L:EIC / SO2223
Trabalho Prático

Q1. Write a program `samples` that given one text file and two integers, `n` and `m`, on the command line, prints a sequence of `n` text fragments, each with `m` characters from the original file. Fragments must be obtained in `n` different positions in the file at random and be written sequentially in `stdout` without any separator or line change, and between `>` and `<` characters.

```
$ samples
usage: samples file numberfrags maxfragsize
$ cat quote.txt
Look again at that dot. That's here. That's home. That's us. On it everyone
you love, everyone you know, everyone you ever heard of, every human
being who ever was, lived out their lives. The aggregate of our joy and
suffering, thousands of confident religions, ideologies, and economic
doctrines, every hunter and forager, every hero and coward, every creator
and destroyer of civilization, every king and peasant, every young couple
in love, every mother and father, hopeful child, inventor and explorer,
every teacher of morals, every corrupt politician, every "superstar," every
"supreme leader," every saint and sinner in the history of our species
lived there - on a mote of dust suspended in a sunbeam.
$ samples quote.txt 5 7
>ry king<
>yone yo<
>unbeam.<
>king an<
>y teach<
```

Tip: use the functions `fseek()`, `srandom()` and `random()` from `libc`. Use the value `0` as the seed for generating the random sequence with `random()`.

Q2. Write a program `txt2epub` that given a list of `n` text files - `f1.txt`, `f2.txt`,..., `fn.txt` - on the command line, applies the `pandoc` command to each of the files generating EPUB versions:

```
pandoc f1.txt -o f1.epub
...
pandoc fn.txt -o fn.epub
```

The conversion from text to EPUB must be done in parallel by `n` processes created for this purpose. Each process gets a name from `argv[]` and converts it as described. After the processes have finished converting all the files, the last step of the program `txt2epub` will be the generation of a `.zip` file with the `n` files in EPUB format.

```
zip ebooks.zip f1.epub ... fn.epub
```

For example:

```
$ txt2epub iliad.txt odyssey.txt aeneid.txt ... metamorphoses.txt
[pid2751] converting iliad.txt ...
[pid2749] converting metamorphoses.txt ...
[pid2752] converting odyssey.txt ...
...
$ ls
ebooks.zip
aeneid.epub
iliad.epub
metamorphoses.epub
odyssey.epub
```

Tip: get a set of books in text format from the Internet. Use only legal copies of books. See here for example: Project Gutenberg.

Q3. Write a program `tokenring` that gets 3 integers - `n`, `p` and `t` - from the command line. When executed, it creates `n` processes connected to each other by "named pipes". The named pipes are designated `pipe1to2`, `pipe2to3`,..., `pipento1` and each allows one-way communication between the `i`-th and the `i+1`-th processes. The last named pipe closes the ring allowing process `n` to communicate with process `1`. Once this process ring is created, `p1` should send a token (a message with an integer with an initial value 0) to the next process (`p1 > p2`) and so on (`p2 > p3 >... > pn > p1 >...`). The token must circulate between processes endlessly, increasing its value at each hop. Each time a process receives the token, it must immediately resend it to the next process or, with a probability of `p`, block its submission during `t` seconds and print a message alerting to this fact (see the following example). In either case, the value of the token must be incremented.

```
$ tokenring 5 0.01 10
[p2] lock on token (val = 2867)
[p2] unlock token
[p5] lock on token (val = 9213)
[p5] unlock token
...
```

Tip: use the function `mkfifo()` from the kernel API to create the named pipes.

General considerations. To ensure that the results obtained can be reproduced you should perform a final test for all programs on the machine `gnomo.fe.up.pt`.

When the assignment is finished, each group must send by e-mail, to the teacher of the respective PL class, a `.zip` file whose name must contain the group number and the number of the PL class to which the group members belong, e.g., `G2PL6.zip`. The file must contain a directory with the same name - `G2PL6` - within which there will be three sub-directories - `Q1`, `Q2` and `Q3` - each containing the C source code for the respective problem as well as a `makefile` with rules to build the program and to clean temporary and binary files. The folder `G2PL6` should also include a text file with the full names and Sigarra IDs of the group members.

In addition to sending the source code for the three questions, as described above, each group must make a presentation of the working programs for the teacher of the respective PL class on a date to be established. In this event, the presence of all group members is mandatory.

Enjoy your work!

The "Sistemas Operativos" Team