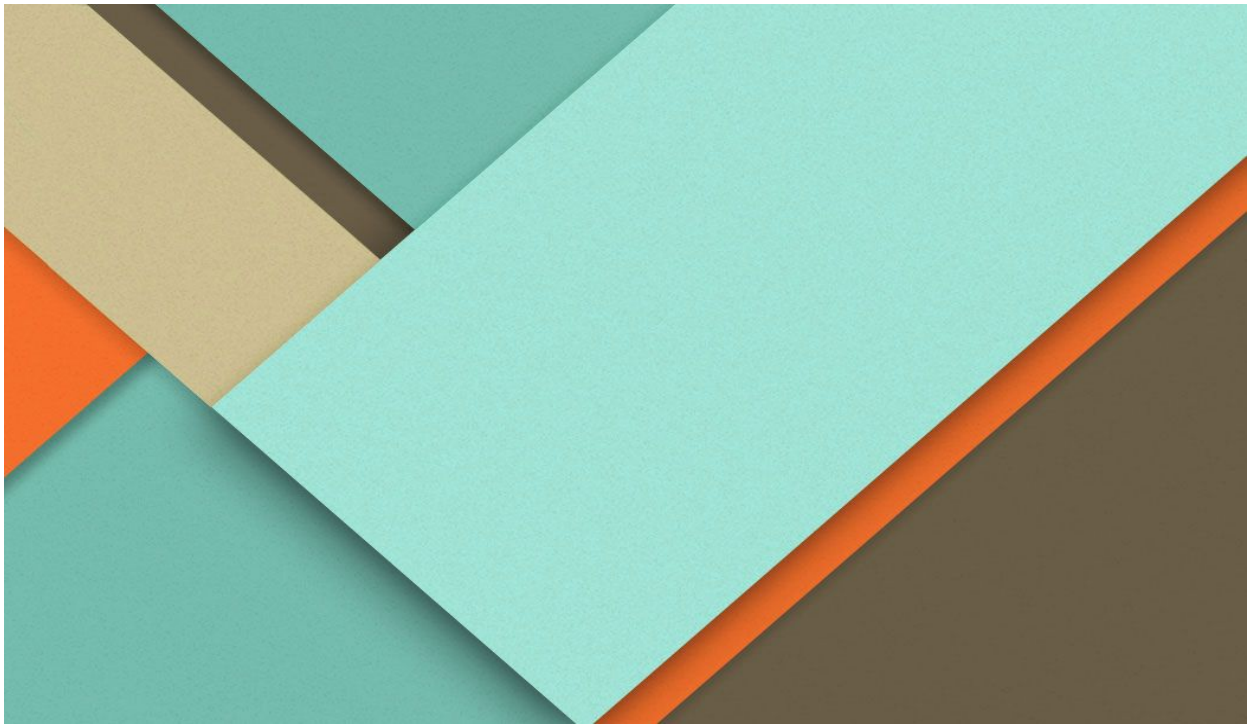


Controlador de sistema de aquecimento

09/09/2019

Gabriel de Souza Nogueira da Silva

Gabriel Ribeiro Camelo

Universidade Federal do Ceará

Rua Cel. Estanislau Frota, 563

Sobral, Ceará, 62010-560

Visão geral

Implementar um software em tempo real na linguagem C para controlar a temperatura de um aquecedor simulado obedecendo os requisitos físicos do sistema e usando como código base o exemplo “controlemanual.c” dado junto com os arquivos do simulador.

Objetivos

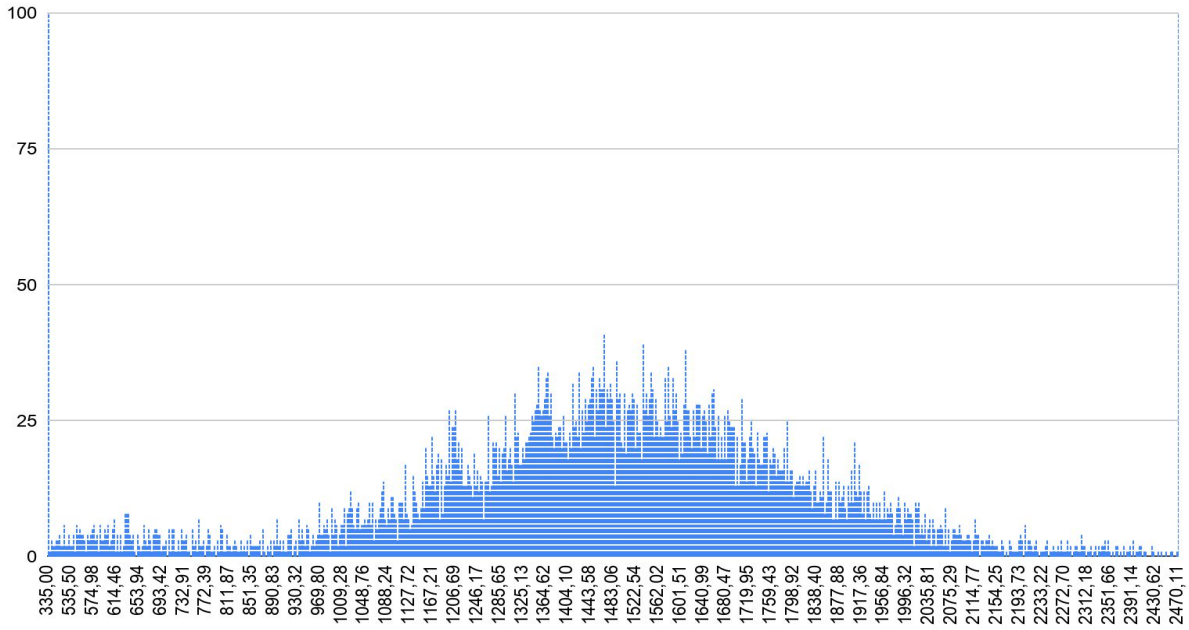
1. Criar um laço de controle como tarefa periódica para a temperatura;
2. Uso das entradas Na e Ni e da saída de água Nf para o controle;
3. Informações na tela sobre a situação corrente.

Especificações

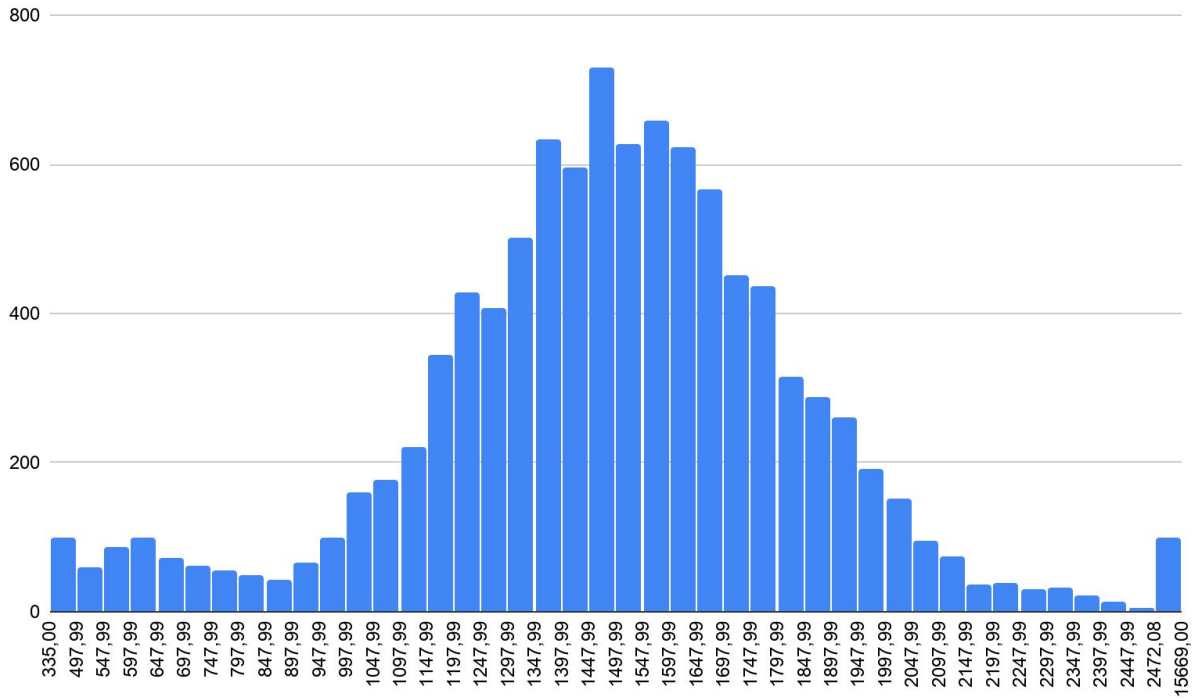
Quando o programa é iniciado é necessário que o usuário digite a temperatura desejada e o software se encarregará de manter a temperatura no valor desejado.

OBS.: Para mais informações sobre o código consultar a seção de anexo (Página 3).

Análise gráfica dos tempos de resposta



No gráfico acima foi utilizado um total de 10000 amostras de tempo de resposta do mesmo programa enquanto ele funcionava controlando a temperatura do aquecedor. Como são 10000 valores a exibição valor por valor tem uma visão prejudicada pois o gráfico fica muito extenso; logo no gráfico abaixo temos um histograma mais simplificado do programa fazendo uma média entre os valores e exibindo para melhor análise.



Portanto, concluímos que o período de 30 ms é mais que suficiente para o funcionamento do sistema visto que o maior tempo foi de 15669 us, 15,6 ms aproximadamente, fazendo com que o deadline da tarefa sempre seja cumprido. Além disso, se ainda quisermos podemos diminuir o deadline para 20 ms e no pior caso ainda estaria cumprindo o deadline estabelecido.

Anexos

```
/*
*   EQUIPE:
*       Gabriel de Souza Nogueira da Silva
*       Gabriel Ribeiro Camelo
*   Execute o programa em uma máquina linux com o seguinte comando:
*                   gcc -o trab_final trab_final.c -lrt
*
*
*/

#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <netdb.h>

#define         NSEC_PER_SEC    (1000000000)        // Numero de nanosegundos em um
milissegundo

#define         N_AMOSTRAS 10000            // Numero de amostras (medicoes) coletadas

#define FALHA 1
```

```
char teclado[1000];
double valor;
char msg_enviada[1000], msg_enviada_clone[1000];
char msg_recebida[1000];
int nrec;
long atraso_inicio[N_AMOSTRAS];		// Medicoes do atraso ateh inicio em microsegundos
long atraso_fim[N_AMOSTRAS];		// Medicoes do atraso ateh o fim em microsegundos

int cria_socket_local(void)
{
	int socket_local;		/* Socket usado na comunicacao */

	socket_local = socket( PF_INET, SOCK_DGRAM, 0);
	if (socket_local < 0) {
		perror("socket");
		return -1;
	}
	return socket_local;
}


struct sockaddr_in cria_endereco_destino(char *destino, int porta_destino)
{
	struct sockaddr_in servidor; /* Endereco do servidor incluindo ip e porta */
	struct hostent *dest_internet;	/* Endereco destino em formato proprio */
	struct in_addr dest_ip;			/* Endereco destino em formato ip numerico */

	if (inet_aton ( destino, &dest_ip ))
```

```
		dest_internet = gethostbyaddr((char *)&dest_ip, sizeof(dest_ip), AF_INET);
	else
		dest_internet = gethostbyname(destino);

	if (dest_internet == NULL) {
		fprintf(stderr,"Endereco de rede invalido\n");
		exit(FALHA);
	}

	memset((char *) &servidor, 0, sizeof(servidor));
	memcpy(&servidor.sin_addr, dest_internet->h_addr_list[0],
sizeof(servidor.sin_addr));
	servidor.sin_family = AF_INET;
	servidor.sin_port = htons(porta_destino);

	return servidor;
}



void envia_mensagem(int socket_local, struct sockaddr_in endereco_destino, char
*mensagem)
{
	/* Envia msg ao servidor */

	if (sendto(socket_local, mensagem, strlen(mensagem)+1, 0, (struct sockaddr *)
&endereco_destino, sizeof(endereco_destino)) < 0 )
	{
		perror("sendto");
		return;
	}
}
```

```
int recebe_mensagem(int socket_local, char *buffer, int TAM_BUFFER)
{
	int bytes_recebidos;		/* Numero de bytes recebidos */

	/* Espera pela msg de resposta do servidor */
	bytes_recebidos = recvfrom(socket_local, buffer, TAM_BUFFER, 0, NULL, 0);
	if (bytes_recebidos < 0)
	{
		perror("recvfrom");
	}

	return bytes_recebidos;
}


int main(int argc, char* argv[])
{
	struct timespec t, t_inicio, t_fim;
	int amostra = 0;			// Amostra corrente
	long int periodo = 30000000;		// 30ms

    if (argc < 3) {
		fprintf(stderr,"Uso: controlemanual <endereco> <porta>\n");
		fprintf(stderr,"<endereco> eh o endereco IP da caldeira\n");
		fprintf(stderr,"<porta> eh o numero da porta UDP da caldeira\n");
		fprintf(stderr,"Exemplo de uso:\n");
		fprintf(stderr,"   controlemanual localhost 12345\n");
		exit(FALHA);
	}
```

```
   int porta_destino = atoi( argv[2]);

   int socket_local = cria_socket_local();

         struct sockaddr_in endereco_destino = cria_endereco_destino(argv[1],
porta_destino);

         // Tarefa periodica iniciará em 1 segundo
         //t.tv_sec++;

   int temperatura_user;
   float haltura, temperatura_sist;

   printf("digite o valor da temperatura desejada\n");
   scanf("%d", &temperatura_user);

   char* ler(char* consulta){//ler as variaveis do simulador
      strcpy( msg_enviada, consulta);

      envia_mensagem(socket_local, endereco_destino, msg_enviada);

                  nrec = recebe_mensagem(socket_local, msg_recebida, 1000);
                  msg_recebida[nrec] = '\0';

      return msg_recebida;
   }

   void altera(char* consulta, float valor){//altera os valores das variaveis no simulador
      sprintf( msg_enviada, consulta, valor);
```

```
    envia_mensagem(socket_local, endereco_destino, msg_enviada);

                nrec = recebe_mensagem(socket_local, msg_recebida, 1000);
                msg_recebida[nrec] = '\0';
}

void altera_altura(){//equilibra a altura entre 2.6 e 2 m
    if(haltura >= 2.6){
        altera("anf%lf", 100.0);
        altera("ani%lf", 0.0);
        altera("ana%lf", 0.0);
      }

      if(haltura <= 2){
        altera("anf%lf", 0);
      }
}

// Le a hora atual, coloca em t
clock_gettime(CLOCK_MONOTONIC ,&t);

      while(amostra < N_AMOSTRAS) {

      // Espera ateh inicio do proximo periodo
                clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);

    // Realiza seu trabalho
                printf("Coletada a amostra %d\n", amostra);

                // Le a hora atual, coloca em t_inicio
                clock_gettime(CLOCK_MONOTONIC ,&t_inicio);
```

```
                // Calcula atraso observado em microsegundos
                atraso_inicio[amostra] = 1000000*(t_inicio.tv_sec - t.tv_sec)   +
(t_inicio.tv_nsec - t.tv_nsec)/1000;

    char* msg_rec = ler("st-0");

    temperatura_sist = atof(&msg_rec[3]);
    printf(">>>%f<<<\n", temperatura_sist);

    if (temperatura_user > temperatura_sist){
        msg_rec = ler("sh-0");
        haltura = atof(&msg_rec[3]);

        altera_altura();

        altera("ani%lf", 0.0);
        altera("ana%lf", 10.0);
    }

    if(temperatura_user < temperatura_sist){
        msg_rec = ler("sh-0");
        haltura = atof(&msg_rec[3]);

        altera_altura();

        altera("ana%lf", 0.0);
        altera("ani%lf", 10.0);
    }

                // Le a hora atual, coloca em t_fim
```

```
                clock_gettime(CLOCK_MONOTONIC ,&t_fim);

                // Calcula o tempo de resposta observado em microsegundos
                atraso_fim[amostra++] = 1000000*(t_fim.tv_sec - t.tv_sec)   +   (t_fim.tv_nsec - t.tv_nsec)/1000;

                // Calcula inicio do proximo periodo
                t.tv_nsec += periodo;
                while (t.tv_nsec >= NSEC_PER_SEC) {
                        t.tv_nsec -= NSEC_PER_SEC;
                        t.tv_sec++;
                }
        }

        printf("Atrasos ateh o inicio da execucao incluem overhead,release jitter, interferencias\n");
        printf("Tempo de resposta vai desde a chegada ateh a conclusao\n\n");
        printf("Atraso inicio          Tempo de resposta\n");

   FILE* dados_i;
   FILE* dados_f;

   dados_i = fopen("dados_inicio.txt", "w+");
   if(dados_i == NULL){
      printf("Erro, nao foi possivel abrir o arquivo\n");
      exit(1);
   }

   dados_f = fopen("dados_fim.txt", "w+");
   if(dados_f == NULL){
      printf("Erro, nao foi possivel abrir o arquivo\n");
      exit(1);
```

```
    }

        for( int i=0; i<N_AMOSTRAS; ++i){
                printf("Atraso inicio=%4ldus     Tempo de resposta=%4ldus\n",
atraso_inicio[i], atraso_fim[i]);
      fprintf(dados_i, "%4ld \n", atraso_inicio[i]);
      fprintf(dados_f, "%4ld \n", atraso_fim[i]);
    }

    fclose(dados_i);
    fclose(dados_f);
}



*   EQUIPE:
*      Gabriel de Souza Nogueira da Silva
*      Gabriel Ribeiro Camelo
*   Execulte o programa em uma maquina linux com o seguinte comando:
*                gcc -o trab_final trab_final.c -lrt
*
*
*/

#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
```

```
#include <errno.h>
#include <stdlib.h>
#include <netdb.h>

#define NSEC_PER_SEC (1000000000) // Numero de nanosegundos em um milissegundo

#define N_AMOSTRAS 10000 // Numero de amostras (medicoes) coletadas

#define FALHA 1


char teclado[1000];
double valor;
char msg_enviada[1000], msg_enviada_clone[1000];
char msg_recebida[1000];
int nrec;
long atraso_inicio[N_AMOSTRAS]; // Medicoes do atraso ateh inicio em microsegundos
long atraso_fim[N_AMOSTRAS]; // Medicoes do atraso ateh o fim em microsegundos

int cria_socket_local(void)
{
	int socket_local; /* Socket usado na comunicacao */

	socket_local = socket( PF_INET, SOCK_DGRAM, 0);
	if (socket_local < 0) {
		perror("socket");
		return -1;
	}
	return socket_local;
```

```
}


struct sockaddr_in cria_endereco_destino(char *destino, int porta_destino)
{
        struct sockaddr_in servidor; /* Endereco do servidor incluindo ip e porta */
        struct hostent *dest_internet;      /* Endereco destino em formato proprio */
        struct in_addr dest_ip;                     /* Endereco destino em formato ip
numerico */

        if (inet_aton ( destino, &dest_ip ))
                dest_internet = gethostbyaddr((char *)&dest_ip, sizeof(dest_ip), AF_INET);
        else
                dest_internet = gethostbyname(destino);

        if (dest_internet == NULL) {
                fprintf(stderr,"Endereco de rede invalido\n");
                exit(FALHA);
        }

        memset((char *) &servidor, 0, sizeof(servidor));
        memcpy(&servidor.sin_addr, dest_internet->h_addr_list[0],
sizeof(servidor.sin_addr));
        servidor.sin_family = AF_INET;
        servidor.sin_port = htons(porta_destino);

        return servidor;
}


void envia_mensagem(int socket_local, struct sockaddr_in endereco_destino, char
*mensagem)
```

```
{
	/* Envia msg ao servidor */

	if (sendto(socket_local, mensagem, strlen(mensagem)+1, 0, (struct sockaddr *)
&endereco_destino, sizeof(endereco_destino)) < 0 )
	{
		perror("sendto");
		return;
	}
}


int recebe_mensagem(int socket_local, char *buffer, int TAM_BUFFER)
{
	int bytes_recebidos;		/* Numero de bytes recebidos */

	/* Espera pela msg de resposta do servidor */
	bytes_recebidos = recvfrom(socket_local, buffer, TAM_BUFFER, 0, NULL, 0);
	if (bytes_recebidos < 0)
	{
		perror("recvfrom");
	}

	return bytes_recebidos;
}


int main(int argc, char* argv[])
{
	struct timespec t, t_inicio, t_fim;
	int amostra = 0;		// Amostra corrente
```

```
        long int periodo = 30000000;          // 30ms

    if (argc < 3) {
                fprintf(stderr,"Uso: controlemanual <endereco> <porta>\n");
                fprintf(stderr,"<endereco> eh o endereco IP da caldeira\n");
                fprintf(stderr,"<porta> eh o numero da porta UDP da caldeira\n");
                fprintf(stderr,"Exemplo de uso:\n");
                fprintf(stderr,"   controlemanual localhost 12345\n");
                exit(FALHA);
        }

   int porta_destino = atoi( argv[2]);

   int socket_local = cria_socket_local();

        struct sockaddr_in endereco_destino = cria_endereco_destino(argv[1],
porta_destino);

        // Tarefa periodica iniciará em 1 segundo
        //t.tv_sec++;

   int temperatura_user;
   float haltura, temperatura_sist;

   printf("digite o valor da temperatura desejada\n");
   scanf("%d", &temperatura_user);

   char* ler(char* consulta){//ler as variaveis do simulador
      strcpy( msg_enviada, consulta);

      envia_mensagem(socket_local, endereco_destino, msg_enviada);
```

```
                nrec = recebe_mensagem(socket_local, msg_recebida, 1000);
                msg_recebida[nrec] = '\0';

    return msg_recebida;
}


void altera(char* consulta, float valor){//altera os valores das variaveis no simulador
    sprintf( msg_enviada, consulta, valor);

    envia_mensagem(socket_local, endereco_destino, msg_enviada);

                nrec = recebe_mensagem(socket_local, msg_recebida, 1000);
                msg_recebida[nrec] = '\0';
}

void altera_altura(){//equilibra a altura entre 2.6 e 2 m
     if(haltura >= 2.6){
          altera("anf%lf", 100.0);
          altera("ani%lf", 0.0);
          altera("ana%lf", 0.0);
        }

        if(haltura <= 2){
          altera("anf%lf", 0);
        }
}

// Le a hora atual, coloca em t
clock_gettime(CLOCK_MONOTONIC ,&t);
```

```
        while(amostra < N_AMOSTRAS) {

        // Espera ateh inicio do proximo periodo
                clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);

    // Realiza seu trabalho
                printf("Coletada a amostra %d\n", amostra);

                // Le a hora atual, coloca em t_inicio
                clock_gettime(CLOCK_MONOTONIC ,&t_inicio);

                // Calcula atraso observado em microsegundos
                atraso_inicio[amostra] = 1000000*(t_inicio.tv_sec - t.tv_sec)   +
(t_inicio.tv_nsec - t.tv_nsec)/1000;

    char* msg_rec = ler("st-0");

    temperatura_sist = atof(&msg_rec[3]);
    printf(">>>%f<<<\n", temperatura_sist);

    if (temperatura_user > temperatura_sist){
        msg_rec = ler("sh-0");
        haltura = atof(&msg_rec[3]);

        altera_altura();

        altera("ani%lf", 0.0);
        altera("ana%lf", 10.0);
    }
```

```
    if(temperatura_user < temperatura_sist){
        msg_rec = ler("sh-0");
        haltura = atof(&msg_rec[3]);

        altera_altura();

        altera("ana%lf", 0.0);
        altera("ani%lf", 10.0);
    }

                // Le a hora atual, coloca em t_fim
                clock_gettime(CLOCK_MONOTONIC ,&t_fim);

                // Calcula o tempo de resposta observado em microsegundos
                atraso_fim[amostra++] = 1000000*(t_fim.tv_sec - t.tv_sec)   +   (t_fim.tv_nsec -
t.tv_nsec)/1000;

                // Calcula inicio do proximo periodo
                t.tv_nsec += periodo;
                while (t.tv_nsec >= NSEC_PER_SEC) {
                        t.tv_nsec -= NSEC_PER_SEC;
                        t.tv_sec++;
                }
        }

        printf("Atrasos ateh o inicio da execucao incluem overhead,release jitter,
interferencias\n");
        printf("Tempo de resposta vai desde a chegada ateh a conclusao\n\n");
        printf("Atraso inicio          Tempo de resposta\n");

  FILE* dados_i;
  FILE* dados_f;
```

```
    dados_i = fopen("dados_inicio.txt", "w+");
    if(dados_i == NULL){
        printf("Erro, nao foi possivel abrir o arquivo\n");
        exit(1);
    }

    dados_f = fopen("dados_fim.txt", "w+");
    if(dados_f == NULL){
        printf("Erro, nao foi possivel abrir o arquivo\n");
        exit(1);
    }

            for( int i=0; i<N_AMOSTRAS; ++i){
                    printf("Atraso inicio=%4ldus     Tempo de resposta=%4ldus\n",
atraso_inicio[i], atraso_fim[i]);
        fprintf(dados_i, "%4ld \n", atraso_inicio[i]);
        fprintf(dados_f, "%4ld \n", atraso_fim[i]);
    }

    fclose(dados_i);
    fclose(dados_f);
}
```