DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E DE SISTEMAS
INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Engenharia Informática

Sistemas Operativos 2

Relatório

Trabalho Prático – Meta 1

Ano Letivo: 2019/2020
Curso: Engenharia Informática
Autores: Ricardo Pereira – 2015015815
Disciplina: Sistemas Operativos 2
Turma: P4
Professores: João Durães, Ricardo Pereira
Data de Submissão: 23/05/2020

Índice

1. Introdução

Este projeto foi realizado no âmbito da disciplina de Sistemas Operativos 2 e o tema abordado é sobre um gestor de transportes de passageiros e taxistas em consola. O projeto foi desenvolvido em C no Windows.

Neste relatório apenas serão descritos e apresentados todos os requisitos da meta 1 do trabalho prático.

A primeira meta consiste em desenvolver e planear as estruturas de dados que serão utilizadas ao longo do projeto e em desenvolver os mecanismos e estratégias base do programa de modo a completar a maior parte das funcionalidades da aplicação CenTaxi e ConTaxi.

Todos os requisitos solicitados foram implementados, no entanto, poderá faltar alguns detalhes ou algo que não me apercebi da necessidade, e, ao longo do desenvolvimento da segunda meta poderão ser feitas algumas alterações ao que foi feito até à meta 1.

Para além do que foi requerido, também foi introduzida uma DLL (CenDLL) que fornece funções gerais. Algumas estruturas gerais podem ser utilizadas por várias aplicações, sendo assim, estas foram colocadas em ficheiros *header* no mesmo diretório da DLL.

2. 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.

2.1. 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.

2.2. 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.

2.3. 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.

2.4. 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.

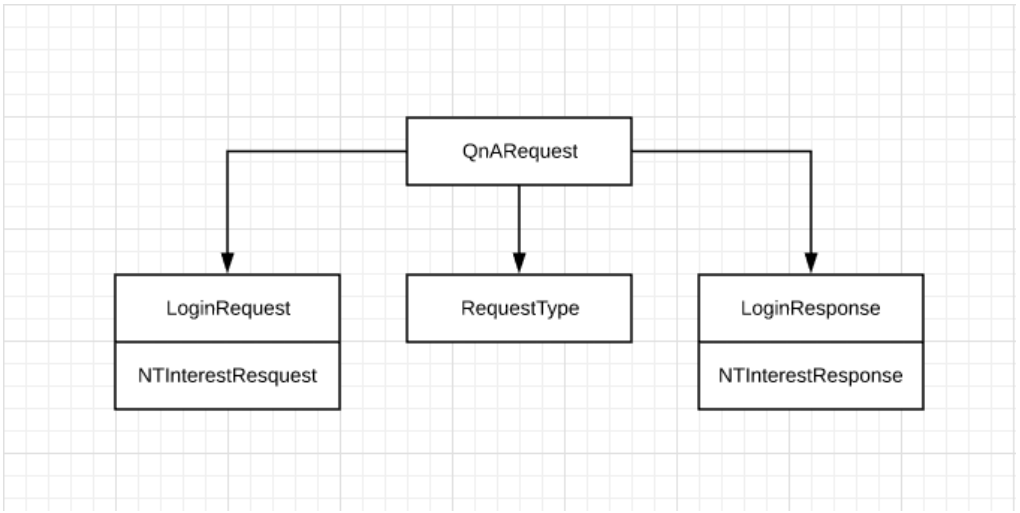
A enumeração usada é a seguinte:

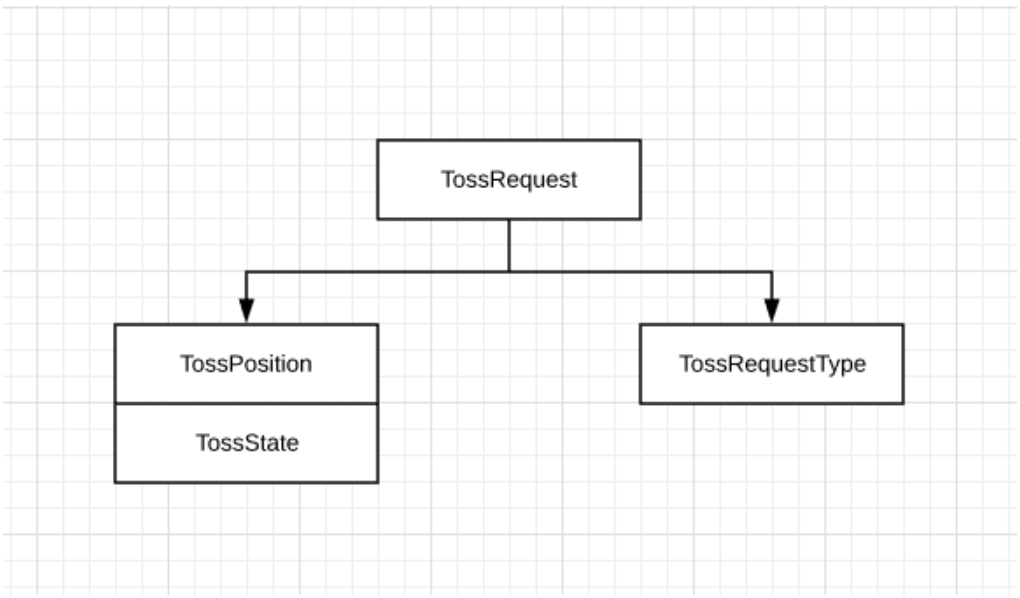
```
enum TaxiState{
	TS_EMPTY, //Is empty
	TS_OTW_PASS, //On The Way to the passenger
	TS_WITH_PASS, //Currently transporting a passenger
	TS_STATIONARY //Currently stationary
};
```

2.5. 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.





2.6. 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;

2.7. 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;

2.8. 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.

2.9. 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;

2.10. 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*;

2.11. 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;

2.12. 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;

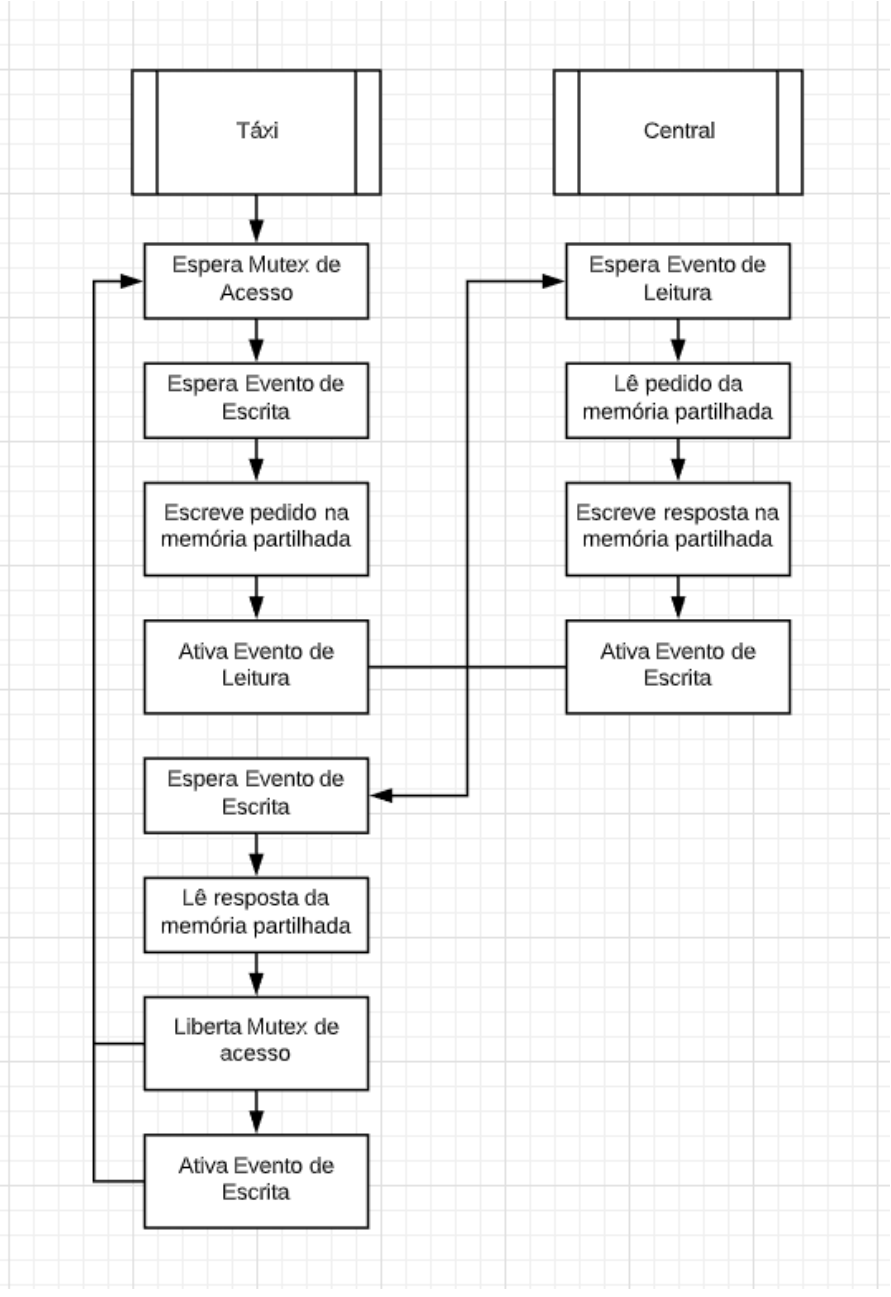
NTBuffer_Tail
Inteiro que representa a posição do táxi no buffer circular das notificações de transporte;

3. Mecanismos usados

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

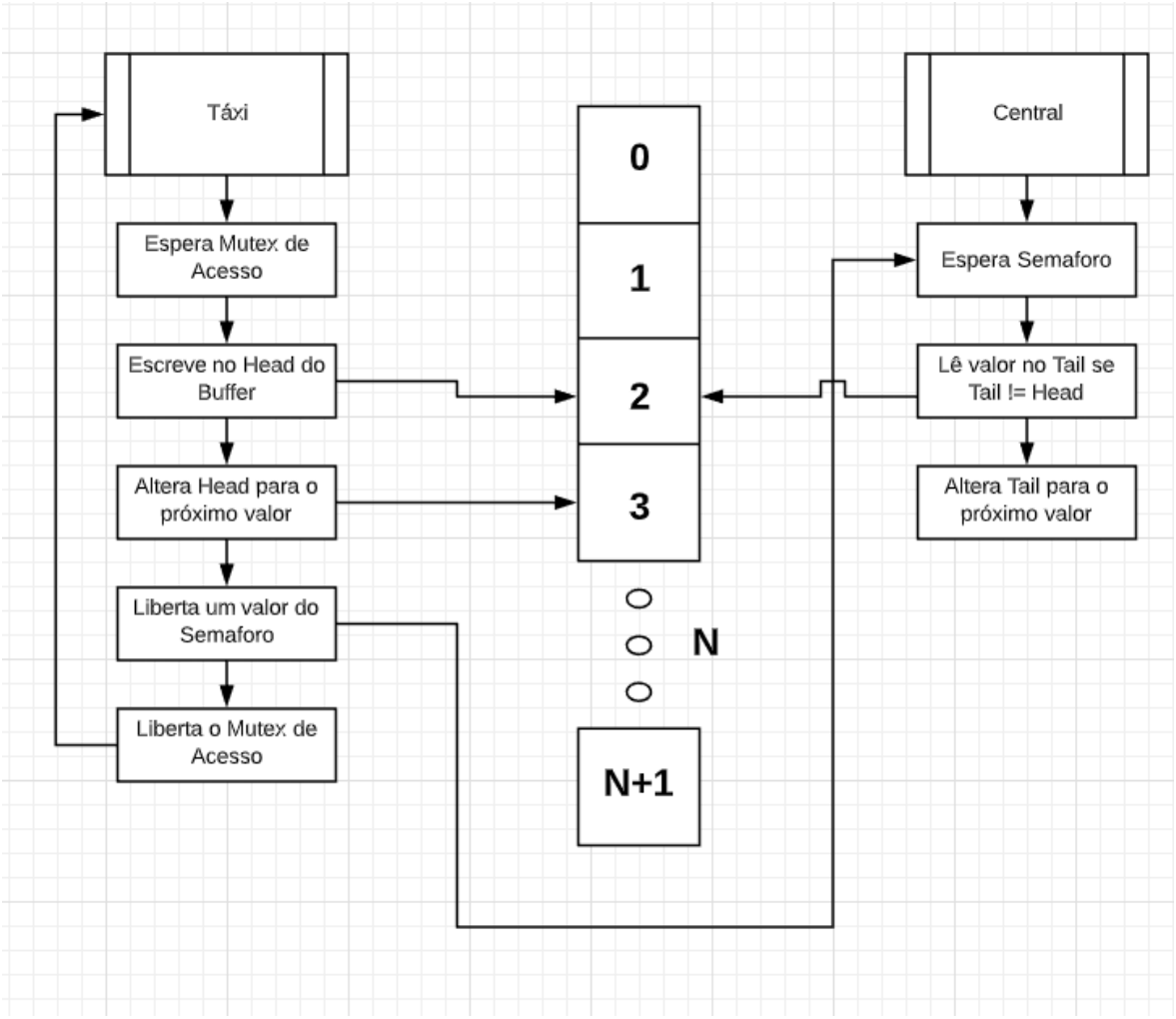
- QnARequest: Um mecanismo desenvolvido de maneira que cada táxi possa efetuar um pedido e esperar por resposta;
- TossRequest: Um mecanismo desenvolvido de maneira que cada táxi envie informação para a central e é apenas consumida, sem resposta;
- NewTransportNotification: Um mecanismo desenvolvido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

3.1. QnARequest



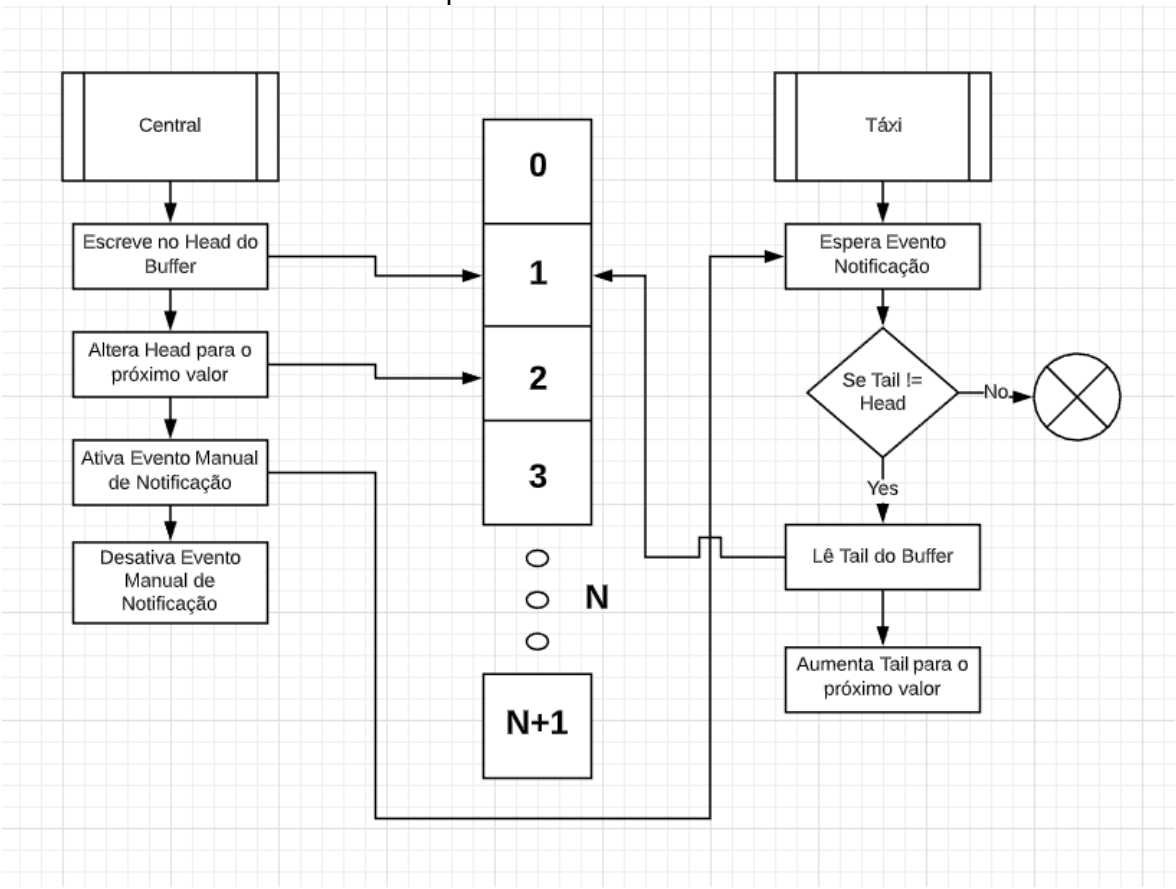
3.2. TossRequest

Mecanismo buffer circular produtor consumidor N -> 1



3.3. NewTransportNotification

Mecanismo buffer circular produtor consumidor 1 -> N



4. 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.