CECS 472, Fall 2011, 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) Modify your `browserd.c` to add a `d` command, which returns to the client the name of the current directory followed by an ASCII zero. Show only the new "d" case of the `browserd` switch. To get the name of the current directory use the `getcwd` system call (use the online manual page).

2) Modify Comer's `UDPtimed.c` so that it will receive and respond to multicasts, broadcasts and (still respond to) unicasts.

```
#define UNIXEPOCH       2208988800UL
int main(int argc, char *argv[])
{
  struct sockaddr_in fsin;
  char    *service = "time";
  char    buf[1];
  int     sock;
  time_t  now;
  int     alen;
  switch (argc) {
  case    1:
          break;
  case    2:
          service = argv[1];
          break;
  default:
          errexit("usage: UDPtimed [port]\n");
  }
  sock = passiveUDP(service);
  while (1) {
    alen = sizeof(fsin);
    if (recvfrom(sock, buf, sizeof(buf), 0,
           (struct sockaddr *)&fsin, &alen) < 0)
      errexit("recvfrom: %s\n", strerror(errno));
    (void) time(&now);
    now = htonl((u_long)(now + UNIXEPOCH));
    (void) sendto(sock, (char *)&now, sizeof(now), 0,
      (struct sockaddr *)&fsin, sizeof(fsin));
  }
}
```

3) Modify Comer's `TCPmechod` given below (parts removed to fit on page). Add a UDP socket. When ever anything arrives on the UDP socket, echo it to all connected TCP clients.

```
int main(int argc, char *argv[]) {
  char *service = "echo";
  struct sockaddr_in fsin;
  int     msock;
  fd_set rfds;   fd_set afds;
  int alen;  int fd, nfds;
  switch (argc) {
    case    1: break;
    case    2: service = argv[1]; break;
    default: errexit("usage: TCPmechod [port]\n");
  }
  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;
}
```

4) Write a TCP addition client. You client should be built inside the function `addinterface` This function has three parameters; the socket (the Comer switch is in the main program), an array of integers of arbitrary size containing the numbers to be added (filled in by the main program) and an integer indicating how many integers are in the array (passed to you by the main program). Your client should send the integers to your server (see next question), read and print the answer (an integer) returned by the server. The server will need to be sent an integer indicating how many integers are going to be sent, that is, how many are in the array.

```
void addinterface(int fd; int addthese[]; int howmany) {
```

5) Write a TCP addition server function (`addinterfaced`) that adds an array of integers. The function (in the Comer tradition) will be passed the slave socket. The function will read an integer indicating how big the array is, then it will read the integers, add them, and send back the answer. You may read the integers one at a time. If you wish to read them all before adding, I'll guarantee there always less than `BUFSIZE` integers in the array. Write the `addinterfaced` procedure and show the line that would need to be added to the `svent` array of your `multid` so that this function would become part of that server.

```
/* Give svent line here */


int addinterfaced(int fd) {
/* Give function here */
```