FACULDADE DE CIÊNCIAS EXATAS E DA ENGENHARIA

Licenciatura em Engenharia Informática

3º Ano

Sistemas Operativos

Projeto prático 2015/2016

Simulação de uma discoteca

Docente: Eduardo Marques

Discentes: José Filipe Silva nº 2070313
Paulo Gil Freitas nº 2012413
Ricardo Pereira nº 2034008

Funchal, 7 de janeiro de 2016

Índice

1.Introdução

No âmbito da unidade curricular Sistemas Operativos, lecionada pelo professor Eduardo Marques, foi pedido a implementação de um simulador de uma discoteca com o objetivo de empregar os conceitos aprendidos nas aulas teóricas e práticas na conceção e desenvolvimento de um sistema simples, empregando os mecanismos de concorrência, sincronização e comunicação existentes na linguagem C.

A avaliação deste projeto fora dividida em três fases:

- 1ª fase: deveria conter as bibliotecas para a gestão da informação nos ficheiros de texto e carregamento de parâmetros pelo Monitor e Simulador;

- 2ª fase: envolvia a implementação das bibliotecas para a comunicação entre o Simulador e o Monitor, criação da interface e ainda a elaboração de um relatório descrevendo as funcionalidades a implementar e como seria feito a sincronização;

- 3ª fase: nesta fase final, teria de agregar todas as bibliotecas anteriores, conter o código com os mecanismos e políticas de sincronização e ainda a entrega do relatório final fundamentando a solução escolhida e conclusões gerais do trabalho.

Tendo estes pontos em atenção, implementou-se uma simulação de uma discoteca que será descrita ao longo deste relatório.

2. Descrição da discoteca

Diferenciando a segunda da terceira fase, na simulação da discoteca pode-se verificar que ocorreu a diminuição do número de salas. Na segunda fase tinha-se a criação de três salas, duas comuns e uma VIP, no entanto com o desenrolar do trabalho estas três salas foram simplificadas para duas salas, uma sem prioridades (sala normal ou 0) e outra com prioridades (sala VIP ou 1). As salas atualmente regem-se pela sua capacidade e pelo tempo que cada pessoa fica no seu interior, tal como no relatório na fase anterior. A capacidade da sala é definida no ficheiro de configuração e tem de ser dada logo de início. Qualquer alteração efetuada neste ficheiro após o começo da simulação será descartada e apenas aplicada na seguinte. Quanto à descriminação entre os clientes criados estes são separados entre normais e VIPs.

- **Sala Normal**

Esta sala é composta por uma fila de espera que é composta por clientes normais e VIPs, não havendo prioridades e tem n número de lugares disponíveis.

Funcionamento: um cliente (normal ou VIP) ao chegar a fila, se houver lugares disponíveis na sala, este entra e fica lá um determinado tempo, caso contrário fica a espera na fila que um cliente deixe um lugar disponível na sala ao sair desta.

- **Sala VIP**

Esta sala é semelhante à sala Normal com a exceção do seu funcionamento quando está cheia, havendo prioridades.

Funcionamento: enquanto houver lugares disponíveis na sala, qualquer tipo de cliente (VIP ou normal) pode entrar, não interessando a prioridade. A partir do momento em que a sala fica cheia é gerado duas filas de espera, em que uma é composta apenas por clientes normais e a outra apenas por clientes VIPs, sendo que estes (os VIPs) têm prioridade na entrada para a sala. A partir deste momento, no caso dos clientes VIPs, a sua entrada na sala apenas é restringida pelo número de lugares disponíveis nesta, ou seja, se houver lugares disponíveis o cliente VIP entra, caso contrário fica à espera que um cliente deixe um lugar disponível na sala quando sair desta. No caso dos clientes normais, se não houver clientes prioritários na fila espera e haja lugares disponíveis na sala, o cliente normal entra e fica na sala um determinado tempo, caso contrário fica a espera na fila.

No que respeita ao funcionamento da discoteca com o parâmetro tempo de abertura e fecho, esta opção foi alterada passando a funcionar da seguinte forma, a discoteca abre com o aparecimento do primeiro cliente e é encerrada quando o último cliente sai da discoteca.

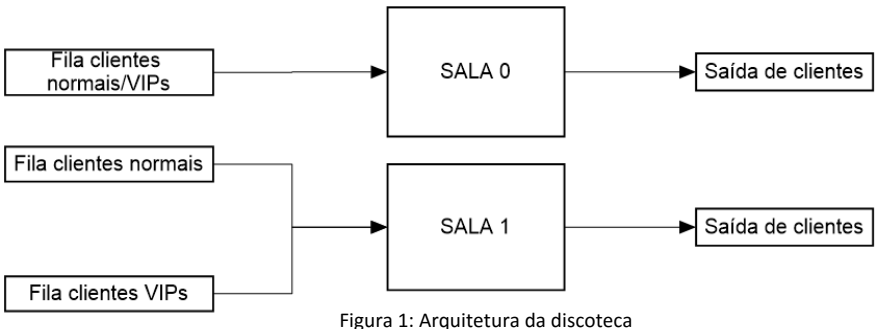

Figura 1: Arquitetura da discoteca

3. Implementação

A implementação da simulação da discoteca foi elaborada, tendo como base a linguagem C. Para a implementação foram criados os ficheiros:

- simulador.c;
- monitor.c;
- configSim.txt;
- Makefile;
- util.h;

➢ **Simulador**

O simulador gera aleatoriamente chegada dos clientes a discoteca e permite a correta coordenação e sincronização dos clientes enviando mensagens dos diversos eventos.

Figura 2: Fluxograma do simulador

Para o carregamento de parâmetros, implementou-se a função **leitura()**, que abre o ficheiro configSim.txt e para cada linha do ficheiro só é copiado o conteúdo após ':', sendo feito em seguida uma conversão do tipo *string* para o tipo *int.*

O ficheiro de configuração configSim.txt contém:

FICHEIRO	DESCRIÇÃO
N_CLIENTES:80	Número total de clientes.
CAPACIDADE_SALA:10	Capacidade máxima de clientes admitida na sala.
PROB_DESISTE_SALA:10	Probabilidade de um cliente desistir da fila de espera.
FREQ_CLIENTES:5	Quantidade de clientes que são criados num determinado espaço de tempo.

Após o carregamento de parâmetros do ficheiro, é feita a ligação com o monitor a partir de um *socket stream*. Esta comunicação será necessária sempre que feche a discoteca e caso um cliente:

- seja criado;
- tenha entrado ou saído de uma sala;
- tenha desistido de uma fila de espera;

O envio de mensagem é feito através da função prepara_mensg(int tipo, int id, int tempo, int sala):

VARIAVEL	DESCRIÇÃO
TIPO	Representa o tipo de evento
ID	ID do cliente
TEMPO	Tempo que ocorreu determinado evento após a abertura da discoteca
SALA	Número da sala

Depois aquando da criação de semáforos, um para a sala normal e dois para a sala VIP. Estes vão assegurar a sincronização limitando quantos clientes podem entrar e quais (no caso de prioritários).

Para a sala normal foi criado o semáforo sala0 que foi inicializado com o valor lido do ficheiro configSim.txt que tem a função de sincronizar o acesso ao recurso que é a sala0:

- **sem_init(&sala0, 0, capacidadeSala)**;

Para a sala VIP foram criados os semáforos semCliVIP e semCliNorm que foram inicializados a 0 e que é no pós-protocolo que ocorre a sinalização da libertação de recursos para estes semáforos:

- **sem_init(&semCliVIP, 0, 0)**;
- **sem_init(&semCliNorm, 0, 0)**;

Como já foi referido anteriormente, a discoteca fecha quando o último cliente sai da discoteca, e o número de clientes é definido através da criação de tarefas, onde o **nºclientes** é o valor lido do ficheiro configSim.txt e o valor **i** é utilizado num ciclo *for* para gerar os clientes numa determinada frequência, ou seja, em cada 5 segundos cria x clientes:

- **pthread_t thread[nºclientes]**: define o número de clientes;
- **pthread_create(&thread[i], NULL, &cliente, NULL)**: cria os clientes.

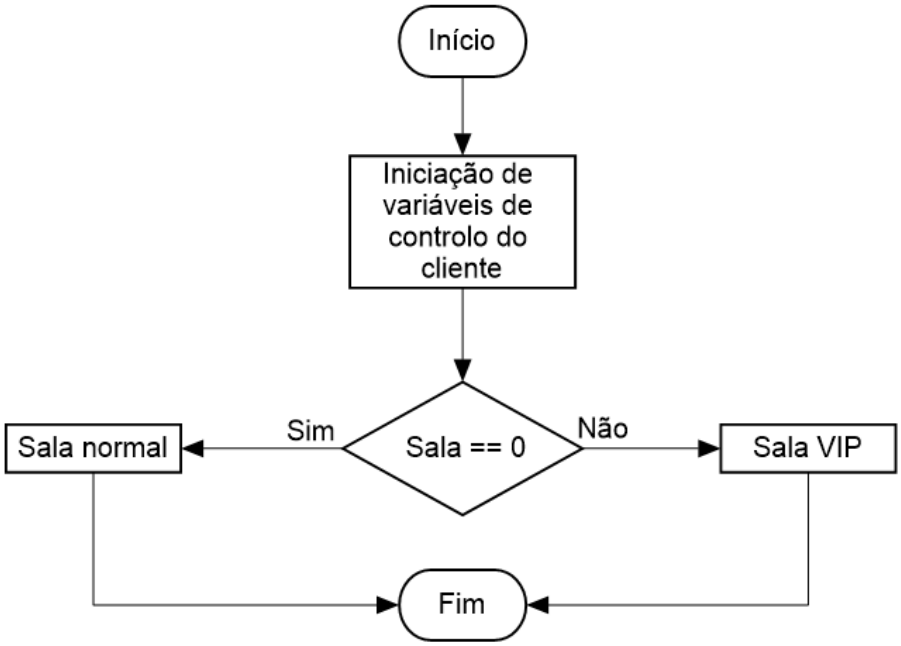

Figura 3: Fluxograma da escolha de sala

Após a criação de clientes é lhes atribuído um id, estes escolhem a sala a partir de um *random,* que cria números inteiros aleatórios de 0 a 1, em que 0 corresponde a sala Normal e 1 a sala VIP.

VARIAVEL	DESCRIÇÃO
CLINTEID	Identificador único do cliente.
SALAID	Identificador da sala escolhida pelo cliente.
CLIENTEPRIO	Prioridade de cada cliente para a entrada de uma sala (utilizado apenas na sala VIP).
PROBDESISTENCIA	Probabilidade que um cliente tem de desistir da fila de espera (utilizado apenas na sala Normal).

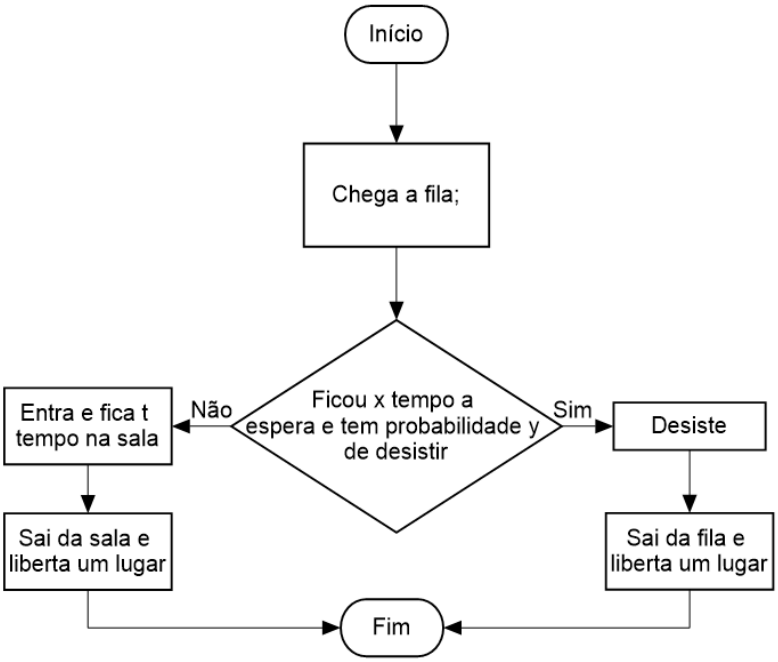

Figura 4: Fluxograma do cliente na sala Normal

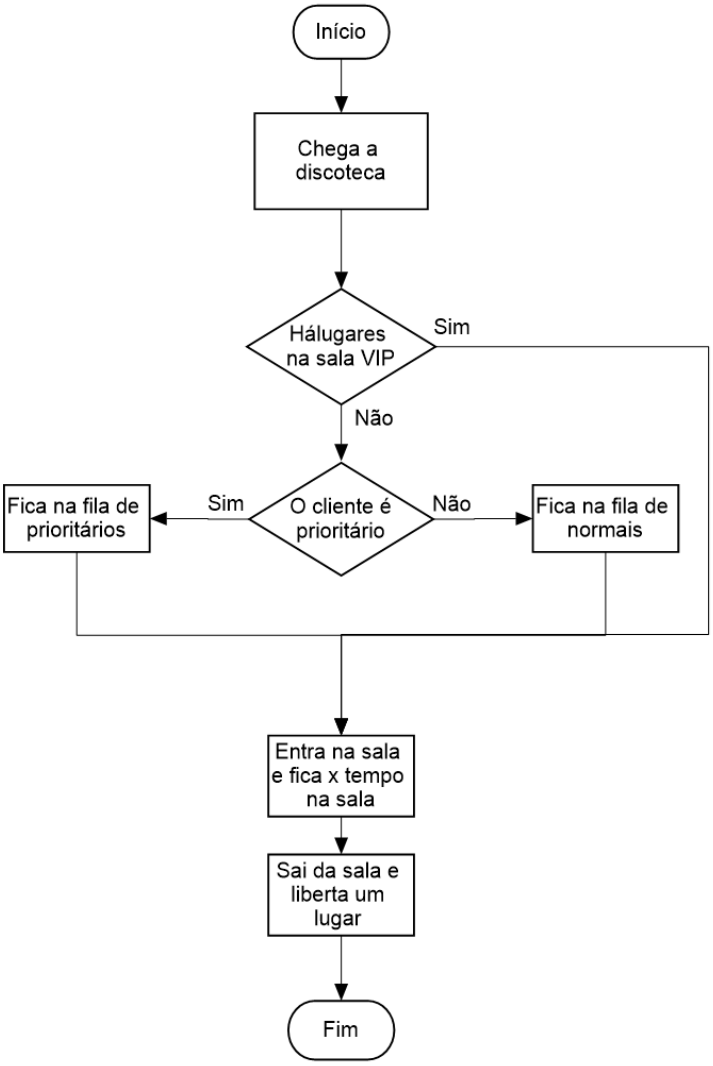

Figura 5: Fluxograma do cliente na sala VIP

- **Monitor**

O monitor aceita a ligação do simulador, descodifica as mensagens recebidas e guarda no ficheiro Logs.txt todos os registos enviados por este. O monitor faz ainda o tratamento das estatísticas:

- A média de espera dos clientes para entrar numa das salas;
- Quantos clientes entraram em cada sala;
- Quantos clientes desistiram na fila de espera;
- Quantos clientes entraram nas salas e o lucro gerado em bilhetes de entrada;
- Duração total da simulação.

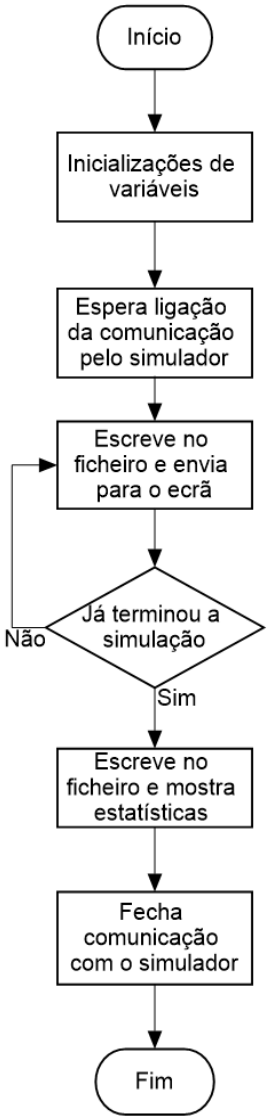

Figura 6: Fluxograma do monitor

- Trincos/ Semáforos

A utilização de trincos e semáforos foi determinante para a implementação de certas opções de funcionamento da discoteca.

Os trincos foram implantados com a função verificadora de variáveis globais, para fazer incrementos e decrementos (numero de clientes, numero de chegadas e saídas das salas) e envio de mensagens para o monitor.

No que respeita à implementação de semáforos, tem-se em diversas situações, alguns exemplos crassos delas são nas filas de entrada para as salas, onde estes limitam as entradas, no momento em que a capacidade máxima da sala foi atingida.

4.Testes

Segundo os testes efetuados foram encontradas diversas limitações no que respeita aos números que podemos utilizar nos diferentes campos.

As restrições são todas relacionadas com a forma como enviamos as mensagens para o monitor, devido a tal, temos que o máximo de tempo de simulação é de 255 segundos, e se ultrapassar este valor o próximo mostrado é zero, fazendo então uma espécie de *reset* onde volta a fazer a contagem até ao valor 255, este processo é repetido.

Dentro dos testes que tivemos em conta podemos exemplificar os números de pessoas que são criadas por espaço de tempo, o qual tal como o numero de pessoas criadas não pode ultrapassar o valor de 128, se ultrapassar obtemos valores negativos. Sendo que o numero de pessoas criadas por espaço de tempo apenas é restringido devido à limitação existente pelo numero total de pessoas criadas (N_CLIENTES). No caso da ultrapassagem desses valores anteriormente referidos existem erros como o caso de *overflow.* Quando ocorre este problema os valores obtidos tornam-se negativos (ID do cliente).

Abaixo é apresentado os testes mais pertinentes que foram realizados para a deteção de erros e limitações da simulação da discoteca.

PARÂMETROS	TEMPO MEDIO ENTRADA	DESISTÊNCIAS	CLIENTES SALA NORMAL	CLIENTES SALA VIP	DINHEIRO GERADO	TEMPO DE SIMULAÇÃO
N_CLIENTES:80 CAPACIDADE_SALA:10 PROB_DESISTE_SALA:10 FREQ_CLIENTE:5	5 min	3	41	36	565 €	121 min
N_CLIENTES:80 CAPACIDADE_SALA:5 PROB_DESISTE_SALA:10 FREQ_CLIENTE:5	55 min	0	32	48	640 €	254 min
N_CLIENTES:80 CAPACIDADE_SALA:10 PROB_DESISTE_SALA:10 FREQ_CLIENTE:10	19 min	2	41	37	575 €	111 min
N_CLIENTES:120 CAPACIDADE_SALA:10 PROB_DESISTE_SALA:10	10 min	4	53	63	895 €	180 min

FREQ_CLIENTE:5						
N_CLIENTES:127 **CAPACIDADE_SALA:10** **PROB_DESISTE_SALA:10** **FREQ_CLIENTE:50**	59 min	6	64	57	890 €	186 min
N_CLIENTES:127 **CAPACIDADE_SALA:30** **PROB_DESISTE_SALA:10** **FREQ_CLIENTE:50**	10 min	2	63	62	935 €	95 min
N_CLIENTES:127 **CAPACIDADE_SALA:127** **PROB_DESISTE_SALA:10** **FREQ_CLIENTE:127**	0 min	0	59	68	975 €	42 min
N_CLIENTES:127 **CAPACIDADE_SALA:30** **PROB_DESISTE_SALA:10** **FREQ_CLIENTE:127**	11 min	8	66	53	860 €	81 min
N_CLIENTES:50 **CAPACIDADE_SALA:10** **PROB_DESISTE_SALA:50** **FREQ_CLIENTE:5**	3 min	4	25	21	335 €	96 min

5.Considerações finais

Com o decorrer da realização do projeto ocorreram diversas problemáticas, no entanto a maior parte das ideias para o funcionamento da discoteca, enunciadas na fase anterior foram colocadas em prática.

Tendo em conta as implementações feitas e as ideias iniciais feitas anteriormente pode-se concluir que os pontos fulcrais presentes nessas ideias e os objetivos propostos pelo professor foram cumpridos.

Conclui-se assim, que o desenvolvimento até à fase final teve sucesso no campo da conclusão dos objetivos propostos por ambas as partes.

6.Anexos

- Código do Simulador

```
// variaveis globais
#include "util.h"
int numCliente;
int nClientes, capacidadeSala,
probDesisteEspera, freqClientes;
int sockfd, dim_serv;
    struct sockaddr_un end_serv;

//variaveis de controlo
int salaVIP_lugares = 0;
int TAM_vip;
int numCliPrio = 0;
int numCliNorm = 0;

//semaforos
sem_t sala0;
sem_t semCliVIP;
sem_t semCliNorm;
//sem_t salaVIP;

//trincos
pthread_mutex_t trincoC; //trinco para
cliente

//tempo
time_t start;

#define CRIACAO 1
#define ENTRADA 2
#define DESISTENCIA 3
#define SAIDA 4
#define TERMINAR 6

//função que le ficheiro
void leitura()
{
    FILE* config;

    config = fopen("configSim.txt", "r");

    if(config != NULL) {
        char linha[50];
        char* valor;
        while(fgets(linha, sizeof(linha),
config) != NULL) {
     valor = strtok(linha, ":");
     if(strcmp(valor, "N_CLIENTES") == 0)
     {
         valor = strtok(NULL, ":");
         nClientes = atoi(valor);
     }
     if(strcmp(valor, "CAPACIDADE_SALA") ==
0) {
          valor = strtok(NULL, ":");
          capacidadeSala = atoi(valor);
    }
    if(strcmp(valor, "PROB_DESISTE_FILA")
== 0) {
          valor = strtok(NULL, ":");
          probDesisteEspera = atoi(valor);
       }
    if(strcmp(valor, "FREQ_CLIENTES") ==
0){
           valor = strtok(NULL, ":");
           freqClientes = atoi(valor);
  }
 }
}
else
    {
        printf("ERRO AO ABRIR FICHEIRO\n");
    }
    fclose(config);
}
void envia_recebe_stream (FILE *fp, int
sockfd, int a, int id)
{
    int n;
    char buffer[MAXLINHA+1];

    if((fgets(buffer, MAXLINHA, fp) ==
NULL) && ferror(fp))
     perror("erro cliente ao ler input");
    n = strlen(buffer) + 1;
    if (a == 1)
    {
        if(write(sockfd, buffer, n) != n)
        perror("erro cliente no write");
    }

    if(write(sockfd, buffer, n) != n)
        perror("erro cliente no write");
    }

void envia_stream(char buff[], int
sockfdes)
{
   if(write(sockfdes, buff, strlen(buff)+1)
== -1)
    {
    perror("ERROR: envia_stream:
simulador");
    }
}

void prepara_mensg(int tipo, int id, int
tempo, int sala)
{
    char buff[5];
    buff[1] = id;
    buff[3] = sala;
    switch(tipo)
    {
        case CRIACAO:
            buff[0] = 'A';
            buff[2] = tempo;
            envia_stream(buff, sockfd);
            break;
        case ENTRADA:
            buff[0] = 'B';
            buff[2] = tempo;
            envia_stream(buff, sockfd);
            break;
        case DESISTENCIA:
            buff[0] = 'C';
            buff[2] = tempo;
            envia_stream(buff, sockfd);
            break;
        case SAIDA:
            buff[0] = 'D';
            buff[2] = tempo;
            envia_stream(buff, sockfd);
            break;
        case TERMINAR:
            buff[0] = 'F';
            buff[2] = tempo;
            envia_stream(buff, sockfd);
            break;
```

```
        default:
        printf("erro evento desconhecido de
cliente");
    }
}
void *cliente(void *ptr)
{
    int hora_criacao;
    int clienteID;
    int salaID = rand()%2;
    int cliente_prio = rand()%2 +1;
    int tempo_espera;
    int probDesistencia = rand()%100 + 1;
    time_t startC;
    startC = time(0);
    hora_criacao = time(0) - start;

    pthread_mutex_lock(&trincoC);

    clienteID = numCliente++;

    pthread_mutex_unlock(&trincoC);

    prepara_mensg(CRIACAO, clienteID,
hora_criacao, salaID);
    printf("\ncliente nº: %d, sala
escolhida: %d sua prioridade e:%d!\n",
clienteID, salaID, cliente_prio);
    printf("tempo quando chegou: %d\n",
hora_criacao);

    if(salaID == 0)
    {
        tempo_espera = time(0);//variavel
com o valor com o tempo que esta na fila a
espera
        sem_wait(&sala0);

        if(probDesistencia <
probDesisteEspera && (int)(time(0) -
tempo_espera) > 5)
        {
            printf("o cliente %d
DESISTIU!\n", clienteID);
            usleep(5000);//sleep critico

            pthread_mutex_lock(&trincoC);
            prepara_mensg(DESISTENCIA,
clienteID, (int)(time(0) - tempo_espera),
salaID);
            pthread_mutex_unlock(&trincoC);
            sem_post(&sala0);
            return NULL;
        }
        usleep(5000);//SLEEP CRITICO

      pthread_mutex_lock(&trincoC);//TRINCO
PARA EXPERIMENTAR
prepara_mensg(ENTRADA, clienteID,
(int)(time(0) - startC), salaID);//CUIDADE
COM A POSICAO DESTA LINHA SE FOR PARA METER
DESISTENCIAS
        pthread_mutex_unlock(&trincoC);
        printf("o cliente %d escolheu
sala0\n", clienteID);
        printf("acabou de entrar o cliente
%d\n", clienteID);


        usleep(25000000);//fica na disco 20
seg+/-
        printf("vai sair o cliente %d da
sala0 ao %d min\n", clienteID,
(int)(time(0) - start));

      sem_post(&sala0);

      pthread_mutex_lock(&trincoC);//TRINCO
PARA EXPERIMENTAR
        prepara_mensg(SAIDA, clienteID,
(int)(time(0) - start), salaID);
        pthread_mutex_unlock(&trincoC);

    }
    else//escolha da sala vip
    {
        pthread_mutex_lock(&trincoC);
        if(salaVIP_lugares >= TAM_vip)
        {
//             if(salaVIP_lugares > 5)
//             {
//
pthread_mutex_unlock(&trincoC);
//                 printf("o cliente %d
DESISTIU, apos %d min a ESPERA\n",
clienteID, tempo_espera);//valor de
tempo_espera errado
//
prepara_mensg(DESISTENCIA, clienteID,
(int)(time(0) - startC));
//
//
//printf("antes_return_desiste cliente
%d\n", clienteID);
//                 return NULL;
//             }
 if(cliente_prio == 2)
 {
        numCliPrio++;

        pthread_mutex_unlock(&trincoC);

        sem_wait(&semCliVIP);

        pthread_mutex_lock(&trincoC);
  }
  else
  {
        numCliNorm++;
        pthread_mutex_unlock(&trincoC);
        sem_wait(&semCliNorm);
        pthread_mutex_lock(&trincoC);
            }
        }

        salaVIP_lugares++;

        pthread_mutex_unlock(&trincoC);

        usleep(5000);//sleep CRITICO para
poder enviar o prepara_mensg

pthread_mutex_lock(&trincoC);//TRINCO PARA
EXPERIMENTAR
        prepara_mensg(ENTRADA, clienteID,
(int)(time(0) - startC), salaID);
        pthread_mutex_unlock(&trincoC);

        printf("ENTROU o cliente %d PRIO:%d
na salaVIP\n", clienteID, cliente_prio);

        usleep(30000000);//fica na sala VIP
15 seg+/- lol
        pthread_mutex_lock(&trincoC);
```

```
        if(salaVIP_lugares >= TAM_vip)
        {
         if(numCliPrio > 0)
         {
            sem_post(&semCliVIP);
            numCliPrio--;
         }
         else
         {
            if(numCliNorm > 0)
         {

        sem_post(&semCliNorm);
        numCliNorm--;
       }
      }
     }

        salaVIP_lugares--;
        pthread_mutex_unlock(&trincoC);

        pthread_mutex_lock(&trincoC);//TRIN
CO PARA EXPERIMENTAR
        prepara_mensg(SAIDA, clienteID,
(int)(time(0) - start), salaID);
        pthread_mutex_unlock(&trincoC);

        printf("SAIU o cliente %d PRIO:%d
da salaVIP\n", clienteID, cliente_prio);

    }

    return NULL;
}

int main()
{

    leitura();
    printf("O nº clientes a ser criado e':
%d\n",nClientes);//NÃO USADO DE MOMENTO
    printf("A capacidade da sala e':
%d\n",capacidadeSala);
    printf("A probablidade de desistencia
da fila e': %d\n",probDesisteEspera);
    printf("A frequencia de criação de
clientes por segundo e': %d\n",
freqClientes);

    //cria socket stream
    if((sockfd = socket (AF_UNIX,
SOCK_STREAM, 0)) < 0)
        perror("erro ao criar socket
cliente");

    bzero((char *) &end_serv,
sizeof(end_serv));

    end_serv.sun_family = AF_UNIX;
    strcpy(end_serv.sun_path,
UNIXSTR_PATH);
    dim_serv = strlen(end_serv.sun_path) +
sizeof(end_serv.sun_family);

    if(connect(sockfd, (struct sockaddr *)
&end_serv, dim_serv) < 0)
        perror("erro ao fazer connect no
cliente");

    //Inicializar semáforos
    sem_init(&sala0, 0, capacidadeSala);//
(semaforo, partilhado==0, valor)tamanho
maximo da sala 0
    sem_init(&semCliVIP, 0, 0);
    sem_init(&semCliNorm, 0, 0);

    //varivel de tamanho da sala VIP
    TAM_vip = capacidadeSala;//tamanho
maximo da sala vip

    start = time(0);
    printf("o tempo de iniciacao: %ld",
start - start);

    srand(time(0));
    //preparaçao das tarefas
    pthread_t thread[nClientes];//colocar
variavel "nClientes"

    for(int i = 0; i <nClientes; i++)
    {
        pthread_create(&thread[i], NULL,
&cliente, NULL);
        usleep(50000);
        if(i%freqClientes == 0)//quantos
clientes sao gerados de "uma só vez"
        {
            usleep(5000000);
        }
    }
    for(int j = 0; j < nClientes; j++)
    {
        pthread_join(thread[j], NULL);
        printf("Tarefa %d terminou\n",j);
    }

    usleep(5000);
    prepara_mensg(TERMINAR, 30,
(int)(time(0) - start), 1);

    close(sockfd);
    exit(0);0return
```

- Código do Monitor

```
#include "util.h"

int pararsimulacao = 0;
int sockfd, novosockfd, dim_cli, dim_serv,
pid_filho;
struct sockaddr_un end_cli, end_serv;

//variaveis de controlo
int clienteTotal, desistenciasTotais,
tempoTotalSimulacao;
int contadorTempoEntradas,
tempoMedioEntrada, contadorDesistencias,
contadorEntradas;
int arrayCliChegadas[200];
int arrayCliTempoEntradas[200];
int contadorSala0, contadorSala1;
int contadorChegadas;
void escrita()
{
    FILE* logs;

    logs = fopen("Logs.txt", "a");

    if(logs == NULL)
    {
        printf("ERRO AO CRIAR FICHEIRO");
    }
    else
    {
        fprintf(logs, "\n---SIMULAÇÃO
INICIADA---\n");


        //int a = 0;
        while(1)
        {
            char buffer[5];
            read(novosockfd, buffer,
sizeof(buffer));

            //printf("%s\n", buffer);

            if(buffer[0] == 'A')//criacao
de cliente
            {
                if(buffer[2] < 0)
                {
                    printf("Chegou um
cliente a fila seu numero é: %d, chegou
aos: %d minutos!\n", buffer[1], (256 +
buffer[2]));

arrayCliChegadas[contadorChegadas] =
buffer[2];
                    contadorChegadas++;
                    fprintf(logs, "Chegou
um cliente a fila seu numero é: %d, chegou
aos: %d minutos!\n", buffer[1], (256 +
buffer[2]));
                    clienteTotal++;
                }
                else
                {
                    printf("Chegou um
cliente a fila seu numero é: %d, chegou
aos: %d minutos!\n", buffer[1], buffer[2]);

arrayCliChegadas[contadorChegadas] =
buffer[2];
                    contadorChegadas++;
                    fprintf(logs, "Chegou
um cliente a fila seu numero é: %d, chegou
aos: %d minutos!\n", buffer[1], buffer[2]);
                    clienteTotal++;
                }
            }
            if(buffer[0] == 'B')//entrada
de cliente na disco
            {
                if(buffer[2] < 0)
                {
                    printf("\nEntrou na
discoteca o cliente: %d, na sala %d, entrou
apos %d minutos na fila de
espera\n",buffer[1], buffer[3], (256 +
buffer[2]));

arrayCliTempoEntradas[contadorTempoEntradas
] = (256 + buffer[2]);

contadorTempoEntradas++;
         if(buffer[3] == 0)
         {
            contadorSala0++;
          }//contador da sala
          else
          {
            contadorSala1++;
          }
             fprintf(logs, "\nEntrou na
discoteca o cliente: %d, entrou apos %d
minutos na fila de espera\n", buffer[1],
(256 + buffer[2]));
         contadorEntradas++;
          }
          else
          {
         printf("\nEntrou na discoteca o
cliente: %d,na sala %d entrou apos %d
minutos na fila de espera\n",
buffer[1],buffer[3],  buffer[2]);

arrayCliTempoEntradas[contadorTempoEntradas
] = buffer[2];

contadorTempoEntradas++;
       if(buffer[3] == 0)
      {
         contadorSala0++;
       }//contador da sala
       else
     {
      contadorSala1++;
      }//contador da sala
     fprintf(logs, "\nEntrou na discoteca o
cliente: %d, entrou apos %d minutos na fila
de espera\n", buffer[1], buffer[2]);
      contadorEntradas++;
                }
            }
            if(buffer[0] ==
'C')//desistencia de um cliente
            {
                if(buffer[2] < 0)
                {
                    printf("\nDesistiu da
discoteca o cliente: %d, entrou apos %d
minutos na fila de espera\n", buffer[1],
(256 + buffer[2]));
                    fprintf(logs,
"\nDesistiu da discoteca o cliente: %d,
```

```
entrou apos %d minutos na fila de
espera\n", buffer[1], (256 + buffer[2]));
                    contadorDesistencias++;
                }
                else
                {
                    printf("\nDesistiu da
discoteca o cliente: %d, entrou apos %d
minutos na fila de espera\n", buffer[1],
buffer[2]);
                    fprintf(logs,
"\nDesistiu da discoteca o cliente: %d,
entrou apos %d minutos na fila de
espera\n", buffer[1], buffer[2]);
                    contadorDesistencias++;
                }
            }
            if(buffer[0] == 'D')//saida da
disco
            {
                if(buffer[2] < 0)
                {
                    printf("\nSaiu da
discoteca o cliente: %d, sala %d no
minuto:%d \n", buffer[1], buffer[3], (256 +
buffer[2]));
                    fprintf(logs, "\nSaiu
da discoteca o cliente: %d, no minuto:%d
\n", buffer[1], (256 + buffer[2]));
                }
                else
                {
                    printf("\nSaiu da
discoteca o cliente: %d, sala %d no
minuto:%d \n", buffer[1], buffer[3],
buffer[2]);
                    fprintf(logs, "\nSaiu
da discoteca o cliente: %d, no minuto:%d
\n", buffer[1], buffer[2]);
                }
            }
            if(buffer[0] == 'F')
            {
                if(buffer[2] < 0)
                {
                    printf("\nFoi Encerrado
a discoteca, apos %d minutos!\n", (256 +
buffer[2]));
                    tempoTotalSimulacao =
(256 + buffer[2]);
                    break; //este break
estara numa letra que significara que foi
terminado a simulacao
                }
                else
                {
                    printf("\nFoi Encerrado
a discoteca, apos %d minutos!\n",  +
buffer[2]);
                    tempoTotalSimulacao =
buffer[2];
                    break; //este break
estara numa letra que significara que foi
terminado a simulacao
                }
            }

        }
        fprintf(logs, "---SIMULAÇÃO
TERMINADA---\n\n");

    }

    fclose(logs);
}

void estatisticas()
{
    FILE* logs;

    logs = fopen("Logs.txt", "a");

    if(logs == NULL)
    {
        printf("ERRO AO CRIAR FICHEIRO");
    }
    else
    {
        fprintf(logs, "\n\nINICIO DAS
ESTATISTICAS\n");
        printf("\n\nINICIO DAS
ESTATISTICAS\n\n");
        fprintf(logs, "Foram gerados %d
clientes ao todo\n", clienteTotal);
        printf("Foram gerados %d clientes
ao todo\n", clienteTotal);
        for(int i = 0; i <
contadorTempoEntradas; i++)
        {
            tempoMedioEntrada =
tempoMedioEntrada +
arrayCliTempoEntradas[i]; //formula esta
errada, voltar a rever
        }
        tempoMedioEntrada =
tempoMedioEntrada/clienteTotal;
        fprintf(logs, "A media de espera
para entrar dos clientes foi %d min!\n",
tempoMedioEntrada);
        printf("A media de espera para
entrar dos clientes foi %d min!\n",
tempoMedioEntrada);
        fprintf(logs, "Entraram na sala0 %d
e na salaVIP %d\n", contadorSala0,
contadorSala1);
        printf("Entraram na sala0 %d e na
salaVIP %d\n", contadorSala0,
contadorSala1);
        fprintf(logs, "Número de
desistencias %d!\n", contadorDesistencias);
        printf("Número de desistencias
%d!\n", contadorDesistencias);
        fprintf(logs, "Número de clientes
que entrou %d, foram gerados %d€ em
bilhetes de entrada\n", contadorEntradas,
(contadorSala0*5 + contadorSala1*10));
        printf("Número de clientes que
entrou %d, foram gerados %d€ em bilhetes de
entrada\n", contadorEntradas,
(contadorSala0*5 + contadorSala1*10));
        fprintf(logs, "Duração total da
simulação foi de %d minutos!\n",
tempoTotalSimulacao);
        printf("Duração total da simulação
foi de %d minutos!\n",
tempoTotalSimulacao);
        fprintf(logs, "\n\nFIM DAS
ESTATISTICAS\n");
        printf("\nFIM DAS ESTATISTICAS\n");
    }
}

void trata_cliente_stream(int sockfd)
{
    int n =0;
    char buffer[MAXLINHA+1];
```

```

    n = read(sockfd, buffer, MAXLINHA+1);
    if(n < 0)
    {
        perror("erro servidor no read");
    }

    printf("%s", buffer);
    printf("asdfsa");

    if(write(sockfd, buffer,n) != n)
        perror("erro servidor no write");

}

int main()
{


    if((sockfd= socket(AF_UNIX,
SOCK_STREAM, 0)) < 0)
    {
            perror("erro ao criar socket
stream servidor");
    }

    unlink(UNIXSTR_PATH);

    bzero((char *)&end_serv,
sizeof(end_serv));
    end_serv.sun_family =AF_UNIX;
    strcpy(end_serv.sun_path,
UNIXSTR_PATH);
    dim_serv = strlen(end_serv.sun_path) +
sizeof(end_serv.sun_family);
    if(bind(sockfd, (struct sockaddr *)
&end_serv, dim_serv) <0)
    {
            perror("erro ao atribuir nome
socket servidor");
    }

    listen(sockfd, 1);

    for(;;)
    {
        dim_cli = sizeof(end_cli);

        novosockfd = accept(sockfd, (struct
sockaddr *) &end_cli, &dim_cli);
        if(novosockfd < 0)
        {
            perror("erro ao criar ligacao
dedicada: accept");
        }

        if((pid_filho = fork()) < 0)
        {
            perror("erro ao criar processo
para atender cliente");
        }
        else if(pid_filho == 0)
        {
            close(sockfd);
            exit(0);
        }
        printf("INICIO DA SIMULACAO\n\n");
        escrita();
        estatisticas();

        close(novosockfd);
        printf("FIM DA SIMULACAO\n");
        break;
    }


    printf("Ficheiro Logs.txt criado com
exito!\n");

    return 0;

}
```