CECS 472, Fall 2012, Midterm 2, Dennis Volper Name:_____

Open Book, open notes. All questions equal weight.
Reassemble loops must be shown in every case where one is needed. All numbers sent over the network must be in network standard order. You may assume appropriate includes are already part of the code.
1) One way to reduce network chat traffic is for the chat server to use broadcasts when it want's to echo to everyone. (As side effect the sender gets a copy of his message sent back to him.) Build a chat server that uses broadcasts instead of TCP unicasts to send to chat messages back to all clients. You are given a copy of a portion Comer's `TCPmechod` as a starting point, show the modifications. (Assume the unshown portion of Comer's code is present.)

```
int main(int argc, char *argv[]) { /* TCPmechod */
  char *service = "echo";
  struct sockaddr_in fsin;
  int     msock;
  fd_set rfds;   fd_set afds;
  int alen;  int fd, nfds;
  switch (argc) { /* not shown */ }
  }
  signal(SIGPIPE, SIG_IGN);
  msock = passiveTCP(service, QLEN);
  nfds = getdtablesize();
  FD_ZERO(&afds);
  FD_SET(msock, &afds);
  while (1) {
    memcpy(&rfds, &afds, sizeof(rfds));
    if (select(nfds,&rfds,(fd_set *)0,(fd_set *)0,(struct timeval *)0) < 0)
      errexit("select: %s\n", strerror(errno));
    if (FD_ISSET(msock, &rfds)) {
      int ssock;
      alen = sizeof(fsin);
      ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
      if (ssock < 0) errexit("accept: %s\n",strerror(errno));
      FD_SET(ssock, &afds);
    }
    for (fd=0; fd<nfds; ++fd)
      if (fd != msock && FD_ISSET(fd, &rfds))
        if (echo(fd) == 0) {
          (void) close(fd);
          FD_CLR(fd, &afds);
        }
  }
}
int echo(int fd) {
  char buf[BUFSIZ]; int  cc;
  cc = recv(fd, buf, sizeof buf, 0);
  if (cc < 0) errexit("echo read: %s\n", strerror(errno));
  if (cc && send(fd, buf, cc, 0) < 0)
    errexit("echo write: %s\n", strerror(errno));
  return cc;
}
```

2) Another way to reduce network chat client is for the server to multicast replies. The chat client needs to be modified to handle this. Modify the following (simplified) chat client so it will use multicast to receive responses from the server.

```
/* Includes not shown */
/*  main  not shown */
int chatc(const char *host, const char *service)
{
  char    buf[LINELEN+1];         /* buffer for one line of text  */
  int     s, n;                   /* socket descriptor, read count*/
  int     outchars, inchars;      /* characters sent and received */
  fd_set  rfds;                   /* read file descriptor set     */
  fd_set  afds;                   /* active file descriptor set   */
  int  nfds;
  s = connectTCP(host, service);
```

```
  nfds = s + 1;
  FD_ZERO(&afds);
  FD_SET(s, &afds);
  FD_SET(0, &afds);
  while (1) {
    memcpy(&rfds, &afds, sizeof(rfds));
    if (select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct timeval *)0) < 0)
          errexit("select: %s\n", strerror(errno));
    if (FD_ISSET(0, &rfds)) {
       if (fgets(buf, sizeof(buf), stdin)==NULL) exit(0);
       buf[LINELEN] = '\0';    /* insure line null-terminated  */
       outchars = strlen(buf);
       (void) send(s, buf, outchars, MSG_NOSIGNAL);
    }
    if (FD_ISSET(s, &rfds)) {
      int  n = recv(s, buf,  LINELEN, MSG_NOSIGNAL);
      if (n <= 0) exit(0);
      buf[n]='\0';
      fputs(buf, stdout);
    }
  }
}
```

3) Modify your browser server so that if it receives a “t” it sends back the time of day (8 bytes, seconds and microseconds) Show only the case that needs to be added to the switch plus any necessary variable declarations.

4) Write a threaded `multid`. Assume you did returns in your returns in your three functions and not exits so that they will not need to be modified. Show the necessary changes to the portion of Comer's `superd` code given below.

```
/* show additional necessary global declarations here */









int main(int argc, char *argv[])
{
  struct service  *psv,    /* service table pointer  */
    *fd2sv[NOFILE];    /* map fd to service pointer  */
  int  fd, nfds;
  fd_set  afds, rfds;    /* readable file descriptors  */
  /* show any additions to main here */









}
/* Includes, prototypes and svent not shown */
void doTCP(struct service *psv)
{
  struct sockaddr_in fsin;  /* the request from address  */
  int  alen;      /* from-address length    */
  int  fd, ssock;
  alen = sizeof(fsin);
  ssock = accept(psv->sv_sock, (struct sockaddr *)&fsin, &alen);
  if (ssock < 0)
    errexit("accept: %s\n", strerror(errno));
  switch (fork()) {
  case 0:
    break;
  case -1:
    errexit("fork: %s\n", strerror(errno));
  default:
    (void) close(ssock);
    return;    /* parent */
  }
  for (fd = NOFILE; fd >= 3; --fd)
    if (fd != ssock)
      (void) close(fd);
  exit(psv->sv_func(ssock));









}
```

5) Write a UT echo server. It should respond to both TCP and UDP clients. Be careful, echo clients don't disconnect after sending a message. A portion of Comer's (UT) daytime is given as a starting point, show the necessary modifications. Note: no reassembly loop is required.

```
int main(int argc, char *argv[])
{
  char *service = "daytime"; /* service name or port number */
  char buf[LINELEN+1]; /* buffer for one line of text */
  struct sockaddr_in fsin; /* the request from address */
  int alen; /* from-address length */
  int tsock;  /* TCP master socket */
  int usock; /* UDP socket */
  int nfds;
  fd_set rfds; /* readable file descriptors */
  switch (argc) { /* not shown */ }
  tsock = passiveTCP(service, QLEN);
  usock = passiveUDP(service);
  nfds = MAX(tsock, usock) + 1; /* bit number of max fd */
  FD_ZERO(&rfds);
  while (1) {
    FD_SET(tsock, &rfds);
    FD_SET(usock, &rfds);
    if (select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct timeval *)0) < 0)
      errexit("select error: %s\n", strerror(errno));
    if (FD_ISSET(tsock, &rfds)) {
      int  ssock;    /* TCP slave socket  */
      alen = sizeof(fsin);
      ssock = accept(tsock, (struct sockaddr *)&fsin, &alen);
      if (ssock < 0) errexit("accept failed: %s\n", strerror(errno));
      /*daytime(buf);*/
      (void) write(ssock, buf, strlen(buf));
      (void) close(ssock);
    }









    if (FD_ISSET(usock, &rfds)) {
      alen = sizeof(fsin);
      if (recvfrom(usock, buf, sizeof(buf), 0,
           (struct sockaddr *)&fsin, &alen) < 0)
        errexit("recvfrom: %s\n", strerror(errno));
      /*daytime(buf);*/
      (void) sendto(usock, buf, strlen(buf), 0,
           (struct sockaddr *)&fsin, sizeof(fsin));
    }
  }
}
```