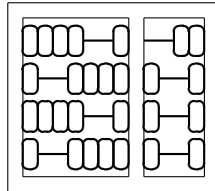


# UNIVERSIDADE ESTADUAL DE CAMPINAS

## INSTITUTO DE COMPUTAÇÃO



### SERVIDOR DE AGENDA BASEADO EM SOCKET UDP

*Relatório do segundo laboratório de MC823*

**Aluno:** Marcelo Keith Matsumoto    **RA:** 085937

**Aluno:** Tiago Chedraoui Silva    **RA:** 082941

#### **Resumo**

O protocolo UDP (User Datagram Protocol) é um protocolo do nível da camada de transporte. Ele é apropriado para aplicações tempo real, como telefonia e transferência de áudio e vídeo sobre a Internet. Dentre suas principais características temos: Não orientado à conexão, sem transferência de dados garantida, não guarda o estado de conexão.

Utilizando esse protocolo desenvolveu-se uma aplicação distribuída que simulava uma agenda para o mês de abril, com diversas funções (inserção de dados na agenda, recuperação de dados, remoção de dados), através da qual foi possível observar o funcionamento de uma comunicação via UDP entre um cliente e um servidor em máquinas diferentes. Para isso, a interação entre ambos ocorria através do envio de dados do cliente para o servidor, assim como na outra direção, utilizando sockets criados.

Por fim, foi realizada uma análise dos tempos de comunicação, processamento e total do programa, no qual concluiu-se que os tempos de processamento são significativamente maiores que os tempos de conexão.

# Sumário

<b>1</b>	<b>Motivação</b>	<b>2</b>
1.1	Teoria . . . . .	2
<b>2</b>	<b>Servidor de agenda</b>	<b>3</b>
2.1	Menu inicial . . . . .	3
2.1.1	Login . . . . .	3
2.1.2	Novo usuário . . . . .	3
2.2	Menu usuário . . . . .	3
2.2.1	Inserção de compromisso . . . . .	4
2.2.2	Remoção de compromisso . . . . .	4
2.2.3	Pesquisas . . . . .	4
<b>3</b>	<b>Ambiente de implementação</b>	<b>4</b>
<b>4</b>	<b>Tempos de comunicação e total</b>	<b>6</b>
4.1	Comparação de tecnologias . . . . .	6
<b>5</b>	<b>Conclusão</b>	<b>6</b>
<b>6</b>	<b>Anexo</b>	<b>6</b>

# 1 Motivação

Atualmente, com o crescente aumento de dispositivos móveis e computadores conectados à rede, o conhecimento da comunicação tem se tornado cada vez mais importante para quem trabalha na área de tecnologia. Um dos maiores exemplos da importância da comunicação entre computadores é em um sistema distribuído.

Um sistema distribuído é uma "coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente"[?]. Para que isso seja possível, diversos computadores estão interligados por uma rede de computadores, através da qual compartilham entre si objetos (como arquivos, informações, processamento, etc) e são responsáveis por manter uma consistência nesses objetos. Portanto, hoje, um servidor não é apenas um computador, mas sim vários computadores em locais diferentes que aparentam ser, para um usuário, um único sistema.

Com o objetivo de otimizar a comunicação entre computadores que venham a requerir os dados na rede, esse trabalho visa estudar o tempo de comunicação entre máquinas que utilizam sockets para tal finalidade.

## 1.1 Teoria

O UDP (User Datagram Protocol) é um protocolo do nível da camada de transporte [?]. Dentre suas principais características temos:

**Não orientado à conexão** Não há abertura de conexão para o envio de dados. Não introduzindo, portanto, atrasos para esta tarefa.

**Sem transferência de dados garantida** Não há garantia de entrega de dados livre de erros entre o processo emissor e receptor;

**Não estado de conexão** Não mantém estado da conexão, que implicaria em buffers de envio e recepção, números de sequência e reconhecimento.

**informações de controle** Tem informações de controle pequeno no cabeçalho.

**Sem controle de fluxo** Taxa de envio sem regulação ou controle de fluxo.

Cada um dos segmentos da camada transporte tem em seu cabeçalho campos denominado número de portas que indicam a qual processo o mesmo deve ser entregue, existindo um número de porta do emissor e o número de porta do receptor.

Possuindo as duas portas, pode-se realizar uma conexão entre elas conhecida por socket. Com o socket é possível diferenciar os canais utilizados por cada processo de aplicação.

O protocolo UDP é apropriado para aplicações tempo real, como telefonia e transferência de áudio e vídeo sobre a Internet.

Devido a importância do protocolo, este laboratório tem o objetivo de medir o tempo total e de comunicação de uma conexão TCP entre um cliente e um servidor.

## **2 Servidor de agenda**

O sistema implementado, uma agenda distribuída, se baseia numa comunicação cliente-servidor. Nele o servidor possui todas as informações da agenda que estão armazenadas em um banco de dados, assim como as opções de interações com os dados que são apresentadas aos clientes em formas de um menu. O cliente só escolhe alguma opção de interação com os dados de acordo com menu.

### **2.1 Menu inicial**

No menu inicial pode-se:

- Logar
- Criar novo usuário
- Sair

#### **2.1.1 Login**

O servidor pede ao usuário o nome de usuário, caso o nome estiver no banco de dados ele pede uma senha que é comparada ao valor do banco de dados, se o usuário não existir é avisado sobre a inexistência, se a senha não conferir é avisado que a senha não confere, caso contrário o usuário consegue logar no sistema, e o servidor recupera sua agenda (cada usuário possui sua agenda).

#### **2.1.2 Novo usuário**

O servidor pede um nome de usuário, o servidor verifica se o nome já não existe, se não existir pede a senha e armazena o usuário no sistema, assim como cria uma agenda vazia para o mesmo.

### **2.2 Menu usuário**

Dentre as possibilidades de interações para um usuário logado tem-se:

- Inserção de um compromisso que possui um nome, dia, hora, e minuto.
- Remoção de um compromisso através de seu nome
- Pesquisa de compromisso por dia

- Pesquisa de compromisso por dia e hora
- Ver todos os compromisso de mês de abril

### 2.2.1 Inserção de compromisso

O usuário deve fornecer o nome do compromisso, o dia, a hora e o minutos em que ele ocorrerá. Caso o compromisso seja possível de ser alocado o servidor avisa com um “OK”, se não for possível também é avisado de tal impossibilidade. Um compromisso é inserido ordenado na agenda se não existir um compromisso com mesmo horário.

### 2.2.2 Remoção de compromisso

O usuário deve fornecer o nome do compromisso que deve ser removido. Caso o compromisso seja encontrado ele é removido, caso contrário é dito que tal compromisso não existe. Se existirem dois compromissos de mesmo nome, o primeiro é removido. Logo é esperado que compromissos possuam nomes diferentes.

### 2.2.3 Pesquisas

O servidor faz um requerimento interativo, ou seja, se for selecionado a pesquisa por dia e hora, o servidor pergunta primeiramente o dia e depois a hora. Logo, é uma pesquisa em etapas no qual o servidor interage com nosso usuário.

## 3 Ambiente de implementação

O sistema de agenda foi implementado e executado nos seguintes sistemas operacionais :

- FC14 - Fedora Laughlin Linux 2.6.35.11

O sistema de agenda foi implementado na linguagem C. Para o armazenamento dos dados, utilizou-se arquivos. Cada usuário possui um arquivo, a sua agenda, no qual armazena-se o nome do compromisso, o dia, a hora e o minuto do mesmo. O sistema lê esse arquivo quando o usuário loga e transfere-o à memória principal, e a cada alteração na agenda o servidor atualiza as informações dos arquivos.

O servidor aceita diversas conexões de clientes, funcionando perfeitamente para interações em diferentes agendas, pois cada cliente possui além de um processo único, que foi criado em um fork, possui um ponteiro para sua agenda. Assim, o servidor consegue alterar todas as agendas independentemente.

O nosso sistema, além disso, apresenta transparência ao usuário. Os tipos de transparência a serem destacados são:

**Acesso:** Esconde as diferenças nas representações de dados e na invocação de funções ou métodos para facilitar a comunicação entre objetos da rede.

**Localização:** Esconde o local em que o objeto se encontra.

**Concorrência:** Esconde como as atividades são coordenadas entre os objetos para obter consistência em um nível mais alto.

Listaremos a seguir algumas funções implementadas de interação:

- Funções de interação com o banco de dados são:

```
1      /* Encontra usuario que ja esta cadastrado no servidor e verifica
2      senha*/
3      int findUser(char nome[], char pwd[]);
4
5      /* Insere novo usuario (nome e senha) no banco de dados*/
6      int newUser(char nome[], char senha[]);
7
8      /*Carrega agenda de usuario*/
9      int loadCal(User *user);
10
11     /*Salva agenda*/
12     int saveCal(User *user);
```

- Funções de interação com a agenda são:

```
1      /*Cria agenda para usuario*/
2      User * agenda_init(char nome[]);
3
4      /*Apaga agenda da memoria principal do servidor */
5      void user_destroy(User *u);
6
7      /*Insere compromisso na agenda*/
8      int set_task(int dia,int hora, int min,char task[], User *u);
9
10     /*Cria compromisso */
11     Agenda * task_init(int dia,int hora, int min,char task[]);
12
13     /*Retorna compromissos do mes de abril*/
14     int verMes(int new_fd, User *u);
15
16     /*Retorna compromissos do mes de abril em determinado dia*/
17     int verDia(int new_fd, User *u, int dia);
18
19     /*Retorna compromissos do mes de abril em determinado dia e
20     determinada hora*/
21     int verHora(int new_fd, User *u, int dia, int hora);
22
```

```

23  /*Remove compromisso da agenda pelo nome*/
24  int delTask( User *u, char nome[]);
25
26  /*Comapara data de compromissos*/
27  int compData(Agenda *newTasks, Agenda *tasks);

```

- Funções de interação servidor-cliente criadas foram:

```

1
2  /*Envia mensagem ao cliente*/
3  void sendStr(int sockfd, char str[]);
4
5  /*Le mensagem do cliente*/
6  int leOpcao(struct sockaddr_storage their_addr, int sockfd);
7
8  /*Apresenta opcoes de login ou criacao novo usuario*/
9  void menu(int new_fd, struct sockaddr_storage their_addr);
10
11 /*Apresenta opcoes de interacao com agenda*/
12 void menu2(int new_fd, struct sockaddr_storage their_addr, User *user);
13
14 /*Envia mensagem para servidor*/
15 void envia_pct( int sockfd, char s[], int size){

```

## 4 Tempos de comunicação e total

### 4.1 Comparação de tecnologias

## 5 Conclusão

## 6 Anexo

Listing 1: Agenda

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include "agenda.h"
4
5  /* INICIALIZA AGENDA
6   * gera o no cabeca e o devolve */
7  User * agenda_init(char nome[]) {
8      User *y = (User *) malloc(sizeof(User));
9      strcpy(y->name, nome);
10     y->tasks = NULL;
11     return y;
12 }
13
14 /* DESTROI AGENDA
15 * desaloca todos os nos */
16 void user_destroy(User *u) {
17     Agenda *next;
18     Agenda *a;
19
20     next = u->tasks;
21     free(u);
22     for (a = next; a != NULL; a = next) {
23         next = a->next;
24         free(a);
25     }
26 }

```

```

27
28 /* funcao booleana que verifica se a agenda esta vazia */
29 int agenda_vazia(User *a) {
30     return (a->tasks == NULL)?(1):(0);
31 }
32
33 /* remover um compromisso pelo seu nome */
34 int delTask( User *u, char nome[]) {
35     int cmp;
36     Agenda *a,*ant;
37     Agenda *newTask;
38
39     a=u->tasks;
40     if (a ==NULL) /* Agenda vazia*/
41         return 0;
42
43     /* primeiro no da cabeca */
44     if (cmp=strcmp(nome,a->task)==0) {
45         printf("sou eu!\n");
46         u->tasks=a->next;
47         free(a);
48         return 1;
49     }
50     ant=a;
51     /* Percorre a lista ligada procurando pelo compromisso a ser removido
        */
52     for (a=a->next; a != NULL; a =a->next) {
53         if( strcmp(nome,a->task)==0) {
54             ant->next=a->next;
55             free(a);
56             return 1;
57         }
58         ant=a;
59     }
60
61     return 0;
62 }
63
64 /* Aloca um novo no com as informacoes do novo compromisso, e o insere na
    ordenadamente na lista ligada
65 * Retorna 1: se o compromisso marcado foi inserido com sucesso
66 * Retorna 0: caso contrario */
67 int set_task(int dia,int hora,int min,char task[], User *u){
68     int cmp;
69     Agenda *a,*next,*ant;
70     Agenda *newTask = task_init(dia, hora,min,task);
71
72     next=u->tasks;
73     if (next ==NULL)
74         u->tasks=newTask;
75     else if (next!=NULL) { /* Agenda vazia? */
76         /* Ja eh a menor? */
77         if (compData(newTask,next)==1) {
78             printf("sou a menor!\n");
79             newTask->next=u->tasks;
80             u->tasks=newTask;
81             return 1;
82         }
83
84         /* insere ordenado usando insertion */
85         for (a = next; a != NULL; a = next) {
86             cmp=compData(newTask,a); /* Verifica se a data eh maior ou menor */
87             if (cmp==-1) { /* Se eh maior */
88                 ant=a;
89                 next = a->next;
90             }
91             else if (cmp==1) { /* Se eh menor */
92                 newTask->next = a;
93                 ant->next=newTask;
94                 break;
95             }
96             else { /* Se forem simultaneos, o compromisso nao eh inserido */
97                 free(newTask);
98                 return 0;
99             }
100         }
101         /*ultimo compromisso*/
102         ant->next=newTask;
103     }
104     return 1;
105 }
106
107 /*
108     Retorna:
109     1 se dia do novo compromisso eh menor
110     -1 se dia do novo compromisso eh maior
111     0 se dia do novo compromisso tem mesmo horario
112 */
113 int compData(Agenda *newTasks,Agenda *tasks) {
114
115     if(newTasks->dia < tasks->dia)
116         return ANTES;
117     else if (newTasks->dia > tasks->dia)
118         return DEPOIS;
119     else if (newTasks->hora < tasks->hora)
120         return ANTES;
121     else if (newTasks->hora > tasks->hora)
122         return DEPOIS;
123     else if (newTasks->min < tasks->min)
124         return ANTES;
125     else if (newTasks->min > tasks->min)
126         return DEPOIS;
127
128     return SIMULTANEO;
129 }
130
131 /* INICIALIZA COMPROMISSOS
132 * Aloca o no compromisso com as informacoes e o devolve o seu apontador
        */
133 Agenda * task_init(int dia,int hora,int min,char task[]) {
134     Agenda *newTask = (Agenda *) malloc(sizeof(Agenda));
135
136     newTask->dia=dia;
137     newTask->hora=hora;
138     newTask->min=min;
139     strcpy(newTask->task,task);
140     newTask->next=NULL;
141
142     return newTask;
143 }
144
145 /* Imprime todos os compromissos do mes envia para o cliente */
146 int verMes(int new_fd, User *u,struct sockaddr_storage their_addr){
147     Agenda *next,*a;
148     char mes[1000]="=== Mes de ABRIL ===\n";
149     char comp[1000],num[5];
150
151     next=u->tasks;
152
153     /* Percorre a lista ligada e concatena cada compromisso numa string */
154     for (a = next; a != NULL; a = a->next) {
155         cpComp(a,comp);
156         strcat(mes,comp);
157         strcpy(comp,"");
158     }
159     printf("%s",mes);

```



```

160     strcat(mes, "\nDigite m para voltar ao menu anterior ou q para sair\n");
161     sendMsg(new_fd, mes, their_addr); /* Envia para o cliente toda as
        informacoes */
162
163     return 0;
164 }
165
166 /* Dado um dia, a funcao retorna todos os compromissos daquele dia */
167 int verDia(int new_fd, User *u, int dia, struct sockaddr_storage
        their_addr){
168     Agenda *next, *a;
169     char mes[1000]="=== Mes de ABRIL ===\n";
170     char comp[1000];
171
172     next=u->tasks;
173
174     /* Percorre a lista ligada em busca dos compromissos daquele dia */
175     for (a = next; a != NULL; a=a->next) {
176         if(a->dia==dia){
177             cpComp(a, comp);
178             strcat(mes, comp);
179         }
180         else if(a->dia>dia)/* Dias ordenados - ultrapassou data */
181             break;
182     }
183
184     strcat(mes, "\nDigite m para voltar ao menu anterior ou q para sair\n");
185     sendMsg(new_fd, mes, their_addr); /* Envia para o cliente os
        compromissos */
186
187     return 0;
188 }
189
190 /* Dado um dia e uma hora, retorna para o cliente todos os compromissos
        correspondentes */
191 int verHora(int new_fd, User *u, int dia, int hora, struct
        sockaddr_storage their_addr){
192     Agenda *next, *a;
193     char mes[1000]="=== Mes de ABRIL ===\n";
194     char comp[1000];
195
196     next=u->tasks;
197
198     /* Percorre a lista ligada procurando pelos compromissos correspondentes
        a hora e o dia */
199     for (a = next; a != NULL; a=a->next) {
200         if(a->dia==dia && a->hora==hora){
201             cpComp(a, comp);
202             strcat(mes, comp);
203             strcat(mes, "\n");
204             strcpy(comp, ""); /* Limpeza da variavel */
205         }
206         else if(a->dia>dia)/* Dias ordenados - ultrapassou data */
207             break;
208     }
209
210     strcat(mes, "\nDigite m para voltar ao menu anterior ou q para sair\n");
211     sendMsg(new_fd, mes, their_addr); /* Envia para o cliente */
212     return 0;
213 }
214 }
215
216 /* Copia compromisso para visualizacao */
217 void cpComp(Agenda *a, char comp[]){
218     char num[5];
219
220     strcpy(comp, ""); /* Limpeza de variaveis */
221

```

```

222     strcat(comp, "\nCompromisso: ");
223     strcat(comp, a->task);
224
225     strcat(comp, "\nDia:");
226     snprintf(num, sizeof(num)-1, "%d", a->dia);
227     strcat(comp, num);
228
229     strcat(comp, "\nHora:");
230     snprintf(num, sizeof(num)-1, "%d", a->hora);
231     strcat(comp, num);
232
233     strcat(comp, "\nMin:");
234     snprintf(num, sizeof(num)-1, "%d", a->min);
235     strcat(comp, num);
236
237     return;
238 }

```

## Listing 2: banco de dados

```

1  #include <stdio.h>
2  #include <string.h>
3  #include "agenda.h"
4
5  /* Verifica USUARIOS
6   * compara no arquivo fp uma lista de usuarios
7   * se usuario existe retorna 1, senao 0 */
8  int findUser(char nome[], char pwd[])
9  {
10     char user [30], arq[20] = "";
11     FILE * pFile;
12
13     /* Formato agruivo: usuario\nsenha\n */
14     pFile = fopen("users.txt", "r"); /*arquivo com nome de usuarios*/
15
16     if (pFile == NULL) {
17         printf("\nFIND USER NULL FILE");
18         return 0;
19     }
20     else {
21
22         /*Le 100 caracteres ou ate o final da linha*/
23         while (fscanf(pFile, "%[^\\n]", user) != EOF)
24         {
25             fgetc(pFile);
26             fscanf(pFile, "%[^\\n]", pwd); /* senha do usuario, nao eh usado,
                somente para leitura do arquivo */
27             fgetc(pFile);
28             if (strcmp(user, nome) == 0) /* Verifica se o eh o usuario buscado
                */
29                 {
30                     fclose(pFile);
31                     /* Cria o arquivo do usuario, caso aquele nao exista */
32                     pFile = fopen(nome, "a");
33                     fclose(pFile);
34                     return 1; /* Devolve 1 se o usuario buscado foi encontrado no
                        arquivo users.txt */
35                 }
36             }
37         }
38         fclose(pFile);
39         return 0; /* Devolve 0 caso o usuario buscado nao esteja cadastrado */
40     }
41
42     /* Insere USUARIO
43     * Retorna 1: se usuario foi inserido
44     * Retorna 0: caso contrario*/

```

```

45 int newUser(char nome[], char senha[])
46 {
47     FILE * pFile;
48     char pwd[20], arq[20] = "";
49
50     if (findUser(nome, pwd) == 0) /* Verifica se o usuario que se deseja
        cadastrar ja existe */
51     {
52         pFile = fopen("users.txt", "a"); /*arquivo com nome de usuarios*/
53         if (pFile == NULL)
54             perror("Error opening file");
55         else
56         {
57             fseek(pFile, 0, SEEK_END); /* O novo usuario eh colocado no final
                do arquivo */
58             fputs(nome, pFile); /* Nome */
59             fputs("\n", pFile);
60             fputs(senha, pFile); /* Senha */
61             fputs("\n", pFile);
62             fclose(pFile);
63
64             /* Cria a agenda para o usuario */
65             pFile = fopen(nome, "w");
66             fclose(pFile);
67
68             return 1;
69         }
70     }
71
72     fclose(pFile);
73     return 0;
74 }
75
76 /*Le toda a agenda do usuario em arquivo e passa para memoria*/
77 int loadCal(User *user)
78 {
79     FILE * pFile;
80     char nome[20]="";
81     char dia[5], hora[5], min[5], task[100], arq[100]="";
82     Agenda *atual;
83     int i = 0; /*numero de compromissos*/
84
85     /*Abre agenda do usuario*/
86     strcpy(nome, user->name);
87     pFile = fopen(nome, "r"); /*arquivo com nome de usuarios*/
88     if (pFile == NULL){
89         printf("\nnome: %s --- %s", user->name, user->name[strlen(user->name) -
                1]);
90         printf("\nnome: %s", nome);
91         printf("\nERROR -- LOAD CAL NULL FILE\n");
92         return i;
93     }
94     else
95     {
96         /*Primeiro evento*/
97         if (fscanf(pFile, "%[^\\n]", task) != EOF)
98             /*evento*/
99             fgetc(pFile);
100             fscanf(pFile, "%[^\\n]", dia); /*dia*/
101             fgetc(pFile);
102             fscanf(pFile, "%[^\\n]", hora); /*hora*/
103             fgetc(pFile);
104             fscanf(pFile, "%[^\\n]", min); /*minuto*/
105             fgetc(pFile);
106             printf("\nInserindo:%s %s %s %s", task, dia, hora, min);
107             user->tasks = task_init(atoi(dia), atoi(hora), atoi(min), task);
108             atual = user->tasks;
109             i++;
110
111             /* Percorre o arquivo lendo os compromissos */
112             while (fscanf(pFile, "%[^\\n]", task) != EOF)
113             {
114                 fgetc(pFile);
115                 fscanf(pFile, "%[^\\n]", dia); /*dia*/
116                 fgetc(pFile);
117                 fscanf(pFile, "%[^\\n]", hora); /*hora*/
118                 fgetc(pFile);
119                 fscanf(pFile, "%[^\\n]", min); /*minuto*/
120                 fgetc(pFile);
121                 printf("\n\\n\\n\\n %s", arq);
122                 printf(" task %s\\n\\n", task);
123                 /* Cria um novo no na lista ligada com as informacoes do
                    compromisso */
124                 atual->next = task_init(atoi(dia), atoi(hora), atoi(min), task);
125                 atual = atual->next;
126                 i++;
127                 strcpy(arq, "");
128                 strcpy(task, "");
129
130             }
131         }
132         fclose(pFile);
133         return i;
134     }
135
136     fclose(pFile);
137
138
139     return i;
140 }
141
142
143 /* Insere Compromissos na agenda, passando da memoria para arquivo
144  * Retorna 1: se compromissos inseridos
145  * Retorna 0: caso contrario*/
146 int saveCal(User *user)
147 {
148     FILE * pFile;
149     char pwd[20], arq[20] = "", nome[20];
150     Agenda *atual;
151
152     strcpy(nome, user->name);
153     pFile = fopen(nome, "w"); /*arquivo com nome de usuarios*/
154     if (pFile == NULL){
155         printf("\nNULL - SaveCal\n");
156         return 0;
157     }
158     else
159     {
160         /* Percorre a lista ligada e imprime as infomacoes de cada no no
            arquivo */
161         for (atual = user->tasks; atual != NULL; atual = atual->next)
162         {
163             fputs(atual->task, pFile);
164             fprintf(pFile, "\\n\\n\\n\\n\\n\\n", atual->dia, atual->hora, atual->
                min);
165         }
166         fclose(pFile);
167         return 1;
168     }
169
170     fclose(pFile);
171     return 0;
172 }

```

### Listing 3: Servidor

```

1  /*
2  ** server.c -- a stream socket server demo
3  */
4  #include "agenda.h"
5  #include <sys/time.h>
6
7  /* Estrutura para analise de tempo em microsegundos */
8  struct timeval first, second, lapsed;
9  struct timezone tzp;
10
11 void serverTimeRecv(struct timeval first, struct timeval second){
12
13     double t2=first.tv_sec+(first.tv_usec/1000000.0);
14     double t3=second.tv_sec+(second.tv_usec/1000000.0);
15
16     FILE * pFile;
17     pFile = fopen("serverTime.dat", "a"); /*arquivo com tempos do servidor
18                                     */
19
20     if (pFile == NULL)
21         return ;
22
23     /* if (first.tv_usec > second.tv_usec) {
24         second.tv_usec += 1000000;
25         second.tv_sec--;
26     } */
27
28     fseek(pFile, 0, SEEK_END);
29     fprintf(pFile, "%f \n", t3-t2);
30     fclose(pFile);
31
32 }
33
34
35 void sigchld_handler(int s)
36 {
37     while(waitpid(-1, NULL, WNOHANG) > 0);
38 }
39
40 // get sockaddr, IPv4 or IPv6:
41 void *get_in_addr(struct sockaddr *sa)
42 {
43     if (sa->sa_family == AF_INET) {
44         return &(((struct sockaddr_in*)sa)->sin_addr);
45     }
46
47     return &(((struct sockaddr_in6*)sa)->sin6_addr);
48 }
49
50 int main(void)
51 {
52     int sockfd, new_fd; // listen on sock_fd, new connection on new_fd
53     struct addrinfo hints, *servinfo, *p;
54     struct sockaddr_storage their_addr; // connector's address information
55     socklen_t addr_len;
56     struct sigaction sa;
57     int yes=1;
58     char s[INET6_ADDRSTRLEN], tempo[5], str[5];
59     int rv;
60     char buf[ MAXDATASIZE];
61
62     memset(&hints, 0, sizeof hints);
63     hints.ai_family = AF_UNSPEC;
64     hints.ai_socktype = SOCK_DGRAM; // TCP stream sockets
65     hints.ai_flags = AI_PASSIVE; // use my IP

```

```

66
67     if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {
68         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
69         return 1;
70     }
71
72     // loop through all the results and bind to the first we can
73     for(p = servinfo; p != NULL; p = p->ai_next) {
74         if ((sockfd = socket(p->ai_family, p->ai_socktype,
75             p->ai_protocol)) == -1) {
76             perror("server: socket");
77             continue;
78         }
79
80         if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
81             close(sockfd);
82             perror("server: bind");
83             continue;
84         }
85
86         break;
87     }
88
89     if (p == NULL) {
90         fprintf(stderr, "server: failed to bind\n");
91         return 2;
92     }
93
94     freeaddrinfo(servinfo); // all done with this structure
95
96     printf("server: waiting for connections...\n");
97     int numbytes;
98     while(1) { // main accept() loop
99         addr_len = sizeof their_addr;
100         if ((numbytes = recvfrom(sockfd, buf, MAXDATASIZE-1, 0,
101             (struct sockaddr *)&their_addr, &addr_len)) == -1) {
102             perror("recvfrom");
103             exit(1);
104         }
105         printf("meu their adress tem:%d\n", their_addr);
106
107         inet_ntop(their_addr.ss_family,
108             get_in_addr((struct sockaddr *)&their_addr),
109             s, sizeof s);
110         printf("server: got connection from %s\n", s);
111
112         // strcpy(str, "0123"); // tamanho de um inteiro bytes
113         // recv(new_fd, tempo, 5, 0);
114         // gettimeofday (&first, &tzp);
115         menu(sockfd, their_addr);
116         // gettimeofday (&second, &tzp);
117         // sendto(sockfd, str, strlen(str), 0, (struct sockaddr *)&their_addr,
118             addr_len);
119         // serverTimeRecv(first, second);
120     }
121     close(new_fd); // parent doesn't need this
122     return 0;
123 }
124
125 void menu(int new_fd, struct sockaddr_storage their_addr){
126     User *user;
127     char nome[20], senha[20], pwd[20], again[1];
128     char str[1000];
129     while(1){
130         sendMsg(new_fd, "Escolha uma opcao:\n\
131             Opcao 1 - Entrar como um usuario\n\
132             Opcao 2 - Criar um usuario\n\

```

```

133         Opcao q - Sair\n0", their_addr);
134     switch(leOpcao(their_addr, new_fd)){
135     case 1:
136         /* Ler usuario */
137         sendMsg(new_fd, "Digite o nome do usuario a ser buscado:\0",
138             their_addr);
139         leString(their_addr, new_fd, nome);
140         /* Busca nome no banco de dados */
141         if(findUser(nome,pwd)){
142             /*verifica senha*/
143             sendMsg(new_fd, "Digite a senha do usuario:\0",their_addr);
144             leString(their_addr, new_fd, senha);
145
146             if(!strcmp(senha,pwd)){
147                 user=agenda_init(nome);
148                 menu2(new_fd, their_addr, user);
149             }
150         }else{
151             sendMsg(new_fd, "Senha nao confere! Digite m para voltar ou q para
152                 sair:\0", their_addr);
153             leString(their_addr, new_fd, again);
154             if(strcmp("q",again)==0)
155                 exit(1);
156         }
157     }else{
158         sendMsg(new_fd, "Usuario inexistente! Digite m para voltar ou q para
159             sair:\0", their_addr);
160         leString(their_addr, new_fd, again);
161     }
162     /*saida do programa*/
163     if(strcmp("q",again)==0)
164         exit(1);
165
166     break;
167 }
168 break;
169
170 case 2:
171     /* Criar um usuario */
172     sendMsg(new_fd, "Digite o nome do usuario a ser criado:\0",
173         their_addr);
174     leString(their_addr, new_fd, nome);
175     sendMsg(new_fd, "Digite a senha do usuario:\0", their_addr);
176     leString(their_addr, new_fd, senha);
177
178     /* Verifica se nome ja existe */
179     if(newUser(nome,senha)==1){
180         user=agenda_init(nome);
181         menu2(new_fd, their_addr, user);
182     }
183     else{
184         sendMsg(new_fd, "Usuario ja existente! Digite m para voltar ou q para
185             sair:\0", their_addr);
186         leString(their_addr, new_fd, again);
187     }
188     /*saida do programa */
189     if(strcmp("q",again)==0)
190         close(new_fd); // mata conexao com cliente
191     exit(1);
192 }
193 break;
194 default:
195     return;

```

```

196     break;
197 }
198 }
199 return;
200
201 }
202
203 void menu2(int new_fd, struct sockaddr_storage their_addr, User *user){
204     char nome[20]="", dia[5]="", hora[5]="", minuto[5]="", task[1000]="",
205         again[1]="";
206     char str[1000]="";
207     char menu[1000]="Escolha uma opcao:\n\
208         Opcao 1 - Marcar um compromisso\n\
209         Opcao 2 - Desmarcar um compromisso\n\
210         Opcao 3 - Obter um compromisso marcado para um horario
211         de um dia\n\
212         Opcao 4 - Obter todos os compromissos marcados para um
213         dia\n\
214         Opcao 5 - Obter todos os compromissos do mes\n\
215         Opcao 6 - Voltar\0";
216
217     /*Recupera agenda do usuario, apos login*/
218     loadCal(user);
219
220     while(1){
221         sendMsg(new_fd, "Escolha uma opcao:\n\
222             Opcao 1 - Marcar um compromisso\n\
223             Opcao 2 - Desmarcar um compromisso\n\
224             Opcao 3 - Obter um compromisso marcado para um horario
225             de um dia\n\
226             Opcao 4 - Obter todos os compromissos marcados para um
227             dia\n\
228             Opcao 5 - Obter todos os compromissos do mes\n\
229             Opcao 6 - Voltar\0", their_addr);
230
231         switch(leOpcao(their_addr, new_fd)){
232         case 1:
233             /* Marcar um compromisso */
234             sendMsg(new_fd, "Digite o nome do compromisso:\0", their_addr);
235             leString(their_addr, new_fd, task);
236             sendMsg(new_fd, "Digite o dia do compromisso:\0", their_addr);
237             leString(their_addr, new_fd, dia);
238             sendMsg(new_fd, "Digite o hora do compromisso:\0", their_addr);
239             leString(their_addr, new_fd, hora);
240             sendMsg(new_fd, "Digite os minutos do compromisso:\0", their_addr);
241             leString(their_addr, new_fd, minuto);
242             set_task(atoi(dia), atoi(hora), atoi(minuto), task, user);
243             verMes(new_fd,user,their_addr);
244
245             /*Se m retorna ao menu, se q salva agenda sai*/
246             leString(their_addr, new_fd,again);
247             if(strcmp("q",again)==0) {
248                 saveCal(user);
249                 user_destroy(user);
250                 close(new_fd); // mata conexao com cliente
251                 exit(1);
252             }
253             break;
254         case 2:
255             /* Desmarcar um compromisso */
256             sendMsg(new_fd, "Digite o nome do compromisso a ser desmarcado:\0",
257                 their_addr);
258             leString(their_addr, new_fd, str);
259             if(delTask(user, str))
260                 sendMsg(new_fd, "\nCompromisso desmarcado\nDigite m para voltar
261                     ao menu anterior ou q para sair\n\0", their_addr);
262             else

```

```

256     sendMsg(new_fd, "\nNao foi encontrado nenhum compromisso
        registrado com esse nome\nDigite m para voltar ao menu
        anterior ou q para sair\n0", their_addr);

257
258     /*Se m retorna ao menu, se q salva agenda sai*/
259     leString(their_addr, new_fd, again);
260     if(strcmp("q", again) == 0) {
261         saveCal(user);
262         user_destroy(user);
263         close(new_fd); // mata conexao com cliente
264         exit(1);
265     }
266     break;
267     case 3:
268         /* Obter compromissos de um dia em determinada hora */
269         sendMsg(new_fd, "Digite o dia:\0", their_addr);
270         leString(their_addr, new_fd, dia);
271         sendMsg(new_fd, "Digite as horas:\0", their_addr);
272         leString(their_addr, new_fd, hora);
273         verHora(new_fd, user, atoi(dia), atoi(hora), their_addr);
274
275         /*Se m retorna ao menu, se q salva agenda sai*/
276         leString(their_addr, new_fd, again);
277         if(strcmp("q", again) == 0) {
278             saveCal(user);
279             user_destroy(user);
280             close(new_fd); // mata conexao com cliente
281             exit(1);
282         }
283         break;
284     case 4:
285         /* Obter todos os compromissos marcados para um dia */
286         sendMsg(new_fd, "Digite o dia:\0", their_addr);
287         leString(their_addr, new_fd, dia);
288         verDia(new_fd, user, atoi(dia), their_addr);
289
290         /*Se m retorna ao menu, se q salva agenda sai*/
291         leString(their_addr, new_fd, again);
292         if(strcmp("q", again) == 0) {
293             saveCal(user);
294             user_destroy(user);
295             close(new_fd); // mata conexao com cliente
296             exit(1);
297         }
298         break;
299     case 5:
300         /* Obter todos os compromissos do mes */
301         verMes(new_fd, user, their_addr);
302
303         /*Se m retorna ao menu, se q salva agenda sai*/
304         leString(their_addr, new_fd, again);
305         if(strcmp("q", again) == 0) {
306             saveCal(user);
307             user_destroy(user);
308             close(new_fd); // mata conexao com cliente
309             exit(1);
310         }
311         break;
312     default:
313         saveCal(user);
314         user_destroy(user);
315         return;
316         break;
317     }
318 }
319 }
320
321 void leString(struct sockaddr_storage their_addr, int sockfd, char string
    []) {
322
323     int numbytes;
324     char s[INET6_ADDRSTRLEN];
325     socklen_t addr_len = sizeof their_addr;
326
327     if ((numbytes = recvfrom(sockfd, string, MAXDATASIZE-1, 0,
328         (struct sockaddr *)&their_addr, &addr_len)) == -1) {
329         perror("recvfrom");
330         exit(1);
331     }
332     printf("listener: got packet from %s\n",
333         inet_ntop(their_addr.ss_family,
334             get_in_addr((struct sockaddr *)&their_addr),
335             s, sizeof s));
336     printf("listener: packet is %d bytes long\n", numbytes);
337     printf("listener: packet contains \"%s\\n\"", string);
338     return;
339 }
340
341 int leOpcao(struct sockaddr_storage their_addr, int sockfd) {
342     int numbytes;
343     char buf[MAXDATASIZE];
344     char s[INET6_ADDRSTRLEN];
345     socklen_t addr_len = sizeof their_addr;
346
347     if ((numbytes = recvfrom(sockfd, buf, MAXDATASIZE-1, 0,
348         (struct sockaddr *)&their_addr, &addr_len)) == -1) {
349         perror("recvfrom");
350         exit(1);
351     }
352     printf("listener: got packet from %s\n",
353         inet_ntop(their_addr.ss_family,
354             get_in_addr((struct sockaddr *)&their_addr),
355             s, sizeof s));
356     printf("listener: packet is %d bytes long\n", numbytes);
357     buf[numbytes] = '\0';
358     printf("listener: packet contains \"%s\\n\"", buf);
359     printf("%c", buf[0]);
360     return atoi(buf);
361 }
362
363 void sendMsg(int new_fd, char str[], struct sockaddr_storage their_addr) {
364     if (sendto(new_fd, str, strlen(str) + 1, 0, (struct sockaddr *)&
365         their_addr, sizeof their_addr) == -1)
366         perror("send");
367 }

```

Listing 4: Cliente

```

1 /*
2 ** client.c -- a stream socket client demo
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <unistd.h>
8 #include <errno.h>
9 #include <string.h>
10 #include <netdb.h>
11 #include <sys/types.h>
12 #include <netinet/in.h>
13 #include <sys/socket.h>
14 #include <arpa/inet.h>
15 #include <sys/time.h>

```

```

16
17 #define PORT "35555" // the port client will be connecting to
18
19 #define MAXDATASIZE 1000 // max number of bytes we can get at once
20 char opcao[256];
21
22 /* Estrutura para analise de tempo em microssegundos */
23 struct timeval first, second, lapsed;
24 struct timezone tzp;
25
26 void connecttime(struct timeval first, struct timeval second){
27
28     double t1=first.tv_sec+(first.tv_usec/1000000.0);
29     double t4=second.tv_sec+(second.tv_usec/1000000.0);
30
31     FILE * pFile;
32     pFile = fopen("conntime.dat", "a"); /*arquivo com tempos do servidor*/
33
34     if (pFile == NULL)
35         return ;
36
37     /* if (first.tv_usec > second.tv_usec) {
38         second.tv_usec += 1000000;
39         second.tv_sec--;
40     } */
41
42     fseek(pFile, 0, SEEK_END);
43     fprintf(pFile, "%f \n", t4-t1);
44     fclose(pFile);
45
46     return;
47 }
48
49
50 void clienteTimeRecv(struct timeval first, struct timeval second){
51
52     double t1=first.tv_sec+(first.tv_usec/1000000.0);
53     double t4=second.tv_sec+(second.tv_usec/1000000.0);
54
55     FILE * pFile;
56     pFile = fopen("clientTime.dat", "a"); /*arquivo com tempos do servidor
57         */
58
59     if (pFile == NULL)
60         return ;
61
62     /* if (first.tv_usec > second.tv_usec) {
63         second.tv_usec += 1000000;
64         second.tv_sec--;
65     } */
66
67     fseek(pFile, 0, SEEK_END);
68     fprintf(pFile, "%f \n", t4-t1);
69     fclose(pFile);
70
71     return;
72 }
73
74 void envia_pct( int sockfd, char s[], int size){
75     if (( send(sockfd, s ,size, 0)) == -1) {
76         perror("talker: sendto");
77         exit(1);
78     }
79     return;
80 }
81
82 // get sockaddr, IPv4 or IPv6:

```

```

83 void *get_in_addr(struct sockaddr *sa)
84 {
85     if (sa->sa_family == AF_INET) {
86         return &(((struct sockaddr_in*)sa)->sin_addr);
87     }
88
89     return &(((struct sockaddr_in6*)sa)->sin6_addr);
90 }
91
92 int main(int argc, char *argv[])
93 {
94     int sockfd, numbytes;
95     char buf[MAXDATASIZE]="";
96     struct addrinfo hints, *servinfo, *p;
97     int rv;
98     char s[INET6_ADDRSTRLEN]="", tempo[5], str[5];
99
100     if (argc != 2) {
101         fprintf(stderr, "usage: client hostname\n");
102         exit(1);
103     }
104
105     memset(&hints, 0, sizeof hints);
106     hints.ai_family = AF_UNSPEC;
107     hints.ai_socktype = SOCK_DGRAM;
108
109     if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
110         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
111         return 1;
112     }
113
114     gettimeofday (&first, &tzp);
115     // loop through all the results and connect to the first we can
116     for(p = servinfo; p != NULL; p = p->ai_next) {
117         if ((sockfd = socket(p->ai_family, p->ai_socktype,
118             p->ai_protocol)) == -1) {
119             perror("client: socket");
120             continue;
121         }
122
123         if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
124             close(sockfd);
125             perror("client: connect");
126             continue;
127         }
128         break;
129     }
130
131     if (p == NULL) {
132         fprintf(stderr, "client: failed to connect\n");
133         return 2;
134     }
135
136     inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
137         s, sizeof s);
138     printf("client: connecting to %s\n", s);
139     gettimeofday (&second, &tzp);
140     connecttime(first, second);
141
142     freeaddrinfo(servinfo); // all done with this structure
143     int size;
144
145     /* Teste de tempo */
146     strcpy(str, "0123"); // tamanho de um inteiro bytes
147     gettimeofday (&first, &tzp);
148     send(sockfd, str , strlen(str), 0);
149
150     while(1){

```

```

151  /* Esperando resposta do servidor*/
152  if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
153      perror("recv");
154      exit(1);
155  }
156
157
158      system("clear");
159      printf("\n%s\n",buf); //client received
160
161      /* Espera resposta do servidor*/
162      strcpy(opcao,"");
163      scanf("%s[%^n]", opcao );
164      getchar();
165
166      envia_pkt(sockfd, opcao ,strlen(opcao) + 1);
167
168      if(strcmp("q",opcao)==0) {
169          break;
170      }
171  }
172
173      recv(sockfd, tempo, MAXDATASIZE-1, 0);
174      gettimeofday (&second, &tzp);
175
176      clienteTimeRecv(first,second);
177
178      close(sockfd);
179
180      return 0;
181  }

```

Listing 5: Cabeçalhos Banco de dados

```

1  #ifndef BD_H_
2  #define BD_H_
3
4  int findUser(char nome[], char pwd[]);
5  int newUser(char nome[], char senha[]);
6  int loadCal();
7  int saveCal();
8
9  #endif /* BD_H_ */

```

Listing 6: Cabeçalhos Agenda

```

1  #ifndef AGENDA_H_
2  #define AGENDA_H_
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <netinet/in.h>
12 #include <netdb.h>
13 #include <arpa/inet.h>
14 #include <sys/wait.h>
15 #include <signal.h>
16 #include "bd.h"
17
18 #define PORT "35555" // the port users will be connecting to
19 #define MAXDATASIZE 1000
20 #define BACKLOG 10 // how many pending connections queue will hold

```

```

21
22 /*Comaracao de compromissos*/
23 #define ANTES 1 // how many pending connections queue will hold
24 #define DEPOIS -1 // how many pending connections queue will hold
25 #define SIMULTANEO 0 // how many pending connections queue will hold
26
27
28 typedef struct agenda {
29     struct agenda *next;
30     int dia;
31     int hora, min;
32     char task[256];
33 } Agenda;
34
35
36 typedef struct user {
37     struct agenda *tasks;
38     char name[20];
39 } User;
40
41 /* Funcoes */
42 void menu(int new_fd, struct sockaddr_storage their_addr);
43 void menu2(int new_fd, struct sockaddr_storage their_addr, User *user);
44 void sendStr(int sockfd, char str[]);
45 void sendMsg(int new_fd, char str[], struct sockaddr_storage their_addr);
46 int leOpcao(struct sockaddr_storage their_addr, int sockfd);
47 void leString(struct sockaddr_storage their_addr, int sockfd, char
    string[]);
48 User * agenda_init(char nome[]);
49 void user_destroy(User *u);
50 int agenda_vazia(User *a);
51 int compData(Agenda *newTasks, Agenda *tasks);
52 int set_task(int dia, int hora, int min, char task[], User *u);
53 Agenda * task_init(int dia, int hora, int min, char task[]);
54 int verMes(int new_fd, User *u, struct sockaddr_storage their_addr);
55 int verDia(int new_fd, User *u, int dia, struct sockaddr_storage
    their_addr);
56 int verHora(int new_fd, User *u, int dia, int hora, struct
    sockaddr_storage their_addr);
57 int delTask( User *u, char nome[]);
58 void cpComp(Agenda *a, char comp[]);
59 #endif /*CONJUNTO_H_*/

```

Listing 7: Makefile

```

1  CC = gcc
2  CFLAGS = -g -ggdb
3  LIBS = -lm
4
5  OBJJS_C = client.o
6  OBJJS_S = agenda.o server.o bd.o
7  ECHO_S = echo_server.c
8  ECHO_C = echo_client.c
9
10 EXEC_C_ECHO = echoc
11 EXEC_S_ECHO = echos
12 EXEC_C = c
13 EXEC_S = s
14
15 all:
16     make c
17     make s
18
19 c: $(OBJJS_C)
20     $(CC) $(CFLAGS) $(OBJJS_C) $(LIBS) -o $(EXEC_C)
21
22 s: $(OBJJS_S)

```

```
23 $(CC) $(CFLAGS) $(OBSJS_S) $(LIBS) -o $(EXEC_S)
24
25
26 run: all
27 ./$(EXEC) $(ATB)
28
29 $(OBSJS): %.o: %.c
30 $(CC) -c $(CFLAGS) $<
31
32 echo:
```

```
33 $(CC) $(CFLAGS) $(ECHO_C) $(LIBS) -o $(EXEC_C_ECHO)
34 $(CC) $(CFLAGS) $(ECHO_S) $(LIBS) -o $(EXEC_S_ECHO)
35
36 clean:
37 rm -f $(OBSJS_C) $(OBSJS_S) $(EXEC_C) $(EXEC_S) *~
38
39 reset: clean
40 rm -f $(FILES)
```