INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

**INSTITUTO POLITÉCNICO DE COIMBRA**

**Licenciatura em Engenharia Informática 2º Ano – 1º Semestre 2021/2022**

**“SOBay”**

**Rafael Couto** **Nº 2019142454**

**Filipa Pimentel Nº 2018011698**

**COIMBRA**

07 de jáneiro de 2022

Índice

[Introdução 3](#_Toc124001107)

[Estruturas de dados 4](#_Toc124001108)

[StructThreadCredentials 4](#_Toc124001109)

[Backend 5](#_Toc124001110)

[User 5](#_Toc124001111)

[Comms 6](#_Toc124001112)

[Item 6](#_Toc124001113)

[Promotor 7](#_Toc124001114)

[Variáveis de Ambiente 8](#_Toc124001115)

[Sinais 9](#_Toc124001116)

[Threads 10](#_Toc124001117)

[Cumprimento dos Requisitos **Error! Bookmark not defined.**](#_Toc124001118)

[Conclusão 12](#_Toc124001119)

# Introdução

A elaboração deste trabalho prático visa consolidar conhecimentos em sistemas operativos UNIX.

O objetivo é implementar uma plataforma de gestão para um sistema de leilões, “*SOBay*”. Esta plataforma permitirá comunicação entre clientes e servidor, que gere itens à venda, verifica preços e determina quem adquire os itens.

Esta plataforma estará distribuída entre três programas essenciais:

* ***Frontend:*** programa responsável por utilizadores, que permitirá compra de itens, venda de itens e gestão de saldo. Cada utilizador será representado por um programa *frontend*.
* ***Promotores:*** programas responsáveis pelo lançamento de promoções. Apenas comunicam com o *backend*. No início da aplicação são lançados dois: um *promotor\_oficial* e um *black\_friday*.
* *Backend****:*** programa responsável pelo sistema, que permite toda a comunicação com clientes e promotores. Existe apenas uma instância deste programa a correr.

# Estruturas de dados

## StructThreadCredentials

A estrutura de dados *“StructThreadCredentials”* é responsável por estabelecer a ponte de ligação entre todas as outras estruturas essenciais ao bom funcionamento do sistema. O principal objetivo da criação desta estrutura é servir de elo entre o programa principal e funções a correr em *thread* que necessitem de mais que uma estrutura. Existe apenas uma única estrutura.

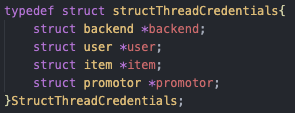


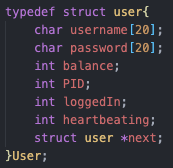
Figura 1- Estrutura StructThreadCredentials

## Backend

A estrutura de dados *“Backend”* é responsável por guardar os valores armazenados das variáveis ambientes e acima de tudo por armazenar o ponteiro para o *mutex*, consequentemente permitindo o bom funcionamento das *threads*. Existe apenas uma única estrutura.

Figura 2- Estrutura Backend

## User

A estrutura de dados *“User”* é responsável por guardar todo os dados necessários para cada utilizador do programa.

Trata-se de uma lista ligada que é criada sempre que o programa *Backend* é iniciado e dado que o foco deste trabalho não são conceitos de programação C e poupança de recursos é alocada memória para o número máximo de utilizadores.

Figura 3- Estrutura User

## Comms

A estrutura *“Comms”* é responsável por armazenar todo o tipo de dados necessários para comunicações entre o programa *Backend* e o programa *Frontend*.

Figura 4- Estrutura Comms

## Item

A estrutura *“Item”* é responsável por armazenar todo o tipo de dados de um item do nosso programa. Fui desenvolvido numa perspetiva funcional e como tal usa a mesma lógica de criação da estrutura *“User”* e é alocado espaço para o número máximo de itens possíveis de gerir.

Figura 5- Estrutura Item

## Promotor

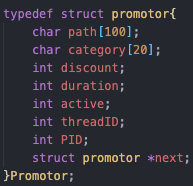
A estrutura *“Promotor”* é responsável por armazenar todo o tipo de dados de um promotor. Dado que todos os promotores são lançados em *thread*, guarda-se também na estrutura o seu *threadID* de modo que seja mais fácil posteriormente matar a *thread*, se necessário. De igual modo, funciona seguindo a mesma metodologia da estrutura *“User”* e *“Item”*, alocando memória para o total de promotores possíveis logo no início do programa *Backend*.

Figura 6- Estrutura Promotor

# Variáveis de Ambiente

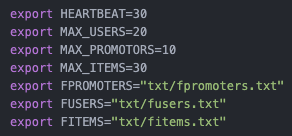


Figura 7 – Variáveis Ambiente

As variáveis ambiente definidas tratam constantes necessárias ao bom funcionamento do programa *Backend* que são verificadas e executadas no início do programa.

Armazenam os valores limite para a criação das estruturas e também a diretório para os ficheiros usados para carregar informações para as estruturas.

# Sinais

Text

Description automatically generated Em relação a implementações de sinais, foi dado uso dos sinais essencialmente para execução de funções. Relativamente ao *Backend*, foi configurado o *SIGINT* para execução da função *ctrlSignal()* que envia um sinal *SIGUSR1* para todos os *Frontend*, de modo a que sejam avisados que o *Backend* fechou e que serão igualmente interrompidos de desempenharem mais ações.

Figura 8 – Função ctrlSignal()

Em ambos os casos executam funções de *quit()* que encerram os seus programas.

Text

Description automatically generated Funções como, *cancelPromotor()* e *quit()* dão uso aos sinais para matar *promotores* e *users*, respetivamente.

Relativamente ao *Frontend*, são igualmente chamadas as funções *receiveSignal()* e *quit()*, ativadas através da receção de sinais *SIGINT* e *SIGUSR1*, respetivamente.

Figura 9 – Excerto da função cancelPromotor()

# Threads

As threads são a grande base deste trabalho, tendo sido implementadas threads para todo o tipo de funções que necessitavam correr em background.

Dada a maior compreensão relativa aos conteúdos programáticos de threads, optou-se por fazer uso deste recurso no seu esplendor.

No *Backend*, permite que funções como *frontendComms()*, *promotorComms()*, *verifyCredentials()*, *itemActions()*, *verifyUserAlive()* e *removeUserNotAlive*. Todas estas funções incorporam ciclos infinitos necessários às verificações recorrentes do programa.

Text

Description automatically generated De igual modo, o *Frontend*, corre em thread as funções, *frontendCommadReader()*, *receiveMessages()*, *threadAlive()* e novamente, portenciam ciclos infinitos de verificações a realizar.

Figura 10 – Exemplo de função usada em Thread

# Decisões de implementação

Alguns aspetos de implementação são passíveis de explicação e, portanto, relativamente à utilização de *threads* invés de *selects* deve-se única e exclusivamente ao facto de, a nosso ver, serem de menor complexidade de compreensão e implementação.

Para a questão de “informar todos os clientes que o *Backend* saiu”, foi escolhido o uso de sinais por ser mais direto, “envia sinal, executa função para sair”.

Em relação ao envio e receção de *Heartbeat*, recorre-se ao envio constante do *PID* do *Frontend*. Se este não enviar o hearbeat no tempo delimitado, o *Backend* coloca o user como *“logged out”* através da variável ambiente definida.

Como já mencionado acima neste documento, visto que o intuito do trabalho e da unidade curricular não é o uso eficiente de recursos, mas sim de mecanismos de sincronização com sinais e aplicações modelo cliente servidor, como esta. De modo a ter fácil acessibilidade a todos os elementos que compõem este programa, decidiu-se alocar a memória para as listas ligadas para o máximo de estruturas que estas podem albergar, e, portanto, não se olha à poupança de memória, mas sim ao bom funcionamento do programa.

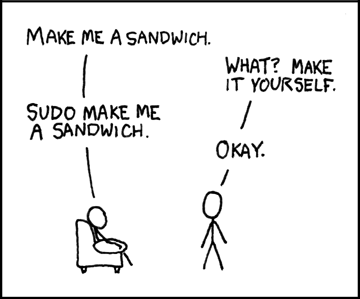
# Conclusão

A termo de conclusão, este trabalho é de tamanha importância para a compreensão de sistemas operativos na sua generalidade e em como tudo se desenrola por de trás do que é representado num ecrã.

Ainda de referir que o trabalho se torna muito mais fácil desde o momento em que tornam o tema interessante e sem aspetos desnecessários e ou sem aplicação de matérias lecionadas.

Em termos de conceitos usados, as *threads* foi de facto um aspecto valorizado e reconhecida a sua importância, bem como os *mutexes* que permitem bloquear conteúdos que possam ser acessados por duas ou mais *threads* em simultâneo. Igualmente os sinais são uma metodologia muito interesse em ambiente Unix de avisar um processo e ou até desempenhar alguma ação predefinida.

Os pipes também foi um conceito de excelente compreensão após a implementação no trabalho, tanto o seu funcionamento síncrono como assíncrono.

 Por fim, salientar a importância desta unidade curricular no programa da Licenciatura em Engenharia Informática, que potencia o melhor conhecimento dos sistemas operativos.