> may vary based on how you have set up your project; the folder hierarchy may be deeper or shallower than the one depicted here. The important part is that the innermost folder under src contains your Java code.

Here, <sources>.java and <subfolders> refer to any extra source files or subfolders your program needs. Name the source file with your main() method (where program execution will start) Main.java.

You do not need to include any files of type .class or .jar in your submission.

Do not submit a paper copy.

Late and improper submissions will receive a maximum of 50% credit for the first 24 hours after the deadline and zero credit afterward.

Submit your code as well as a snapshot of your code execution.

## Hints

You are expected to validate user input. The same concerns about acceptable input types and edge cases within that type apply. Under no circumstances should your code segfault, initiate a stack dump, or terminate with an assertion failure or an uncaught exception. It will be thoroughly tested with a wide range of valid and invalid inputs to check for this.

Your code should gracefully handle any errors that crop up. Graceful handling means not letting the program crash, but instead catching and managing the error so that, at the very least, the program can exit normally. Any crash, segfault, or otherwise abnormal program termination will result in a penalty, no matter the cause, so it is in your best interest to use good error-handling practices.