UFRJ

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Faculty of Computer Science

Course: Computação Concorrente

Report Regarding The Writers And Readers Problem

Student:
Abraham Banafo Ampah
DRE : 117074396

1st December, 2019.

SUMMARY

1 INTRODUCTION

The readers and writers problem was solved using C programming language and Python. The main part of the code was written in C and the auxiliary part of the problem was written in Python. After running the main program, an auxiliary Python file in generated in addition with other files (a log file and other files that contain the identification of a thread during that process). These files tests if all the conditions for the readers and writers problem have been met successfully.

2 DEVELOPMENT

2.1 Objective

To implement a simplified version of the readers and writers problem, ensuring no thread starvation. Reader and writer threads should be given a unique integer. The shared reading and writing space will be a single integer variable. The reader threads should read the value of this variable and write it to an output file (a unique file for each reader thread). The writing threads should write its unique identifier value in the shared variable.

2.2 Justification

Pthread.h library was used in developing the program to solve the readers and writers problem concurrently with writer priority. The following C libraries were also used (stdio.h, stdlib.h and string.h) in solving the problem.
The pthread.h library gives us the ability to create threads, which was used in creating our solution.
The string.h library was used to aid the creation of our files generated after reading (conversion of integer to a string and concatenating them to the string ".txt").
The stdlib.h library was used to aid memory allocation to thread identifiers in the system.
When the program is compiled, our main function begins to run. In the main function we take the necessary input from the user. We then proceed to create the log file and auxiliary program file which will be in Python, using the C method of creating files. The functions needed in the auxiliary program are also written in the main function, using the C method of writing a text in a file.
With this part concluded, we then proceed to creating our threads for both readers and writers. While creating the threads the functions **void *leitor (void *arg)** and **void *escritor (void *arg)** are called for readers and writers. These functions then also call the functions **void initLeitura(int tid)**, **void fimLeitura(int tid)**, **void initEscrita(int tid)** and **void fimEscrita(int tid)**. Any time, any of these functions are called, the name of the function is written in both the log file and auxiliary program. After this, we then join the threads in order to end the program.

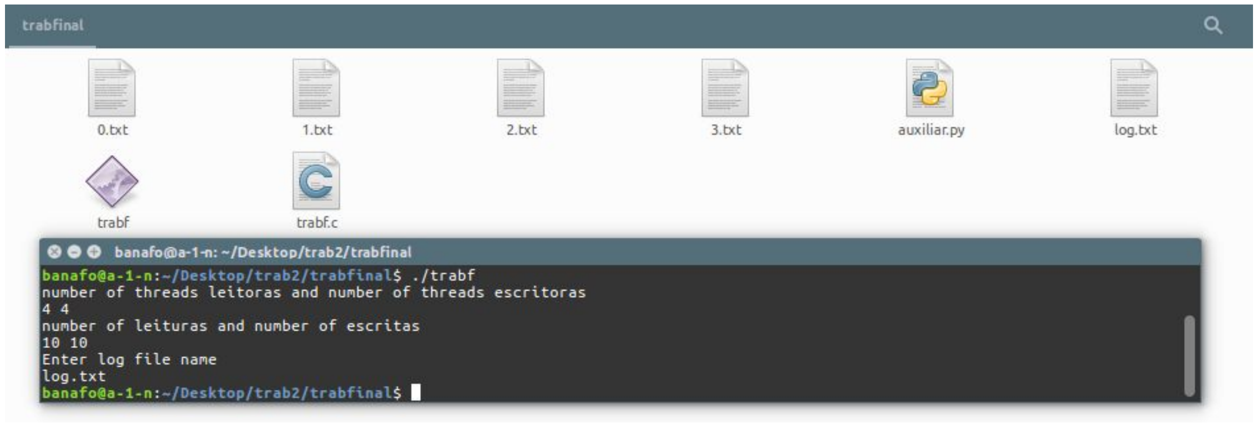
We then have an auxiliary program which we can run to test if really our reader and writer problem conditions were met successfully.

2.3 Functions Created

void initLeitura(int tid);	Initializes reading
void fimLeitura(int tid);	Terminates reading
void initEscrita(int tid);	Initializes Writing
void fimEscrita(int tid);	Terminates Writing
void *leitor (void *arg);	contains void initLeitura(int tid) and void fimLeitura(int tid).
void *escritor (void *arg);	contains void initEscrita(int tid) and void *escritor (void *arg)

3 CONCLUSIONS(TEST CASES)

1st test
Number of reader threads: 4
Number of writer threads: 4
Number of readings: 10
Number of writing: 10
Name of log file: log.txt

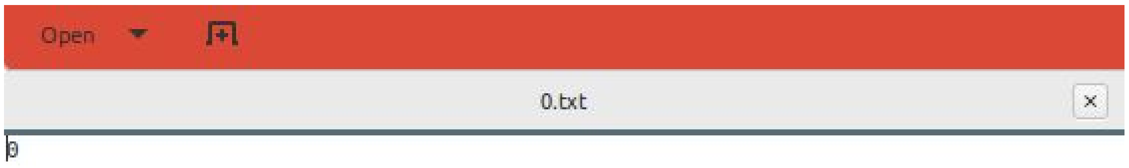


```
banafo@a-1-n:~/Desktop/trab2/trabfinal$ ./trabf
number of threads leitoras and number of threads escritoras
4 4
number of leituras and number of escritas
10 10
Enter log file name
log.txt
banafo@a-1-n:~/Desktop/trab2/trabfinal$
```

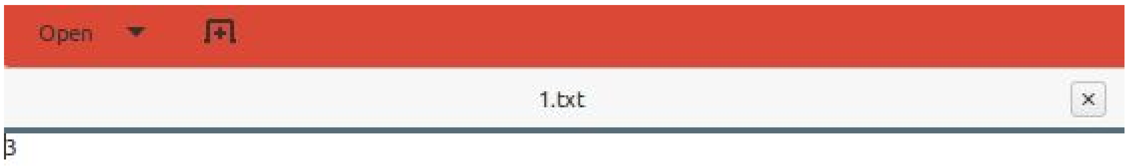
Output
(0.txt, 1.txt, 2.txt. 3.txt)
log.txt
auxiliar.py

File 0.txt contained 0
File 1.txt contained 3

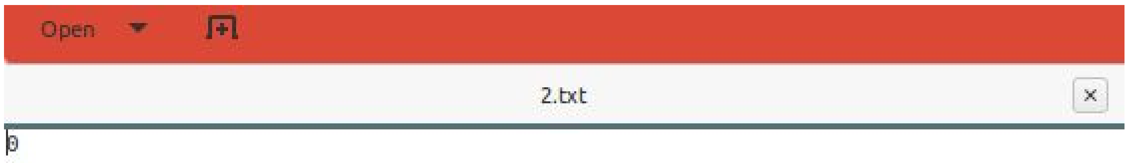
File 2.txt contained 0
File 3.txt contained 0



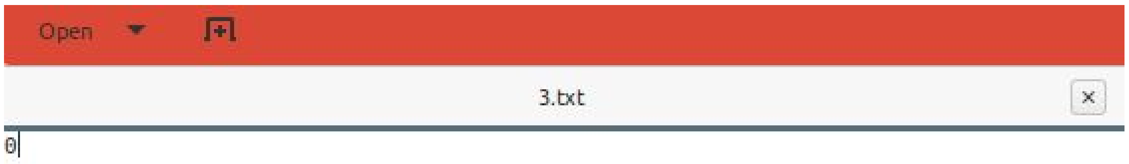
```
0
```



```
3
```

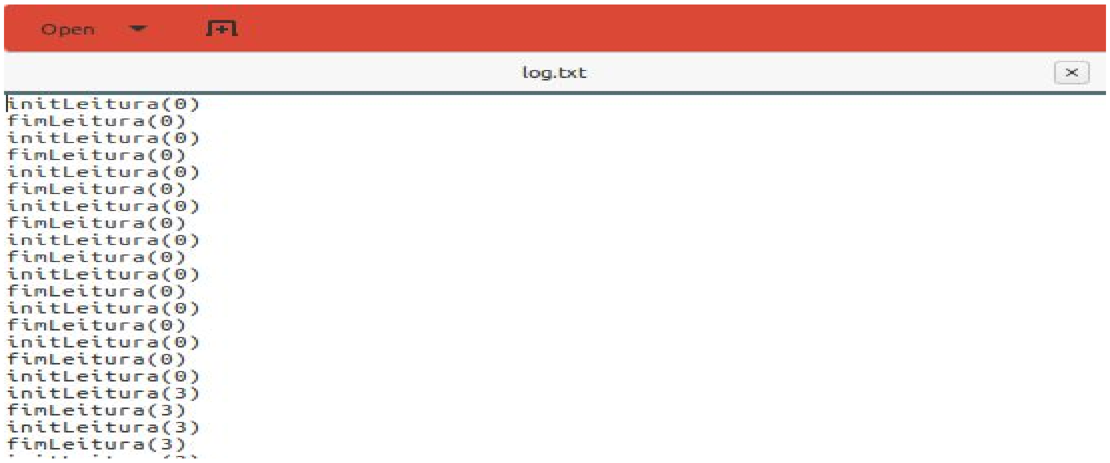


```
0
```



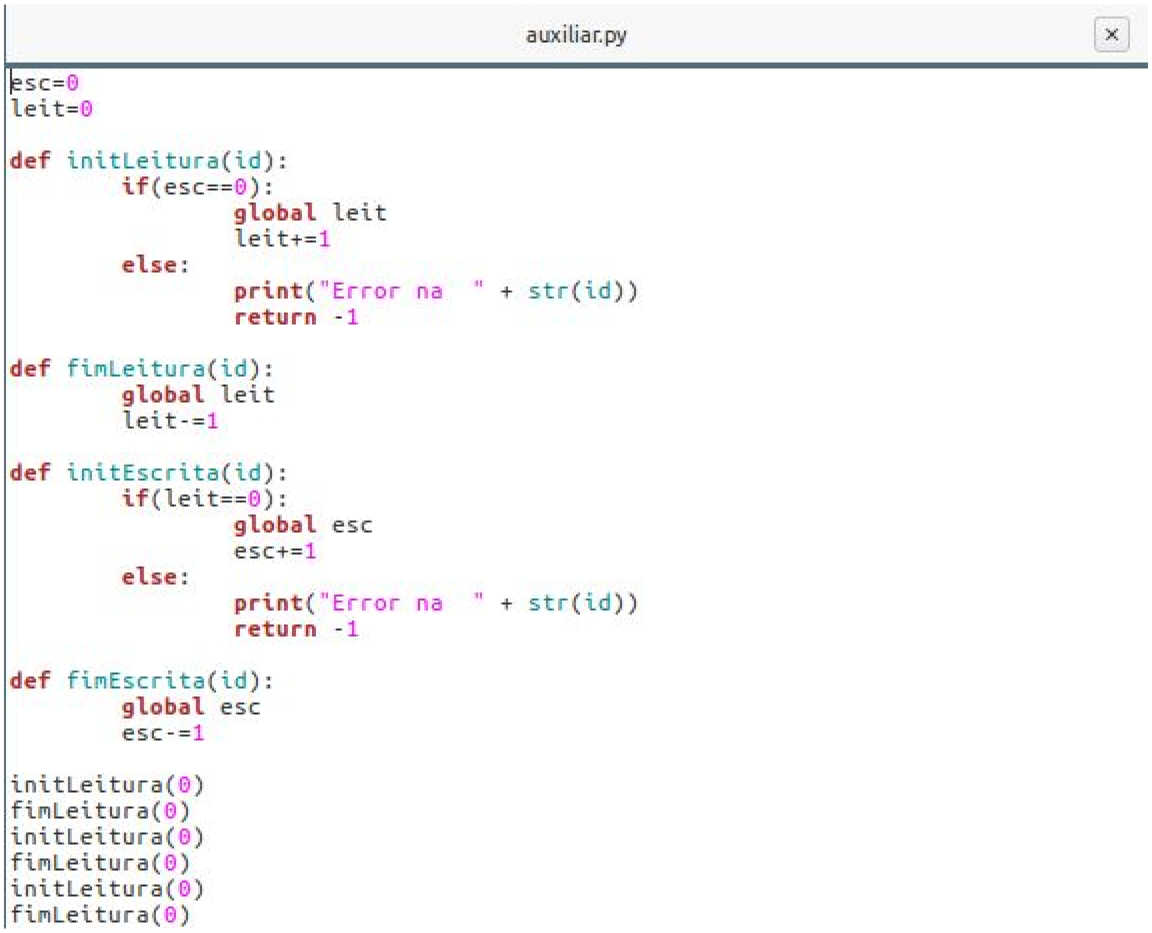
```
0
```

log.txt



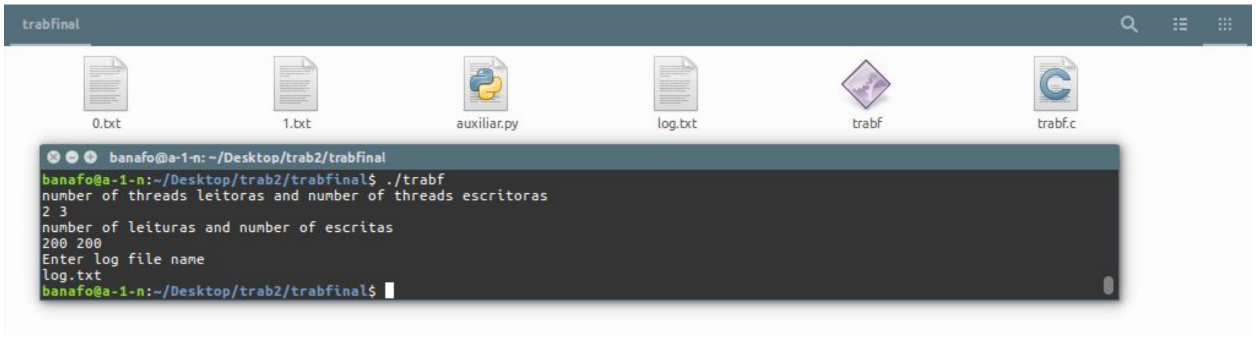
```
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
initLeitura(3)
fimLeitura(3)
initLeitura(3)
fimLeitura(3)
```

auxiliar.py



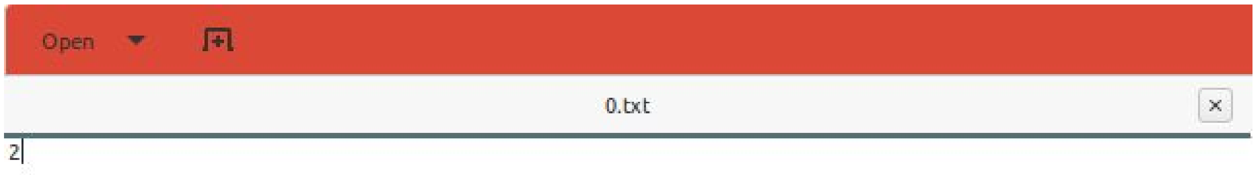
```python
esc=0
leit=0

def initLeitura(id):
        if(esc==0):
                global leit
                leit+=1
        else:
                print("Error na  " + str(id))
                return -1

def fimLeitura(id):
        global leit
        leit-=1

def initEscrita(id):
        if(leit==0):
                global esc
                esc+=1
        else:
                print("Error na  " + str(id))
                return -1

def fimEscrita(id):
        global esc
        esc-=1

initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
initLeitura(0)
fimLeitura(0)
```

2nd Test
Number of reader threads: 2
Number of writer threads: 3
Number of readings: 200
Number of writing: 200
Name of log file: log.txt

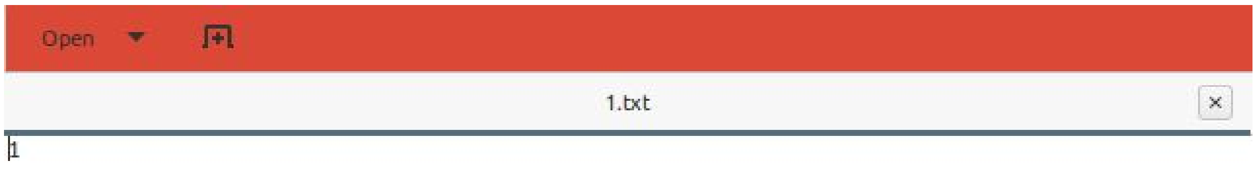


```
banafo@a-1-n:~/Desktop/trab2/trabfinal$ ./trabf
number of threads leitoras and number of threads escritoras
2 3
number of leituras and number of escritas
200 200
Enter log file name
log.txt
banafo@a-1-n:~/Desktop/trab2/trabfinal$ 
```

File 0.txt contained 2
File 1.txt contained 1

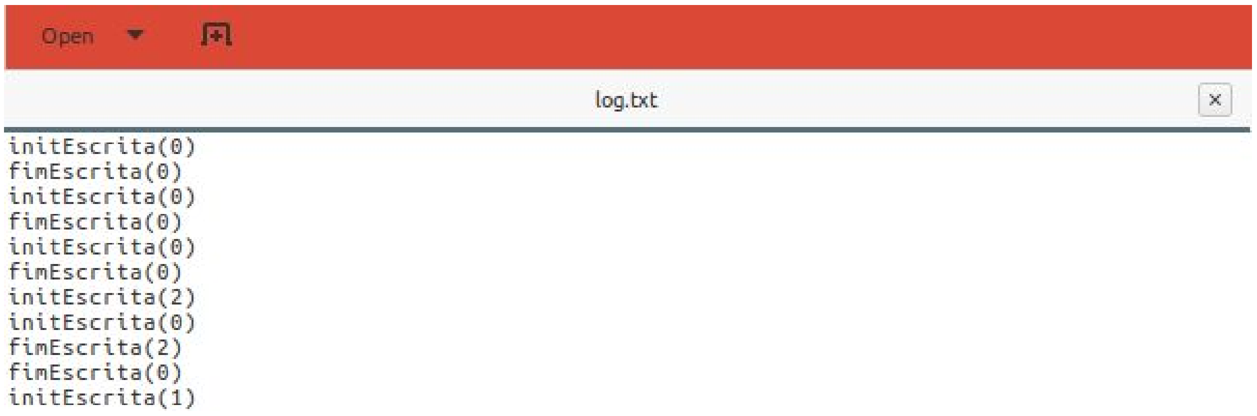


```
2
```



```
1
```

log.txt



```
initEscrita(0)
fimEscrita(0)
initEscrita(0)
fimEscrita(0)
initEscrita(0)
fimEscrita(0)
initEscrita(2)
initEscrita(0)
fimEscrita(2)
fimEscrita(0)
initEscrita(1)
```

4 REFERENCES

1 William Stallings. “Operating System Internals and Design Principles”, 7th edition, 2012.

2 Peter Pacheco. “An Introduction to Parallel Programming”, Morgan Kaufmann, 2011.