netclient.c:

```
/*  Make the necessary includes and set up the variables.  */

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

int
main ()
{
        int sockfd;
        socklen_t len;
        struct sockaddr_in address;
        int result;
        char ch = 'A';

        /*  Create a socket for the client.  */

        sockfd = socket (AF_INET, SOCK_STREAM, 0);

        /*  Name the socket, as agreed with the server.  */

        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr ("127.0.0.1");
        address.sin_port = htons (9734);
        len = sizeof (address);

        /*  Now connect our socket to the server's socket.  */

        result = connect (sockfd, (struct sockaddr *) &address,
len);

        if (result == -1)
        {
                perror ("oops: netclient");
                exit (1);
        }

        /*  We can now read/write via sockfd.  */

        write (sockfd, &ch, 1);
        read (sockfd, &ch, 1);
        printf ("char from server = %c\n", ch);
        close (sockfd);
        exit (0);
}
```

server1.c:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>

int
main ()
{
        int server_sockfd, client_sockfd;
        socklen_t server_len, client_len;
        struct sockaddr_in server_address;
        struct sockaddr_in client_address;

        server_sockfd = socket (AF_INET, SOCK_STREAM, 0);

        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = htonl (INADDR_ANY);
        server_address.sin_port = htons (9734);
        server_len = sizeof (server_address);
        bind (server_sockfd, (struct sockaddr *)
&server_address, server_len);

        /*  Create a connection queue and wait for clients.  */

        listen (server_sockfd, 5);

        while (1)
        {
                char ch;

                printf ("server waiting\n");

                /*  Accept connection.  */

                client_len = sizeof (client_address);
                client_sockfd = accept (server_sockfd,
                                (struct sockaddr *)
&client_address,
                                &client_len);

                /*  We can now read/write to the client on
client_sockfd.
                        The five second delay is just for this
demonstration.  */

                read (client_sockfd, &ch, 1);
                sleep (5);
                ch++;
                write (client_sockfd, &ch, 1);
                close (client_sockfd);
        }
}
```

fork1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int
main ()
{
        pid_t res;
        int status;
        res = fork ();
        if (res == 0)
        {
                printf ("Hello from child\n");
                sleep (1);
                exit (1);
        };
        if (res == -1)
                perror ("fork");
        wait (&status);
        if (WIFEXITED (status))
                printf ("Child exited normally with exit code %
d\n",
                                WEXITSTATUS (status));
        else
                printf ("Child exited abnormally\n");
        exit (0);
}
```

fork2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int
main ()
{
        pid_t res;
        int status;
        res = fork ();
        if (res == 0)
        {
                printf ("Hello from child\n");
                sleep (1);
                abort ();
        };
        if (res == -1)
                perror ("fork");
        wait (&status);
        if (WIFEXITED (status))
                printf ("Child exited normally with exit code %
d\n",
                                WEXITSTATUS (status));
        else
                printf ("Child exited abnormally\n");
        exit (0);
}
```

fork3.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int
main ()
{
        pid_t res;
        int i;
        res = fork ();
        if (res == 0)
        {
                for (i = 0; i < 200; ++i)
                {
                        printf ("Hello #%d from child\n", i);
                }
                abort ();
        };
        if (res == -1)
                perror ("fork");
        for (i = 0; i < 200; ++i)
        {
                printf ("Hello #%d from parent\n", i);
                sleep (1);
        };
        exit (0);
```

```
}
```

fork4.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int
main ()
{
        pid_t res;
        int status;
        int i;
        res = fork ();
        if (res == 0)
        {
                for (i = 0; i < 200; ++i)
                        printf ("Hello #%d from child\n", i);
                abort ();
        };
        if (res == -1)
                perror ("fork");
        for (i = 0; i < 200; ++i)
        {
                printf ("Hello #%d from parent\n", i);
                if (waitpid (res, &status, WNOHANG) == res)
                        printf ("Child exited\n");
                sleep (1);
        };
        exit (0);
}
```

fork5.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

void
childsignal (int what)
{
        pid_t v;
        int old_errno = errno;
        printf ("New signal\n");
        while ((v = waitpid (-1, NULL, WNOHANG)) != 0
                        && !(v == -1 && errno != EINTR))
                printf ("%d\n", v);
        if (v == -1)
                perror ("waitpid");
        printf ("Signal exit\n");
        errno = old_errno;
}

int
main ()
{
        int i;
        struct sigaction sig;
        sig.sa_handler = childsignal;
        sigemptyset (&sig.sa_mask);
        sig.sa_flags = SA_NOCLDSTOP;
        sigaction (SIGCHLD, &sig, NULL);
        for (i = 0; i < 200; i++)
        {
                pid_t res;
                printf ("Starting child #%d\n", i);
                res = fork ();
                if (res == 0)
                {
                        sleep (5);
                        exit (i + 1);
                };
                if (res == -1)
                        perror ("fork");
        };
        for (i = 0; i < 360; i++)
                sleep (1);
        exit (0);
}
```

fork6.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int
main ()
{
        int i;
        signal (SIGCHLD, SIG_IGN);
        for (i = 0; i < 200; i++)
        {
                pid_t res;
                printf ("Starting child #%d\n", i);
                res = fork ();
                if (res == 0)
                {
                        sleep (5);
                        exit (i + 1);
                };
                if (res == -1)
                        perror ("fork");
        }
        for (i = 0; i < 360; i++)
                sleep (1);
        exit (0);
}
```

server2.c:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>

int
main ()
{
        int server_sockfd, client_sockfd;
        socklen_t server_len, client_len;
        struct sockaddr_in server_address;
        struct sockaddr_in client_address;
        server_sockfd = socket (AF_INET, SOCK_STREAM, 0);

        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = htonl (INADDR_ANY);
        server_address.sin_port = htons (9734);
        server_len = sizeof (server_address);
        bind (server_sockfd, (struct sockaddr *)
&server_address, server_len);

        /*  Create a connection queue, ignore child exit
details and wait for clients.  */

        listen (server_sockfd, 5);

        signal (SIGCHLD, SIG_IGN);

        while (1)
        {
                char ch;

                printf ("server waiting\n");

                /*  Accept connection.  */

                client_len = sizeof (client_address);
                client_sockfd = accept (server_sockfd,
                                        (struct sockaddr *)
&client_address,
                                        &client_len);

                /*  Fork to create a process for this client
and perform a test to see
                        whether we're the parent or the child.
*/

                if (fork () == 0)
                {

                        /*  If we're the child, we can now
read/write to the client on client_sockfd.
                                The five second delay is just
for this demonstration.  */

                        read (client_sockfd, &ch, 1);
                        sleep (5);
                        ch++;
                        write (client_sockfd, &ch, 1);
                        close (client_sockfd);
                        exit (0);
                }

                /*  Otherwise, we must be the parent and our
work for this client is finished.  */

                else
                {
                        close (client_sockfd);
                }
        }
}
```

eintrs.c:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>
```

```
int main ()
{
        ssize_t count;
        char buffer[1024 * 1024];
        int server_sockfd, client_sockfd;
        socklen_t server_len, client_len;
        struct sockaddr_in server_address;
        struct sockaddr_in client_address;

        server_sockfd = socket (AF_INET, SOCK_STREAM, 0);

        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = htonl (INADDR_ANY);
        server_address.sin_port = htons (9734);
        server_len = sizeof (server_address);
        bind (server_sockfd, (struct sockaddr *)
&server_address, server_len);

        /*  Create a connection queue and wait for clients.  */

        listen (server_sockfd, 5);

        printf ("server waiting\n");

        /*  Accept connection.  */

        client_len = sizeof (client_address);
        client_sockfd = accept (server_sockfd,
                        (struct sockaddr *) &client_address,
&client_len);

        while (1)
        {
                count = read (client_sockfd, buffer, 1024 *
1024);
                printf ("Read %d bytes\n", count);
        }
}
```

eintrc.c:

```
/*  Make the necessary includes and set up the variables.  */

#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>

void alarmhandler (int s)
{
        signal (SIGALRM, alarmhandler);
        alarm (1);
}

int main ()
{
        int sockfd;
        socklen_t len;
        struct sockaddr_in address;
        int result;
        ssize_t count;
        char buffer[1024 * 1024];
        signal (SIGALRM, alarmhandler);
        alarm (1);

        /*  Create a socket for the client.  */

        sockfd = socket (AF_INET, SOCK_STREAM, 0);

        /*  Name the socket, as agreed with the server.  */

        address.sin_family = AF_INET;
        address.sin_addr.s_addr = inet_addr ("127.0.0.1");
        address.sin_port = htons (9734);
        len = sizeof (address);

        /*  Now connect our socket to the server's socket.  */

        result = connect (sockfd, (struct sockaddr *) &address,
len);

        if (result == -1)
        {
                perror ("oops: eintrc");
                exit (1);
        }

        /*  We can now read/write via sockfd.  */
        while (1)
        {

                count = write (sockfd, buffer, 1024 * 1024);
                if (count == -1)
                        perror ("write");
                else
                        printf ("Wrote %d bytes\n", count);
        }
}
```

threadsafe1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

char *
unsafe_itoa (int number)
{
        static char buf[16];
        int i = 0, j;
        do
        {
                buf[i++] = number % 10 + '0';
                number /= 10;
        }
        while (number);
        buf[i] = 0;
        for (j = 0; j < i / 2; j++)
        {
                char c = buf[j];
                buf[j] = buf[i - j - 1];
                buf[i - j - 1] = c;
        };
        return buf;
}


void *
thread (void *param)
{
        int start = *((int *) (param));
        int i;
        printf("Hello from thread %d\n",start);
        for (i = 0; i < 10000; i++)
                printf ("%d %s\n", i + start, unsafe_itoa (i +
start));
        pthread_exit (param);
}

int
main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
        void *retval;
        a = 0;
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        exit (0);
}
```

threadsafe2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

char *
safe_itoa (int number, char *buf)
{
        int i = 0, j;
        do
        {
                buf[i++] = number % 10 + '0';
                number /= 10;
        }
        while (number);
        buf[i] = 0;
        for (j = 0; j < i / 2; j++)
        {
                char c = buf[j];
                buf[j] = buf[i - j - 1];
                buf[i - j - 1] = c;
        };
        return buf;
}


void *
thread (void *param)
{
        char buf[16];
        int start = *((int *) (param));
        int i;
        printf ("Hello from thread %d\n", start);
        for (i = 0; i < 10000; i++)
                printf ("%d %s\n", i + start, safe_itoa (i +
start, buf));
        pthread_exit (param);
}
int
main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
```

```
        void *retval;
        a = 0;
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        exit (0);
}
```

threadsafe3.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

char *
unsafe_itoa (int number)
{
        static char buf[16];
        int i = 0, j;
        do
        {
                buf[i++] = number % 10 + '0';
                number /= 10;
        }
        while (number);
        buf[i] = 0;
        for (j = 0; j < i / 2; j++)
        {
                char c = buf[j];
                buf[j] = buf[i - j - 1];
                buf[i - j - 1] = c;
        };
        return buf;
}

pthread_mutex_t itoa_mutex;

void *
thread (void *param)
{
        int start = *((int *) (param));
        int i;
        printf("Hello from thread %d\n",start);
        for (i = 0; i < 10000; i++)
        {
                pthread_mutex_lock(&itoa_mutex);
                printf ("%d %s\n", i + start, unsafe_itoa (i +
start));
                pthread_mutex_unlock(&itoa_mutex);
        }
        pthread_exit (param);
}

int
main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
        void *retval;
        a = 0;
        pthread_mutex_init(&itoa_mutex,NULL);
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        pthread_mutex_destroy(&itoa_mutex);
        exit (0);
}
```

threadsafe4.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int c;

void * thread (void *param)
{
        int i;
        for (i = 0; i < 10000; i++)
        {
                c=c*123413123+7;;
        }
        pthread_exit (param);
}

int main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
        void *retval;
        a = 0;
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        printf("c=%d\n",c);
        exit (0);
}
```

threadsafe5.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int c;

pthread_mutex_t itoa_mutex;

void *
thread (void *param)
{
        int i;
        for (i = 0; i < 10000; i++)
        {
                pthread_mutex_lock(&itoa_mutex);
                c=c*123413123+7;;
                pthread_mutex_unlock(&itoa_mutex);
        }
        pthread_exit (param);
}

int
main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
        void *retval;
        a = 0;
        pthread_mutex_init(&itoa_mutex,NULL);
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        pthread_mutex_destroy(&itoa_mutex);
        printf("c=%d\n",c);
        exit (0);
}
```

threadsafe6.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

volatile int c;

pthread_mutex_t itoa_mutex;

void *
thread (void *param)
{
        int i;
        for (i = 0; i < 10000; i++)
        {
                pthread_mutex_lock(&itoa_mutex);
                c=c*123413123+7;;
                pthread_mutex_unlock(&itoa_mutex);
        }
        pthread_exit (param);
}

int
main (int argc, char *argv[])
{
        pthread_t t1, t2;
        pthread_attr_t attr;
        int a, b;
        void *retval;
        a = 0;
        pthread_mutex_init(&itoa_mutex,NULL);
        pthread_attr_init (&attr);
        pthread_create (&t1, &attr, thread, &a);
        if (argc > 1)
        {
                b = 10000;
                pthread_create (&t2, &attr, thread, &b);
                pthread_join (t2, &retval);
        };
        pthread_join (t1, &retval);
        pthread_mutex_destroy(&itoa_mutex);
        printf("c=%d\n",c);
        exit (0);
}
```