// **System Programming III (Chapter 20, Unix book) (Page 863)**

// **Program-1: Machine type in your PC (Page 869-870)**

#include <stdio.h>

int main(void) {

int n;

char \*cp;

n = 0x12345678;

cp = (char \*) (&n);

if (\*cp != 0x78) printf("Big Endian\n");

else printf("Little Endian\n");

return 0;

}

**// Output:**

~/Week-7$ cc Prog1-byteorder.c -w -o Prog1-byteorder

~/Week-7$ ./Prog1-byteorder

Little Endian

~/Week-7$ uname -p

x86\_64

~/Week-7$

**// Program-2: Create pipe (Page 874-875)**

#include<stdio.h>

#include <unistd.h>

int main(void) {

int data\_channel[2];

if (pipe(data\_channel) == -1) {

printf("Pipe failed.\n");

return (1);

}

printf("The pipe descriptors are: \n");

printf("\t Read-end: %d\n", data\_channel[0]);

printf("\t Write-end: %d\n", data\_channel[1]);

return(0);

}

**// Output:**

~/Week-7$ cc Prog2-create\_pipe.c -w -o Prog2-create\_pipe

~/Week-7$ ./Prog2-create\_pipe

The pipe descriptors are:

Read-end: 3

Write-end: 4

3. **// Program-3: Pipe\_talk (Page 875-876)**

#include <stdio.h>

#include<string.h>

#include <unistd.h>

#define SIZE 32

const char \*Child\_Greeting = "Hello, mom!\n";

int main(void) {

int data\_channel[2], pid, nr, nw, nbytes;

char buf[SIZE];

if (pipe(data\_channel) == -1) {

printf("Pipe failed\n");

return(1);

}

pid = fork();

if (pid == -1) {

printf("Fork failed\n");

return(1);

}

nbytes = strlen(Child\_Greeting);

if (pid == 0) {

close(data\_channel[0]);

nw = write(data\_channel[1], Child\_Greeting, nbytes);

if (nw == -1) {

printf("Write error\n");

return(1);

}

return(0);

}

/\* Parent process \*/

close(data\_channel[1]);

nr = read(data\_channel[0], buf, nbytes);

if (nr == -1) {

printf("Read error!\n");

return(1);

}

nw = write(1, buf, nr);

if (nw == -1) {

printf("Write to stdout failed\n");

return(1);

}

printf("Well done, son!\n");

return(0);

}

// **Output:**

~/Week-7$ cc Prog3-pipe\_talk.c -w -o Prog3-pipe\_talk

~/Week-7$ ./Prog3-pipe\_talk

Hello, mom!

Well done, son!

4. **// Program-4: Pipe 2-way talk (Page 876-878)**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#define SIZE 32

const char \*Child\_Greeting = "Hello, mom! \n";

const char \*Parent\_Greeting = "Well done, son!\n";

int main(void){

int pipe1[2], pipe2[2];

int pid, nr,nw, status, sizec, sizep;

char buf[SIZE];

sizec = strlen(Child\_Greeting);

sizep = strlen(Parent\_Greeting);

if (pipe(pipe1) == -1) {

printf("Pipe1 failed\n");

return(1);

}

if(pipe(pipe2) == -1) {

printf("Pipe2 failed\n");

return(1);

}

pid = fork();

if (pid == -1) {

printf("Fork failed\n");

return(1);

}

if (pid == 0) {

close(pipe1[0]);

close(pipe2[1]);

nw = write(pipe1[1], Child\_Greeting, sizec);

if (nw == -1) {

printf("Write to pipe1 error in child \n");

return(1);

}

nr = read(pipe2[0],buf, sizep);

if (nr == -1) {

printf("Read pipe2 error in child\n");

return(1);

}

nw = write(1, buf, nr);

if (nw == -1) {

printf("Write to stdout failed in child\n");

return(1);

}

close(pipe1[1]);

close(pipe2[0]);

return(0);

}

/\* Parent process \*/

close(pipe1[1]);

close(pipe2[0]);

nr = read(pipe1[0], buf, sizec);

if (nr == -1) {

printf("Read pipe1 error in parent\n");

return(1);

}

nw = write(1, buf, nr);

if (nw == -1) {

printf("Write to stdout failed in parent\n");

return(1);

}

nw = write(pipe2[1], Parent\_Greeting, sizep);

if (nw == -1) {

printf("Write to pipe2 error in parent\n");

return(1);

}

close(pipe1[0]);

close(pipe2[1]);

wait(&status);

return(0);

}

**// Output**

~/Week-7$ !c

cc Prog4-pipe\_2way\_talk.c -w -o p2wt

~/Week-7$ !.

./p2wt

Hello, mom!

Well done, son!

5. **// Program-5: Broken pipe (Page 879)**

**#include <unistd.h>**

**#include <stdio.h>**

int main(void)

**{**

int data\_channel[2], nw;

if (pipe(data\_channel) == -1) {

printf("Pipe failed\n");

return(1);

}

printf("The pipe descriptors are: \n");

printf("\t\t Read end: %d\n", data\_channel[0]);

printf("\t\t Write end: %d\n", data\_channel[1]);

close(data\_channel[0]);

nw = write(data\_channel[1], "Hello, world!\n", 14);

printf("This and subsequent statements are never executed.\n");

return(0);

}

**// Output:**

~/Week-7$ !c

cc Prog5-broken\_pipe.c -w -o Prog5-brokenp

~/Week-7$ !.

./Prog5-brokenp

The pipe descriptors are:

Read end: 3

Write end: 4

6. // **Miscellaneous commands (Page 881)**

~/Week-7$ mkfifo fifo1

~/Week-7$ touch greeting

~/Week-7$ ls -i fifo1 greeting

164 fifo1 165 greeting

~/Week-7$ ls -l fifo1 greeting

prw-r--r-- 1 user user 0 May 29 22:21 fifo1

-rw-r--r-- 1 user user 0 May 29 22:21 greeting

~/Week-7$ du fifo1 greeting

1 fifo1

1. greeting

~/Week-7$ cat greeting > fifo1 &

[1] 1079

~/Week-7$ ls -l fifo1 greeting

prw-r--r-- 1 user user 0 May 29 22:21 fifo1

-rw-r--r-- 1 user user 1 May 29 22:28 greeting

~/Week-7$ du fifo1 greeting

1 fifo1

1 greeting

~/Week-7$ cat fifo1

[1]+ Done cat greeting > fifo1

~/Week-7$ du fifo1 greeting

1 fifo1

1. greeting

7. // **Program-6: Prog6-sockets.c (Page 893)**

#include <sys/types.h>

#include <sys/socket.h>

int main(void) {

int s1, s2;

if ((s1 = socket(PF\_INET, SOCK\_STREAM, 0)) == -1) {

printf("SOCK\_STREAM socket failed\n");

return (1);

}

if ((s2 = socket(PF\_INET, SOCK\_DGRAM, 0)) == -1) {

printf("SOCK\_DGRAM socket failed\n");

return(1);

}

printf("The socket descriptor for the stream socket is: %d\n", s1);

printf("The socket descriptor for the datagram socket is: %d\n", s2);

return(0);

}

// Output:

~/Week-7$ cc Prog6-sockets.c -w -o Prog6-sockets

~/Week-7$ ./Prog6-sockets

The socket descriptor for the stream socket is: 3

The socket descriptor for the datagram socket is: 4

8. // Program-7: Server\_TCP

/\* Usage: Server-name Protocol Port

\* Here, Protocol is the transport level protocol and Port is the protocol port number where the service is to be offered \*/

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#define BUFF 256

#define QLEN 16

int main(int argc, char \*argv[]) {

int psock, asock, caddrlen;

struct sockaddr\_in caddr;

/\* Exit if program is not run with two command line arguments \*/

if(argc != 3) {

printf("Usage: server protocol port\n");

return(1);

}

/\* Create a TCP socket, bind name to it, and put it in passive mode \*/

psock = CreatePassiveSock(argv[1], argv[2], QLEN);

caddrlen = sizeof(caddr);

while(1) {

/\* Accept actual connection from the client \*/

if ((asock = accept(psock, (struct sockaddr \*) &caddr, &caddrlen)) == -1) {

printf("Accept failed\n");

return(1);

}

/\* TCP echo service code \*/

EchoServerTCP(asock);

/\* Close active socket \*/

close (asock);

} /\* while \*/

}

/\* Create a TCP or UDP socket socket, bind a name to it, and put it in passive model if it is a TCP socket \*/

/\* 'protocol' is transport layer protocol ("tcp", "udp", etc). \*/

/\* 'portptr' is a pointer to port number as a character string \*/

/\* 'qlen' is the queue length associated with the passive socket. \*/

int CreatePassiveSock(char \*protocol, char \*portstr, int qlen) {

int s, port, type, saddrlen;

char \*endptr;

struct sockaddr\_in saddr;

/\* Convert portstr to port number as integer. Display error message and exit if portstr is not a number. \*/

port = (int) strtol(portstr, &endptr, 10);

if(strcmp("tcp",protocol) == 0)

type = SOCK\_STREAM;

else if (strcmp("udp",protocol) == 0)

type = SOCK\_DGRAM;

else {

printf("Unsupported protocol \n");

return(1);

}

/\* Create a TCP or UDP socket for IPv4 \*/

if ((s = socket(PF\_INET, type, 0)) == -1) {

printf("socket call failed\n");

return(1);

}

/\* Bind address to socket \*/

if (bind(s, (struct sockaddr \*)&saddr, saddrlen) == -1) {

printf("Bind failed\n");

return(1);

}

/\* If it is a TCP socket, put it in passive mode. i.e., redy to listen for incoming connection \*/

if (type == SOCK\_STREAM) {

if (listen(s,qlen) == -1) {

printf("Listen failed\n");

return(1);

/\* Provide echo service to a client, 'sock' is the active socket connected to the client-side socket \*/

void EchoServerTCP(int sock) {

int nr, nw;

char buf[BUFF];

/\* Communicate with client: read and write back \*/

memset(buf, 0, BUFF);

if ((nr = read(sock, buf, BUFF-1)) == -1) {

printf("socket read error\n");

return(1);

}

/\* Write back (echo) the same data to client \*/

if ((nw = write(sock, buf, nr)) == -1) {

printf("socket write error\n");

return(1);

}}

~/Week-7$ !c

cc Prog7-echo-server\_TCP.c -w -o tcpechos

~/Week-7$ !.

./tcpechos tcp 6001 &

[1] 808

~/Week-7$ Bind failed

Accept failed