My Project

Generated by Doxygen 1.9.5

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 myqueue Struct Reference

```
#include <fila_circular.h>
```

Data Fields

- float ∗ queue
- unsigned int capacity
- int front
- int rear

3.1.1 Field Documentation

3.1.1.1 capacity

```
unsigned int capacity
```

present queue capacity

3.1.1.2 front

```
int front
```

front of the queue -1 if empty

3.1.1.3 queue

```
float* queue
```

pointer to queue block of size capacity

3.1.1.4 rear

```
int rear
```

rear of the queue -1 if empty

The documentation for this struct was generated from the following file:

- fila_circular.h

3.2 QueueSystem Struct Reference

Data Fields

- float sim_time
- float time_last_event
- int server_status [12]
- float time_next_event [12]
- float mean_service
- int num_servers
- int lost_customers
- float mean_interarrival
- float area_num_in_q
- int num_custs_delayed
- float total_of_delays
- int num_events
- float time_arrival [Q_LIMIT]
- int area_server_status [12]
- int next_event_type
- int idseq [12]
- int num_delays_required
- int with_queue
- myQueue circularQueue

3.2.1 Field Documentation

3.2.1.1 area_num_in_q

```
float area_num_in_q
```

Média do tempo entre chegadas dos clientes

3.2.1.2 area_server_status

```
int area_server_status[12]
```

Tabela que armazena os tempos de chegada dos clientes na fila usando with_queue=1 Variaveis-Estatisticas

3.2.1.3 circularQueue

```
myQueue circularQueue
```

Indica se há fila (1-sim, 0-não)

3.2.1.4 idseq

```
int idseq[12]
```

3.2.1.5 lost_customers

```
int lost_customers
```

Nº total de servidores no sistema Variaveis-Clientes

3.2.1.6 mean_interarrival

```
float mean_interarrival
```

conta o nºde clientes perdidos

3.2.1.7 mean_service

```
float mean_service
```

Tabela que armazena o tempo do prox servidor

3.2.1.8 next_event_type

```
int next_event_type
```

Tabela que armazena a area 0 - a lista de eventos esta vazia; 1 é uma chegada; 2 ate num_srvers+1 é uma partida desse servidor

3.2.1.9 num_custs_delayed

```
int num_custs_delayed
```

Aramezena a area do nº de clientes na fila

3.2.1.10 num_delays_required

```
int num_delays_required
```

Tabela que aramazena os indentificadores de sequencia para a geração de variaveis aleatorias

3.2.1.11 num_events

```
int num_events
```

soma de total de atrasos de todos os clientes perdidos

3.2.1.12 num_servers

```
int num_servers
```

media de tempo de serviço

3.2.1.13 server_status

```
int server_status[12]
```

armazena o tempo do ultimo evento ocorrido Variaveis-Servidor

3.2.1.14 sim_time

```
float sim_time
```

Variaveis-TEMPO

3.2.1.15 time_arrival

```
float time_arrival[Q_LIMIT]
```

Nº de eventos, é dinamico pois depende dos clientes

3.2.1.16 time_last_event

```
float time_last_event
```

tempo atual da simulação

3.2.1.17 time_next_event

```
float time_next_event[12]
```

Tabela que indica o estado de cada servidor (IDLE ou BUSY)

3.2.1.18 total_of_delays

```
float total_of_delays
```

conta o nºde clientes que foram atendidos

3.2.1.19 with_queue

```
int with_queue
```

Nº total de clientes que devem ser atendidos antes da execução do codigo terminar

The documentation for this struct was generated from the following file:

- fila1s.c

Chapter 4

File Documentation

4.1 fila1s.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "lcgrand.h"
#include "fila_circular.h"
```

Data Structures

- struct QueueSystem

Macros

- #define Q_LIMIT 200
- #define BUSY 1
- #define IDLE 0
- #define INFINITO 1e+30

Functions

- float expon (float mean, int stream)
- int selectFreeServer (const QueueSystem ∗system)
- void initialize (QueueSystem ∗system)
- void report (const QueueSystem ∗system, FILE ∗outfile)
- void update_time_avg_stats (QueueSystem ∗system)
- void timing (QueueSystem ∗system)
- void arrive (QueueSystem ∗system)
- void depart (QueueSystem ∗system)
- int main ()

4.1.1 Macro Definition Documentation

4.1.1.1 BUSY

```
#define BUSY 1
```

4.1.1.2 IDLE

```
#define IDLE 0
```

4.1.1.3 INFINITO

```
#define INFINITO 1e+30
```

4.1.1.4 Q_LIMIT

```
#define Q_LIMIT 200
```

4.1.2 Function Documentation

4.1.2.1 arrive()

```
void arrive (
             QueueSystem * system )
```

idseq[1] está o indentificadir da sequencia usada para as chegadas

Verificar se o servidor está ocupado

Usando a função select_server que verifica se existe um servidor não ocupado se o resultado for -1 então os servidotres estão todos ocupados logo o cliente não é logo atendido if (select_server(server_status, ∗num_servers) == -1) {

Se ele estiver ocupado adiciona mais um cliente à fila circular

incrementa o numero de chamadas perdidas

Servidor estiver Disponivel o atraso do cliente começa a zero seleciona o servidor disponivel se o servidor que for ocupado, passa a estar ocupado

4.1.2.2 depart()

```
void depart (
            QueueSystem * system )
```

função responsavel por processar o cliente que sai da fila Teste Debug– Mensagens de ERRO

variavel auxiliar

Verifica se a fila esta vazia

fila vazia ou sistema com perda

se tem perda acabou (não faz mais nada)

Se não há clientes à espera significa q o servidor concluiu o serviço
passa a estar desocupado

vem para aqui se for um sistema com fila e esta não estiver vazia

int inf= para testar se correu bem

front_time corresponde ao time_arrival[1] na fila não circular

calcula o atraso do cliente q esta a iniciar o serviço e atualiza o contador de atraso

aumenta o numero de clientes atrasados e agenda a saida

antes era sempre 2 agora é do 2 ate ao num_server+1

4.1.2.3 expon()

```
float expon (
            float mean,
            int stream )
```

Retorna uma variavel aleatoria exponencial com media "mean" Função para gerar uma variavel exponencial

4.1.2.4 initialize()

```
void initialize (
            QueueSystem * system )
```

a tabela server_status só é usada no indice 2 até num_servers+1 (se o numero de servidores for 1 só é usado o indice 2) time_next_event indice zero indica que não há mais eventos indice 1 indica uma chegada indice 2 até num_servers+1 indica uma partida nesse servidor se o time_next_event nesse indice for diferente de infinito inicializa as variaveis com valores iniciais

tempo de simulação começa a zero

os clientes na fila começam a zero

tempo do ultimo evento começa a zero

numero de clientes atendidos começa a zero

o total de atrasos começa a zero

Atualização do número de eventos

Inicializa a fila circular

estado inicial dos servidores como IDLE

isto são as partidas

inicializa as sementes a usar para chegadas (1) e tempos de serviço (2 ate num_servers+1) zero n é usado

gera a primeira chegada

4.1.2.5 main()

```
int main ( )
```

zero n é usado a capacidade minima deve de ser smp num_serveres+1 uma vez que vamos precisar de ter eventos por servidor mais um para as chegadas, podemos usar um truque seguinte: time_next_event[0... numservers -1] tem o instante de partida desse servidor ou um valor muito elevado "infinito " (chamamos uma constante omeuinfinito=biblioteca-->climits=floats=FLT_MAX (incluir biblioteca)) time_next_event[numserveers] passa ser a chegada seguinte

Modificacoes para lidar com os multiplos servidores

Inicilaiza as variaveis do sistema

Vamos usar um while para percorrer todos os clientes atendidos ate atingir o numero de clientes pretendidos

Verificar se há eventos agendados

se o proximo evento for uma chegada

se o proximo evento for uma saida

se n for uma saida ou uma chegada então ERRO

Gera o relatorio final

Fecha os ficheiros

4.1.2.6 report()

```
void report (
            const QueueSystem * system,
            FILE * outfile )
```

relatorio do desempenho do sistema de fila

imprime a media do atraso na fila por cliente

imprime a media do numero de clientes na fila

imprime a utilização media dos servidores

imprime o tempo total da simulação.

imprime o numero de clientes perdidos

4.1.2.7 selectFreeServer()

```
int selectFreeServer (
            const QueueSystem * system )
```

função para selecionar um servidor disponivel o estado de cada servidor ou IDLE ou BUSY

Returns

devolve o primeiro servidor livre ou -1 se tiverem todos ocupados

percorrre a lista de servidores

server_status–um array que indica on estado de cada servidor (IDLE ou BUSY) num_servers–numero total de servidores no sistema

retorna o indice do primeiro servidor diponivel

retorna -1 se todos os servidores estiverem ocupados

4.1.2.8 timing()

```
void timing (
            QueueSystem * system )
```

determinar o prox evento e avanca o tempo da simulacao

experimentar depois só com infinito

caso em que a fila está vazia

identifica o servidor a libertar para saber na função depart (é o que tem menor tempo)

fprintf(stderr, "\nTimming: system->next_event_type==%d", system->next_event_type);

if (system->server_status[system->next_event_type]>=2){ fprintf(stderr, "\nTimming: system->server_↩ status[system->next_event_type]==%d", system->server_status[system->next_event_type]); }

Verifica se a lista de eventos esta vazia.

A lista de eventos está vazia, então a simulação para.

Se a lista de eventos nao esta vazia então a simulacao para

avanca o tempo da simulacao para o tempo do prox evento

Para calcular e atualizar as estatisticas de tempo medio relacionadas ao nº de clientes na fila e estado dos servidores

4.1.2.9 update_time_avg_stats()

```
void update_time_avg_stats (
            QueueSystem * system )
```

Função que atualiza ass estatisticas do codigo tempo decorrido desde do ultimo evento

atualiza o tempo do ultimo evento para o tempo atual

se o sistema tem fila de espera atualiza a area da função number_queue permite calcular o n de utilizadores na fila de espera

atualiza a area media ponderada do numero de clientes na fila para cada servidor

4.2 fila_circular.c File Reference

```
#include "fila_circular.h"
```

Functions

- void inicQueue (myQueue ∗q)

    create empty queue of capacity zero
- int getCapacity (const myQueue ∗q)

    To get the queue capacity.
- int getSize (const myQueue ∗q)

    To get the queue ocupancy.
- int resizeQueue (myQueue ∗q)

    if zero becomes CAPACITY else double queue capacity
- int checkFull (const myQueue ∗q)

    Here we check if the Circular queue is full or not.
- int checkEmpty (const myQueue ∗q)

    Here we check if the Circular queue is empty or not.
- int enQueue (myQueue ∗q, float value)

    Addition in the Circular Queue.
- int deQueue (myQueue ∗q, float ∗value)

    Removal from the Circular Queue.
- void printQueue (const myQueue ∗q)

    Display the queue.
- void freeQueue (myQueue ∗q)

    free memory alocated to the queue This queue returns to inic state

4.2.1 Function Documentation

4.2.1.1 checkEmpty()

```
int checkEmpty (
            const myQueue * q )
```

Here we check if the Circular queue is empty or not.

Returns

Information about the queue: 1 is the queue is empty, 0 otherwise

< queue is empty

< queue is noy empty

4.2.1.2 checkFull()

```
int checkFull (
            const myQueue * q )
```

Here we check if the Circular queue is full or not.

Returns

Information about the queue: 1 is the queue is full, 0 otherwise

< queue is full due to ZERO capacity

< queue is full

< queue is not full

4.2.1.3 deQueue()

```
int deQueue (
            myQueue * q,
            float * value )
```

Removal from the Circular Queue.

Parameters

out	*value*	of the element that was at the head or the queue.

Returns

-1 try to deque on a empty queue

+1 if successfull

4.2.1.4 enQueue()

```
int enQueue (
            myQueue * q,
            float value )
```

Addition in the Circular Queue.

Parameters

	q	where value will be added to the rear
in	*value*	to be added to the queue

Returns

Information about succes (+1) or not, due to full queue and then failure of realloc (-1)

< By omission, it will enQueue

< Did not enQueue, due failing to resize

< Will enQueue, after successuful resize

< It will enQueue

< Status of enQueue operation, in cicular fashion

4.2.1.5 freeQueue()

```
void freeQueue (
            myQueue * q )
```

free memory alocated to the queue This queue returns to inic state

< all space was released

< the queue is now NULL for consistency

< capacity is ZERO

< queue is iniatlly empty

4.2.1.6 getCapacity()

```
int getCapacity (
            const myQueue * q )
```

To get the queue capacity.

number of elements in queue

Returns

queue capacity

4.2.1.7 getSize()

```
int getSize (
             const myQueue * q )
```

To get the queue ocupancy.

number of elements in queue

Returns

number of elements in the queue

< queue is empty

< queue has one element

< queue is full

4.2.1.8 inicQueue()

```
void inicQueue (
             myQueue * q )
```

create empty queue of capacity zero

< ZERO capacity

< the queue is now NULL for consistency

< queue is iniatlly empty

4.2.1.9 printQueue()

```
void printQueue (
             const myQueue * q )
```

Display the queue.

4.2.1.10 resizeQueue()

```
int resizeQueue (
             myQueue * q )
```

if zero becomes CAPACITY else double queue capacity

Returns

0 if the queue was allocated succesfully, -1 if unable to resize queue

< If no capacity reserves base capacity

< base capacity

< unable to resize queue

< queue was allocated succesfully

< doubles previous capacity

< unable to resize queue

< queue was resized succesfully

4.3 fila_circular.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

Data Structures

- struct myqueue

Macros

- #define CAPACITY 2

Typedefs

- typedef struct myqueue myQueue

Functions

- void inicQueue (myQueue ∗q)

  create empty queue of capacity zero
- int getCapacity (const myQueue ∗q)

  number of elements in queue
- int getSize (const myQueue ∗q)

  number of elements in queue
- int resizeQueue (myQueue ∗q)

  if zero becomes CAPACITY else double queue capacity
- int checkFull (const myQueue ∗q)

  Here we check if the Circular queue is full or not.
- int checkEmpty (const myQueue ∗q)

  Here we check if the Circular queue is empty or not.
- int enQueue (myQueue ∗q, float value)

  Addition in the Circular Queue.
- int deQueue (myQueue ∗q, float ∗value)

  Removal from the Circular Queue.
- void printQueue (const myQueue ∗q)

  Display the queue.
- void freeQueue (myQueue ∗q)

  free memory alocated to the queue This queue returns to inic state

4.3.1 Macro Definition Documentation

4.3.1.1 CAPACITY

```
#define CAPACITY 2
```

uncoment (comment) definition of DEBUG e debug messafes are (NOT) desired

4.3.2 Typedef Documentation

4.3.2.1 myQueue

```
typedef struct myqueue myQueue
```

4.3.3 Function Documentation

4.3.3.1 checkEmpty()

```
int checkEmpty (
            const myQueue * q )
```

Here we check if the Circular queue is empty or not.

Returns

Information about the queue: 1 is the queue is empty, 0 otherwise

< queue is empty

< queue is noy empty

4.3.3.2 checkFull()

```
int checkFull (
            const myQueue * q )
```

Here we check if the Circular queue is full or not.

Returns

Information about the queue: 1 is the queue is full, 0 otherwise

< queue is full due to ZERO capacity

< queue is full

< queue is not full

4.3.3.3 deQueue()

```
int deQueue (
            myQueue * q,
            float * value )
```

Removal from the Circular Queue.

Parameters

out	*value*	of the element that was at the head or the queue.

Returns

-1 try to deque on a empty queue

+1 if successfull

4.3.3.4 enQueue()

```
int enQueue (
            myQueue * q,
            float value )
```

Addition in the Circular Queue.

Parameters

	q	where value will be added to the rear
in	*value*	to be added to the queue

Returns

Information about succes (+1) or not, due to full queue and then failure of realloc (-1)

< By omission, it will enQueue

< Did not enQueue, due failing to resize

< Will enQueue, after successuful resize

< It will enQueue

< Status of enQueue operation, in cicular fashion

4.3.3.5 freeQueue()

```
void freeQueue (
            myQueue * q )
```

free memory alocated to the queue This queue returns to inic state

< all space was released

< the queue is now NULL for consistency

< capacity is ZERO

< queue is iniatlly empty

4.3.3.6 getCapacity()

```
int getCapacity (
             const myQueue * q )
```

number of elements in queue

number of elements in queue

Returns

queue capacity

4.3.3.7 getSize()

```
int getSize (
             const myQueue * q )
```

number of elements in queue

number of elements in queue

Returns

number of elements in the queue

< queue is empty

< queue has one element

< queue is full

4.3.3.8 inicQueue()

```
void inicQueue (
             myQueue * q )
```

create empty queue of capacity zero

< ZERO capacity

< the queue is now NULL for consistency

< queue is iniatlly empty

4.3.3.9 printQueue()

```
void printQueue (
            const myQueue * q )
```

Display the queue.

4.3.3.10 resizeQueue()

```
int resizeQueue (
            myQueue * q )
```

if zero becomes CAPACITY else double queue capacity

Returns

0 if the queue was allocated succesfully, -1 if unable to resize queue

< If no capacity reserves base capacity

< base capacity

< unable to resize queue

< queue was allocated succesfully

< doubles previous capacity

< unable to resize queue

< queue was resized succesfully

4.4 fila_circular.h

Go to the documentation of this file.

```
// Addated from https://prepinsta.com/data-structures-algorithms/circular-queue-using-array-in-c/

#include <stdio.h>
#include <stdlib.h>

#define CAPACITY 2

// #define DEBUG

typedef struct myqueue
{
    float * queue;
    unsigned int capacity;
    int front;
    int rear;
} myQueue;


void inicQueue(myQueue * q);

int getCapacity(const myQueue * q);

int getSize(const myQueue * q);


int resizeQueue(myQueue * q);


int checkFull (const myQueue * q);


int checkEmpty (const myQueue *q);


int enQueue (myQueue * q, float value);


int deQueue (myQueue * q, float * value);


void printQueue (const myQueue * q);


void freeQueue (myQueue * q);

```

4.5 lcgrand.c File Reference

Macros

- #define MODLUS 2147483647
- #define MULT1 24112
- #define MULT2 26143

Functions

- float lcgrand (int stream)
- void lcgrandst (long zset, int stream)
- long lcgrandgt (int stream)

4.5.1 Macro Definition Documentation

4.5.1.1 MODLUS

```
#define MODLUS 2147483647
```

4.5.1.2 MULT1

```
#define MULT1 24112
```

4.5.1.3 MULT2

```
#define MULT2 26143
```

4.5.2 Function Documentation

4.5.2.1 lcgrand()

```
float lcgrand (
             int stream )
```

4.5.2.2 lcgrandgt()

```
long lcgrandgt (
             int stream )
```

4.5.2.3 lcgrandst()

```
void lcgrandst (
             long zset,
             int stream )
```

4.6 lcgrand.h File Reference

Functions

- float lcgrand (int stream)
- void lcgrandst (long zset, int stream)
- long lcgrandgt (int stream)

4.6.1 Function Documentation

4.6.1.1 lcgrand()

```
float lcgrand (
             int stream )
```

4.6.1.2 lcgrandgt()

```
long lcgrandgt (
              int stream )
```

4.6.1.3 lcgrandst()

```
void lcgrandst (
              long zset,
              int stream )
```

4.7 lcgrand.h

Go to the documentation of this file.

```
/* The following 3 declarations are for use of the random-number generator
lcgrand and the associated functions lcgrandst and lcgrandgt for seed
management.  This file (named lcgrand.h) should be included in any program
using these functions by executing
#include "lcgrand.h"
before referencing the functions.  */
float lcgrand(int stream);
void lcgrandst(long zset, int stream);
long lcgrandgt(int stream);
```

4.8 testa_fila_circular.c File Reference

```
#include "fila_circular.h"
```

Functions

- int main ()

4.8.1 Function Documentation

4.8.1.1 main()

```
int main ( )
```

< For testing

Index