C:\Users\tiago\AppData\Local\Microsoft\Windows\INetCache\Content.Word\logo-isec-transparente.png

**Engenharia Informática**

**Sistemas Operativos**

**Relatório**

**Trabalho Prático – Meta 1**

**Ano Letivo**: 2020/2021

**Curso**: Engenharia Informática

**Autores**: Ricardo Pereira – 21250780, Pedro Ruivo - X

**Disciplina**: Sistemas Operativos

**Turma**: P1

**Professores**: João Durães, Jorge Rodrigues

**Data de Submissão:** 08/11/2020

**Índice**

[1. Introdução 3](#_Toc41168519)

[2. Estruturas de dados 4](#_Toc41168521)

[2.1. CenDLL - Map 4](#_Toc41168522)

[2.2. CenDLL – XYObject 5](#_Toc41168523)

[2.3. CenDLL - Passenger 6](#_Toc41168524)

[2.4. CenDLL - Taxi 7](#_Toc41168525)

[2.5. CenDLL - Requests 8](#_Toc41168526)

[2.6. CenTaxi – TParam 9](#_Toc41168527)

[2.7. CenTaxi – CenPassenger 10](#_Toc41168528)

[2.8. CenTaxi – Settings 11](#_Toc41168529)

[2.9. CenTaxi – CenService 12](#_Toc41168530)

[2.10. CTDLL e ConTaxi – TParam 13](#_Toc41168531)

[2.11. CTDLL – Settings 14](#_Toc41168532)

[2.12. CTDLL – Service 15](#_Toc41168533)

[3. Mecanismos usados 16](#_Toc41168534)

[3.1. QnARequest 16](#_Toc41168535)

[3.2. TossRequest 17](#_Toc41168536)

[3.3. NewTransportNotification 18](#_Toc41168537)

[4. Requisitos 19](#_Toc41168538)

1. Referee (Árbitro)

Na meta 1, o árbitro apenas prepara a sua aplicação. Onde absorve os parâmetros recebidos pela consola e armazena na sua estrutura de dados, esta funcionalidade não está totalmente testada, mas foi desenvolvida de modo a que a ordem não seja importante, mas a aplicação será encerrada (corretamente) se os parâmetros não forem fornecidos como esperado. Juntamente com os parâmetros, esta aplicação também obtém variáveis de ambiente. Se não conseguir obter as variáveis de ambiente, esta irá usar os valores pré-definidos no seu ficheiro *header*.

* 1. Constantes

#define DEFAULT\_GAMEDIR "~/Documents/SO/SO\_CHAMPION/Games/"

#define DEFAULT\_MAXPLAYER 10

#define MIN\_CHAMP\_DURATION 60000

#define MAX\_CHAMP\_DURATION 600000

#define MIN\_WAITING\_DURATION 30000

#define MAX\_WAITING\_DURATION 120000

#define MAX\_MAXPLAYER 30

Estas constantes são autoexplicativas e usadas para evitar falhas no uso da aplicação.

* 1. Estruturas de dados

typedef struct Referee Referee;

typedef struct Application Application;

struct Referee{

char gameDir[STRING\_MEDIUM];

int maxPlayers;

int championshipDuration;

int waitingDuration;

};

struct Application{

Referee referee;

};

A estrutura de dados principal, *Application* é usada para armazenar a informação necessária para o funcionamento da aplicação. Esta contem uma outra estrutura

1. Estruturas de dados

Neste capítulo serão apresentadas e explicadas as estruturas de dados definidas para a meta 1 do programa. Mas, para facilitar a explicação de cada estrutura é necessário conhecer o projeto completo, sendo este constituído por:

* CenDLL – Uma DLL (não requerida), responsável por fornecer funções gerais.
* CTDLL – Uma DLL responsável de gerir a comunicação entre ConTaxi e CenTaxi (apenas contém comportamentos a serem utilizados por ConTaxi)
* CenTaxi – Uma aplicação de consola, que é a aplicação principal onde é gerido grande parte do projeto.
* ConTaxi – Uma aplicação de consola, que é usada pelos taxistas, de modo a que estes se possam movimentar e transportar passageiros.
  1. CenDLL - Map

struct Map{

int width;

int height;

char\* cellArray;

};

Esta estrutura é utilizada para armazenar informações de um mapa, nomeadamente largura, altura e um *array* de caracteres, sendo assim, a estrutura é composta por:

**width**

Inteiro para armazenar a largura do mapa.

**height**

Inteiro para armazenar a altura do mapa.

**cellArray**

*Array* dinâmico para guardar todas as células do mapa.

* 1. CenDLL – XYObject

struct XYObject{

double coordX;

double coordY;

double speedX;

double speedY;

double speedMultiplier;

};

Esta estrutura é utilizada para armazenar informações sobre objetos 2D, foi definido que todos os objetos visíveis no mapa deveriam conter esta estrutura. A estrutura é composta por:

**coordX**

Double para armazenar a posição da coordenada X.

**coordY**

Double para armazenar a posição da coordenada Y.

**speedX**

Double para armazenar a velocidade da coordenada X, indicando a direção.

**speedY**

Double para armazenar a velocidade da coordenada Y, indicando a direção.

**speedMultiplier**

Double para armazenar a velocidade do objeto, sendo possível calcular a próxima posição adicionando (speedX/Y \* speedMultiplier) à posição atual.

* 1. CenDLL - Passenger

struct Passenger{

bool empty;

TCHAR Id[STRING\_SMALL];

XYObject object;

};

Esta estrutura é utilizada para armazenar informações sobre um passageiro. A estrutura é composta por:

**empty**

Bool para identificar se o passageiro está a ser usado ou se é lixo.

**Id**

String para armazenar o *ID* do passageiro.

**object**

*XYObject* para armazenar o objeto 2D do passageiro.

* 1. CenDLL - Taxi

struct Taxi{

bool empty;

TCHAR LicensePlate[STRING\_SMALL];

XYObject object;

TaxiState state;

};

Esta estrutura é utilizada para armazenar informações sobre um táxi. A estrutura é composta por:

**empty**

Bool para identificar se o táxi está a ser usado ou se é lixo.

**LicensePlate**

String para armazenar a matrícula do táxi.

**object**

*XYObject* para armazenar o objeto 2D do táxi.

**state**

*Enumeração* para identificar o estado do táxi.

A enumeração usada é a seguinte:

enum TaxiState{

TS\_EMPTY, //Is empty

TS\_OTW\_PASS, //On The Way to the passenger

TS\_WITH\_PASS, //Currently transporting a passenger

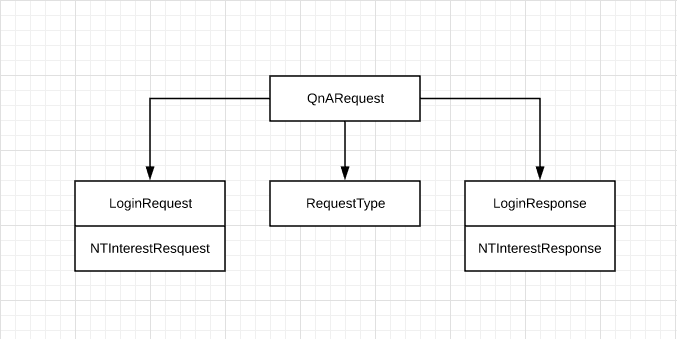
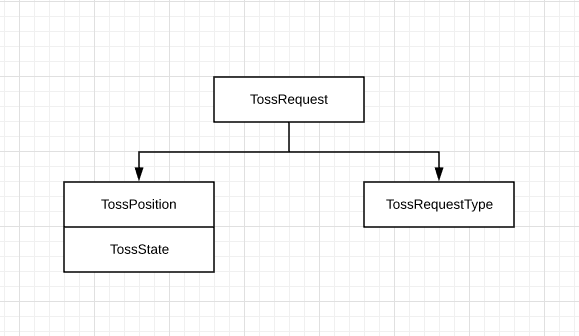
TS\_STATIONARY //Currently stationary

};

* 1. CenDLL - Requests

Estas estruturas são idênticas em questão ao funcionamento, contudo é necessário tomar nota que:

* QnARequest e TossRequest são as estruturas principais, sendo que QnARequest envia pedido e recebe resposta enquanto TossRequest apenas envia informação;
* QnARequest é constituída por uma enumeração que identifica o tipo de pedido, por uma das estruturas de *request* (usando *union*) e por uma das estruturas de *response* (usando *union*);
* TossRequest é constituída por uma enumeração que identifica o tipo de pedido, e por uma das estruturas *toss* (usando *union*;
* Cada estrutura de *request* ou *response* é constituída por informações necessária de modo a completar a sua finalidade.

* 1. CenTaxi – TParam

typedef struct TParam\_QnARequest TParam\_QnARequest;

typedef struct TParam\_TaxiAssignment TParam\_TaxiAssignment;

typedef struct TParam\_ConsumeTossRequests TParam\_ConsumeTossRequests;

Estas estruturas são utilizadas para enviar argumentos para as *threads*, de modo que a estas tenham tudo o que é necessário para completar o seu objetivo. Sendo estas:

struct TParam\_QnARequest{

Application\* app;

};

struct TParam\_TaxiAssignment{

Application\* app;

int myIndex;

};

struct TParam\_ConsumeTossRequests{

Application\* app;

};

**TParam\_QnARequest**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização;

**TParam\_TaxiAssignment**

Esta estrutura é constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização e por um inteiro que identifica o passageiro associado à thread;

**TParam\_ConsumeTossRequests**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização;

* 1. CenTaxi – CenPassenger

typedef struct CenPassenger CenPassenger;

typedef struct CPThreadHandles CPThreadHandles;

Estas estruturas servem de modificação para a estrutura de passageiros já existente, com o objetivo de adicionar *handles* para a *thread* que irá tratar de escolher o táxi a transportar o respetivo passageiro. Sendo estas:

struct CPThreadHandles{

HANDLE hTaxiAssignment;

DWORD dwIdTaxiAssignment;

};

struct CenPassenger{

Passenger passengerInfo;

CPThreadHandles cpThreadHandles;

int\* interestedTaxis;

};

**CP\_ThreadHandles**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização;

**CenPassenger**

Esta estrutura é constituída por um passageiro, estrutura *CPThreadHandles* e um *array* de inteiros que guarda os taxistas interessados a transportar o passageiro;

* 1. CenTaxi – Settings

struct Settings{

int secAssignmentTimeout;

bool allowTaxiLogins;

};

Esta simples estrutura é usada para gerir as definições da aplicação CenTaxi, sendo composta por:

**secAssignmentTimeout**

Inteiro que representa o tempo em segundos necessário para fechar um transporte, escolhendo um táxi para o passageiro respetivo e notificar ambos;

**allowTaxiLogins**

Bool que representa o acesso a novos táxis de fazer login, caso seja *false*, os táxis não poderão efetuar login.

* 1. CenTaxi – CenService

typedef struct Application Application;

typedef struct ThreadHandles ThreadHandles;

typedef struct SyncHandles SyncHandles;

typedef struct ShmHandles ShmHandles;

Estas estruturas são usadas para gerir grande parte da funcionalidade da aplicação, contudo é necessário tomar nota que:

* Application é a estrutura principal que irá incorporar as restantes;
* ThreadHandles apenas guarda os *handles* e *Ids* das *threads* criadas;
* SyncHandles apenas guarda os *handles* dos mecanismos criados ou abertos;
* ShmHandles apenas guarda os *handles* e ponteiros para a memória partilhada criada e aberta;

Portanto, só é necessário falar da estrutura application:

struct Application{

Settings settings;

Taxi\* taxiList;

Map map;

CenPassenger\* passengerList;

ThreadHandles threadHandles;

SyncHandles syncHandles;

ShmHandles shmHandles;

int maxTaxis;

int maxPassengers;

};

**settings**

Estrutura de *settings*;

**taxiList e passengerList**

*Arrays* dinâmicos para guardar todos os táxis e passageiros do sistema;

**Map**

Estrutura de mapa, para guardar o mapa carregado a partir de um ficheiro;

**maxTaxis**

Inteiro que representa a quantidade máxima de táxis aceite ao mesmo tempo;

**maxPassengers**

Inteiro que representa a quantidade máxima de passageiros aceite ao mesmo tempo;

* 1. CTDLL e ConTaxi – TParam

typedef struct TParam\_StepRoutine TParam\_StepRoutine;

typedef struct TParam\_QnARequest TParam\_QnARequest;

typedef struct TParam\_NotificationReceiver\_NT TParam\_NotificationReceiver\_NT;

typedef struct TParam\_TossRequest TParam\_TossRequest;

Tal como mencionado em estruturas anteriormente, estas estruturas são utilizadas para enviar argumentos para as *threads*, de modo que a estas tenham tudo o que é necessário para completar o seu objetivo. Sendo estas:

struct TParam\_StepRoutine{

Application\* app;

};

struct TParam\_QnARequest{

Application\* app;

QnARequest request;

};

struct TParam\_NotificationReceiver\_NT{

Application\* app;

};

struct TParam\_TossRequest{

Application\* app;

TossRequest tossRequest;

};

**TParam\_StepRoutine**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso às informações do táxi a processar;

**TParam\_QnARequest**

Esta estrutura é constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização e o pedido a ser processado pela *thread*;

**TParam\_NotificationReceiver\_NT**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização;

**TParam\_TossRequest**

Esta estrutura é apenas constituída por *Application*, de modo a que a *thread* tenha acesso aos *handles* de sincronização e o pedido a ser processado pela *thread*;

* 1. CTDLL – Settings

struct Settings{

int CDN;

bool automaticInterest;

};

Esta simples estrutura é usada para gerir as definições da aplicação ConTaxi, sendo composta por:

**CDN**

Inteiro que representa a distância máxima em células para enviar um pedido de interesse de um transporte à central, de forma automática;

**automaticInterest**

Bool que representa se o interesse deverá ser feito de forma automática ou não;

* 1. CTDLL – Service

typedef struct Application Application;

typedef struct ThreadHandles ThreadHandles;

typedef struct SyncHandles SyncHandles;

typedef struct ShmHandles ShmHandles;

Tal como as estruturas do CenTaxi Service, estas estruturas são usadas para gerir grande parte da funcionalidade da aplicação, contudo é necessário tomar nota que:

* Application é a estrutura principal que irá incorporar as restantes;
* ThreadHandles apenas guarda os *handles* e *Ids* das *threads* criadas;
* SyncHandles apenas guarda os *handles* dos mecanismos criados ou abertos;
* ShmHandles apenas guarda os *handles* e ponteiros para a memória partilhada criada e aberta;

Portanto, só é necessário falar da estrutura application:

struct Application{

Settings settings;

Taxi loggedInTaxi;

HANDLE taxiMovementRoutine;

Map map;

ThreadHandles threadHandles;

SyncHandles syncHandles;

ShmHandles shmHandles;

int maxTaxis;

int maxPassengers;

int NTBuffer\_Tail;

};

**settings**

Estrutura de *settings*;

**loggedInTaxi**

Estrutura que representa o táxi ligado à aplicação;

**taxiMovementRoutine**

*Handle* para guardar o *handle* do *WaitableTimer* responsável pelo movimento do táxi;

**Map**

Estrutura de mapa, para guardar o mapa carregado a partir da memória partilhada;

**maxTaxis**

Inteiro que representa a quantidade máxima de táxis aceite ao mesmo tempo;

**maxPassengers**

Inteiro que representa a quantidade máxima de passageiros aceite ao mesmo tempo;

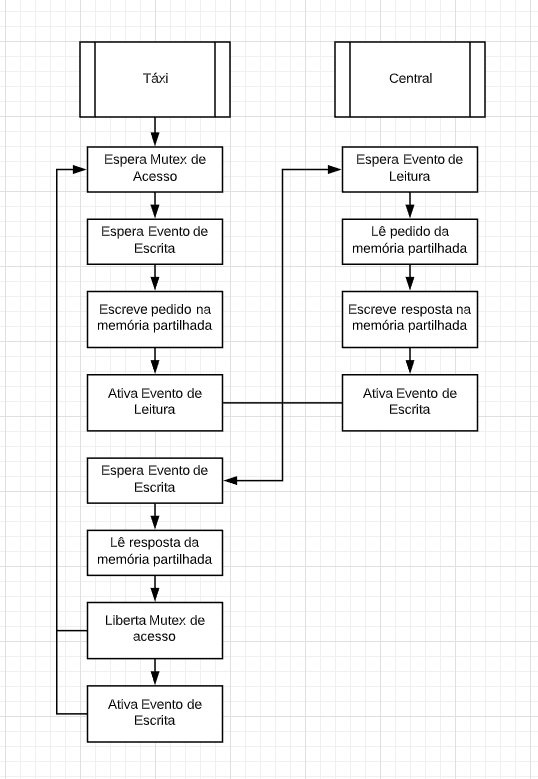
**NTBuffer\_Tail**

Inteiro que representa a posição do táxi no buffer circular das notificações de transporte;

1. Mecanismos usados

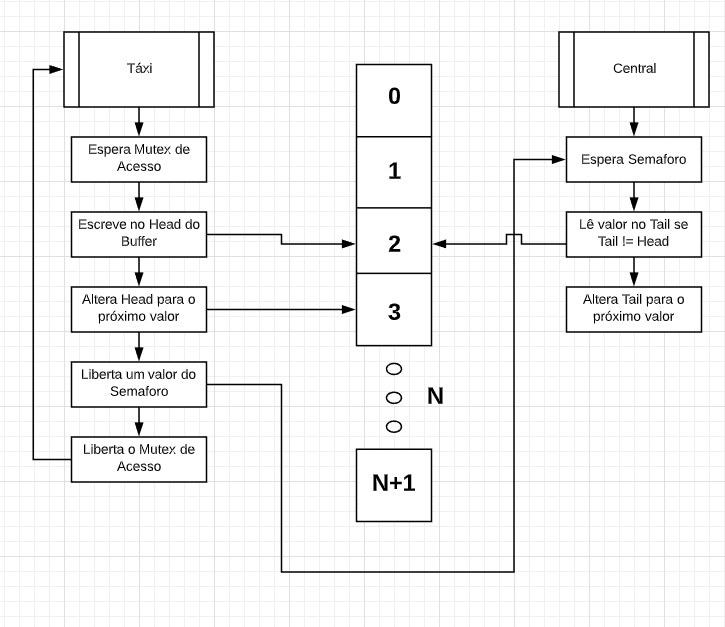
Foram usados três mecanismos complexos, de modo a que a comunicação entre ConTaxi e CenTaxi não tenha problemas, sendo estes:

* QnARequest: Um mecanismo desenvolvido de maneira que cada táxi possa efetuar um pedido e esperar por resposta;
* TossRequest: Um mecanismo desenvolvido de maneira que cada táxi envie informação para a central e é apenas consumida, sem resposta;
* NewTransportNotification: Um mecanismo desenvolvido de maneira que a central informe os táxis quando algum passageiro efetuar um pedido de transporte.
  1. QnARequest



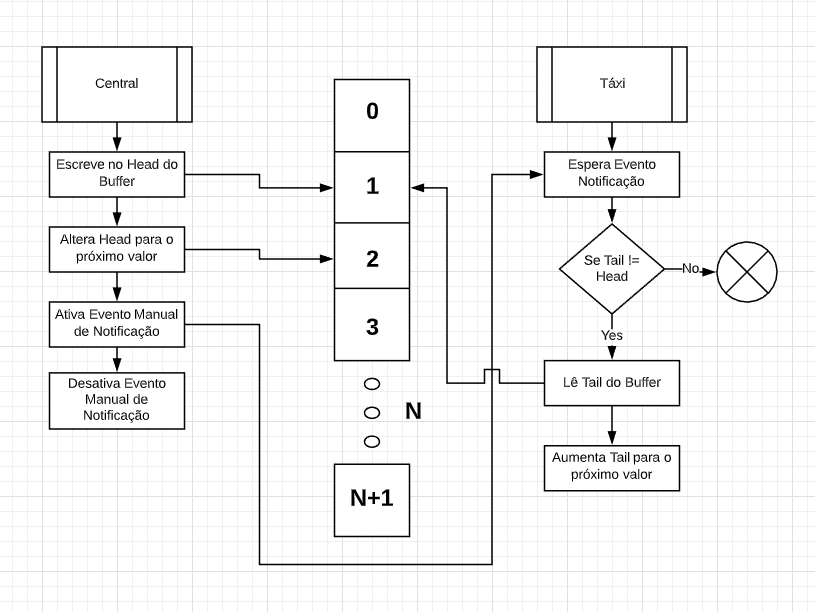
* 1. TossRequest

Mecanismo buffer circular produtor consumidor N -> 1



* 1. NewTransportNotification

Mecanismo buffer circular produtor consumidor 1 -> N



1. Requisitos

Neste capítulo serão apresentadas as funcionalidades solicitadas e desenvolvidas para a meta 1. Todos os requisitos solicitados foram cumpridos, confio que as funcionalidades pedidas para desenvolver foi interpretado de forma correta e foram concluídas tais funcionalidades com sucesso e ainda implementaram funcionalidades extras em relação às solicitadas para esta meta.

É importante referir que todas as implementações aplicadas nesta meta poderão ser mais tarde alteradas devido a falhas de lógica ou inconsistências perante os requisitos futuros, *bugs* não detetados, melhores formas de implementação, entre outras razões.

Referente aos requisitos desenvolvidos e não desenvolvidos, será enviado um PDF (Development) adjacente com tabelas Excel de modo a fornecer a informação pedida.