Operating systems: Laboratory practice 1
System calls

uc3m | Universidad Carlos III de Madrid

Héctor Molina Garde.

.

Computer Science Engineering Degree

Nicolás Maire Bravo.

Computer Science Engineering Degree

Guillermo González Avilés.

Computer Science Engineering Degree

Code description:

Every .c file is descriptive by each function they have. In addition, the taken design decisions are explained in the ‘main()’ function of each file description as it employs all the code towards the made decisions.

1. crear.c file:

The crear.c file has just two functions: ‘get_mode()’ and ‘main()’.

a. get_mode()

The ‘*get_mode(char *arg_mode)*’ function analyzes the correctness of the permissions mode input number as a 3-digit octal number. First, it checks whether the length of the input string(arg_mode) is 3 or 4. Additionally, it checks that if the length is 4, the first character of the string must be a ‘0’ which indicates the octal number. Then, a loop checks that the other characters of the strings are numbers in range of [0,7], i.e. the numbers in base 8. Finally, the function uses the function strtol() which converts from ‘**char*’ to ‘*mode_t*’ and returns the value.

b. main()

The `*main()*’ function is the skeleton of the program, it creates a new file with a specified name and permission mode based on the input. It first checks that a correct number of arguments is provided; if not, it prints an error message and exits. The program extracts the file name from the arguments and converts the provided mode string into an octal `*mode_t*` value using `*get_mode()*`. It then saves the current file creation mask (`*umask*`), and sets it to `*0*` to ensure exact permission settings, and attempts to create the file using `*open()*` with the `*O_CREAT | O_EXCL | O_TRUNC*` flags, ensuring the file is newly created and truncated if it already exists. If the file creation fails, it prints an error message, restores the previous `umask`, and exits. If the file was successfully created, it closes it, restores the original `umask`, and exits normally the program.

2. combine.c file:

a. eq_alumno()

The function ‘*eq_alumno()*’ compares two ‘*alumno*’ structures by checking if their ‘nombre’ (name) fields are identical. It uses the strcmp() function, to check if the strings are the same.

b. check_duplicated_alumno()

The function ‘*check_duplicated_alumno()*’ scans the array of ‘*alumno*’ structures to determine if there are any duplicate names. It uses a nested loop, where each student is compared against all subsequent students using the ‘*eq_alumno()’* function. If a duplicate is found, it immediately returns an error; otherwise, after checking all pairs, it returns 0.

c. bubble_sort()

The function ‘*bubble_sort()*’ is a straightforward implementation of the simple algorithm that sorts in ascending order by the student marks.

d. make_csv()

The function ‘*make_csv()*’ generates the requested CSV named "estadisticas.csv" that contains statistical data about student grades. It categorizes grades into five groups: M (10), S (9), N (8-7), A (6-5), and F (4-0). Which counts how many students fall into each category and calculates the percentage of the total student count for each category. It then writes this information into the CSV file, ensuring proper formatting.

e. read_file()

The function ‘*read_file()*’ reads student records from a binary file into an array of struct ‘alumno’. It opens the file in read-only mode and iterates through it, reading ‘*sizeof(struct alumno)*’ bytes at a time. It ensures no more than a hundred students are read and detects potential issues such as incomplete or corrupted data.

f. write_students()

The function ‘*write_students()*’ writes the sorted student array to an output file in binary format. It opens the file in write mode, creating it if it does not exist, and overwriting any existing content. It then writes each student structure to the file using ‘*write()*’. After writing to all students, the file is properly closed to prevent data loss.

g. main()

Finally the function ‘*main()*’ orchestrates the entire program. It first checks if the correct number of arguments is provided, ensuring three file paths are given (two input files and one output file). To avoid the memory leaks due to the use of dynamic memory arrays, a fixed size array of a hundred as it can not be bigger. Then, the number of students is setted by a variable that registers it. The unique disadvantage is the necessity of passing those two variables for each function that needs them. Hence, with that explanation done, the function reads student records from both input files with ‘*read_file()*’ and ensures that the total number of students does not exceed MAX_STUDENTS. After, the array is passed to the function ‘*check_duplicated_alumno()*‘, if duplicate names are found, it prints an error and exits. Otherwise, it sorts the students by grade with ‘*bubble_sort()*’, writes the sorted list to the output file ‘*with write_file()*’, and then generates a CSV file containing grade statistics with ‘*make_csv()*’. This function ensures proper error handling at each step to maintain data integrity.

As a remark, in the whole file proper errors handling were placed after all the function syscalls to ensure the program exit if an exception occurred.

Test cases:

As a starting point, the two programs pass all the tests from the provided tester. The ‘crear.c’ accepts the inputs of the mode_t ‘0000’, ‘0777’ and ‘0644’. The ‘combine.c’ program accepts also: ‘./combine’, ‘./combine f1.dat f2.dat test5.txt’ and "./combine f3.dat f4.dat test6.txt". The unique test that is not being passed is the output in test5.txt due the exact order among people with the same mark. However, it was posteriorly to the code developing clarified that the order among them is not reliable. We expect that with this clarifications done, all the test cases related with the correct cases in ‘crear.c’, and tests about the lack of arguments and correct sorting and outputting in both the output argument file and the ‘estadisticas.csv’ in ‘combine.c’ can be assumed as passed and well-done.

Note: The environment of execution is a virtual machine of Ubuntu x64 in VMWare Workstation, in the Bash`s terminal. When the directory has changed permissions the command used was: ‘chmod 555 ssoo_p1_initial_code/’

crear.c				
Description	Planned input	Expected output.	Terminal input/output	Status
No input arguments	./crear	"Program executing with 2 arguments, the program must run with 0 arguments."	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear Error: Invalid number of arguments. Program executing with 0 arguments, the program must run with 2 arguments.	✅
More arguments than necessary	./crear fd 999 third	"Program executing with 2 arguments, the program must run with 3 arguments."	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd 999 third Error: Invalid number of arguments. Program executing with 3 arguments, the program must run with 2 arguments.	✅
‘ABC’ in mode	./crear fd ABC	the argument: ABC, is not an octal number with just 3 digits	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd ABC the argument: ABC, is not an octal number with just 3 digits	✅
‘0999’ in mode	./crear fd 0999	the argument: 999, is not an octal number with just 3 digits	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd 0999 the argument: 0999, is not an octal number with just 3 digits	✅
‘1005’ in mode	./crear fd 1005	the argument: 1005, is not an octal number with just 3 digits	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd 1005 the argument: 1005, is not an octal number with just 3 digits	✅
‘0x777’ in mode	./crear fd 0x777	the argument: 0x777, is not an octal number with just 3 digits	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd 0x777 the argument: 0x777, is not an octal number with just 3 digits	✅
Failure opening/creating	./crear fd 0777 with restricted permissions directory (555)	Error creating file fd	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./crear fd 0777 Error creating file fd	✅
combine.c				

Description	Planned input	Expected output.	Terminal input/output	Status
More arguments than necessary	./combine file1.dat file2.dat file3.bin file4.bin	Error: Invalid number of arguments. Program running with 4 arguments, the program must tun with 3 arguments.	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine file1.dat file2.dat file3.bin file4.bin Error: Invalid number of arguments. Program running with 4 arguments, the program must tun with 3 arguments.	✅
Input file does not exist	./combine file1.dat file2.dat file3.bin (file1 doesn't exist)	Error reading the file file1.dat	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine file1.dat file2.dat file3.bin Error reading the file file1.dat	✅
Maximum number of students reached	./combine file1.dat file2.dat file3.bin(students of file1 and file2 >100)	Maximun number of students reached	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine file1.dat file2.dat file3.bin Maximun number of students reached	✅
Duplicated students	./combine f1.dat f2.dat output3.txt(f1 and f2 has the same students)	There are duplicated students	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine file1.dat file2.dat file3.bin There are duplicated students	✅
student's mark not in [0,10]	./combine f1.dat f2.dat output3.txt(a student has as mark a weird number)	The mark: 134219508 of the student: Maribel Fernandez is not in the range of [0,10]	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine f1.dat f2.dat output3.txt 7. Maribel Fernandez, 134219508, 5 Error: Invalid mark. The mark: 134219508 of the student: Maribel Fernandez is not in the range of [0,10]	✅
Corrupted data in student	(f1.dat has less bits that i should so the data is corrupted)	Warning: Partial struct read. Data might be corrupted.	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine f1.dat f2.dat output3.txt Warning: Partial struct read. Data might be corrupted.	✅
Error writing file	./combine f1.dat f2.dat output3.txt(file3 doesn't have writing permissions)	Error writing the file output3.txt	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ chmod 555 output3.txt hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine f1.dat f2.dat output3.txt Error writing the file output3.txt	✅
Error reading file, i.e. Input files without permissions	./combine file1.dat file2.dat file3.bin (file1 has not permissions to be read)	Error reading the file f1.dat	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ chmod 300 f1.dat hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine f1.dat f2.dat output3.txt Error reading the file f1.dat	✅
Error creating .csv	./combine f1.dat f2.dat output3.txt	Error creating file estadisticas.csv	hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ chmod 000 estadisticas.csv hector@hector-VMware-Virtual-Platform:~/Oper_sys_lab1/ssoo_p1_initial_code$ ./combine f1.dat f2.dat output3.txt Error creating file estadisticas.csv	✅

Conclusions:

Throughout the development of this laboratory practice, we have gained a deeper understanding of system calls and file handling in C. The two programs, ‘crear.c’ and ‘combine.c’, effectively demonstrate how to work. However, not many problems were found during the development of the practice through the use of Linux terminal or C programming language. Simply, the unique one was the understanding of the system calls *per se* and its implementation into code.

Using system calls such as open(), write(), and read() helped reinforce the importance of robust error handling, as the system calls can fail everywhere. Thus, the systematic verification of inputs, permissions, and file integrity ensured that the program behaved as expected under various conditions. The proper use of umask() to manage file permissions in crear.c allowed for precise control over file creation rights. In addition, extra programming knowledge.

Under our consideration, the practice was a great introduction to C language and the use of the Linux terminal, as well as learning how to use Virtual Machines for the proper development of the practice.