Computer Architecture and Technology Area

Universidad Carlos III de Madrid



OPERATING SYSTEMS

Lab 1. System Calls

**BACHELOR'S DEGREE IN COMPUTER SCIENCE AND**

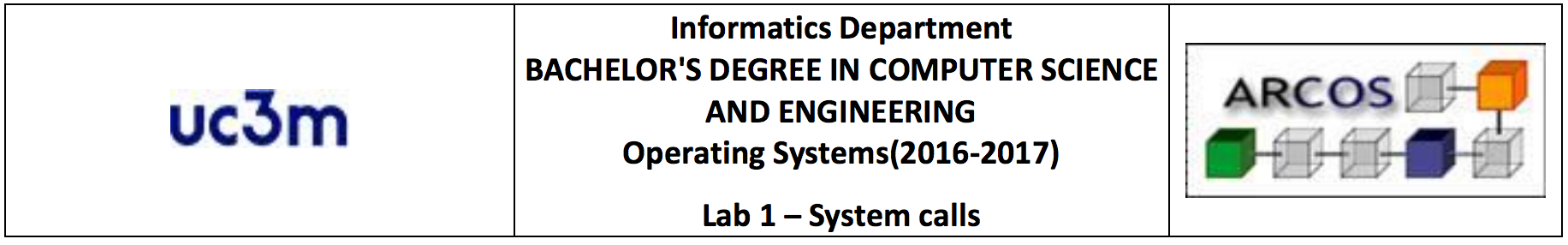
**ENGINEERING**

Year 2016/2017

AUTHORS:

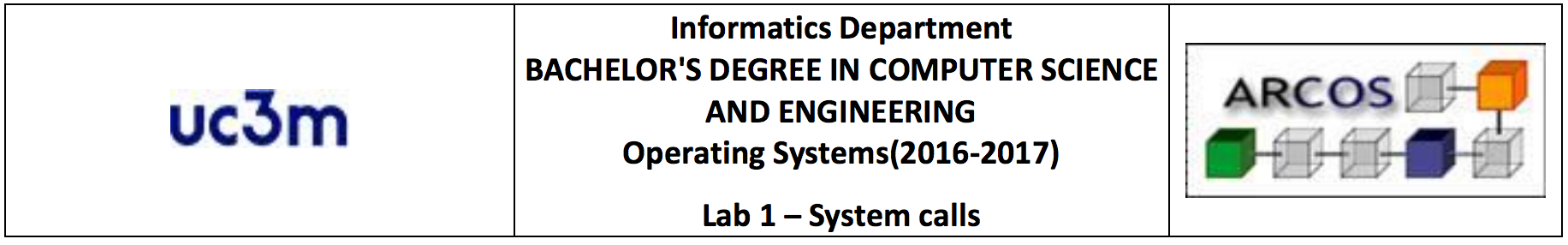
Ryan Cohane (100370386)

Carine Torres (100370405)



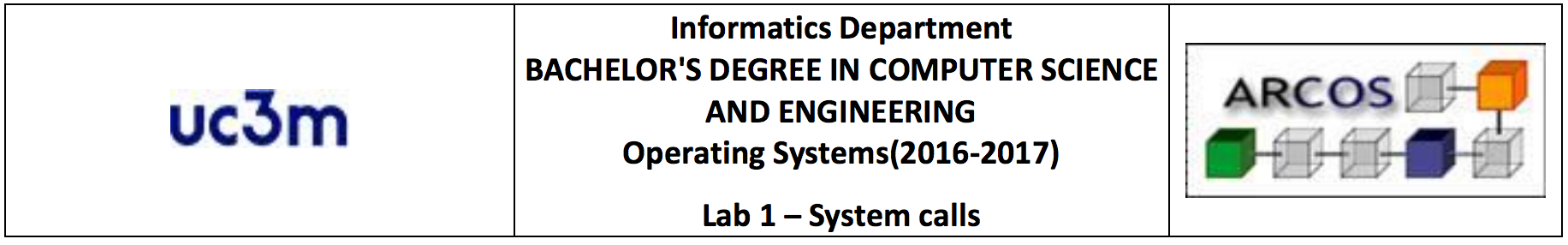
**Index**

1. Description of Code ………………………………………………………………………………… 3
   1. mycat ………………………………………………………………………………… 3
   2. myls …………………………………...……………………………………………… 4
   3. mysize ……………………………………………..………………………………… 5
2. Description of Tests …...…………………………………………………………………………… 6
   1. mycat ………………………………………………………………………………… 6
   2. myls …………………………………...……………………………………………… 7
   3. mysize ……………………………………………..………………………………… 8
3. Conclusions …...……………………………………………………………..………………………… 9

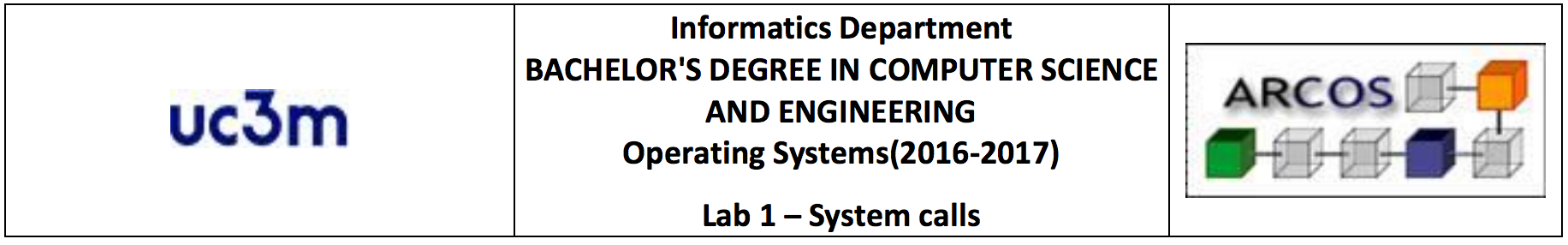


**1. Description of Code**

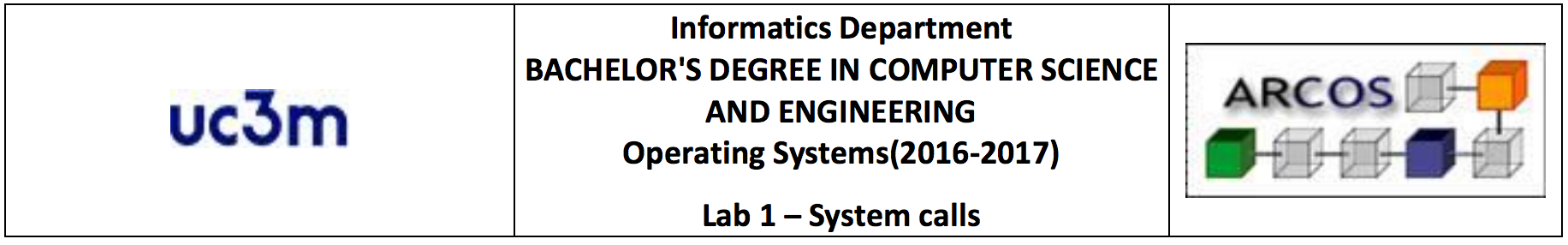
**mycat**



**myls**



**mysize**



**1. Description of Tests**

**mycat**

The first program, **mycat**, was intended to open a file specified by argument and show the “whole contents” of the file through standard output using the calls *open, read, write,* and *close*.

*Test 1: Basic Functionality*

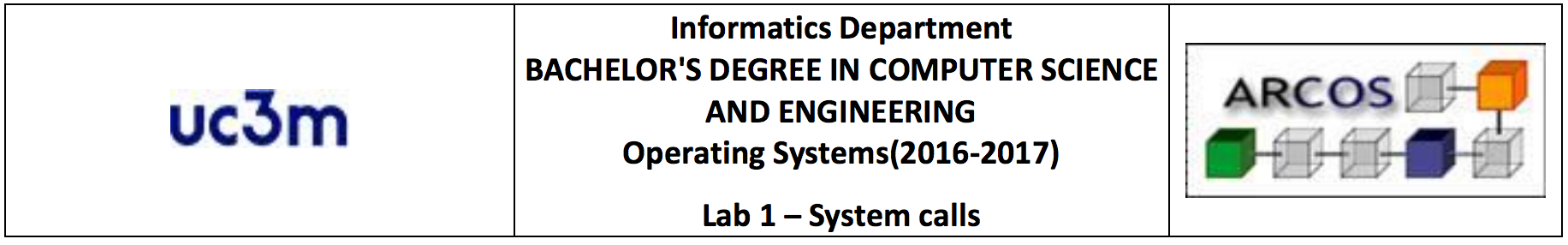
Evidently, **mycat** should thus accomplish the same functionality as the built-in Linux function **cat**. Thus, our first test of **mycat** included a comparison of the output of running the program on f1.txt, a file provided in the p1\_tests directory, with the output of running **cat** on f1.txt. Indeed, the output was the same. This thus proved that **mycat** indeed reads files with content and writes such content to standard output because that is what **cat** does.

*Test 2: Functionality with Empty Files*

One edge case to test is whether or not **mycat** knows how to handle empty files. This is important because having 0 contents in the file should not result in returning -1, but 0, when the file is read. Thus, we tested whether running **mycat** on f3.txt, an empty file added to the p1\_tests directory, resulted in no error, nor any output. Because it resulted in neither, we are confident that **mycat** can correctly read and write the contents of a 0-byte file.

*Test 3: Functionality with Large Files*

Another edge case to test is on the opposite end—whether or not **mycat** knows how to handle large files. This is important because even if the number of bytes in the file surpasses the buffer size, the buffer should have no problem reading and writing a set amount of bytes (1024 in this case), and then reading and writing in the next set amount of 1024B in the file, and so on for however many bytes are in the file. Thus, no matter what, the entire contents of a file should be written out and processed because all that will happen is that 1024B are read at a time, until the end of the file is reached. We thus created f4.txt, a file of 75KB that we added to the p1\_tests directory, and on which we ran **mycat**. Doing so resulted in no errors, and every character of f4.txt was actually output, so we know **mycat** is successful on larger files.



**myls**