

```

1  #include <netinet/in.h>
2  #include <unistd.h>
3  #include <strings.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/socket.h>
7  #include <arpa/inet.h>
8
9
10 /* *****
11 Este é apenas um possível exemplo.
12 No caso de um servidor que possa atender vários tipos de pedidos,
13 a criação de threads ou processos poderá ser feita apenas para alguns
14 tipos de pedidos.
15 ***** */
16
17 int main(int argc, char *argv[]) {
18     int sockfd;
19     struct sockaddr_in serv_addr;
20
21     if(argc!=2) {
22         printf("Usage: %s port_number\n",argv[0]);
23         exit(1);
24     }
25
26     sockfd = socket(PF_INET, SOCK_STREAM, 0);
27     if(sockfd<0){
28         perror("socket");
29         exit(1);
30     }
31
32     bzero((char *) &serv_addr, sizeof(serv_addr));
33     serv_addr.sin_family = AF_INET;
34     serv_addr.sin_addr.s_addr = INADDR_ANY;
35     serv_addr.sin_port = htons(atoi(argv[1]));
36
37     if(bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0){
38         perror("bind");
39         exit(1);
40     }
41
42     listen(sockfd,5);
43
44     //servidor_mono_tarefa_iterativo(sockfd);
45     //servidor_multi_tarefa_com_threads(sockfd);
46     //servidor_multi_tarefa_com_processos(sockfd);
47     //servidor_multi_tarefa_com_processos_sigchld(sockfd);
48
49     return(0);
50 }
51
52
53 /*****
54 //Assume-se que recebe um socket já com ligação estabelecida,
55 //detecta o tipo de pedido (se vários possíveis), satisfaz o pedido e
56 //fecha a ligação antes de terminar (close(s))
57 //Em alguns casos poderá ser necessário definir parâmetros adicionais,
58 //relativos a variáveis do servidor.
59 void atende_pedido(int s) {
60
61     printf("This is a test.\n");
62     sleep(5);
63     printf("Bye.\n");
64
65     close(s);
66 }
67
68
69
70
71
72
73
74
75

```

```

76  /*****
77  void servidor_mono_tarefa_iterativo(int s)
78  {
79      int ns;
80      unsigned int clilen;
81      struct sockaddr_in cli_addr;
82
83      while(1) {
84          clilen = sizeof(cli_addr);
85          ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
86          //Verificação de erros.
87          if(ns<0) {
88              perror("accept");
89              sleep(1);
90              continue;
91          }
92
93          printf("Connection from %s\n", inet_ntoa(*(struct in_addr *) &(cli_addr.sin_addr)));
94          //
95
96
97          atende_pedido(ns);
98      }
99  }
100
101
102
103
104
105
106
107
108
109  /*****
110  #include <signal.h>
111  #include <sys/wait.h>
112  #include <errno.h>
113
114
115  void servidor_multi_tarefa_com_processos(int s) {
116      int ns, r;
117      struct sockaddr_in cli_addr;
118      unsigned int clilen;
119
120      signal(SIGCHLD, SIG_IGN);
121
122      while(1) {
123          clilen = sizeof(cli_addr);
124          ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
125          //código de verificação de erros omitido
126
127          r=fork();
128          if(r==0) {
129              atende_pedido(ns);
130              exit(0);
131          }
132          else if(r<0) {
133              //não conseguiu criar novo processo. Atende em modo iterativo
134              atende_pedido(ns);
135          }
136          else
137              close(ns);
138      }
139  }
140  }
141
142
143
144
145
146
147
148
149
150

```

```

151
152
153 /*****
154 #include <signal.h>
155 #include <sys/wait.h>
156 #include <errno.h>
157
158 void sigchld_handler(int signum) {
159     int pid;
160
161     while((pid=waitpid(-1,NULL,WNOHANG))>0)
162         printf("Child process %d ended\n", pid);
163 }
164
165 void servidor_multi_tarefa_com_processos_sigchld(int s) {
166     int ns, r;
167     struct sockaddr_in cli_addr;
168     unsigned int clilen;
169
170     //usar o SIGCHLD para detectar a terminação dos processos filho
171     struct sigaction act;
172     act.sa_handler=sigchld_handler;
173     act.sa_flags=SA_NOCLDSTOP;
174     sigemptyset(&(act.sa_mask));
175     sigaction(SIGCHLD, &act, NULL);
176
177     while(1) {
178         clilen = sizeof(cli_addr);
179         ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
180         //pode dar-se o caso do accept ser interrompido pelo SIGCHLD (EINTR)
181         if(ns<0) {
182             if(errno!=EINTR)
183                 sleep(1);
184             continue;
185         }
186
187         r=fork();
188         if(r==0) {
189             atende_pedido(ns);
190             exit(0);
191         }
192         else if(r<0) {
193             //não conseguiu criar novo processo. Atende em modo iterativo
194             atende_pedido(ns);
195         }
196         else
197             close(ns);
198     }
199 }
200 }
201 }
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225

```

```

226
227 /*****
228 #include <pthread.h>
229
230 typedef struct {
231     int s;
232     //outras necessárias
233 } args_t;
234
235
236 void *aux(void *args) {
237     pthread_detach(pthread_self()); //necessário para não haver threads zombie
238     atende_pedido( ((args_t *) args)->s );
239     free(args);
240     return(NULL);
241 }
242
243
244
245 void servidor_multi_tarefa_com_threads(int s)
246 {
247     int ns;
248     struct sockaddr_in cli_addr;
249     unsigned int clilen;
250     pthread_t tid;
251     args_t *pargs;
252
253     while(1) {
254         clilen = sizeof(cli_addr);
255         ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
256         //código de verificação de erros omitido
257
258         pargs=malloc(sizeof(args_t));
259         pargs->s = ns;
260
261         if(pthread_create(&tid, NULL, aux, pargs) != 0) {
262             //não consegui criar nova thread. Atende em modo iterativo.
263             atende_pedido(ns);
264         }
265     }
266 }
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300

```

301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321