Instituto Politécnico de Setúbal - Escola Superior de Tecnolgia -2014

TUQUETUQUE ENTERPRISE

MANUAL TÉCNICO

By

Ana Sequeira Nº120221055

Andre Benevides Nº120221013

licenciatura em Engenharia Informática

Docente: Nuno Ribeiro

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUÇÃO

O manual técnico do programa “TuqueTuque” fornerce-lhe a informação necessária à cerca das funcionalidades implementadas e através de que métodos foram implementadas, portanto a forma como foi construído o programa de forma a efetuar o prentendido.

O pretendido com este programa como se pode observar de forma simplicada no manual de utilizador é a partir de um programa de simulação efectuar um estudo relativo ao funcionamento da empresa “TuqueTuque”, relativamente ao número de turistas atendidos por hora, a distância máxima de viagem percorrida por este transporte e o número de tuque tuques que estão em questão, para determinado plano.

TUQUETUQUE

No nosso programa, o tuque tuque consiste num meio de transporte, que leva dois passageiros de cada vez ao seu destino. Em termos técnicos, este tuque tuque consiste apenas num processo consumidor, que funciona em acordo com um processo produtor. Para o processo consumidor funcionar da forma pretendida temos regras de acesso às memórias partilhadas e estas regras são impostas através da utilização de semáforos.

Resumindo, temos um programa simplificado, no qual criámos duas memórias partilhadas (uma para a fila de atendimento e outra para as estatísticas), três semáforos (uma para as pessoas, um para os tuque tuques e o outro para as estatísticas), um processo-pai e diversos processos-filhos (número variável da configuração da simulação).

FUNCIONAMENTO DA SIMULAÇÃO

Ao executar a aplicação deparamo-nos com o seguinte menu principal:

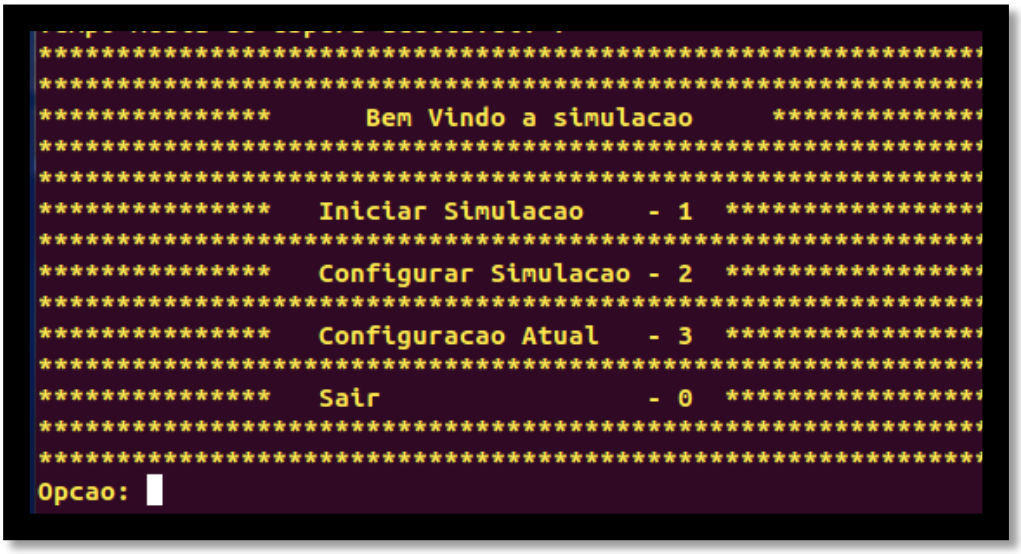

Figura 1-Menu Principal

Se pretendermos observar a simulação em ocorrência, escolhemos a opção 1 e iremos ter algo do género da seguinte imagem.

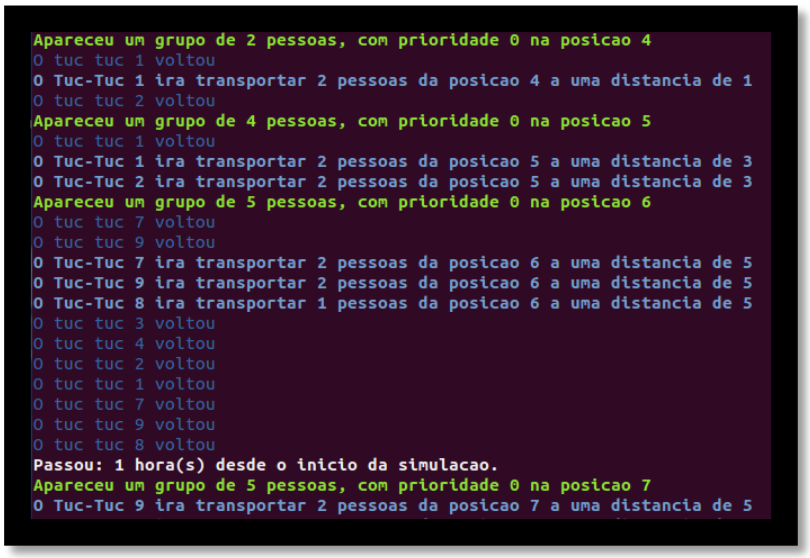

Figura 2- Simulação a decorrer

As outras opções são de consulta, no caso da opção 3, e de alterações na configuração no caso da 2, pois esta simulação deve permitir diversos testes, de modo a chegar a conclusões fiáveis.

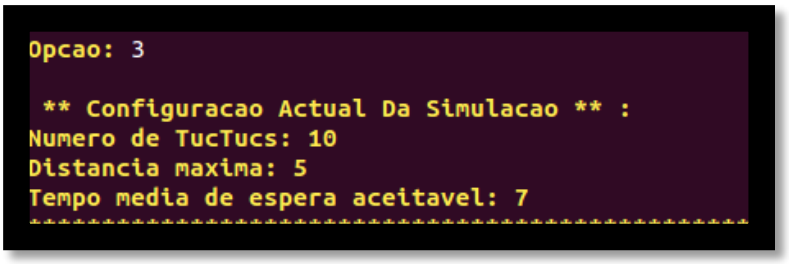

Figura 3-Consulta da configuração

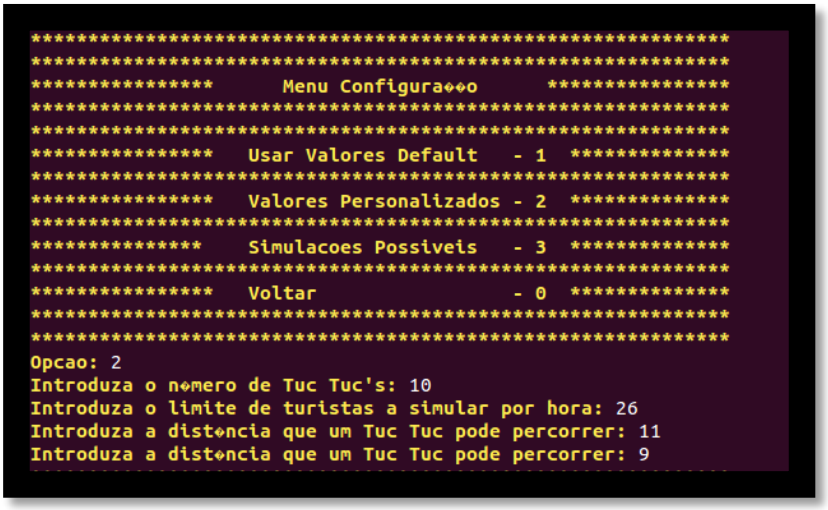

Figura 4-Configuração Personalizada

SIMULAÇÃO PARA VÁRIOS CENÁRIOS

Além da possibilidade de configuração personalizada, oferecemos no nosso programa, exemplos especificos para simulação. Estes exemplos servem para mostrar ao utilizador de como pode vir a configurar a sua simulação e desta forma o que pode alterar para a simulação manter se eficiente. No sentido, de que, se pretendermos aumentar a distancia a qual os Tuc-Tucs se deslocam, ou se o número de turistas aumenta numa determinada altura do ano, se calhar devemos implementar mais Tuc-Tucs e contratar mais motoristas para esta altura especifíca, isto, para que o tempo de espera dos clientes não seja demasiado, e isto não cause aos clientes a escolher outro transporte.

Menu de exemplos de configuração da simulação:

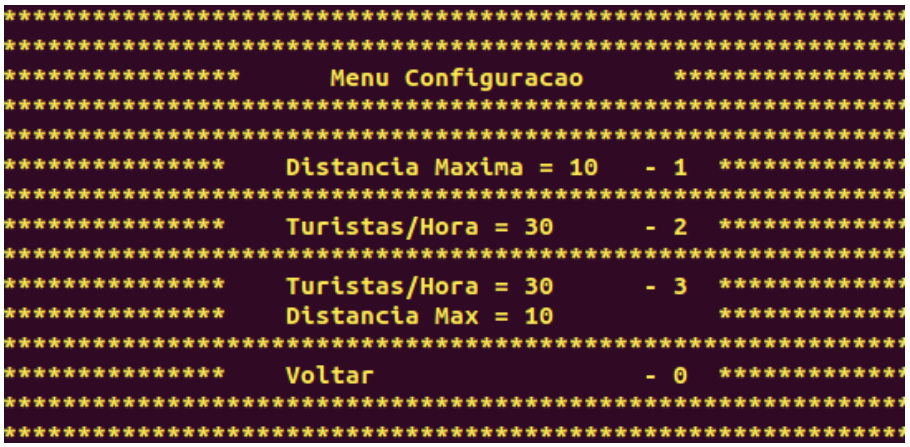

Figura 5-Menu Configuração – Exemplos

ESTASTÍSTICAS

Após a ocorrência de uma simulação, o programa apresenta as estastísticas resultantes dessa mesma simulação.

Considerando as perguntas que nos foram propostas no enunciado, ao nível das estastísticas, testámos, utilizando os nossos exemplos de configuração de simulação (ver figura 5), e obtemos as seguintes estastísticas.

Quanto pressionamos a opção 1 e voltamos a simular em que a distância máxima é alterada para 10km:

Figura 6 - Exemplo de Simulação 1 - Distância Máxima de 10km

Quando pressionamos a opção 2, em que os turistas aparecem com uma média de 30 por hora:

Figura 7-Exemplo de Simulação 2 - Média de turistas por hora é 30

Quando pressionamos a opção 3, que consiste, na união da opção 2 e 1, ou seja, uma distância máxima de 10km e com turistas a aparecerem a 30 por hora.

Figura 8 - Exemplo de Simulação 3 - Média de turistas por hora é 30 e distância máxima é 10km

As conclusões que podemos tirar depois destas 3 simulações diferentes é que o tempo de espera máximo é facilmente afectável pela distância máxima e pelo número de turistas que aparecem por hora. Por outro lado, o tempo de espera médio conseguiu manter-se.

Podemos verificar que quando aumentamos a distância máxima para 10km, como se verifica na figura 6 (Simulação 1), temos uma distância total superior à distância verificada na figura 7(Simulação 2), em que a distância máxima era de 5km. Por outro lado, podemos observar que na figura 7, em que o número médio de turistas por hora é de 30, temos um número de turistas gerados e servidos superior ao número corresponde do da figura 6, simulação em que a média de turistas por hora é de 25.

Se compararmos a última simulação com as duas anteriores, sendo esta uma simualação com mais distância, e mais turistas simultâneamente, temos que, tanto o número de distância total, como o número de turistas gerados e servidos é superior quando comparado com uma simulação das anteriores, em que, no âmbito destas variáveis, tem um valor inferior. Conclui-se a última simulação, tem ao mesmo tempo, mais turistas e mais distância total. Tornando-se numa simulação que “consumiu mais” e “produziu mais”, temos um tempo máximo de espera de 7m, igual ao da simulação 1, em que a distância foi duplicada, ou seja, o tempo que demoramos a fazer cada viagem pode ser duplicado.

Este estudo, é baseado, neste caso apenas num teste com cada uma das configurações, mas temos de ter em conta, que o número de km que um tuque tuque vai percorrer vai depender da distância ao destino que os turistas se prentendem deslocar, ou seja, sendo a distância completamente variável, mais o facto de aparecer turistas com prioridade ser tambem um caso deixado à sorte, e por fim a constituição de um grupo de turistas ser de 1 a 5, pode afectar drásticamente o tempo de espera máximo. Estes dados teriam de ser analisados com um número muito elevado de simulações para tirarmos conclusões mais sólidas. Conforme os nossos conhecimentos actuais, vamos tentar responder ás perguntas do enunciado.

Figura 9- Simulação Default - Estastisticas

Questões de enunciado

a) Se o número de tuc-tucs actual é adequado para a média de turistas por hora que recorrem ao serviço actualmente e para a distância máxima que podem percorrer actualmente. Assumindo que um tempo máximo de espera de 7 minutos é aceitável.

b) O número de tuc-tucs (mantendo a distância máxima e a média de turistas por hora actuais) adequado para um tempo de espera máximo de 5 minutos.

c) O número de tuc-tucs adequado (mantendo a distância máxima actual e com um tempo máximo de espera de 7 minutos) se o número médio de turistas a recorrer ao serviço for de 30 por hora.

d) O tempo de espera médio esperado se a distância máxima for alterada para 10 km e se mantiverem os valores actuais para o número de tuc-tucs e a média de turistas por hora a recorrer ao serviço.

e) O número ideal de tuc-tucs se a distância máxima for alterada para 10 km e se mantiver a média de turistas por hora a recorrer ao serviço actuais. Assumindo que um tempo máximo de espera de 7 minutos é aceitável.

f) Qual o número ideal de tuc-tucs, se a distância máxima for alterada para 10 km e a média de turistas por hora a recorrer ao serviço for de 30 por hora. Assumindo que um tempo máximo de espera de 7 minutos é aceitável.

a) Verificou-se que o número de Tuc-Tucs atual é adequado havendo apenas um tempo de espera maximo de 4min.

b) Para 5 minutos como tempo de espera máximo chegamos a conclusão que o número atual de Tuc-Tucs também é adequado.

c) No caso de aumentar o numero de turistas para 30 por hora, o número de Tuc-Tucs atuais continua a ser adequado e até é possível diminuir o número de Tuc-Tucs.

d) Caso a distância aumente para 10km, o número de Tuc-Tucs manter-se 10 e o número de turistas por hora manter-se 25/h o tempo de espera médio será 1-2 min.

e) No caso de o número de turistas por hora manter-se a 25 e a distancia máxima aumentar para 10km e para um tempo de espera máximo de 7 minutos, o número de Tuc-Tucs ideal seria a partir de 10 Tuc-Tucs. No entanto como se trata de uma simulação em que cada execução da aplicação são gerados dados diferentes este número de Tuc-Tucs pode não ser aceitável e seria melhor considerar aumentar o número de Tuc-Tucs para 12.

f) No caso de o número de turistas por hora aumentar para 30 por hora e a distância máxima para 10km e para um tempo de espera máximo de 7minutos, o número de tuc tucs ideal seria a partir de 10 tuc tucs. No entanto,

para termos mais certezas de que o tempo máximo de espera nao irá exceder estes 7m, o número ideial de tuc tucs seria, por exemplo, 12 tuc tucs. Isto devido á distância percorrida pelos tucs tucs ser completamente dependente da preferência dos turistas, poderiamos ter azar e percorrer apenas

ESTRUTURAS

Para que fosse possível retirar conclusões sobre o que melhorar numa empresa de transporte por Tuc-Tucs optamos pela implementação de estructuras para tornar o código simples, funcional e capaz de reproduzir dados pertinentes.

```
typedef struct{
    int nPessoa;
    int prioridade;
    int tempoEsperaP;
    int tempoInicial;
    int distancia;
    int servido;
}grupoPessoas;
```

Optamos por apenas 1 estructura pois segundo o pensamento que seguimos não há qualquer necessidade de implementar outras estructuras pois apenas iriam tornar o código mais complexo e menos eficiente. Sendo assim esta estructura simula os grupos de turistas que irão recorrer aos serviços da empresa.

SEMÁFOROS

Como anteriormente foi indicado, ao todo existem 3 semáforos a controlar os processos da aplicação. Todos os semáforos são semáforos binários (mutex) e cada um está encarregue de uma função diferente.

Semaphore mutexPessoa, mutexStats, mutexTucTuc

Para inicializar os semáforos utilizamos os comandos:

mutexTucTuc = init_sem(1);

mutexPessoa = init_sem(1);

mutexStats = init_sem(1);

Sendo que se tratam todos de semáforos binários o valor inicial é 1 nos três semáforos.

Dado que o escalonador não executa os processos, por ordem, a implementação de semáforos é essencial para contrariar esse problema, pois com a utilização destes semáforos conseguimos controlar de forma rigorosa a forma como os processos vão sendo executados, ditando assim as regras de que processo deve ser executado primeiro permitindo assim que a aplicação seja executada sem quaisquer problemas.

MEMÓRIAS PARTILHADAS

Para que seja possível guardar e partilhar dados entre os processos. , a aplicação possui duas memórias partilhadas. Para que os processos acedam as mesmas são utilizados semáforos que controlam o acesso de modo a permitir apenas um processo a aceder à memória partilhada de cada vez.

As duas memórias partilhadas são:

1) #define SHMKEYFILAESPERA (key_t)0X10

2) #define SHMKEYFILAESPERA (key_t)0X20

1) Esta memória partilhada é utilizada para guardar os grupos de pessoas que vão sendo gerados ao longo do tempo

2) Esta memória partilahda é utilizada para guardar e atualizar todos os dados que serão necessários para as estatisticas e, também, para sinalizar informação aos processos.

PROCESSO-PAI

O processo pai é responsável por mostrar as estatísticas, libertar memória partilhada e semáforos e também está encarregue de perguntar ao utilizador se deseja fazer uma nova simulação.

```
rel_sem(mutexTucTuc);

rel_sem(mutexPessoa);

rel_sem(mutexStats);


shmdt(fila_addr);

shmctl(fila, IPC_RMID, NULL);

shmdt(stats_addr);

shmctl(stats, IPC_RMID, NULL);
```

Estes passos no final da execução da aplicação são criticos pois caso não fossem libertadas as memórias partilhadas e os semáforos ao fim de algumas execuções da aplicação a máquina iria alertar o utilizador de que não é possivel executar a aplicação pois não possui memória suficiente para tal.

PROCESSOS-FILHO

A nossa aplicação por defeito possui 11 processos filhos, um processo que cria grupos de turistas e os outros 10 processos estao encarregues de transportar os grupos para o seu destino. Caso o utilizador assim o desejar pode aumentar ou diminuir o número de processos filhos, isto é, alterar o numero processos encarregues de transportar grupos (Tuc-Tucs).

```
int child_pid[filhos], wait_pid;

    int i, j, child_stat;


    for (i = 0; i < filhos; i++)

    {

        child_pid[i] = fork();

        switch (child_pid[i])

        {

        case -1:

            perror("\e[1;37mO fork falhou \e[0m\n");

            exit(1);

            break;


        case 0:

            if (i == 0) gerador();

            if (i>0) consumidor(i);



            break;
```

```
                default:
                        if (i == (filhos - 1))
                        {
                                for (j = 0; j < filhos; ++j)
                                {
                                        wait_pid = wait(&child_stat);
                                        if (wait_pid == -1)
                                        {
                                                perror("\e[1;37mWait falhou\e[0m");
                                        }
        };
                        };
```

O gerador() funciona como um gerador de grupos de pessoas que continua sempre a criar novos grupos até acabar o tempo previsto de simulação.

O consumidor(i) funciona como se tratasse de um Tuc-Tuc, ou seja, procura um grupo de pessoas a servir e simula o transporte para o seu destino.

OPCOES DE IMPLEMENTAÇÃO

Para o desenvolvimento de uma simulação fiável recorremos a:

- Memórias partilhadas para guardar, atualizar e aceder a dados que necessitam de ser partilhados entre os processos.

- Semáforos, para limitar a entrada nas memórias partilhadas para que sejam evitados os conflitos entre processos ou por exemplo que processos semelhantes trabalhem com dados diferentes.

- Por defeito a simulação tem 11 processos filhos, mas é permitido ao utilizador aumentar ou diminuir o número de Tuc-Tuc's a simular, isto é, é possivel a simulação correr com mais ou menos processos filhos que aqueles que são definidos por defeitos. Decidimos também que os processos filhos iriam colaborar com o processo pai nas estatisticas, ou seja, os processos filhos durante a sua execução irão guardar dados que posteriormente serão tratados pelo processo pai se necessário.

FUNÇÔES IMPORTANTES

Anteriormente, foi referido o funcionamento dos processos-pai e filho e dos semáforos. Vamos agora mencionar e explicar os métodos mais importantes deste programa. Estes métodos são fundamentais ao funcionamento deste programa específico, pois foram desenvolvidos especificamente para funcionar nas circunstâncias do estudo em questão.

Visto que, o nosso objetivo era simular o funcionamento da empresa TuqueTuque, e como consideramos a existência de grupos de turistas, e no nosso estudo escolhemos como “default” uma situação de de uma média de 25 turistas por hora, a aparecer para ser atendidos e viajar no Tuc-Tuc.

Os nossos métodos responsáveis pela criação e trajeto dos turistas são gerarGrupos(), gerador().

Em relação ao funcionamento dos Tuc-Tucss, temos métodos como: consumidor(), grupoParaRetirar() e verificarServidos().

Com o método gerarGrupos() pretendemos gerar grupos de turistas, este grupos não podem ser sempre iguais, daí a utilização da função rand(), para obtermos grupos com um número de turistas que difere de grupo para grupo, e grupos também com diferentes prioridades.

Função *gerarGrupos()*

```
grupoPessoas gerarGrupos() {
srand(time(NULL));
grupoPessoas grupo;
int numero, priori, dist;
int probabilidade = (rand() % 100);
if (probabilidade < 80){
        numero = (rand() % 3) + 3;
}
else{
        numero = (rand() % 2) + 1;
}
```

```
        int probPrioridade = (rand() % 100);
        if (probPrioridade < 90){
                priori = 0;
        }
        else{
                priori = 1;
        }
        dist = (rand() % distanciaMax) + 1;
        if (numero > 2){
                priori = 0;
        }
        grupo.nPessoa = numero;
        grupo.prioridade = priori;
        grupo.tempoEsperaP = 0;
        grupo.tempoInicial = time(0);
        grupo.distancia = dist;
        grupo.servido = 0;
        return grupo;
}
```

Com o método gerador() pretendemos simular a chegada dos turistas e a sua colocação numa fila de espera para atendimento. Pretendemos que esta função pare de executar quando nao houverem mais turistas para atender.

Função gerador()

```
void gerador(){
        /*variaveis necessarias: grupo de turistas gerado,
        variavel qe representa o tempo inicial,
        o tempo atual, o tempo decorrido entre o inicial
        e o atual, o numero de turistas e por fim uma flag*/
```

```
        grupoPessoas gerado;
        int inicio = time(0);
        int agora, tempoDecorrido, flag;
        int tempoDormir;
        int inicioTuristas = time(0);
        flag = 1;
        int turistas;
        int horas = 1;
        turistas = 0; //iniciamos os turistas a 0, pois ainda os vamos gerar
        for (;;){           //ciclo infinito
                agora = time(0);
                tempoDecorrido = difftime(agora, inicio);
 //calculo do tempo decorrido desde o inicio ate ao momento atual
                if (tempoDecorrido < LIFETIME){
                        if (*(stats_ptr + 22) < limiteTuristas){
                                gerado = gerarGrupos();
                                P(mutexPessoa);
                                printf("\e[1;32mApareceu um grupo de %d pessoas, com
prioridade %d na posicao %d \e[0m\n", gerado.nPessoa, gerado.prioridade, in);
                                *(fila_ptr + in) = gerado;
                                *(stats_ptr + 4) = *(stats_ptr + 4) + gerado.nPessoa;
                                *(stats_ptr + 20) = *(stats_ptr + 20) + 1;
                                in = (in + 1) % N;
                                V(mutexPessoa);
                                *(stats_ptr + 22) = *(stats_ptr + 22) + gerado.nPessoa;
                                sleep(3);
                        }
                        else{
```

```
                                        tempoDormir = difftime(agora, inicioTuristas);
                                        tempoDormir = 60 - tempoDormir;
                                        sleep(tempoDormir);
                                        printf("\e[1;37mPassou: %d hora(s) desde o inicio da
simulacao.\n\e[0m", horas);
                                        horas = horas + 1;
                                        inicioTuristas = time(0);
                                        *(stats_ptr + 22) = 0;
                                }
                        }
                        else{
                                if (flag == 1){
                                        printf("\e[1;34mPararam de chegar turistas. \e[0m\n");
                                }
                                flag = 0;
                                *(stats_ptr + 21) = 1;
                                exit(0);
                        }
                }
}
```

A função consumidor() está encarregue de servir os turistas na fila de espera. Se já não existirem mais turistas para servir, a função termina.

Função consumidor()

```
void consumidor(int i){
        int pos, flagTucTuc, switchop;
        grupoPessoas grupo;
        flagTucTuc = 0;
```

```
        for (;;){
                switchop = 0;
                if (flagTucTuc == 1){
                        printf("\e[0;34mO tuc tuc %d voltou \e[0m\n", i);
                        flagTucTuc = 0;
                }
                if (verificarServidos() == 1)
                {
                        printf("\e[1;31mNao existem mais pessoas para servir, o tuc tuc %d vai
para casa \e[0m\n", i);
                        exit(0);
                }
                P(mutexTucTuc);
                tt_contador = tt_contador + 1;
                if (tt_contador == 1){
                        P(mutexPessoa);
                }
                V(mutexTucTuc);
                pos = grupoParaRetirar();
                if (pos >= 0){
                        grupo = *(fila_ptr + pos);
                        if (grupo.nPessoa > 2)
                        {
                                flagTucTuc = 1;
                                grupo.nPessoa = grupo.nPessoa - 2;
                                printf("\e[1;34mO Tuc-Tuc %d ira transportar %d pessoas da
posicao %d a uma distancia de %d \e[0m\n", i, 2, pos, grupo.distancia);
                                *(fila_ptr + pos) = grupo;
                                *(stats_ptr + 5) = *(stats_ptr + 5) + 2;
```

```
                    *(stats_ptr + 3) = *(stats_ptr + 3) + grupo.distancia;
                    switchop = grupo.distancia;
                }
                else{
                    flagTucTuc = 1;
                    printf("\e[1;34mO Tuc-Tuc %d ira transportar %d pessoas da
posicao %d a uma distancia de %d \e[0m\n", i, grupo.nPessoa, pos, grupo.distancia);
                    int tempoAtual = time(0);
                    grupo.tempoEsperaP = difftime(tempoAtual, grupo.tempoInicial);
                    grupo.servido = 1;
                    *(fila_ptr + pos) = grupo;
                    *(stats_ptr + 3) = *(stats_ptr + 3) + grupo.distancia * 2;
                    *(stats_ptr + 5) = *(stats_ptr + 5) + grupo.nPessoa;
                    *(stats_ptr + 2) = *(stats_ptr + 2) + grupo.tempoEsperaP;
                    switchop = grupo.distancia;
                } //Fim else
            }//Fim if(pos >= 0)

            tt_contador = tt_contador - 1;
            if (tt_contador == 0){
                V(mutexPessoa);
            }
            V(mutexTucTuc);
            sleep(1);
            switch (switchop){
            case 1:
                sleep(2);
                break;
```

```
case 2:
        sleep(5);
        break;
case 3:
        sleep(8);
        break;
case 4:
        sleep(11);
        break;
case 5:
        sleep(14);
        break;
case 6:
        sleep(17);
        break;
case 7:
        sleep(20);
        break;
case 8:
        sleep(23);
        break;
case 9:
        sleep(26);
        break;
case 10:
        sleep(29);
        break;
}//Fim Switch
```

```
        }//Fim for

}//Fim consumidor(int i)
```

A função grupoParaRetirar() tem a função de retornar a posição do grupo da fila de espera que deverá ser atendido, ou seja, o método que verifica qual o grupo que deve ser atendido primeiro, considerando a sua prioridade e tempo de espera.

Função grupoParaRetirar()

```
int grupoParaRetirar()
{
        int t;
        int prioriAux, tempoAux, posRetornar, tempoEsperaGrupo;
// variaveis auxiliares para encontrar a posição a remover
        int tempoAtual = time(0);  //variavel que vai buscar o tempo atual
        posRetornar = -1;
        prioriAux = 0;
        tempoAux = 0;
        grupoPessoas grupo;
        P(mutexStats);
        int gruposGerados = *(stats_ptr + 20);
        V(mutexStats);
        for (t = 0; t < gruposGerados; t++){
                grupo = *(fila_ptr + t);
//verificar se existe um grupo nessa posicao
                if (grupo.servido == 0){
                        tempoEsperaGrupo = difftime(tempoAtual, grupo.tempoInicial);//variavel
que guarda o tempo de espera atual do grupo
                        if (grupo.prioridade >= prioriAux){
                                prioriAux = grupo.prioridade;
//caso a prioridade do grupo seja igual ou superior a prioridade do grupo anterior
                                if (tempoEsperaGrupo > tempoAux){
                                        tempoAux = tempoEsperaGrupo;
                                        posRetornar = t;
//guardar posicao do grupo que pode ser o proximo a ser servido
                                }
                        }
                }
        }
        return posRetornar;             //retornar posicao desejada
}
```

A função verificarServidos() verifica se todos os grupos de espera na fila estão servidos.

Função verificarServidos()

```
int verificarServidos(){
        int flag = 0;
        int i;
        if (*(stats_ptr + 21) == 1){
                grupoPessoas grupo;
                int grupos = *(stats_ptr + 20);
                for (i = 0; i < grupos; i++){
                        grupo = *(fila_ptr + i);
                        if (grupo.servido == 1){
                                flag = 1;
                        }
                        else{
                                flag = 0;
                                return flag;
                        }
                }
        }
        return flag;
}
```

LIMITAÇÕES DO PROGRAMA

A limitação da aplicação prende-se pela linguagem onde foi criado a linguagem C. Esta linguagem não é orientada a objeto e como tal não faz uso de classes , objectos, polimorfismo e esconder os dados. Como tal é uma linguagem que nao consegue representar objetos da vida real e que não é segura. Para além disso como o código é todo criado de forma sequencial conforme o código cresce vai se tornando cada vez mais dificil corrigir os bugs que possam surgir.

Como em C não existe uma verificação de tipos de dados é possivel passar um inteiro (int) para uma variavel que deveria ser float.

IMPLEMENTAÇÃO CHAVE

Na nossa opinião a implementação mais importante é os semáforos. Entre as implementações mais importantes, há que considerar o fork(), pois é essencial para a criação de múltiplos processos e memória partilhada. Porém, um elemento fundamental para os múltiplos processos funcionarem com dados iguais, quando falamos de programação, em sistemas Unix, são os semáforos.

Os semáforos, sendo uma das ferramentas usadas para a comunicação entre processos (IPC), tornam-se numa implementação fundamental para coordenar e sincronizar actividades em que multiplos processos competem pelos mesmos recursos, pois permitem ao programador escolher de que forma quer que os processos funcionem fazendo assim uma execução linear e com sentido mesmo que o escalonador não o tenho decidido dessa maneira.

CÓDIGO-FONTE

```
/*
----------Projeto SO----------
---Autores: André Benevides nº120221013; Ana Sequeira nº120221055---
---Docente: Nuno Ribeiro                                    ---
---Disciplina: Sistemas Operativos                          ---
---Tema: Estudo para a empresa "TUQUETUQUE"                                  ---
-------------------------------------------------------------------
-------INDICE - LINHA ONDE SE ENCONTRA OS PROCESSOS E FUNCÕES-------
--- BIBLIOTECAS NECESSARIAS.....21---
--- CONSTANTES UTILIZADAS.....29---
--- ESTRUTURAS UTILIZADAS.....44---
--- DECLARACAO VARIAVEIS.....54---
--- PLACEHOLDER.....55---
--- PLACEHOLDER.....66---
--- PLACEHOLDER.....77---
--- PLACEHOLDER.....88---
--- PLACEHOLDER.....99---
-------------------------------------------------------------------
/*bibliotecas necessarias*/
#include "sema.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <unistd.h>


#define LIFETIME                300         /* max. lifetime of a process */
```

```
#define DEFAULT_VELOCIDADE      40        /* Velocidade Media do tuc tuc */

#define DEFAULT_TUCTUC          10        /* Numero de tuc tuc */

#define DEFAULT_DISTANCIA       5         /* Distancia percorrida */

#define DEFAULT_TEMPOESPERA     7             /* Tempo de espera de uma pessoa */

#define N                                                   150         /* numero de pessoas no array */


#define SHMKEYFILAESPERA (key_t)0X10

#define SHMKEYSTATS (key_t)0X20

/*estrutura que representa um grupo de turistas, considerando as variáveis; nPessoa,com 1 a 5
pessoas, a prioridade do grupo, ou, seja

, se existe algum deficiente ou idoso, na variavel prioridade, o tempo de espera do grupo em
tempoEsperaP, o tempoInicial para o calculo do

tempo em espera, a distancia a qual o grupo se prentende deslocar, de 1 a distancia maxima, e o
estado deste grupo guardado em servido,

que indica se o grupo ja foi atendido ou nao   */

typedef struct{

        int nPessoa;

        int prioridade;

        int tempoEsperaP;

        int tempoInicial;

        int distancia;

        int servido;

}grupoPessoas;


/*declaracao de variaveis*/

int gruposGerados,

filhos,

flag,

tempoEspera,
```

```
nTucTuc,
distanciaMax,
turistasServidos,
distanciaT,
limiteTuristas,
in, out,
tt_contador;
int op, opC;

/*memorias partilhadas*/
int fila;
char *fila_addr;
grupoPessoas *fila_ptr;
int stats;
char *stats_addr;
int *stats_ptr;

/*semaforos*/
semaphore mutexPessoa, mutexTucTuc, mutexStats;

/*metodos declarados*/
int simular();
void inicio();
void configurar();
void configAtual();
void configDefault();
void configDefaultS();
void configPersonalizada();
```

```
void mostrarInfo();

int tempoMaximoEspera();

void calcTempoEspera();

void gerador();

grupoPessoas gerarGrupos();

void consumidor(int i);

int grupoParaRetirar();

void gerador();

void consumidor(int i);

int verificarServidos();

void menuSecundario();

void mudarDistancia();

void mudarTuristasHora();

void mudarTuristasDistancia();


/*metodo main*/

int main(void)

{

        gruposGerados = 1;

        flag = 0;

        inicio();

}


/*

Metodo que consiste na inicializacao das variaveis de memoria,semaforos, criacao de processo-pai
e processo filho.

*/

int simular()
```

```
{
        mutexTucTuc = init_sem(1);
        mutexPessoa = init_sem(1);
        mutexStats = init_sem(1);

        fila = shmget(SHMKEYFILAESPERA, N, 0777 | IPC_CREAT);
        fila_addr = (char*)shmat(fila, 0, 0);
        fila_ptr = (grupoPessoas*)fila_addr;

        stats = shmget(SHMKEYSTATS, N, 0777 | IPC_CREAT);
        stats_addr = (char*)shmat(stats, 0, 0);
        stats_ptr = (int*)stats_addr;

        filhos = nTucTuc + 1;
        *(stats_ptr + 1) = 0;                   //Tempo de espera medio
        *(stats_ptr + 2) = 0;                   //tempo de espera total
        *(stats_ptr + 3) = 0;                   //Distancia percorrida por todos os TTs
        *(stats_ptr + 4) = 0;                   //Nº turistas gerados
        *(stats_ptr + 5) = 0;                   //Nº turistas servidos
        *(stats_ptr + 20) = 0;                  //Grupos gerados
        *(stats_ptr + 21) = 0;                  //flag que sinaliza os consumidores que o gerador
parou
        *(stats_ptr + 22) = 0;                  //contador de turistas

        int child_pid[filhos], wait_pid;
        int i, j, child_stat;
        in = 0;
        out = 0;
```

```
for (i = 0; i < filhos; i++)
{
        child_pid[i] = fork();
        switch (child_pid[i])
        {
        case -1:
                perror("\e[1;37mO fork falhou \e[0m\n");
                exit(1);
                break;

        case 0:
                if (i == 0) gerador();
                if (i>0) consumidor(i);
                break;
        default:
                if (i == (filhos - 1))
                {
                        for (j = 0; j < filhos; ++j)
                        {
                                wait_pid = wait(&child_stat);
                                if (wait_pid == -1)
                                {
                                        perror("\e[1;37mWait falhou\e[0m");
                                };
                        };
                        printf("\e[1;37mFim da simulacao. \e[0m\n");
                        mostrarInfo();
                        rel_sem(mutexTucTuc);
```

```
                                        rel_sem(mutexPessoa);
                                        rel_sem(mutexStats);
                                        shmdt(fila_addr);
                                        shmctl(fila, IPC_RMID, NULL);
                                        shmdt(stats_addr);
                                        shmctl(stats, IPC_RMID, NULL);
                                };
                        }; //Switch
                }; //For
                return 0;
}
/*
Processo que simula a chegada de grupos de turistas e coloca-os numa fila de espera.
O metodo começa por verificar o tempo decorrido desde o inicio da simulaçao e
compara-o com o tempo LIFETIME que foi definido para 300 segundos. Caso o tempo
decorrido seja igual ou superior ao LIFETIME o metodo ira notificar o utilizador,
atraves de um printf, que o nao serao gerados mais grupos de turistas e os consumidores
de que nao irao chegar mais grupos atraves de uma flag guardada em memoria partilhada
*/
void gerador(){
        /*variaveis necessarias: grupo de turistas gerado,
        variavel qe representa o tempo inicial,
        o tempo atual, o tempo decorrido entre o inicial
        e o atual, o numero de turistas e por fim uma flag*/
        grupoPessoas gerado;
        int inicio = time(0);
        int agora, tempoDecorrido, flag;
        int tempoDormir;
```

```
int inicioTuristas = time(0);

flag = 1;

int turistas;

int horas = 1;

turistas = 0; //iniciamos os turistas a 0, pois ainda os vamos gerar

for (;;){		//ciclo infinito

	agora = time(0);

	tempoDecorrido = difftime(agora, inicio); //calculo do tempo decorrido desde o inicio ate ao momento atual

	if (tempoDecorrido < LIFETIME){

		if (*(stats_ptr + 22) < limiteTuristas){

			gerado = gerarGrupos();

			P(mutexPessoa);

			printf("\e[1;32mApareceu um grupo de %d pessoas, com prioridade %d na posicao %d \e[0m\n", gerado.nPessoa, gerado.prioridade, in);

			*(fila_ptr + in) = gerado;

			*(stats_ptr + 4) = *(stats_ptr + 4) + gerado.nPessoa;

			*(stats_ptr + 20) = *(stats_ptr + 20) + 1;

			in = (in + 1) % N;

			V(mutexPessoa);

			*(stats_ptr + 22) = *(stats_ptr + 22) + gerado.nPessoa;

			sleep(3);

		}

		else{

			tempoDormir = difftime(agora, inicioTuristas);

			tempoDormir = 60 - tempoDormir;

			sleep(tempoDormir);

			printf("\e[1;37mPassou: %d hora(s) desde o inicio da simulacao.\n\e[0m", horas);
```

```
                        horas = horas + 1;
                        inicioTuristas = time(0);
                        *(stats_ptr + 22) = 0;
                    }
                }
                else{
                    if (flag == 1){
                        printf("\e[1;34mPararam de chegar turistas. \e[0m\n");
                    }
                    flag = 0;
                    *(stats_ptr + 21) = 1;
                    exit(0);
                }
            }
        }
        /*
        Processo que vai servir os grupos de pessoas em lista de espera.
        O metodo começa por entrar num for infinito em que primeiro
        verifica se a flag flagTucTuc encontra-se com o valor 1,
        caso seja 1 o metodo ira imprimir a frase "O tuc tuc voltou" e
        volta a colocar o valor da flagTucTuc a 0. Em seguida o metodo
        verifica se ainda existem turistas por servir, caso nao existam
        o metodo imprime a frase "Nao existem mais pessoas para servir,
        o tuc tuc x vai para casa" e executa o comando exit(0) para fechar
        o processo. Caso haja turistas por servir o metodo continua o seu codigo.
        O metodo para saber que grupo de turistas tem de servir a seguir
        chama o metodo grupoParaRetirar(), que retorna um inteiro que indica
        em que posicao se encontra o grupo que deve ser servido a seguir.
```

Em seguida o metodo verifica quantas pessoas compoem o grupo, caso sejam mais de 2 pessoas o metodo ira decrementar 2 no numero desse grupo e adicionar dados as estasticas(nº de turistas servidos e distancia percorrida), caso sejam 2 ou menos turistas o metodo ira executar as açoes que foram mencionadas anteriormente e tambem ira mudar a flag "servido" para 1 e calcular o tempo de espera do grupo recorrendo ao difftime(time_t, time_t)
Para finalizar uma iteracao no ciclo infinito o metodo, a partir do atributo distancia dos grupos, ira calcular por quantos segundos devera suspender o processo
*/

```
void consumidor(int i){

        int pos, flagTucTuc, switchop;
        grupoPessoas grupo;
        flagTucTuc = 0;
        for (;;){
                switchop = 0;
                if (flagTucTuc == 1){
                        printf("\e[0;34mO tuc tuc %d voltou \e[0m\n", i);
                        flagTucTuc = 0;
                }
                if (verificarServidos() == 1)
                {
                        printf("\e[1;31mNao existem mais pessoas para servir, o tuc tuc %d vai para casa \e[0m  \n", i);
                        exit(0);
                }
                P(mutexTucTuc);
                tt_contador = tt_contador + 1;
```

```
                if (tt_contador == 1){
                        P(mutexPessoa);
                }
                V(mutexTucTuc);
                pos = grupoParaRetirar();
                if (pos >= 0){
                        grupo = *(fila_ptr + pos);
                        if (grupo.nPessoa > 2)
                        {
                                flagTucTuc = 1;
                                grupo.nPessoa = grupo.nPessoa - 2;
                                printf("\e[1;34mO Tuc-Tuc %d ira transportar %d pessoas da
posicao %d a uma distancia de %d \e[0m\n", i, 2, pos, grupo.distancia);
                                *(fila_ptr + pos) = grupo;
                                *(stats_ptr + 5) = *(stats_ptr + 5) + 2;
                                *(stats_ptr + 3) = *(stats_ptr + 3) + grupo.distancia;
                                switchop = grupo.distancia;
                        }
                        else{
                                flagTucTuc = 1;
                                printf("\e[1;34mO Tuc-Tuc %d ira transportar %d pessoas da
posicao %d a uma distancia de %d \e[0m\n", i, grupo.nPessoa, pos, grupo.distancia);
                                int tempoAtual = time(0);
                                grupo.tempoEsperaP = difftime(tempoAtual, grupo.tempoInicial);
                                grupo.servido = 1;
                                *(fila_ptr + pos) = grupo;
                                *(stats_ptr + 3) = *(stats_ptr + 3) + grupo.distancia * 2;
                                *(stats_ptr + 5) = *(stats_ptr + 5) + grupo.nPessoa;
                                *(stats_ptr + 2) = *(stats_ptr + 2) + grupo.tempoEsperaP;
```

```
                    switchop = grupo.distancia;
            } //Fim else
}//Fim if(pos >= 0)

tt_contador = tt_contador - 1;
if (tt_contador == 0){
            V(mutexPessoa);
}
V(mutexTucTuc);
sleep(1);
switch (switchop){
case 1:
            sleep(2);
            break;
case 2:
            sleep(5);
            break;
case 3:
            sleep(8);
            break;
case 4:
            sleep(11);
            break;
case 5:
            sleep(14);
            break;

case 6:
```

```
                    sleep(17);
                    break;
            case 7:
                    sleep(20);
                    break;
            case 8:
                    sleep(23);
                    break;
            case 9:
                    sleep(26);
                    break;
            case 10:
                    sleep(29);
                    break;
            }//Fim Switch
        }//Fim for
}//Fim consumidor(int i)
/*
Metodo que gera e retorna um grupo com valores aleatorios
para a: prioridade, numero de turistas e distancia que
pretende viajar recorrendo a funcao rand(void).
Os restantes atributos, tempoEsperaP, tempoInicial
e servido sao inicializados com os valores: 0, time(0)
e 0 respectivamente.
*/
grupoPessoas gerarGrupos()
{
        srand(time(NULL));
```

```
grupoPessoas grupo;
int numero, priori, dist;
int probabilidade = (rand() % 100);
if (probabilidade < 80){
        numero = (rand() % 3) + 3;
}
else{
        numero = (rand() % 2) + 1;
}

int probPrioridade = (rand() % 100);
if (probPrioridade < 90){
        priori = 0;
}
else{
        priori = 1;
}
dist = (rand() % distanciaMax) + 1;
if (numero > 2){
        priori = 0;
}

grupo.nPessoa = numero;
grupo.prioridade = priori;
grupo.tempoEsperaP = 0;
grupo.tempoInicial = time(0);
grupo.distancia = dist;
grupo.servido = 0;
```

```
	return grupo;
}
/*
Metodo que percorre a memoria partilhada *(fila_ptr)
que representa a fila de espera para encontrar
qual o proximo grupo que deve ser servido pelo
Tuc Tuc ( consumidor() ).
O metodo ira recorrer a um ciclo for que inicia na
posicao 0 e acaba na posicao gruposGerados, que é
obtido a partir da memoria partilhada das estatisticas
*(stats_ptr) e quando encontrar um grupo, que pareca ser
o grupo com mais prioridade e que se encontra em espera
a mais tempo, ira colocar a posicao do mesmo na variavel
posRetornar. Assim que o metodo percorrer a fila de espera
toda ira retornar o valor da variavel posRetornar
*/
int grupoParaRetirar()
{
	int t;
	int prioriAux, tempoAux, posRetornar, tempoEsperaGrupo;		/*variaveis
auxiliares para encontrar a posição a remover*/
	int tempoAtual = time(0);
		/*variavel que vai buscar o tempo atual*/
	posRetornar = -1;
	prioriAux = 0;
	tempoAux = 0;
	grupoPessoas grupo;
	P(mutexStats);
```

```
	int gruposGerados = *(stats_ptr + 20);

	V(mutexStats);

	for (t = 0; t < gruposGerados; t++){

		grupo = *(fila_ptr + t);				/*verificar se existe um grupo nessa posicao*/

		if (grupo.servido == 0){

			tempoEsperaGrupo = difftime(tempoAtual, grupo.tempoInicial); /*variavel que guarda o tempo de espera atual do grupo*/

			if (grupo.prioridade >= prioriAux){

				prioriAux = grupo.prioridade;		/*caso a prioridade do grupo seja igual ou superior a prioridade do grupo anterior*/

				if (tempoEsperaGrupo > tempoAux){

					tempoAux = tempoEsperaGrupo;

					posRetornar = t;		/*guardar posicao do grupo que pode ser o proximo a ser servido*/

				}

			}

		}

	}	/*quando o metodo acabar de percorrer a fila de espera*/

	return posRetornar;					/*retornar posicao desejada*/

}

/*

Metodo que mostra uma interface compreensivel para

um utilizador com conhecidos basicos de informatica,

sendo que este e apresentado com 4 opcoes

1- Iniciar Simulacao

2- Configurar Simulacao

3- Configuracao Atual

0 - Sair
```

```
Caso o utilizador decida recorrer a primeira opcao
sem ter primeiro ido a segunda opcao(Configurar Simulacao)
a simulacao ira ser iniciada com os valores standard
10 - Tuc-Tuc  ( consumidor () )
25 - Turistas por hora
5- - Distancia maxima que podem viajar
*/
void inicio()
{
        printf("\e[1;33m*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("***************      Bem Vindo a simulacao     **************\n");
        printf("*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("***************   Iniciar Simulacao    - 1  *****************\n");
        printf("*****************************************************************\n");
        printf("***************   Configurar Simulacao - 2  *****************\n");
        printf("*****************************************************************\n");
        printf("***************   Configuracao Atual   - 3  *****************\n");
        printf("*****************************************************************\n");
        printf("***************   Sair                 - 0  *****************\n");
        printf("*****************************************************************\n");
        printf("*****************************************************************\e[0m\n");
        do
        {
                printf("\e[1;33mOpcao: \e[0m");
                scanf("%d", &op);
        } while (op > 3);
```

```
        switch (op)
        {
        case 1:
                if (flag == 0){                         /* Se a configuracao nao foi alterada, usar
configuracao default */
                        configDefaultS();               /* n.TucTucs=10 Distancia maxima=5km
Tempo de espera aceitavel=7min */
                }
                simular();
                break;

        case 2:
                configurar();
                break;
        case 3:
                configAtual();
                break;

        case 0:
                exit(0);
                break;
        }
}
/*
Metodo que mostra uma interface compreensivel para
um utilizador com conhecidos basicos de informatica,
sendo que este e apresentado com 4 opcoes
1- Usar Valores Default
```

2- Valores Personalizados

3- Simulacoes Possiveis

0 - Voltar

Caso o utilizador opte pela opcao 1 a simulacao ira usar

os valores standard 10 - Tuc-Tuc (consumidor ())

25 - Turistas por hora

5- - Distancia maxima que podem viajar.

Caso opte pela opcao 2 a simulacao ira pedir ao utilizador

para inserir os dados que desejar nos campos numero Tuc Tuc,

turistas por hora e distancia maxima.

Caso opte pela opcao 3 este sera apresentado a um novo menu

com 3 opcoes de simulacoes diferentes.

Caso opte pela opcao 0 este sera levado de volta ao menu inicial

*/

void configurar(){

```
        printf("\e[1;33m*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("****************      Menu Configuração      ****************\n");
        printf("*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("****************   Usar Valores Default   - 1  **************\n");
        printf("*****************************************************************\n");
        printf("****************   Valores Personalizados - 2  **************\n");
        printf("*****************************************************************\n");
        printf("****************    Simulacoes Possiveis   - 3  **************\n");
        printf("*****************************************************************\n");
        printf("****************   Voltar                 - 0  **************\n");
```

```
        printf("*****************************************************************\n");
        printf("*****************************************************************\e[0m\n");
        int opC;
        do
        {
                printf("\e[1;33mOpcao: \e[0m");
                scanf("%d", &opC);
        } while (opC < 0 || opC>3);
        switch (opC)
        {
        case 1:
                configDefault();
                break;

        case 2:
                configPersonalizada();
                break;

        case 3:
                menuSecundario();
                break;

        case 0:
                inicio(0);
                break;
        }
}
/*
```

```
Metodo que mostra uma interface compreensivel para
um utilizador com conhecidos basicos de informatica,
sendo que este e apresentado com 4 opcoes
1 - Distancia Maxima = 10
2 - VTuristas/Hora = 30
3 - Turistas/Hora = 30 e Distancia Max = 10
0 - Voltar
*/
void menuSecundario(){
        int opcao;

        printf("\e[1;33m*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("****************      Menu Configuracao      ****************\n");
        printf("*****************************************************************\n");
        printf("*****************************************************************\n");
        printf("***************    Distancia Maxima = 10   - 1  *************\n");
        printf("*****************************************************************\n");
        printf("***************    Turistas/Hora = 30      - 2  *************\n");
        printf("*****************************************************************\n");
        printf("***************    Turistas/Hora = 30      - 3  *************\n");
        printf("***************    Distancia Max = 10          *************\n");
        printf("*****************************************************************\n");
        printf("***************    Voltar                  - 0  *************\n");
        printf("*****************************************************************\n");
        printf("*****************************************************************\e[0m\n");
        do{
                printf("\e[1;33mOpcao: \e[0m");
```

```
            scanf("%d", &opcao);
        } while (opcao > 3 || opcao < 0);

        switch (opcao){

        case 1:
            mudarDistancia();
            break;

        case 2:
            mudarTuristasHora();
            break;

        case 3:
            mudarTuristasDistancia();
            break;

        case 0:
            inicio();
            break;

        default:
            printf("\e[1;33mIntroduza uma opcao valida (exemplo: 1,2,3 ou 0):\e[0m\n");
            break;

        }
}
/*
```

```
Metodo que ira mostrar ao utilizador uma lista

das estatisticas retiradas da simulacao.

O metodo ira mostrar o:

Tempo de espera max

Tempo de espera total

Distancia total

Tempo media de espera

Turistas servidos

Turistas gerados

Tempo simulacao

*/

void mostrarInfo()

{

        int a;

        int b;

        int calculo;

        calculo = 7 * nTucTuc;

        calculo = (int)calculo / tempoMaximoEspera();

        calculo = calculo + nTucTuc;

        a = (*(stats_ptr + 2));

        b = (*(stats_ptr + 20));

        printf("\e[1;33m************************************************************\n");

        printf("************************************************************\n");

        printf("******************       Estatisticas      ******************\n");

        printf("************************************************************\n");

        printf("************************************************************\n");

        printf("***** Tempo de espera max.:  %d min          **************\n",
tempoMaximoEspera());
```

```
        printf("*****************************************************************\n");
        printf("***** Tempo de espera total: %d min          **************\n", *(stats_ptr + 2));
        printf("*****************************************************************\n");
        printf("***** Distancia Total:       %d km           **************\n", *(stats_ptr + 3));
        printf("*****************************************************************\n");
        printf("***** Tempo medio de espera: %d min          **************\n", (a / b));
        printf("*****************************************************************\n");
        printf("***** Turistas servidos:     %d              **************\n", *(stats_ptr + 5));
        printf("*****************************************************************\n");
        printf("***** Turistas gerados:      %d              **************\n", *(stats_ptr + 4));
        printf("*****************************************************************\n");
        printf("***** Tempo Simulacao:       %d              **************\n", LIFETIME);
        printf("*****************************************************************\n");
        printf("*****************************************************************\e[0m\n");
}
/*
Metodo que mostra a configuracao atual da simulacao ao utilizador
*/
void configAtual()
{
        printf("\n \e[1;33m** Configuracao Actual Da Simulacao ** : \n");
        printf("Numero de TucTucs: %d\n", nTucTuc);
        printf("Distancia maxima: %d\n", distanciaMax);
        printf("Tempo media de espera aceitavel: %d \e[0m\n", tempoEspera);
        inicio();
}
/*
Metodo que introduz os valores default da simulacao
```

```
*/
void configDefault()
{
	flag = 0;
	nTucTuc = DEFAULT_TUCTUC;
	distanciaMax = DEFAULT_DISTANCIA;
	tempoEspera = DEFAULT_TEMPOESPERA;
	printf("\e[1;37mA definir parametros standard...\n");
	printf("Numero TucTucs -> 10 \n Distancia maxima -> 5km \n Tempo de espera ->
7min ...\e[0m");
	limiteTuristas = 25;
	inicio();
}
/*
Metodo que introduz os valores default da simulacao e que e
chamado caso a simulacao seja iniciada sem terem sido introduzidos
quaisquer dados
*/
void configDefaultS()
{
	flag = 0;
	nTucTuc = DEFAULT_TUCTUC;
	distanciaMax = DEFAULT_DISTANCIA;
	tempoEspera = DEFAULT_TEMPOESPERA;
	limiteTuristas = 25;
}
/*
Metodo que permite ao utilizador definir os valores da simulacao e que e
```

```
chamado caso a simulacao seja iniciada sem terem sido introduzidos
quaisquer dados
*/
void configPersonalizada()
{
	flag = 1;
	do{
		printf("\e[1;33mIntroduza o número de Tuc Tuc's: \e[0m");
		scanf("%d", &nTucTuc);
	} while (nTucTuc < 1 || nTucTuc>100);

	do{
		printf("\e[1;33mIntroduza o limite de turistas a simular por hora: \e[0m");
		scanf("%d", &limiteTuristas);
	} while (limiteTuristas < 25 || limiteTuristas>100);

	do{
		printf("\e[1;33mIntroduza a distância que um Tuc Tuc pode percorrer: \e[0m");
		scanf("%d", &distanciaMax);
	} while (distanciaMax < 1 || distanciaMax>10);
	inicio();
}
/*
Metodo que percorre a memoria partilhada que
guarda os grupos de turistas *(fila_ptr) para
encontrar o grupo que estou mais tempo para ser
servido.
Retorna o maior tempo de espera que encontrar
```

```
*/
int tempoMaximoEspera()
{
	int i;
	int aux = 0;
	grupoPessoas grp;
	for (i = 0; i<*(stats_ptr + 20); i++){
		grp = *(fila_ptr + i);
		if (grp.tempoEsperaP>aux){
			aux = grp.tempoEsperaP;
		}
	}
	return aux;
}
void calcTempoEspera()
{
	int gruposG = 0;
	grupoPessoas grupoP;
	gruposG = *(stats_ptr + 20);
	int i;
	for (i = 0; i < gruposG; i++)
	{
		grupoP = *(fila_ptr + i);
		if (grupoP.servido == 0){
			grupoP.tempoEsperaP = difftime(time(0), grupoP.tempoInicial);
			*(stats_ptr + 1) = *(stats_ptr + 1) + grupoP.tempoEsperaP;
			*(fila_ptr + i) = grupoP;
		}
```

```
        }
}
/*Metodo produtor
gera os grupos de turistas, evocando o metodo gerarGrupos()
, insere os na memória, sendo no nosso contexto pratico,
a fila de atendimento, utilizando o auxilio de semaforos*/
int verificarServidos(){
        int flag = 0;
        int i;
        if (*(stats_ptr + 21) == 1){
                grupoPessoas grupo;
                int grupos = *(stats_ptr + 20);
                for (i = 0; i < grupos; i++){
                        grupo = *(fila_ptr + i);
                        if (grupo.servido == 1){
                                flag = 1;
                        }
                        else{
                                flag = 0;
                                return flag;
                        }
                }
        }
        return flag;
}
/*
Metodo que muda a distancia maxima para 10km
mantendo o numero de tuc tuc's e limite de turistas
```

```
a 10 e 25 respectivamente
*/
void mudarDistancia()
{
        flag = 1;
        nTucTuc = 10;
        limiteTuristas = 25;
        distanciaMax = 10;
        inicio();
}
/*
Metodo que muda o numero de turistas por hora
para 30 mantendo o numero de tuc tuc's e
a distancia a 10 e 5 respectivamente
*/
void mudarTuristasHora()
{
        flag = 1;
        nTucTuc = 10;
        distanciaMax = 5;
        limiteTuristas = 30;
        inicio();
}
/*
Metodo que muda o numero de turistas por hora
para 30 e a distancia para 10 mantendo o
numero de tuc tuc's a  10;
*/
```

```
void mudarTuristasDistancia()
{
	flag = 1;
	nTucTuc = 10;
	distanciaMax = 10;
	limiteTuristas = 30;
	inicio();
}
```