THEORY OF INFORMATION, ARCHITECTURE OF COMPUTERS AND OPERATING SYSTEMS (TIACOS)
Bioinformatics, ESCI
DAC, UPC

IOLab **Spring 2023**

Objectives

The purpose of this assignment is to understand how I / O system calls work. They deal with bytes and you need to program how to interpret them (as integers, strings, etc.). In order to do that, we will use input files in FASTQ format, a common file format in bioinformatics. You will read and interpret them and eventually syncronize two processes to read and compute quality errors concurrently.

Skills

- Properly use of the system calls `open`, `read`, `write`, `close` and `pipe`.
- Be able to program byte level manipulations in Python.
- Debug Python programs.

Handout Instructions

> This wording has a companion file, `iolab-files.tar`, with two Python scripts `ioLab.py` and `seqFastqParser.py` that you must edit and deliver. Besides, there are some FASTQ files that you can use as input for your tests.

Start by copying `iolab-files.tar` to a (protected) directory on a Linux machine in which you plan to do your work. Then give the command

```
$ tar xvf iolab-files.tar
```

This will cause a number of files to be unpacked in the directory. The files you will be modifying and turning in are `ioLab.py`, and `parFastqParser.py`.

The `ioLab.py` file contains a skeleton for each of the 8 Python functions you have to program. The first part of the assignment is to complete each function. This file also contains a function example, named `showSequence(filename)` that calls the functions you have to implement in order to read the header of a fastq file and print it on screen.

The complete list is:

Name	Description	Rating
`atoi(str)`	Converts the bytestring str to integer	2
`readLine(fd)`	Returns a line, in a bystestring, read from the file descriptor fd.	2
`writeLine(line)`	Prints line to standard output.	1
`readLenght(line)`	Returns the lenght of a sequence.	2
`showHeader(filename)`	Prints name and length of a Fastq file.	1
`showSeqQlty(filename)`	Prints pair base→quality.	2
`worstQlty(seqLine, qltyLine)`	Returns the worst pairs base→quality.	3
`showWorstQlty(filename)`	Prints the worst pair base→quality.	1

Table 1: ioLab functions.

FASTQ format

FASTQ[1] is a text-based format for storing both nucleotide sequence and its corresponding quality score. It has 4 lines per sequence:

- @ the unique sequence name.
- The nucleotide sequence (A, C, G, T, N) on 1 line.
- + The quality line break. Sometimes with the sequence name again.
- The quality scores (ASCII characters) on 1 line.

The quality line is always the same line length as the sequence line.

Phred Quality Scores (Q-Score) in FASTQ files represent the probability of error of each base in the sequence field. A Q-Score of 40 represents a probability of error of 0.00010, while a Q-Score of 0 represents a probability of error of 1.00000. In FASTQ, each Q-Score is represented as an ASCII character, in which the ASCII code to represent a Q-Score q is calculated as: `chr(q + 33)`.

Therefore, a Q-Score of 0 would be represented by the ASCII character in position 33 (‘!’), while a Q-Score of 80, would be represented by the character in position 113 (‘q’).

Example of FASTQ file contents:

```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACC
+HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
efcfffffcfeefffcffffffddf`feed]`]_Ba_^__
```

The first nucleotide, `T` has a quality equal to 68, because ’e’$= 101$ and $101 - 33 = 68$.

[1] See https://en.wikipedia.org/wiki/FASTQ_format.

Quality here means the probability of error. A Phred quality score is a measure of the quality of the identification of the nucleotide generated by automated DNA sequencing. The quality score is associated to a probability of error; the probability of an incorrect base call.

$$P = 10^{(-Q/10)}$$

See these slides for further reading.

Fastq parser

You have to program two Python scripts that deal with a Fastq file. The first one, `seqFastqParser.py`, is just to check that `ioLab.py` works properly. The second one, `parFastqParser.py` uses also `ioLab.py` but needs much more programming work.

The `seqFastqParser.py`, 14 points.

Read this file carefully and understand what it does. Once you have finished the programming of `ioLab.py`, execute `seqFastqParser.py` and it should work properly. You do not need to modify this file.

Example of execution

```
$ ./seqFastqParser.py SRR000049.fastq
File:  SRR000049.fastq Fastq header
Name: @SRR000049.1EM6706F01DGZLJ-Number of bases:  309
File:  SRR000049.fastq First sequence
TCAGTAAGAAAATTCAAGCTTTCTGATTCTATCCTTGTGTTACTGTTGGAACCGAAAGGGTTTGAATTCAAACCC
TTTCGGTTCCAACGATTTTACTTTTCATGCTTTTTAAAGCATAAAATGTTTCTAATGGTATATTTTGAAATGTTTTT
AAAGTACAGAAAAAAATACAATGGACACTCATGTACCTGTTACCCAGATTTAACAATTTAAATAAATGTTCATATT
TTCATTAAGAAATAAAATATAACAAATAGTCCTTAATATGTTGGATATAAATATACATACTTACTGAGCGGGCTGG
CAAGG
File:  SRR000049.fastq First sequence and qualities
T --> 27 C --> 29 A --> 29 G --> 29 T --> 11 A --> 37 A --> 28 G --> 20 A --> 44
A --> 34 A --> 17 A --> 2 T --> 25 T --> 10 C --> 29 A --> 37 A --> 26 G --> 28
C --> 29 T --> 42 T --> 32 T --> 11 C --> 26 T --> 15 G --> 29 A --> 24 T --> 39
T --> 29 C --> 20 T --> 24 A --> 23 T --> 25 C --> 35 C --> 23 T --> 28 T --> 14
G --> 28 T --> 25 G --> 26 T --> 39 T --> 28 A --> 23 C --> 29 T --> 11 G --> 25
T --> 23 T --> 7 G --> 38 G --> 26 A --> 34 A --> 22 C --> 35 C --> 23 G --> 28
A --> 42 A --> 32 A --> 11 G --> 40 G --> 30 G --> 8 T --> 43 T --> 34 T --> 14
G --> 15 A --> 37 A --> 26 T --> 39 T --> 28 C --> 25 A --> 43 A --> 34 A --> 14
C --> 45 C --> 35 C --> 17 T --> 43 T --> 34 T --> 14 C --> 29 G --> 33 G --> 20
T --> 32 T --> 20 C --> 38 C --> 27 A --> 39 A --> 29 C --> 27 G --> 27 A --> 26
T --> 47 T --> 36 T --> 22 T --> 8 A --> 28 C --> 29 T --> 47 T --> 36 T --> 21
```

```
T --> 7 C --> 19 A --> 29 T --> 25 G --> 29 C --> 28 T --> 50 T --> 36 T --> 24
T --> 15 T --> 6 A --> 44 A --> 34 A --> 17 G --> 27 C --> 22 A --> 29 T --> 23
A --> 44 A --> 34 A --> 18 A --> 2 T --> 26 G --> 28 T --> 34 T --> 27 T --> 3
C --> 23 T --> 24 A --> 37 A --> 29 T --> 27 G --> 36 G --> 27 T --> 28 A --> 28
T --> 28 A --> 28 T --> 41 T --> 34 T --> 20 T --> 5 G --> 28 A --> 41 A --> 34
A --> 17 T --> 28 G --> 28 T --> 44 T --> 36 T --> 24 T --> 15 T --> 5 A --> 40
A --> 34 A --> 15 G --> 28 T --> 28 A --> 21 C --> 28 A --> 28 G --> 27 A --> 45
A --> 34 A --> 24 A --> 17 A --> 12 A --> 6 A --> 1 T --> 22 A --> 27 C --> 28
A --> 31 A --> 21 T --> 27 G --> 35 G --> 26 A --> 26 C --> 27 A --> 24 C --> 28
T --> 28 C --> 28 A --> 24 T --> 28 G --> 27 T --> 20 A --> 28 C --> 36 C --> 27
T --> 28 G --> 27 T --> 30 T --> 19 A --> 28 C --> 41 C --> 34 C --> 16 A --> 25
G --> 28 A --> 26 T --> 41 T --> 34 T --> 16 A --> 36 A --> 28 C --> 28 A --> 37
A --> 28 T --> 39 T --> 33 T --> 13 A --> 39 A --> 33 A --> 13 T --> 18 A --> 41
A --> 34 A --> 16 T --> 27 G --> 27 T --> 36 T --> 28 C --> 18 A --> 22 T --> 28
A --> 26 T --> 41 T --> 34 T --> 19 T --> 5 C --> 27 A --> 21 T --> 37 T --> 29
A --> 36 A --> 28 G --> 24 A --> 40 A --> 34 A --> 17 T --> 27 A --> 40 A --> 33
A --> 18 A --> 3 T --> 26 A --> 22 T --> 27 A --> 33 A --> 24 C --> 28 A --> 38
A --> 31 A --> 10 T --> 24 A --> 26 G --> 28 T --> 22 C --> 33 C --> 23 T --> 32
T --> 22 A --> 35 A --> 26 T --> 17 A --> 28 T --> 27 G --> 28 T --> 36 T --> 28
G --> 36 G --> 28 A --> 28 T --> 21 A --> 28 T --> 28 A --> 41 A --> 34 A --> 17
T --> 17 A --> 28 T --> 27 A --> 21 C --> 22 A --> 28 T --> 26 A --> 28 C --> 28
T --> 33 T --> 23 A --> 28 C --> 24 T --> 28 G --> 27 A --> 21 G --> 26 C --> 24
G --> 39 G --> 33 G --> 13 C --> 27 T --> 27 G --> 36 G --> 27 C --> 27 A --> 36
A --> 27 G --> 34 G --> 24
File:  SRR000049.fastq The worst quality of the first sequence
The worst:  A -> 1
```

The `parFastqParser.py`, 6 points

The program `parFastqParser.py` computes and prints the worst quality of all the scores. In order to do that you will have two processes and two pipes. One pipe, named `p2ch` (from parent to child) where the parent writes and the child reads. The other one, named `ch2p` (from child to parent), where the child writes and the parent reads. The parent process executes a loop. Each iteration, it reads the length of a line, a line of sequences and and a line of qualities from the fastq file (passed as an argument 1 to the program). Parent sends this information to its child, via pipe `p2ch`, and waits for its child returns the worst pair (base, quality),via `ch2p`. When there are no more lines to read, the parent process print the worst pair to the standard output, kills its child, clean up and ends.

The child process executes an infinite loop. Its waits reading from the pipe `p2ch`. When the parent sends the appropriate information, the child finds out the worst quality of the line and returns to its parent a pair (base, quality) via pipe `ch2p` and writes the pair to an ASCII text file called `worst.txt`. This file is created by the child process at the beginning with permissions: `rw-rw-r--`.

Figure 1 shows a sequence diagram of the program parFastq. Each vertical rectangle means one or more system calls of one kind. It's up to you to decide, for instance, how many calls to `read()` the parent process needs when it is accessing to the fastq file in order to get all the information it needs to send to its child.

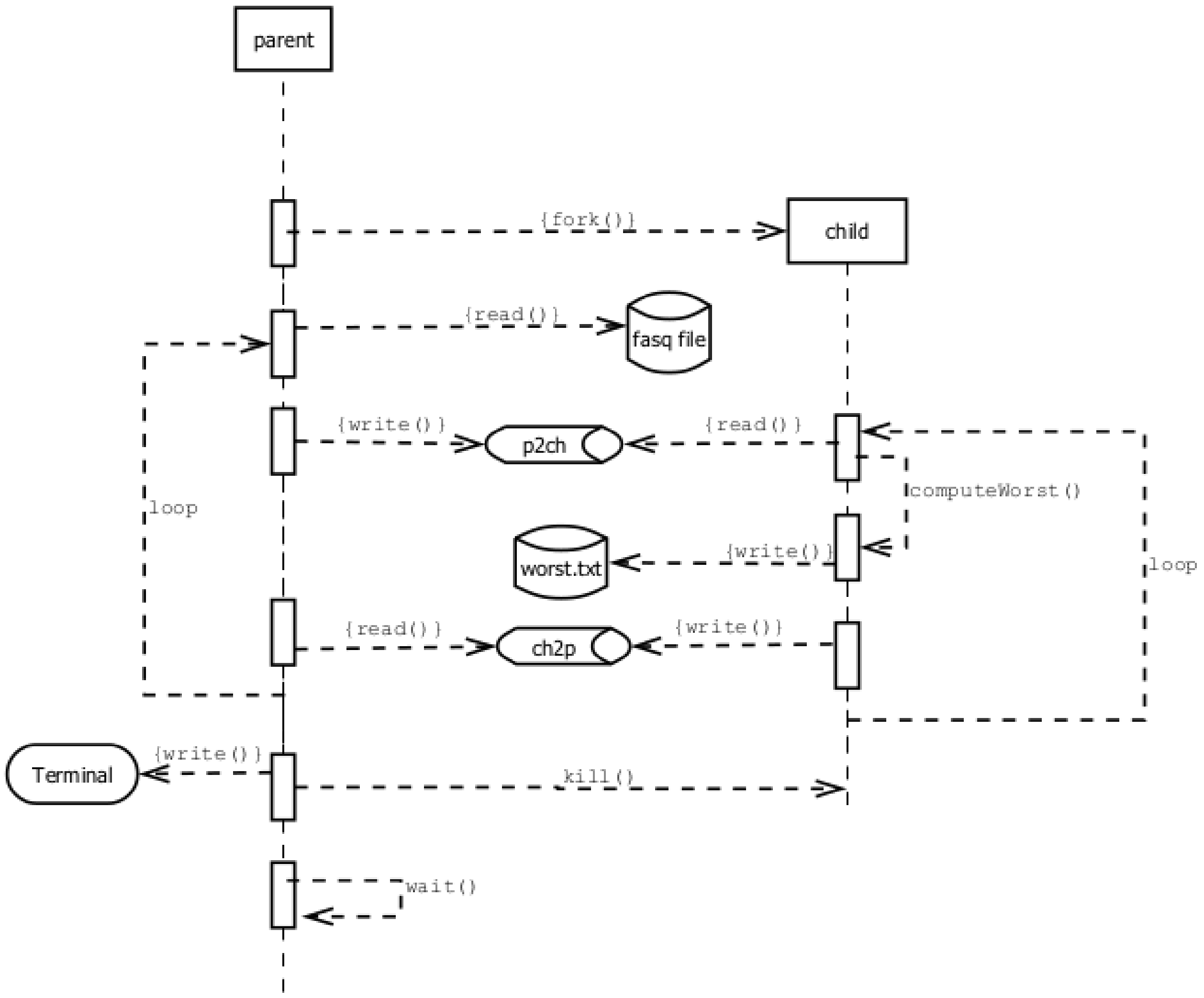

Figure 1: Sequence diagram of parFastqParser.

Note that sometimes you are dealing with strings, sometimes with bytes and sometimes with integers. You can use Python functions to interpret data accordently. For instance, to create an integer from a byte string you can use `int.from_bytes()` and viceversa `int.to_bytes()`. Note that the byte order is important here. See slides 9-12 of Unit *Computer Architecture*.

For example, how to pass the integer 49 to a byte string of two bytes. And from integer 49 to bytes; using little endian byte ordering:

```
>>> int.to_bytes(49,2,'little')
b'1\x00'
```

```
>>> int.from_bytes(b'1\x00','little')
49
```

For instance, how to pass from a string '23' to a byte string b'23':

```
>>> bytes(str(23),"utf-8")
b'23'
```

Recall good practices of programming. Each system call should be protected within a `try: except:`. Two typical errors in this program will be `OSError` and `TypeError`.

And last but not least, as this is concurrent programming, different executions could produce different results. Imagine two sceneários: (a) the parent process closes its pipe for writting (`p2ch`) and kills its child before the child process tries to read from that pipe. (b) the parent process closes its pipe for writting (`p2ch`) and its child process reads from the pipe before its parent kill it. In scenario (a) nobody reads from the pipe `p2ch`, in scenario (b), the child gets 0 bytes from the pipe. Do your programm finishes properly in both cases?

Evaluation

Your score will be computed out of a maximum of 32 points based on the following distribution:

20 Correctness points.

10 Performance points.

2 Style points.

Correctness points. The functions you must program have been given a difficulty rating between 1 and 3, such that their weighted sum totals to 14. You will get full credit for a function if it works, and no credit otherwise. Besides, program `parFastqParser.py` has a weighted of 6 points.

Performance points. Our main concern at this point in the course is that you can get the right answer. However, we want to instill in you a sense of keeping things as short and simple as you can. You must only use system calls, but try to minimize the number of calls.

Style points. Finally, we've reserved 2 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Deliverable

In this lab, you have to upload a tar file named `iolab.tar` via Moodle. To create it, type:

```
$ tar -cvf iolab.tar parFastqParser.py ioLab.py
```

References

[Python 3.9] *The Python Language Reference*
https://docs.python.org/3/reference/index.html