HANDOUT
Embedded ARM Board Client/Server Programming
HL

I. Background

Socket programming for embedded ARM board client/server communication is described in this handout. Socket is a software endpoint (protocol) for inter-process communications:

1. Client/Server program running on ARM board
   i. Create socket, and
   ii. Waits for client connection.
2. Client/Server program running on a host PC
   i. Create socket and establish a connection to the server, and
   ii. Reads server's process ID (PID) and host name.

Generally, the above described operation can be described in Figure 1.

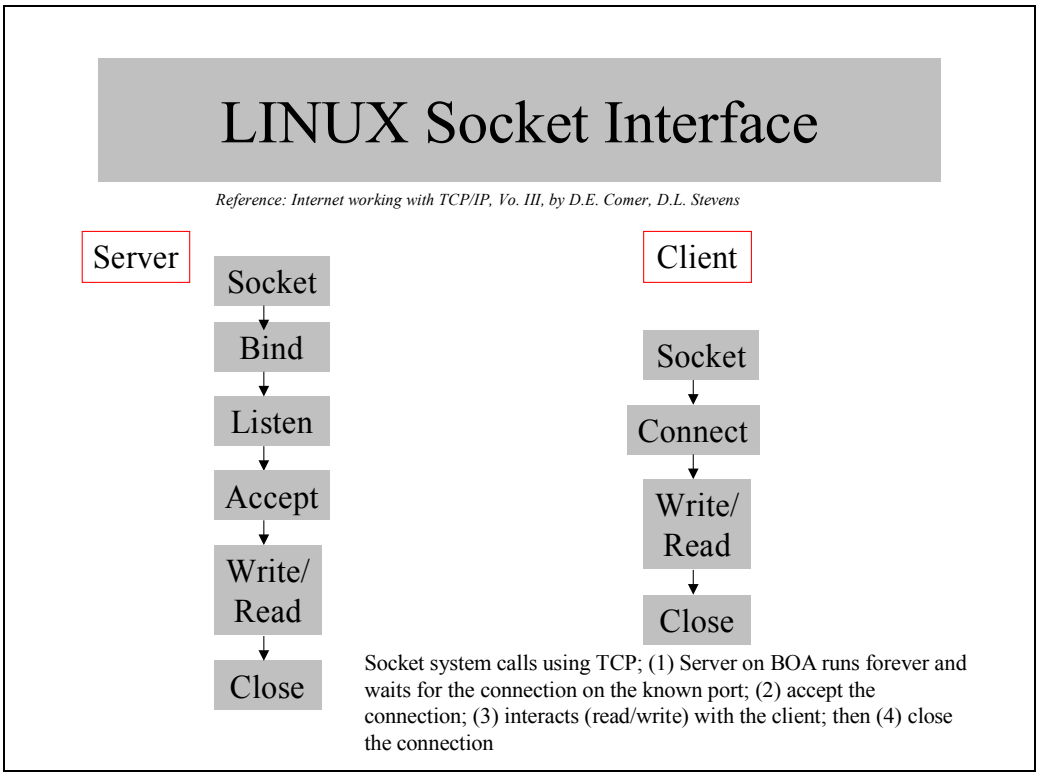

Figure 1. Illustration of Socket Interface using TCP.

The summary of socket functions is given in Table 1.

	Socket	Create a socket for inter-process communications
	connect	Connect to a socket
	write	Send outgoing data via the connection
	read	Receive incoming data via the connection
	close	Terminate communication and de-allocate the descriptor
	bind	Bind a local IP address and protocol port to a socket
	listen	Place a socket in a passive mode and set incoming TCP connections the server

		will en-queue
	Accept	Accept the next incoming connection (server)
	recv	Receiving the next incoming datagram
	recvmsg	Receiving the next incoming datagram
	recvfrom	Receiving the next incoming datagram and recording its source end point address
	send	Send outgoing datagram
	sendmsg	Send outgoing datagram
	sendto	Send outgoing datagram usually to a pre-recorded end point address
	shutdown	Terminate PCP connection
	getpeername	After connection, obtain the remote machines end point address from a socket
	getsockopt	Obtain the current options for a socket
	setsockopt	Change the options for a socket

Table 1. Socket, ***Internet working with TCP/IP, Vo. III, by D.E. Comer, D.L. Stevens.***

II. Server Program Implementation

The basic functionality in the pseudo code for the server program is summarized in the following table, Table 2.

Program module	**Functionality**	**Description**

`int open_socket_and_wait_for_connection()`	`sin.sin_family = AF_INET;`
	`sin.sin_addr.s_addr = INADDR_ANY;`
	`sin.sin_port = htons(SOCKET_PORT);`
	`sinsize = sizeof(sin);`
		A1

	`if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)`	**A2**
	`if (bind(s, (struct sockaddr *)&sin, sinsize) == -1)`	**A3**
	`if (listen(s, 5) == -1)`	**A4**
	`if ((fd = accept(s, (struct sockaddr *)&pin, &sinsize)) == -1)`	**A5**
`void pid_host()`	`pid = getpid();` `if (write(STDOUT_FILENO, &pid, sizeof(int)) != 4)`	**B1**
	`gethostname(hostname, 255);` `if (write(STDOUT_FILENO, hostname, strlen(hostname) + 1) == -1)`	**B2**
`int main (void)`	`fd =open_socket_and_wait_for_connection();`	**C1**
	`pid_host();`	**C2**

Table 2. Basic Functionality of the Server Program

The implementation is given in Figure 2.

```
//*********************************************************************
// Server Program: socket_s.c   Coded by: HL;                         *
// Status: debug;               Version: 0.1;                         *
// Date: March 07;              History: Modified from open source    *
//*********************************************************************
#include <stdio.h>
#include <sys/types.h>     //uClinux: uClibc/include/sys
#include <sys/socket.h>    //uClinux: uClibc/include/sys
#include <netinet/in.h>    //uClinux: uClibc/include/netinet
#include <unistd.h>        //uClinux: uClibc/include/
#define  SOCKET_PORT  6500

int open_socket_and_wait_for_connection()
{
    struct sockaddr_in sin, pin;
    int s, fd, sinsize;
```

```
  sin.sin_family      = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port        = htons(SOCKET_PORT);
  sinsize = sizeof(sin);

    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
      perror("socket"); exit(1);
    }

    if (bind(s, (struct sockaddr *)&sin, sinsize) == -1) {
      perror("bind"); exit(1);
    }

    if (listen(s, 5) == -1) {
      perror("bind") ; exit(1);
    }
    printf("Server started. Waiting for client connection...\n");

    if ((fd = accept(s, (struct sockaddr *)&pin, &sinsize)) == -1) {
      perror("accept"); exit(1);
    }
    printf("Got client connection...\n");
    return(fd);
}

void pid_host()
{
    char hostname[255];
    int pid;

    pid = getpid();
    if (write(STDOUT_FILENO, &pid, sizeof(int)) != 4) {
      perror("pid_host()-write"); exit(1);
    }

    gethostname(hostname, 255);
    if (write(STDOUT_FILENO, hostname, strlen(hostname) + 1) == -1) {
      perror("pid_host()-write"); exit(1);
    }
    printf("Sending my PID (%d) and hostname (%s) to the client...\n",
         pid, hostname);
}

int main (void)
{
    int fd;
    char dummy[255];

    fd = open_socket_and_wait_for_connection();
    pid_host();
    printf("Closing connection...\n");
    close(fd);
}
```

Figure 2. Server Program

III. Client Program Implementation

The basic functionality in the pseudo code for the client program is summarized in Table 3.

Progra	Functionality	Descriptio

m module		n
int get_socket_connection()	hp = gethostbyname("172.19.163.132");	A1

	sin.sin_family = AF_INET;
	sin.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;
	sin.sin_port = htons(SOCKET_PORT);
		A2

	if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)	A3
	if (connect(s, (struct sockaddr *) &sin, sizeof(sin)) == -1)	A4
int main(void)	fd = get_socket_connection();	B1

	read(fd, &cpid, sizeof(int));
	read(fd, hostname, 1);
	while(hostname[i++] != '\0') read(fd,&hostname[i], 1);
	close(fd);
		B2

	ppid = getpid();	B3

Table 3. Client program pseudo code.

The implementation is given in Figure 3.

```
//
// Client Program: socket_c.c
//
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#define  SOCKET_PORT  6500

int get_socket_connection()
{
    struct sockaddr_in sin;
    struct hostent *hp;
    int s;
    hp = gethostbyname("192.168.0.132");
    sin.sin_family      = AF_INET;
    sin.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;
    sin.sin_port        = htons(SOCKET_PORT);

    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)
       { perror("socket"); exit(1); }
    if (connect(s, (struct sockaddr *) &sin, sizeof(sin)) == -1)
       { perror("connect"); exit(1); }
    return(s);
}
int main(void)
{
    int fd;
    int i = 0;
    int cpid, ppid;
    char hostname[255];

    fd = get_socket_connection();

    read(fd, &cpid, sizeof(int));
    read(fd, hostname, 1);
    while(hostname[i++] != '\0') read(fd, &hostname[i], 1);
    close(fd);

    ppid = getpid();

    printf("My pid is %d.  My server on %s has a pid of %d.\n",
          ppid, hostname, cpid);
}
```

Figure 3. Client program.

Appendix A. Generic Address Structure

```
struct  sockaddr {          /* struct to hold an address */
    u_char  sa_len;         /* total length              */
    u_short sa_family;      /* type of address           */
    char    sa_data[14];    /* value of address          */
};
```

Figure 4. *sockaddr* structure.

```
struct  sockaddr_in {           /* struct to hold an address      */
    u_char  sin_len;            /* total length                   */
    u_short sin_family;         /* type of address                */
    u_short sin_port;           /* protocol port number           */
    struct  in_addr sin_addr;   /* IP address (declared to be     */
                                /*  u_long on some systems)       */
    char    sin_zero[8];        /* unused (set to zero)           */
};
```

Figure 5. *sockadd_in* structure for exclusively TCP/IP communications

(End)