```c
#include <netinet/in.h>
#include <unistd.h>
#include <strings.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>


/* ************************************************************************
Este é apenas um possível exemplo.
No caso de um servidor que possa atender vários tipos de pedidos,
a criação de threads ou processos poderá ser feita apenas para alguns
tipos de pedidos.
************************************************************************ */

int main(int argc, char *argv[]) {
  int sockfd;
  struct sockaddr_in serv_addr;

  if(argc!=2) {
    printf("Usage: %s port_number\n",argv[0]);
    exit(1);
  }

  sockfd = socket(PF_INET, SOCK_STREAM, 0);
  if(sockfd<0){
    perror("socket");
    exit(1);
  }

  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(atoi(argv[1]));

  if(bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0){
    perror("bind");
    exit(1);
  }

  listen(sockfd,5);

  //servidor_mono_tarefa_iterativo(sockfd);
  //servidor_multi_tarefa_com_threads(sockfd);
  //servidor_multi_tarefa_com_processos(sockfd);
  //servidor_multi_tarefa_com_processos_sigchld(sockfd);

  return(0);
}


/*************************************************************************/
//Assume-se que recebe um socket já com ligação estabelecida,
//detecta o tipo de pedido (se vários possíveis), satisfaz o pedido e
//fecha a ligação antes de terminar (close(s))
//Em alguns casos poderá ser necessário definir parâmetros adicionais,
//relativos a variáveis do servidor.
void atende_pedido(int s) {

  printf("This is a test.\n");
  sleep(5);
  printf("Bye.\n");

  close(s);
}
```

```c
/*****************************************************************************/
void servidor_mono_tarefa_iterativo(int s)
{
  int ns;
  unsigned int clilen;
  struct sockaddr_in cli_addr;

  while(1) {
    clilen = sizeof(cli_addr);
    ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
   //Verificação de erros.
   if(ns<0) {
      perror("accept");
      sleep(1);
      continue;
    }

    printf("Connection from %s\n", inet_ntoa(*((struct in_addr *) &(cli_addr.sin_addr) )));
    //


    atende_pedido(ns);
  }
}









/*****************************************************************************/
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>


void servidor_multi_tarefa_com_processos(int s) {
  int ns, r;
  struct sockaddr_in cli_addr;
  unsigned int clilen;

  signal(SIGCHLD, SIG_IGN);

  while(1) {
    clilen = sizeof(cli_addr);
    ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
    //código de verificação de erros omitido

    r=fork();
    if(r==0) {
      atende_pedido(ns);
      exit(0);
    }
    else if(r<0) {
      //não conseguiu criar novo processo. Atende em modo iterativo
      atende_pedido(ns);
    }
    else
      close(ns);

  }
}
```

```c

/*****************************************************************************/
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>

void sigchld_handler(int signum) {
  int pid;

  while((pid=waitpid(-1,NULL,WNOHANG))>0)
    printf("Child process %d ended\n", pid);
}

void servidor_multi_tarefa_com_processos_sigchld(int s) {
  int ns, r;
  struct sockaddr_in cli_addr;
  unsigned int clilen;

  //usar o SIGCHLD para detectar a terminação dos processos filho
  struct sigaction act;
  act.sa_handler=sigchld_handler;
  act.sa_flags=SA_NOCLDSTOP;
  sigemptyset(&(act.sa_mask));
  sigaction(SIGCHLD, &act, NULL);

  while(1) {
    clilen = sizeof(cli_addr);
    ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
    //pode dar-se o caso do accept ser interrompido pelo SIGCHLD  (EINTR)
    if(ns<0) {
      if(errno!=EINTR)
        sleep(1);
      continue;
    }


    r=fork();
    if(r==0) {
      atende_pedido(ns);
      exit(0);
    }
    else if(r<0) {
      //não conseguiu criar novo processo. Atende em modo iterativo
      atende_pedido(ns);
    }
    else
      close(ns);

  }
}
```

```c

/*************************************************************************/
#include <pthread.h>

typedef struct {
  int s;
//outras necessárias
} args_t;


void *aux(void *args) {
  pthread_detach(pthread_self());  //necessário para não haver threads zombie
  atende_pedido( ((args_t *) args)->s );
  free(args);
  return(NULL);
}



void servidor_multi_tarefa_com_threads(int s)
{
  int ns;
  struct sockaddr_in cli_addr;
  unsigned int clilen;
  pthread_t tid;
  args_t *pargs;

  while(1) {
    clilen = sizeof(cli_addr);
    ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
    //código de verificação de erros omitido

    pargs=malloc(sizeof(args_t));
    pargs->s = ns;

    if(pthread_create(&tid, NULL, aux, pargs) != 0) {
      //não consegui criar nova thread. Atende em modo iterativo.
      atende_pedido(ns);
    }
  }
}
```