MP09

UF03

MULTIPROCESSING PROPOSED EXERCISES

01. The following program written in Python calculates the Fibonacci sequence for n:

```
# fibonacci_1p.py

import sys

def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)

if len(sys.argv) > 1:
    number = sys.argv[1]
    if number.isnumeric():
        print(fibonacci(int(number)))
    else:
        print('the argument is not a number!')
else:
    print('the number is missing as an argument!')
```

with the following execution examples (Ubuntu 24.04 and Python 3.12.3):

```
ricard@HP-ProBook:~$ python3 fibonacci_1p.py
the number is missing as an argument!
ricard@HP-ProBook:~$ python3 fibonacci_1p.py a
the argument is not a number!
ricard@HP-ProBook:~$ python3 fibonacci_1p.py 0
0
ricard@HP-ProBook:~$ python3 fibonacci_1p.py 1
1
ricard@HP-ProBook:~$ python3 fibonacci_1p.py 3
2
ricard@HP-ProBook:~$ python3 fibonacci_1p.py 4
3
ricard@HP-ProBook:~$ python3 fibonacci_1p.py 5
5
ricard@HP-ProBook:~$ /usr/bin/time -f '%E' python3 fibonacci_1p.py 43
433494437
1:25.90
ricard@HP-ProBook:~$ /usr/bin/time -f '%e' python3 fibonacci_1p.py 43
433494437
85.34
ricard@HP-ProBook:~$
```

Create a Python program (fibonacci_2p) that modify the previous source code to calculate the Fibonacci sequence through multiprogramming (two processes, main and one child that them share the work). Then create a table of programs execution times (fibonacci_1p.py and fibonacci_2p, for values from 25 to 50, using a Shell Script or Python script) and also a bar graph (using Matplotlib) to compare the programs execution times.

02. Create a Python program (fibonacci_3p.py) that modify the source code from the previous exercise to calculate the Fibonacci sequence through multiprogramming (three processes, main and two child that they share the work). Then create a table of programs execution times (fibonacci_1p.py, fibonacci_2p.py and fibonacci_3p.py, for values from 25 to 50) and also a bar graph to compare the programs execution times.

03. The following program written in Python sorts the numbers contained in a text file using bubble sort:

```
# bubble_sort_1p.py

import os
import sys

def bubble_sort(l_unsorted):
    l_sorted = l_unsorted[:]
    n_sorted = len(l_sorted)

    for i in range(n_sorted-1):
        for j in range(0, n_sorted-i-1):
            if l_sorted[j] > l_sorted[j+1]:
                l_sorted[j], l_sorted[j+1] = l_sorted[j+1], l_sorted[j]

    return l_sorted

if len(sys.argv) > 1:
    file_name = sys.argv[1]

    if os.path.exists(file_name):
        unsorted_list = []
        with open(file_name) as file:
            for line in file: unsorted_list.append(int(line))

        print(bubble_sort(unsorted_list))
    else:
        print(f"file with name '{file_name}' does not exist!")
else:
    print('the file name is missing as an argument!')
```

with the following execution examples:

```
ricard@HP-ProBook:~$ seq 1 20 | shuf > numbers
ricard@HP-ProBook:~$ cat numbers | tr '\n' ' ' && echo
3 20 9 12 15 2 16 6 4 5 19 8 18 17 7 11 13 14 1 10
ricard@HP-ProBook:~$ python3 bubble_sort_1p.py numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
ricard@HP-ProBook:~$
```

```
ricard@HP-ProBook:~$ seq 1 10000 | shuf > numbers
ricard@HP-ProBook:~$ /usr/bin/time -f '%E' python3 bubble_sort_1p.py numbers > /dev/null
0:05.51
ricard@HP-ProBook:~$
```

Create a Python program (bubble_sort_3p.py) that modify the previous source code to sort a text file through multiprogramming (three processes, main and two child, where the children do all the sorting work). Then create a table of programs execution times (bubble_sort_1p.py and bubble_sort_3p.py, for text files with 1000, 2000, 3000, …, 20000 numbers) and also a bar graph to compare the programs execution times.

04. Create a Python program (bubble_sort_pool.py) that modify the source code from the previous exercise to sort a text file through multiprogramming with a pool. Then create a table of programs execution times (bubble_sort_1p.py, bubble_sort_3p.py and bubble_sort_pool.py) for text files with 1000, 2000, 3000, …, 20000 numbers and also a bar graph to compare the programs execution times.

05. The following program written in Python sometimes does not return a correct result.

```
# manage_bank_account.py

import multiprocessing as mp
import time

def deposit(b, t):
    for u in range(t):
        time.sleep(0.01)
        b.value += 5

def withdraw(b, t):
    for w in range(t):
        time.sleep(0.01)
        b.value -= 5

if __name__ == '__main__':
    balance = mp.Value('i', 500)
    process_deposit = mp.Process(target=deposit, args=(balance, 100))
    process_withdraw = mp.Process(target=withdraw, args=(balance, 100))
    process_deposit.start()
    process_withdraw.start()
    process_deposit.join()
    process_withdraw.join()

    print(balance.value)
```

```
ricard@HP-ProBook:~$ python3 manage_bank_account.py
475
ricard@HP-ProBook:~$ python3 manage_bank_account.py
510
ricard@HP-ProBook:~$ python3 manage_bank_account.py
405
ricard@HP-ProBook:~$
```

Modify it with some multiprogramming synchronization mechanism so that it returns always the correct result.

06. Create a Python program (dont_guess_the_number.py) that simulates the game “Don't guess the number” through multiprogramming.

The game works as follows:

- The participants of the game are a referee and some players (between two and six).
- All participants will throw a dice, first the referee and then all the players at the same time.
- Once all the dice have been thrown, the number indicated by the referee's dice will be compared with the number indicated by each player's dice. Players with the same number as the referee will be eliminated.
- If there are players (not eliminated), the above process will be repeated until there is one player or none.
- If there is one player at the end of the game, they will be the winner.

Each player will be a child process that:

- When it is ready to start, it will indicate it with a message on the screen.
- While it is playing: it will roll the dice and then communicate the result to the main process which will be in charge of displaying the corresponding message.
- Depending on the result, it will wait for his turn to roll the dice again or it will be eliminated by the main process.

The main process will take the role of the referee and each message that appears on the screen will be preceded by the time that has passed since the game started.

Examples of preliminary execution:

```
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py
  0.000000 Game is starting ...
  0.000052 The number of players must be specified!
  0.000071 Game has been interrupted!
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py a
  0.000001 Game is starting ...
  0.000017 Incorrect number of players!
  0.000040 Game has been interrupted!
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 1
  0.000001 Game is starting ...
  0.000038 The number of players must be a minimum of two and a maximum of six
  0.000043 Game has been interrupted!
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 7
  0.000000 Game is starting ...
  0.000036 The number of players must be a minimum of two and a maximum of six
  0.000041 Game has been interrupted!
ricard@HP-ProBook:~$
```

Execution example with two players:

```
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 2
  0.000000 Game is starting ...
  0.009288 P1 is ready ...
  0.009712 P2 is ready ...
  1.009606 Refere throws a dice and reports that it is 6
  3.010252 P2 throws a dice and reports that it is 4, continues
  3.010344 P1 throws a dice and reports that it is 1, continues
  4.010873 Refere throws a dice and reports that it is 1
  6.011155 P1 throws a dice and reports that it is 2, continues
  6.011209 P2 throws a dice and reports that it is 1, eliminated
  7.011610 Referee determines that the P1 wins
  7.011714 Game over
ricard@HP-ProBook:~$
```

Execution example with three players:

```
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 3
  0.000000 Game is starting ...
  0.008898 P1 is ready ...
  0.009190 P2 is ready ...
  0.009526 P3 is ready ...
  1.009404 Refere throws a dice and reports that it is 2
  3.010307 P1 throws a dice and reports that it is 2, eliminated
  3.010471 P3 throws a dice and reports that it is 3, continues
  3.010508 P2 throws a dice and reports that it is 2, eliminated
  4.011148 Referee determines that the P3 wins
  4.011211 Game over
ricard@HP-ProBook:~$
```

Another execution example with three players:

```
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 3
  0.000000 Game is starting ...
  0.008540 P1 is ready ...
  0.008871 P2 is ready ...
  0.009185 P3 is ready ...
  1.009128 Refere throws a dice and reports that it is 3
  3.009962 P1 throws a dice and reports that it is 2, continues
  3.010058 P2 throws a dice and reports that it is 3, eliminated
  3.010186 P3 throws a dice and reports that it is 4, continues
  4.010422 Refere throws a dice and reports that it is 6
  6.010759 P1 throws a dice and reports that it is 1, continues
  6.010846 P3 throws a dice and reports that it is 6, eliminated
  7.011317 Referee determines that the P1 wins
  7.011406 Game over
ricard@HP-ProBook:~$
```

Execution example with four players:

```
ricard@HP-ProBook:~$ python3 dont_guess_the_number.py 4
  0.000001 Game is starting ...
  0.008780 P1 is ready ...
  0.009124 P2 is ready ...
  0.009480 P3 is ready ...
  0.009827 P4 is ready ...
  1.009812 Refere throws a dice and reports that it is 4
  3.010553 P1 throws a dice and reports that it is 6, continues
  3.010602 P2 throws a dice and reports that it is 3, continues
  3.010641 P4 throws a dice and reports that it is 4, eliminated
  3.010730 P3 throws a dice and reports that it is 6, continues
  4.011052 Refere throws a dice and reports that it is 4
  6.011390 P3 throws a dice and reports that it is 2, continues
  6.011465 P1 throws a dice and reports that it is 2, continues
  6.011485 P2 throws a dice and reports that it is 3, continues
  7.011722 Refere throws a dice and reports that it is 4
  9.012374 P3 throws a dice and reports that it is 2, continues
  9.012433 P2 throws a dice and reports that it is 6, continues
  9.012453 P1 throws a dice and reports that it is 2, continues
 10.012749 Refere throws a dice and reports that it is 4
 12.013281 P3 throws a dice and reports that it is 3, continues
 12.013341 P1 throws a dice and reports that it is 4, eliminated
 12.013482 P2 throws a dice and reports that it is 3, continues
 13.013782 Refere throws a dice and reports that it is 5
 15.014235 P3 throws a dice and reports that it is 2, continues
 15.014291 P2 throws a dice and reports that it is 2, continues
 16.014396 Refere throws a dice and reports that it is 1
 18.014703 P3 throws a dice and reports that it is 1, eliminated
 18.014763 P2 throws a dice and reports that it is 2, continues
 19.014969 Referee determines that the P2 wins
 19.015022 Game over
ricard@HP-ProBook:~$
```

7. Create a program that solves a real-life problem with multiprogramming and indicate the improvement achieved by the program compared to its version without multiprogramming.