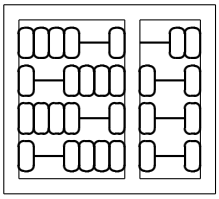
UNIVERSIDADE ESTADUAL DE CAMPINAS

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 usas 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 realizado uma análise dos tempo de 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

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"[1]. 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 [4]. Dentre usas 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 seqüência 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:

```
/* Encontra usuario que ja esta cadastrado no servidor e verifica
senha*/
int findUser(char nome[], char pwd[]);

/* Insere novo usuario (nome e senha) no banco de dados*/
int newUser(char nome[], char senha[]);

/*Carrega agenda de usuario*/
int loadCal(User *user);

/*Salva agenda*/
int saveCal(User *user);
```

- Funções de interação com a agenda são:

```
/*Cria agenda para usuario*/
User * agenda_init(char nome[]);

/*Apaga agenda da memoria principal do servidor */
void user_destroy(User *u);

/*Insere compromisso na agenda*/
int set_task(int dia,int hora, int min,char task[], User *u);

/*Cria compromisso */
Agenda * task_init(int dia,int hora, int min,char task[]);

/*Retorna compromissos do mes de abril*/
int verMes(int new_fd, User *u);

/*Retorna compromissos do mes de abril em determinado dia*/
int verDia(int new_fd, User *u, int dia);

/*Retorna compromissos do mes de abril em determinado dia e
determinada hora*/
int verHora(int new_fd, User *u, int dia, int hora);

```

```
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

Aplicamos o cálculo de tempo ao programa principal de forma a obtermos o tempo total, tempo de comunicação e os tempos da execução de cada função. Para o tempo total, no cliente pega-se o tempo antes do primeiro send e após o último recv. Para o tempo de comunicação, pega-se o tempo total e subtrai-se o tempo de processamento do servidor, que é depois do primeiro recv e antes do último send.

Para o tempo total das funções obteu-se o tempo de inserir um compromisso, remover o compromisso, ver a agenda do mês, ver a agenda de um dia, ver a agenda de uma hora. Os dados e os testes estão exemplificados nas tabelas **??**, **??**, **??**, **??** e **??**.

O resultado obtido para 100 valores foi:

Note que os tempos das tabelas **??**, **??** e **??** estão, respectivamente, em ordem decrescente de tamanho. Isso se deve ao fato de que a operação 3 (tabela **??**) o programa busca pelo dia, hora e minuto do compromisso, fazendo com que o número de comparações feitas seja maior que as operações **??** e **??**, tornando aquela operação mais lenta. O mesmo pode afirmar entre as operações 4 e 5, correspondentes às tabelas **??** e **??**, respectivamente.

Valor	Tempo
Max	?? ms
Min	?? ms
Média	?? ms
Desvio	?? ms

(a) Tempo total

Valor	Tempo
Max	0.526 ms
Min	0.116 ms
Média	0.200 ms
Desvio	0.002 ms

(b) Tempo de comunicação

Tabela I: Conexão e fechamento de conexão com servidor

Valor	Tempo
Max	?? ms
Min	?? ms
Média	?? ms
Desvio	?? ms

(a) Tempo total

Valor	Tempo
Max	0.502 ms
Min	0.113 ms
Média	0.200 ms
Desvio	0.001 ms

(b) Tempo de comunicação

Tabela II: Conexão, login na conta, inserção de compromisso e fechamento de conexão com servidor

Valor	Tempo
Max	?? ms
Min	?? ms
Média	?? ms
Desvio	?? ms

(a) Tempo total

Valor	Tempo
Max	1.124 ms
Min	0.111 ms
Média	0.207 ms
Desvio	0.009 ms

(b) Tempo de comunicação

Tabela III: Conexão, login na conta, ver compromissos de determinado dia, hora e minuto e fechamento de conexão com servidor

Valor	Tempo
Max	?? ms
Min	?? ms
Média	?? ms
Desvio	?? ms

(a) Tempo total

Valor	Tempo
Max	3.154 ms
Min	0.110 ms
Média	0.251 ms
Desvio	0.005 ms

(b) Tempo de comunicação

Tabela IV: Conexão, login na conta, ver compromissos de determinado dia e hora e fechamento de conexão com servidor

Valor	Tempo
Max	?? ms
Min	?? ms
Média	?? ms
Desvio	?? ms

(a) Tempo de total

Valor	Tempo
Max	3.834 ms
Min	0.117 ms
Média	0.241 ms
Desvio	0.001 ms

(b) Tempo de comunicação

Tabela V: Conexão, login na conta, ver todos os compromissos do mês e fechamento de conexão com servidor

4.1 Comparação de tecnologias

5 Conclusão

Referências

[1] Tanenbaum, Andrew S. e Maarten Van Steen Distributed Systems: Principles and Paradigms. Prentice Hall.

[2] Brian "Beej Jorgensen"Hall Beej's Guide to Network Programming - Using Internet Sockets . Disponível em *http://beej.us/guide/bgnet/*, [Último acesso: 07/04/2011].

[3] Mike Muuss Packet Internet Grouper (Groper) . Disponível em *http://linux.die.net/man/8/ping*, [Último acesso: 10/04/2011].

[4] J. Kurose e K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson Addison Wesley, 3 ed., 2005.

6 Anexo

Listing 1: Agenda

```
#include <stdlib.h>
#include <string.h>
#include "agenda.h"

/* INICIALIZA AGENDA
 * gera o no cabeca e o devolve */
User * agenda_init(char nome[]) {
  User *y = (User *) malloc(sizeof(User));
  strcpy(y->name,nome);
  y->tasks = NULL;
  return y;
}

/* DESTROI AGENDA
 * desaloca todos os nos */
void user_destroy(User *u) {
  Agenda *next;
  Agenda *a;

  next = u->tasks;
  free(u);
  for (a = next; a != NULL; a = next) {
    next = a->next;
    free(a);
  }
}

/* funcao booleana que verifica se a agenda esta vazia */
int agenda_vazia(User *a) {
  return (a->tasks == NULL)?(1):(0);
}

/* remover um compromisso pelo seu nome */
int delTask( User *u, char nome[]){
  int cmp;
  Agenda *a,*ant;
  Agenda *newTask;

  a=u->tasks;
  if (a ==NULL) /* Agenda vazia*/
    return 0;

  /* primeiro no da cabeca */
  if(cmp=strcmp(nome,a->task)==0){
    printf("sou eu!\n");
    u->tasks=a->next;
    free(a);
    return 1;
  }
  ant=a;
  /* Percorre a lista ligada procurando pelo compromisso a ser removido
       */
  for (a=a->next; a != NULL; a =a->next) {
    if( strcmp(nome,a->task)==0){
      ant->next=a->next;
      free(a);
      return 1;
    }
    ant=a;
  }

  return 0;
}

/* Aloca um novo no com as informacoes do novo compromisso, e o insere na
      ordenadamente na lista ligada
 * Retorna 1: se o compromisso marcado foi inserido com sucesso
 * Retorna 0: caso contrario */
int set_task(int dia,int hora,int min,char task[], User *u){
  int cmp;
  Agenda *a,*next,*ant;
  Agenda *newTask = task_init(dia, hora,min,task);

  next=u->tasks;
  if (next ==NULL)
    u->tasks=newTask;
  else if(next!=NULL){ /* Agenda vazia? */
    /* Ja eh a menor? */
    if(compData(newTask,next)==1){
      printf("sou a menor!\n");
      newTask->next=u->tasks;
      u->tasks=newTask;
      return 1;
    }

    /* insere ordenado usando insertion */
    for (a = next; a != NULL; a = next) {
      cmp=compData(newTask,a); /* Verifica se a data eh maior ou menor */
      if(cmp==-1){ /* Se eh maior */
  ant=a;
  next = a->next;
      }
      else if(cmp==1){ /* Se eh menor */
  newTask->next = a;
  ant->next=newTask;
  break;
      }
      else{ /* Se forem simultaneos, o compromisso nao eh inserido */
  free(newTask);
  return 0;
      }
    }
    /*ultimo compromisso*/
    ant->next=newTask;
  }
  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 correpondentes
          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 compromiso para visuzalizacao */
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 uxuario 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\narq %s", arq);
```

```
        printf("  task %s\n\n", task);
        /* Cria um novo no na lista ligada com as informacoes do
               compromisso */
        atual->next = task_init(atoi(dia), atoi(hora), atoi(min), task);
        atual = atual->next;
        i++;
        strcpy(arq, "");
        strcpy(task, "");

      }
    }
    fclose(pFile);
    return i;
  }


  fclose(pFile);


  return i;
}

/* Insere Compromissos na agenda, passando da memoria para arquivo
 * Retorna 1: se compromissos inseridos
 * Retorna 0: caso contrario*/
int saveCal(User *user)
{
  FILE * pFile;
  char pwd[20], arq[20] = "", nome[20];
  Agenda *atual;

  strcpy(nome, user->name);
  pFile = fopen(nome, "w"); /*arquivo com nome de usuarios*/
  if (pFile == NULL){
    printf("\nNULL - SaveCal\n");
    return 0;
  }
  else
  {
    /* Percorre a lista ligada e imprime as infomacoes de cada no no
           arquivo */
    for (atual = user->tasks; atual != NULL; atual = atual->next)
    {
      fputs(atual->task, pFile);
      fprintf(pFile, "\n%d\n%d\n%d\n", atual->dia, atual->hora, atual->
             min);
    }
    fclose(pFile);
    return 1;
  }

  fclose(pFile);
  return 0;
}
```

Listing 3: Servidor

```
/*
** server.c -- a stream socket server demo
*/
#include "agenda.h"
#include <sys/time.h>

/* Estrutura para analise de tempo em microsegundos */
struct timeval  first, second, lapsed;
struct timezone tzp;

void serverTimeRecv(struct timeval first,struct timeval second){

  double t2=first.tv_sec+(first.tv_usec/1000000.0);
  double t3=second.tv_sec+(second.tv_usec/1000000.0);

  FILE * pFile;
  pFile = fopen("serverTime.dat", "a"); /*arquivo com tempos do servidor
         */

  if (pFile == NULL)
    return ;

  /* if (first.tv_usec > second.tv_usec) {
    second.tv_usec += 1000000;
    second.tv_sec--;
  } */

  fseek(pFile, 0, SEEK_END);
  fprintf(pFile,"%f \n" ,t3-t2);
  fclose(pFile);

  return;
}


void sigchld_handler(int s)
{
  while(waitpid(-1, NULL, WNOHANG) > 0);
}

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
  if (sa->sa_family == AF_INET) {
    return &(((struct sockaddr_in*)sa)->sin_addr);
  }

  return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(void)
{
  int sockfd, new_fd;  // listen on sock_fd, new connection on new_fd
  struct addrinfo hints, *servinfo, *p;
  struct sockaddr_storage their_addr; // connector's address information
  socklen_t addr_len;
  struct sigaction sa;
  int yes=1;
  char s[INET6_ADDRSTRLEN],tempo[5],str[5];
  int rv;
  char buf[ MAXDATASIZE];

  memset(&hints, 0, sizeof hints);
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_DGRAM;// TCP stream sockets
  hints.ai_flags = AI_PASSIVE; // use my IP

  if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
  }

  // loop through all the results and bind to the first we can
  for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
        p->ai_protocol)) == -1) {
      perror("server: socket");
      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,
          addr_len);
118 //      serverTimeRecv(first,second);
119     }
120     close(new_fd);  // parent doesn't need this
121   return 0;
122 }
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\n\0", 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",
                their_addr);
138       leString(their_addr, new_fd, nome);
139
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
            sair:\0", their_addr);
152     leString(their_addr, new_fd, again);
153     if(strcmp("q",again)==0)
154       exit(1);
155   }
156       }
157       else{
158   sendMsg(new_fd, "Usuario inexistente! Digite m para voltar ou q para
          sair:\0", their_addr);
159   leString(their_addr, new_fd, again);
160
161   /*saida do
162         programa*/
163   if(strcmp("q",again)==0)
164   exit(1);
165
166       break;
167
168       }
169       break;
170
171     case 2:
172
173       /* Criar um usuario */
174       sendMsg(new_fd, "Digite o nome do usuario a ser criado:\0",
                their_addr);
175       leString(their_addr, new_fd, nome);
176       sendMsg(new_fd, "Digite a senha do usuario:\0", their_addr);
177       leString(their_addr, new_fd, senha);
178
179       /* Verifica se nome ja existe */
180       if(newUser(nome,senha)==1){
181   user=agenda_init(nome);
182   menu2(new_fd, their_addr,user);
183       }
184       else{
185   sendMsg(new_fd, "Usuario ja existente! Digite m para voltar ou q para
          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]="",
          again[1]="";
205   char str[1000]="";
206   char menu[1000]="Escolha uma opcao:\n\
```

```
                Opcao 1 - Marcar um compromisso\n\
                Opcao 2 - Desmarcar um compromisso\n\
                Opcao 3 - Obter um compromisso marcado para um horario
                      de um dia\n\
                Opcao 4 - Obter todos os compromissos marcados para um
                      dia\n\
                Opcao 5 - Obter todos os compromissos do mes\n\
                Opcao 6 - Voltar\0";

/*Recupera agenda do usuario, apos login*/
loadCal(user);

while(1){
  sendMsg(new_fd, "Escolha uma opcao:\n\
                Opcao 1 - Marcar um compromisso\n\
                Opcao 2 - Desmarcar um compromisso\n\
                Opcao 3 - Obter um compromisso marcado para um horario
                      de um dia\n\
                Opcao 4 - Obter todos os compromissos marcados para um
                      dia\n\
                Opcao 5 - Obter todos os compromissos do mes\n\
                Opcao 6 - Voltar\0", their_addr);
  switch(leOpcao(their_addr, new_fd)){
  case 1:
    /* Marcar um compromisso */
    sendMsg(new_fd, "Digite o nome do compromisso:\0", their_addr);
    leString(their_addr, new_fd, task);
    sendMsg(new_fd, "Digite o dia do compromisso:\0", their_addr);
    leString(their_addr, new_fd, dia);
    sendMsg(new_fd, "Digite o hora do compromisso:\0", their_addr);
    leString(their_addr, new_fd, hora);
    sendMsg(new_fd, "Digite os minutos do compromisso:\0", their_addr);
    leString(their_addr, new_fd, minuto);
    set_task(atoi(dia), atoi(hora), atoi(minuto), task, user);

    verMes(new_fd,user,their_addr);

    /*Se m retorna ao menu, se q salva agenda sai*/
    leString(their_addr, new_fd,again);
    if(strcmp("q",again)==0) {
saveCal(user);
user_destroy(user);
close(new_fd);  // mata conexao com cliente
exit(1);
    }
    break;
  case 2:
    /* Desmarcar um compromisso */
    sendMsg(new_fd, "Digite o nome do compromisso a ser desmarcado:\0",
           their_addr);
    leString(their_addr, new_fd, str);
    if(delTask(user, str))
      sendMsg(new_fd, "\nCompromisso desmarcado\nDigite m para voltar
            ao menu anterior ou q para sair\n\0", their_addr);
    else
      sendMsg(new_fd, "\nNao foi encontrado nenhum compromisso
            registrado com esse nome\nDigite m para voltar ao menu
            anterior ou q para sair\n\0", their_addr);

    /*Se m retorna ao menu, se q salva agenda sai*/
    leString(their_addr, new_fd,again);
    if(strcmp("q",again)==0){
saveCal(user);
user_destroy(user);
close(new_fd);  // mata conexao com cliente
exit(1);
    }
    break;
  case 3:
    /* Obter compromissos de um dia em determinada hora */
    sendMsg(new_fd, "Digite o dia:\0", their_addr);
    leString(their_addr, new_fd, dia);
    sendMsg(new_fd, "Digite as horas:\0", their_addr);
    leString(their_addr, new_fd, hora);
    verHora(new_fd,user,atoi(dia),atoi(hora),their_addr);

    /*Se m retorna ao menu, se q salva agenda sai*/
    leString(their_addr, new_fd,again);
    if(strcmp("q",again)==0){
saveCal(user);
user_destroy(user);
close(new_fd);  // mata conexao com cliente
exit(1);
    }
    break;
  case 4:
    /* Obter todos os compromissos marcados para um dia */
    sendMsg(new_fd, "Digite o dia:\0", their_addr);
    leString(their_addr, new_fd, dia);
    verDia(new_fd,user,atoi(dia),their_addr);

    /*Se m retorna ao menu, se q salva agenda sai*/
    leString(their_addr, new_fd,again);
    if(strcmp("q",again)==0){
saveCal(user);
user_destroy(user);
close(new_fd);  // mata conexao com cliente
exit(1);
    }
    break;
  case 5:
    /* Obter todos os compromissos do mes */
    verMes(new_fd,user,their_addr);

    /*Se m retorna ao menu, se q salva agenda sai*/
    leString(their_addr, new_fd,again);
    if(strcmp("q",again)==0){
saveCal(user);
user_destroy(user);
close(new_fd);  // mata conexao com cliente
exit(1);
    }
    break;
  default:
    saveCal(user);
    user_destroy(user);
    return;
    break;
  }
 }
}

void leString(struct sockaddr_storage their_addr, int sockfd, char string
      []){

 int numbytes;
 char s[INET6_ADDRSTRLEN];
 socklen_t addr_len = sizeof their_addr;

 if ((numbytes = recvfrom(sockfd, string, MAXDATASIZE-1 , 0,
       (struct sockaddr *)&their_addr, &addr_len)) == -1) {
   perror("recvfrom");
   exit(1);
 }
 printf("listener: got packet from %s\n",
  inet_ntop(their_addr.ss_family,
```

```
        get_in_addr((struct sockaddr *)&their_addr),
        s, sizeof s));
    printf("listener: packet is %d bytes long\n", numbytes);
    printf("listener: packet contains \"%s\"\n", string);
    return;

}

int leOpcao(struct sockaddr_storage their_addr, int sockfd ){
    int numbytes;
    char buf[MAXDATASIZE];
    char s[INET6_ADDRSTRLEN];
    socklen_t addr_len = sizeof their_addr;

    if ((numbytes = recvfrom(sockfd, buf, MAXDATASIZE-1 , 0,
            (struct sockaddr *)&their_addr, &addr_len)) == -1) {
      perror("recvfrom");
      exit(1);
    }
    printf("listener: got packet from %s\n",
     inet_ntop(their_addr.ss_family,
        get_in_addr((struct sockaddr *)&their_addr),
        s, sizeof s));
    printf("listener: packet is %d bytes long\n", numbytes);
    buf[numbytes] = '\0';
    printf("listener: packet contains \"%s\"\n", buf);
    printf("%c", buf[0]);
    return atoi(buf);
}

void sendMsg(int new_fd, char str[],struct sockaddr_storage their_addr){
    if (sendto(new_fd, str , strlen(str) + 1, 0,(struct sockaddr *)&
          their_addr,  sizeof their_addr) == -1)
      perror("send");
}
```

Listing 4: Cliente

```
/*
** client.c -- a stream socket client demo
*/

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

#define PORT "35555" // the port client will be connecting to

#define MAXDATASIZE 1000 // max number of bytes we can get at once
char opcao[256];

/* Estrutura para analise de tempo em microsegundos */
struct timeval  first, second, lapsed;
struct timezone tzp;

void connecttime(struct timeval first,struct timeval second){

   double t1=first.tv_sec+(first.tv_usec/1000000.0);
   double t4=second.tv_sec+(second.tv_usec/1000000.0);


   FILE * pFile;
   pFile = fopen("conntime.dat", "a"); /*arquivo com tempos do servidor*/

   if (pFile == NULL)
     return ;

   /* if (first.tv_usec > second.tv_usec) {
     second.tv_usec += 1000000;
     second.tv_sec--;
   } */

   fseek(pFile, 0, SEEK_END);
   fprintf(pFile,"%f \n" ,t4-t1);
   fclose(pFile);

   return;
}


void clienteTimeRecv(struct timeval first,struct timeval second){

   double t1=first.tv_sec+(first.tv_usec/1000000.0);
   double t4=second.tv_sec+(second.tv_usec/1000000.0);

   FILE * pFile;
   pFile = fopen("clientTime.dat", "a"); /*arquivo com tempos do servidor
         */

   if (pFile == NULL)
     return ;

   /* if (first.tv_usec > second.tv_usec) {
     second.tv_usec += 1000000;
     second.tv_sec--;
   } */

   fseek(pFile, 0, SEEK_END);
   fprintf(pFile,"%f \n" ,t4-t1);
   fclose(pFile);

   return;
}


void envia_pct( int sockfd, char s[], int size){
   if (( send(sockfd, s ,size, 0)) == -1) {
     perror("talker: sendto");
     exit(1);
   }
  return;
}

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE]="";
    struct addrinfo hints, *servinfo, *p;
```

```
    int rv;
    char s[INET6_ADDRSTRLEN]="",tempo[5],str[5];

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;

    if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
   gettimeofday (&first, &tzp);
    // loop through all the results and connect to the first we can
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1) {
            perror("client: socket");
            continue;
        }

        if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("client: connect");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return 2;
    }

    inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),
            s, sizeof s);
    printf("client: connecting to %s\n", s);
   gettimeofday (&second, &tzp);
   connecttime(first,second);

    freeaddrinfo(servinfo); // all done with this structure
    int size;

    /* Teste de tempo */
   strcpy(str,"0123");//tamanho de um inteiro bytes
   gettimeofday (&first, &tzp);
   send(sockfd, str , strlen(str), 0);

    while(1){
      /* Esperando resposta do servidor*/
      if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
        perror("recv");
        exit(1);

      }

      system("clear");
      printf("\n%s\n",buf); //client received

      /* Espera resposta do servidor*/
      strcpy(opcao,"");
      scanf("%[^\n]", opcao );
      getchar();

      envia_pct(sockfd, opcao ,strlen(opcao) + 1);

      if(strcmp("q",opcao)==0){
  break;
      }
    }

    recv(sockfd, tempo, MAXDATASIZE-1, 0);
    gettimeofday (&second, &tzp);

    clienteTimeRecv(first,second);

    close(sockfd);

    return 0;
}
```

Listing 5: Cabeçalhos Banco de dados

```
#ifndef BD_H_
#define BD_H_

int findUser(char nome[], char pwd[]);
int newUser(char nome[], char senha[]);
int loadCal();
int saveCal();

#endif /* BD_H_ */
```

Listing 6: Cabeçalhos Agenda

```
#ifndef AGENDA_H_
#define AGENDA_H_

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

#define PORT "35555"  // the port users will be connecting to
#define MAXDATASIZE 1000
#define BACKLOG 10     // how many pending connections queue will hold

/*Comaracao de compromissos*/
#define ANTES 1     // how many pending connections queue will hold
#define DEPOIS -1     // how many pending connections queue will hold
#define SIMULTANEO 0     // how many pending connections queue will hold


typedef struct agenda {
  struct agenda *next;
  int dia;
  int hora, min;
  char task[256];
} Agenda;

```

```

typedef struct user {
  struct agenda *tasks;
  char name[20];
} User;

/* Funcoes */
void menu(int new_fd, struct sockaddr_storage their_addr);
void menu2(int new_fd, struct sockaddr_storage their_addr, User *user);
void sendStr(int sockfd, char str[]);
void sendMsg(int new_fd, char str[],struct sockaddr_storage their_addr);
int leOpcao(struct sockaddr_storage their_addr, int sockfd);
void leString(struct sockaddr_storage their_addr, int sockfd , char
      string[]);
User * agenda_init(char nome[]);
void user_destroy(User *u);
int agenda_vazia(User *a);
int compData(Agenda *newTasks,Agenda *tasks);
int set_task(int dia,int hora, int min,char task[], User *u);
Agenda * task_init(int dia,int hora, int min,char task[]);
int verMes(int new_fd, User *u,struct sockaddr_storage their_addr);
int verDia(int new_fd, User *u, int dia,struct sockaddr_storage
      their_addr);
int verHora(int new_fd, User *u, int dia, int hora,struct
      sockaddr_storage their_addr);
int delTask( User *u, char nome[]);
void cpComp(Agenda *a, char comp[]);
#endif /*CONJUNTO_H_*/
```

Listing 7: Makefile

```
CC = gcc
CFLAGS =  -g -ggdb
LIBS = -lm

OBJS_C =  client.o
OBJS_S =  agenda.o server.o bd.o
ECHO_S = echo_server.c
ECHO_C = echo_client.c

EXEC_C_ECHO = echoc
EXEC_S_ECHO = echos
EXEC_C = c
EXEC_S = s

all:
  make c
  make s

c: $(OBJS_C)
  $(CC) $(CFLAGS) $(OBJS_C) $(LIBS) -o $(EXEC_C)

s: $(OBJS_S)
  $(CC) $(CFLAGS) $(OBJS_S) $(LIBS) -o $(EXEC_S)


run: all
  ./$(EXEC) $(ATB)

$(OBJS): %.o: %.c
  $(CC) -c $(CFLAGS) $<

echo:
  $(CC) $(CFLAGS)  $(ECHO_C) $(LIBS) -o $(EXEC_C_ECHO)
  $(CC) $(CFLAGS)  $(ECHO_S) $(LIBS) -o $(EXEC_S_ECHO)

clean:
  rm -f $(OBJS_C) $(OBJS_S) $(EXEC_C) $(EXEC_S) *~

reset: clean
  rm -f $(FILES)
```