

RELATÓRIO DO SEGUNDO TRABALHO PRÁTICO PARA DISCIPLINA DE SISTEMAS OPERATIVOS

FUMADORES

30/12/2019

Vinícius Benite Ribeiro (82773)

Professor: José Nuno Panelas Nunes Lau

**Introdução**

O presente trabalho visa implementar na linguagem C um programa que resolve o problema dos fumadores, proposto em 1971 porSuhas Patil.

Existem 3 processos independentes: fumador, agente e o watcher. A sincronização entre eles é feita através de semáforos e memória partilhada.

**Smoker**

Suponha que um cigarro exija três ingredientes para fumar: tabaco, papel e fósforos. Há três fumantes ao redor da mesa, cada um com um suprimento infinito de um dos três ingredientes - um fumante tem um suprimento infinito de tabaco, outro tem papel e o terceiro tem fósforos.

**waitForIngredients()**

**rollingCigarette()**

**smoke()**

**Agent**

Um agente permite que os fumantes façam seus cigarros selecionando arbitrariamente dois dos suprimentos para colocar na mesa. O fumante que tem o terceiro suprimento deve remover os dois itens da mesa, usando-os (junto com o próprio suprimento) para fazer um cigarro, que eles fumam por um tempo. Quando o fumante termina o cigarro, o agente coloca dois novos itens aleatórios na mesa. Esse processo continua para sempre. O agente tem um processo que espera no semáforo de cada fumante para que ele saiba que pode colocar os novos itens na mesa.

**prepareIngredients()**

**waitForCigarette()**

**closeFactory()**

**Watcher**

O watcher é responsável por verificar se após a emissão de um novo ingrediente (um dos elementos do pacote produzido pelo agente) há algum fumador que possa fumar. Existe um watcher por cada tipo de ingrediente.

**waitForIngredient()**

**updateReservations()**

**informSmoker()**

O programa tem funções que geram diferentes dados a partir do comando *last.* As funções ***get\_sessions\_\*\*\*()*** são chamadas de acordo com o argumento passado pelo terminal:

* Default e -f -> get\_sessions\_info()
* -u -> get\_sessios\_by\_user(): filtra o output com os usuários procurados pelo utilizador
* -s -e -> get\_sessions\_by\_date() : filtra o output com o período selecionado.

Os dados são filtrados, para facilitar o posterior tratamento, com as opções do comando last e com os comandos *grep* e *sed*. O retorno dessas funções é feito para ficheiros externos. PS: não importa o número de vezes que correr o programa. Os ficheiros serão sempre iguais, a não ser que haja novas sessões.

**Tratar os dados**

A função ***main()*** lê os ficheiros gerados anteriormente, linha por linha, e cria um array associativo *map{user: tempo\_de\_sessão}*. Além disso, essa função checa as opções e devidas restrições, e atribui flags utilizadas para o print, dependendo dos argumentos passados.

A função ***logic()*** é chamada para fazer os cálculos necessários, como passar os dias para minutos, por exemplo. Para isso faz operações sobre arrays auxiliares, guardando o resultado no array associativo *final{ }*.

A função **max\_min()** calcula os tempos máximos e tempos mínimos de cada user e o número total de sessões, guardando o resultado no array associativo *aux\_array{ }*.

A função ***search\_user()*** procura um user a partir de uma regex passado como argumento ao terminal.

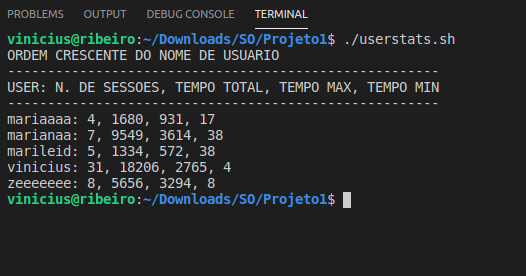
**Output**

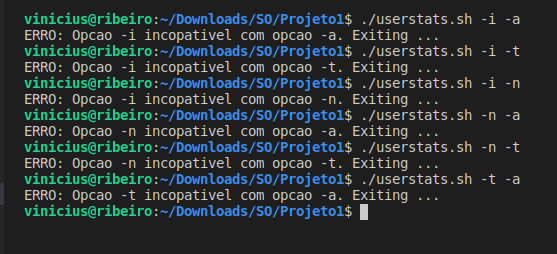
O output é feito com a função ***print().*** Através do flag (atribuído na função main), diferentes outputs, com diferentes tipos de ordem(feito pela função interna sort) são mostrados no terminal.

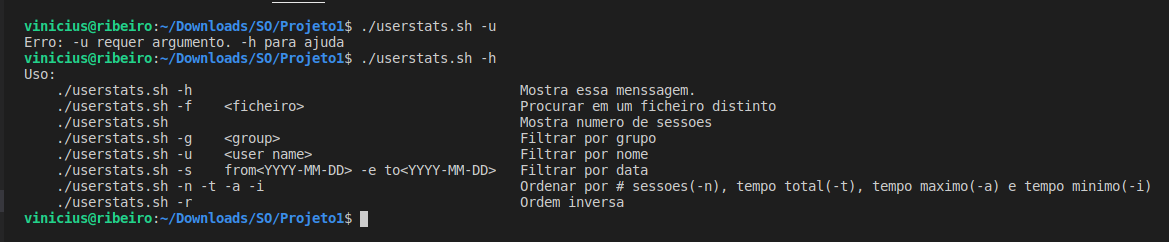
**Segundo script**

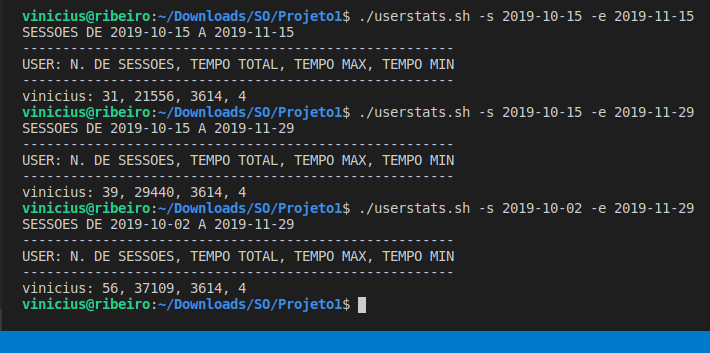
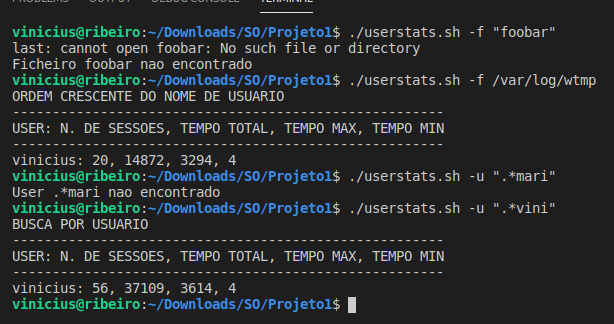
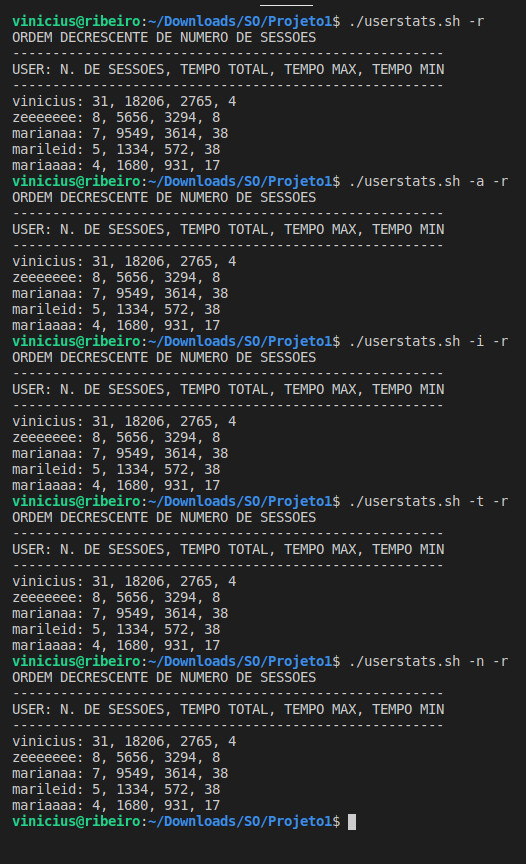
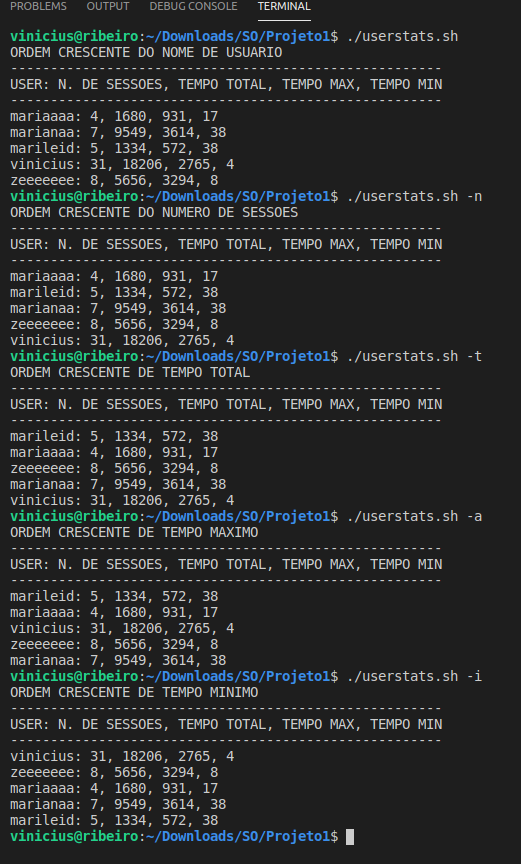
O segundo script também usa o getopts() e os mesmos mecanismos do script anterior. Além disso, checa se os ficheiros passados como argumentos existem e podem ser encontrados.

**Testes**

****

****

****

****

**Fontes**

<http://man7.org/linux/man-pages/man1/sort.1.html>

<http://man7.org/linux/man-pages/man1/last.1.html>

<https://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples>

<https://linuxconfig.org/how-to-use-getopts-to-parse-a-script-options>

<https://www.artificialworlds.net/blog/2012/10/17/bash-associative-array-examples/>

<http://fibrevillage.com/scripting/619-ways-to-remove-empty-lines-in-a-file-on-linux>

Guiões e materiais das aulas P.