



# Wi-Fi Communication

## Mini Project

2170710 - Mobile Communication & Wireless Communication

### Guided by:

Prof. Sunil A. Bakhru

Computer Engineering Department,

BVM Engineering College, Vallabh Vidyanagar

### Group Members (E3/G4):

Rushi Patel (140070107040)

Siddharth Goswami (150073107002)

Rahul Mourya (150073107007)

# Project Definition

---

Write a program that:

1. Identifies the Wi-Fi hot-spots in its wireless range.
2. Gives a counts of Wi-Fi hotspots.
3. Also performs file transfer between given two Wi-Fi devices –across different Wi-Fi hotspots.

# Table of contents

---

1. C Program for Scanning Wi-Fi hotspots.
2. C Program for file transfer between two Wi-Fi devices.
3. Some useful Linux commands for Wi-Fi Scanning.

# Program 1 - *Objective*

---

Write a program to list all the Wi-Fi hotspots available and give total count of them using C programming.

## Modules Covered :

1. Identifies the Wi-Fi hot-spots in its wireless range.
2. Gives a counts of Wi-Fi hotspots.

# Program 1 - *File hierarchy*

---

## Program

- `wifi_scan_all.c`

User Program

`include`

Include Folder (contains libraries)

- `interface_helper.c`

General Wi-Fi Interface Functions

- `wifi_scan.h`

Header File (contains necessary struct)

# Code Explanation

wifi\_scan\_all.c

```
1  #include "include/interface_helper.c"
2  #include "include/wifi_scan.h"
3  #include <stdio.h> // printf, scanf
4  #include <string.h>
5
6  // convert bssid to printable hardware mac address
7  const char *bssid_to_string(const uint8_t bssid[BSSID_LENGTH], char bssid_string[BSSID_STRING_LENGTH])
8  {
9      snprintf(bssid_string, BSSID_STRING_LENGTH, "%02x:%02x:%02x:%02x:%02x:%02x",
10             bssid[0], bssid[1], bssid[2], bssid[3], bssid[4], bssid[5]);
11     return bssid_string;
12 }
13
14 const int BSS_INFOS = 10; //the maximum amounts of APs (Access Points) we want to store
15
16 void Usage(char **argv);
17
18 int main(int argc, char **argv)
19 {
20     struct wifi_scan *wifi=NULL; // this stores all the library information
21     struct bss_info bss[BSS_INFOS]; // this is where we are going to keep informatoin about APs (Access
22     char mac[BSSID_STRING_LENGTH]; // a placeholder where we convert BSSID to printable hardware mac ad
23     int status, i, count;
24
25     if(argc != 2) // if the user doesn't know how to use, we display usage instructions
```

# Code Explanation

wifi\_scan\_all.c

enum constants from **wifi\_scan.h**

```
1  #include "include/interface_helper.c"
2  #include "include/wifi_scan.h"
3  #include <stdio.h> // printf, scanf
4  #include <string.h>
5
6  // convert bssid to printable hardware mac address
7  const char *bssid_to_string(const uint8_t bssid[BSSID_LENGTH], char bssid_string[BSSID_STRING_LENGTH])
8  {
9
10 // some constants - mac address length, mac address string length, max length of wireless network id with
11 enum wifi_constants {BSSID_LENGTH=6, BSSID_STRING_LENGTH=18, SSID_MAX_LENGTH_WITH_NULL=33};
12 // anything >=0 should mean that your are associated with the station
13 enum bss_status{BSS_NONE=-1, BSS_AUTHENTICATED=0, BSS_ASSOCIATED=1, BSS_IBSS_JOINED=2};
14
15 // internal data used by the functions
16 struct wifi_scan;
```

```
17
18 int main(int argc, char **argv)
19 {
20     struct wifi_scan *wifi=NULL; // this stores all the library information
21     struct bss_info bss[BSS_INFOS]; // this is where we are going to keep informatoin about APs (Access
22     char mac[BSSID_STRING_LENGTH]; // a placeholder where we convert BSSID to printable hardware mac ad
23     int status, i, count;
24
25     if(argc != 2) // if the user doesn't know how to use, we display usage instructions
```

wifi\_scan.h

# Code Explanation

wifi\_scan\_all.c

```
1  #include "include/interface_helper.c"
2  #include "include/wifi_scan.h"
3  #include <stdio.h> // printf, scanf
4  #include <string.h>
5
6  // convert bssid to printable hardware mac address
7  const char *bssid_to_string(const uint8_t bssid[BSSID_LENGTH], char bssid_string[BSSID_STRING_LENGTH])
8  {
9      snprintf(bssid_string, BSSID_STRING_LENGTH, "%02x:%02x:%02x:%02x:%02x:%02x",
10             bssid[0], bssid[1], bssid[2], bssid[3], bssid[4], bssid[5]);
11     return bssid_string;
12 }
13
14 const int BSSID_LENGTH = 6; // (number of bytes) we want to store
15
16 void Usage(char **argv);
17
18 int main(int argc, char **argv)
19 {
20     struct wifi_scan *wifi=NULL; // this stores all the library information
21     struct bss_info bss[BSS_INFOS]; // this is where we are going to keep informatoin about APs (Access
22     char mac[BSSID_STRING_LENGTH]; // a placeholder where we convert BSSID to printable hardware mac ad
23     int status, i, count;
24
25     if(argc != 2) // if the user doesn't know how to use, we display usage instructions
```

print to fixed buffer of char



# Code Explanation

wifi\_scan\_all.c

```
1  #include "include/interface_helper.c"
2  #include "include/wifi_scan.h"
3  #include <stdio.h> // printf, scanf
4  #include <string.h>
5
6  // convert bssid to printable hardware mac address
7  const char *bssid_to_string(const uint8_t bssid[BSSID_LENGTH], char bssid_string[BSSID_STRING_LENGTH])
8  {
9      snprintf(bssid_string, BSSID_STRING_LENGTH, "%02x:%02x:%02x:%02x:%02x:%02x",
10             bssid[0], bssid[1], bssid[2], bssid[3], bssid[4], bssid[5]);
11     return bssid_string;
12 }
13
14 const int BSS_INFOS = 10; // the maximum number of BSS infos to scan
15
16 void Usage(char **argv);
17
18 int main(int argc, char **argv)
19 {
20     struct wifi_scan *wifi=NULL; // this stores all the library information
21     struct bss_info bss[BSS_INFOS]; // this is where we are going to keep informatoin about APs (Access
22     char mac[BSSID_STRING_LENGTH]; // a placeholder where we convert BSSID to printable hardware mac ad
23     int status, i, count;
24
25     if(argc != 2) // if the user doesn't know how to use, we display usage instructions
```


constant **struct** size of **bss\_info** array

# Code Explanation

wifi\_scan\_all.c

```
14  const int BSS_INFOS = 10; //the maximum amounts of APs (Access Points) we want to store
15
16  void Usage(char **argv);
17
18  int main(int argc, char **argv)
19  {
20      struct wifi_scan *wifi=NULL; // this stores all the library information
21      struct bss_info bss[BSS_INFOS]; // this is where we are going to keep informatoin about APs (Access
22      char mac[BSSID_STRING_LENGTH]; // a placeholder where we convert BSSID to printable hardware mac ad
23      int status, i, count;
24
25      if(argc != 2) // if the user doesn't know how to use, we display usage instructions
26      {
27          Usage(argv);
28          return 0;
29      }
30
31      printf("Triggering scan needs permissions.\n");
32
33      printf("### Close the program with ctrl+c when you're done ###\n\n");
34
35      // initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
36      wifi = wifi_scan_init(argv[1]);
```

**struct from wifi\_scan.h**



# Code Explanation

wifi\_scan\_all.c

```
14  const int BSS_INFOS = 10; //the maximum amounts of APs (Access Points) we want to store
```

```
15  
16  void Usage(char **argv);
```

```
17  
18  int main(int argc, char **argv)
```

```
19  {
```

```
20      struct wifi_scan *wifi=NULL; // this stores all the library information
```

```
21      struct bss_info bss[BSS_INFOS]
```

```
22      char mac[BSSID_STRING_LENGTH];
```

```
23      int status, i, count;
```

```
24  
25      if(argc != 2) // if the user c
```

```
26      {
```

```
27          Usage(argv);
```

```
28          return 0;
```

```
29      }
```

```
30  
31      printf("Triggering scan needs
```

```
32  
33      printf("### Close the program
```

```
34  
35      // initialize the library with network interface ar
```

```
36      wifi = wifi_scan_init(argv[1]);
```

struct from wifi\_scan.h

```
14  // internal data used by the functions
```

```
15  struct wifi_scan;
```

```
16
```

```
17  // a single wireless network can have multiple BSSes working as
```

```
18  struct bss_info
```

```
19  {
```

```
20      uint8_t bssid[BSSID_LENGTH]; //this is hardware mac address
```

```
21      char ssid[SSID_MAX_LENGTH_WITH_NULL]; //this is the name of
```

```
22      enum bss_status status; //anything >=0 means that your are
```

```
23      int32_t signal_mbm; //signal strength in mBm, divide it by
```

```
24      int32_t seen_ms_ago; //when the above information was collec
```

```
25  };
```

```
26
```

wifi\_scan.h

# Code Explanation

wifi\_scan\_all.c

```
35 // initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
36 wifi = wifi_scan_init(argv[1]);
37
38 status = wifi_scan_all(wifi, bss, BSS_INFOS);
39
40 // it may happen that device is unreachable
41 if(status < 0) {
42     perror("Unable to get scan data");
43 }
44 else { // wifi_scan_all returns the number of found stations, it may be greater than BSS_INFOS
45     count = 0;
46 }
47
48 printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");
49
50 for(i = 0; i < status && i < BSS_INFOS; ++i) {
51     printf("%s %s \t \t\t\t%d dBm \t%d \t%s\n",
52           bssid_to_string(bss[i].bssid, mac),           // BSSID (MAC Address)
53           bss[i].ssid,                                   // ESSID (Access Point Name)
54           bss[i].signal_mbm/100,                         // RSS (Received Signal Strength)
55           bss[i].seen_ms_ago,                            // Last Seen (ms)
56           (bss[i].status==BSS_ASSOCIATED ? "associated" : "")) // is connected
57     );
58     count++;
59 }
60
```

pass network interface given from command line argument as a parameter to **wifi\_scan\_init()** function in **wifi\_scan.h**

# Code Explanation

wifi\_scan\_all.c

```
35 // initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
36 wifi = wifi_scan_init(argv[1]);
37
38 status = wifi_scan_all(wifi, bss, BSS_INFOS);
39
40 // it may happen that device is unreachable (e.g.
41 if(status < 0) {
42     perror("Unable to get scan data");
43 }
44 else { // wifi_scan_all returns the number of found stations, it may be greater than BSS_INFOS
45     count = 0;
46 }
47
48 printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");
49
50 for(i = 0; i < status && i < BSS_INFOS; ++i) {
51     printf("%s %s \t \t\t\t\t%d dBm \t%d \t%s\n",
52           bssid_to_string(bss[i].bssid, mac),           // BSSID (MAC Address)
53           bss[i].ssid,                                   // ESSID (Access Point Name)
54           bss[i].signal_mbm/100,                         // RSS (Received Signal Strength)
55           bss[i].seen_ms_ago,                             // Last Seen (ms)
56           (bss[i].status==BSS_ASSOCIATED ? "associated" : "") // is connected
57     );
58     count++;
59 }
```

pass wifi\_scan struct, bss and BSS\_INFOS  
as parameter to  
**wifi\_scan\_all()** function in **wifi\_scan.h**

# Code Explanation

wifi\_scan\_all.c

```
35 // initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
36 wifi = wifi_scan_init(argv[1]);
37
38 status = wifi_scan_all(wifi, bss, BSS_INFOS);
39
40 // it may happen that device is unreachable (e.g.
41 if(status < 0) {
42     perror("Unable to get scan data");
43 }
44 else { // wifi_scan_all returns the number of found
45     count = 0;
46 }
47
48 printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");
49
50 for(i = 0; i < status && i < BSS_INFOS; ++i) {
51     printf("%s %s \t \t\t\t\t%d dBm \t%d \t%s\n",
52           bssid_to_string(bss[i].bssid, mac),           // BSSID (MAC Address)
53           bss[i].ssid,                                   // ESSID (Access Point Name)
54           bss[i].signal_mbm/100,                         // RSS (Received Signal Strength)
55           bss[i].seen_ms_ago,                            // Last Seen (ms)
56           (bss[i].status==BSS_ASSOCIATED ? "associated" : "")) // is connected
57     );
58     count++;
59 }
```

- **wifi\_scan\_all()** makes a passive scan of all networks around.
- Function requires permissions (sudo).
- returns -1 if errors, else count of stations found.

# Code Explanation

# wifi\_scan\_all.c

```
// initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
wifi = wifi_scan_init(argv[1]);

status = wifi_scan_all(wifi, bss, BSS_INFOS);

// it may happen that device is unreachable
if(status < 0) {
    perror("Unable to get scan data");
}
else { // wifi_scan_all returns the number
    count = 0;
}

printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");

for(i = 0; i < status && i < BSS_INFOS; ++i) {
    printf("%s %s \t \t\t\t\t%d dBm \t%d \t%s\n",
        bssid_to_string(bss[i].bssid, mac), // BSSID (MAC Address)
        bss[i].ssid, // ESSID (Access Point Name)
        bss[i].signal_mbm/100, // RSS (Recieved Signal Strengt
        bss[i].seen_ms_ago, // Last Seen (ms)
        (bss[i].status==BSS_ASSOCIATED ? "associated" : "")) // is connected
    );
    count++;
}
```

- use minimum value of i considering **status** and **BSS\_INFOS** declared earlier.
- In other words we want to limit the number of BSS to display by either **status** or **BSS\_INFOS**, whichever is less.

- use minimum value of i considering **status** and **BSS\_INFOS** declared earlier.
- In other words we want to limit the number of BSS to display by either **status** or **BSS\_INFOS**, whichever is less.

# Code Explanation

# wifi\_scan\_all.c

```
// initialize the library with network interface argv[1] (e.g. wlan0 / wlp3s0)
wifi = wifi_scan_init(argv[1]);

status = wifi_scan_all(wifi, bss, BSS_INFOS);

// it may happen that device is unreachable (e.g. the device works in such way that it doesn't
if(status < 0) {
    perror("Unable to get scan data");
}
else { // wifi_scan_all returns the number of scanned APs
    count = 0;
}

printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");

for(i = 0; i < status && i < BSS_INFOS; ++i) {
    printf("%s %s \t \t\t\t%d dBm \t%d \t%s\n",
        bssid_to_string(bss[i].bssid, mac),           // BSSID (MAC Address)
        bss[i].ssid,                                   // ESSID (Access Point Name)
        bss[i].signal_mbm/100,                         // RSS (Received Signal Strength)
        bss[i].seen_ms_ago,                             // Last Seen (ms)
        (bss[i].status==BSS_ASSOCIATED ? "associated" : "") // is connected
    );
    count++;
}
```

• using dot operator to access values of **bss\_info** struct.

- using dot operator to access values of **bss\_info** struct.



# Code Explanation

wifi\_scan\_all.c

```
47
48 printf("MAC \t AP NAME \t RSS \t LAST SEEN \t STATUS \n");
49
50 for(i = 0; i < status && i < BSS_INFOS; ++i) {
51     printf("%s %s \t \t\t\t%d dBm \t%d \t%s\n",
52           bssid_to_string(bss[i].bssid, mac),           // BSSID (MAC Address)
53           bss[i].ssid,                                   // ESSID (Access Point Name)
54           bss[i].signal_mbm/100,
55           bss[i].seen_ms_ago,
56           (bss[i].status==BSS_ASSOCIATED ? "associated" : "disassociated"));
57     count++;
58 }
59
60 printf("\n-----\n");
61 printf("Total hotspot(s) count: %d", count);
62 printf("\n-----\n");
63
64 // free the library resources
65 wifi_scan_close(wifi);
66
67 return 0;
68
69 }
70
```

Finally we close library resource by **wifi\_scan\_close()** function

# Compiling the program

---

```
$> gcc wifi_scan_all.c -lmnl -o wifi_scan_all
```

GNU  
Compiler



C Program

lib-mnl  
(minimalistic Netlink  
communication  
library)

-o Output  
File Name

# Compiling the program

---

```
$> sudo ./wifi_scan_all wlp3s0
```

Elevated  
Permissions



Executable  
generated by  
compiler

Network Interface  
Name  
(find by **ifconfig**)

[\[1\]](#)

# Extra bits on Network Interface Names [\[2\]](#)

---

wlp3s0

wl

wlan

p3

PCI Bus 3

s0

Slot 0

# Output

```
MCWC_Program_WIFI_AP: ./wifi_scan_all

sidx1024@sidx1024-X450JN:~/Documents/MCWC_Program_WIFI_AP$ gcc wifi_scan_all.c -lmnl -o wifi_scan_all
sidx1024@sidx1024-X450JN:~/Documents/MCWC_Program_WIFI_AP$ sudo ./wifi_scan_all wlp3s0
Triggering scan needs permissions.
### Close the program with ctrl+c when you're done ###

MAC ADDRESS          AP NAME              RSS(dbm)   LAST SEEN   STATUS
cc:d3:1e:50:4a:18     Jio of Siddharth     -58         128         associated
90:8d:78:70:b9:47     nirav                -45         88
f0:d7:aa:aa:20:ac     ADYYTW90b0coNSlQbHVz -58         2656
a8:6b:ad:36:95:d4     R 2 D 2              -40         2632
90:8d:78:cd:da:2f     Bring Beer and get access -89         28972
c8:3a:35:52:00:c8     Don't ask for wifi   -86         28324
98:de:d0:bd:1c:62     -88                 1676
7c:91:22:8f:04:48     Don                 -95         26564
f4:f2:6d:47:46:7c     Prohibited           -84         3944

-----
Total hotspot(s) count: 9
-----
sidx1024@sidx1024-X450JN:~/Documents/MCWC_Program_WIFI_AP$
```

# Program 2 - *Objective*

---

Write a program to transfer a file between two Wi-Fi devices over any available WiFi hotspot, using C programming.

Modules Covered :

3. Also performs file transfer between given two Wi-Fi devices –across different Wi-Fi hotspots.

## Program 2 - *File hierarchy*

---

### Program

server

Server Folder

- server.c

Server C Program

client

Client Folder

- client.c

Client C Program

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <arpa/inet.h> // inet_xxx, ntohs
5
6  int main(int argc, char *argv[]) {
7
8      int sockfd = 0;
9      int bytesReceived = 0;
10     char recvBuff[1024];
11
12     memset(recvBuff, '0', sizeof(recvBuff));
13     struct sockaddr_in serv_addr;
14
15     /* Create a socket first */
16     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\nError : Could not create socket\n");
18         return 1;
19     }
20
21     /* Initialize sockaddr_in data structure */
22     serv_addr.sin_family = AF_INET;
```

**sockfd** -> socket file descriptor

In Unix and related computers operating systems, a file descriptor is an abstract indicator used to access a file or other input/output resource, such as a pipe or network connection.

File descriptors are part of the POSIX application programming interface.

A file descriptor is a non-negative integer, represented in C programming language as the type `int`.



# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <arpa/inet.h> // inet_xxx, ntohs
5
6  int main(int argc, char *argv[]) {
7
8      int sockfd = 0;
9      int bytesReceived = 0;
10     char recvBuff[1024];
11
12     memset(recvBuff, '0', sizeof(recvBuff));
13     struct sockaddr_in serv_addr;
14
15     /* Create a socket first */
16     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\nError : Could not create socket\n");
18         return 1;
19     }
20
21     /* Initialize sockaddr_in data structure */
22     serv_addr.sin_family = AF_INET;
```

**sockfd** -> socket file descriptor

In Unix and related computers operating systems, a file descriptor is an abstract indicator used to access a file or other input/output resource, such as a pipe or network connection.

File descriptors are part of the POSIX application programming interface.

A file descriptor is a non-negative integer, represented in C programming language as the type `int`.

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <arpa/inet.h> // inet_xxx, ntohs
5
6  int main(int argc, char *argv[]) {
7
8      int sockfd = 0;
9      int bytesReceived = 0;
10     char recvBuff[1024];
11
12     memset(recvBuff, '0', sizeof(recvBuff));
13     struct sockaddr_in serv_addr;
14
15     /* Create a socket first */
16     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\nError : Could not create socket \n");
18         return 1;
19     }
20
21     /* Initialize sockaddr_in data structure */
22     serv_addr.sin_family = AF_INET;
```

**recvBuff** -> receive buffer, char array of 1024 bytes

**memset** (*array*, replacement, sizeof *array*) -> clear **recvBuff** and fill with 0 zeros.

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <arpa/inet.h> // inet_xxx, ntohs
5
6  int main(int argc, char *argv[]) {
7
8      int sockfd = 0;
9      int bytesReceived = 0;
10     char recvBuff[1024];
11
12     memset(recvBuff, '0', sizeof(recvBuff));
13     struct sockaddr_in serv_addr;
14
15     /* Create a socket first */
16     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\nError : Could not create socket \n");
18         return 1;
19     }
20
21     /* Initialize sockaddr_in data structure */
22     serv_addr.sin_family = AF_INET;
```

**sockaddr\_in** -> inbuilt-struct provided by inet header file  
**memset** (*array*, replacement, sizeof *array*) -> clear **recvBuff** and fill with 0 zeros.

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <sys/types.h> // socklen_t
5  // sockaddr_in
6  #include <netinet/in.h>
7
8  struct sockaddr_in {
9      short sin_family; // e.g. AF_INET
10     unsigned short sin_port; // e.g. htons(3490)
11     struct in_addr sin_addr; // see struct in_addr, below
12     char sin_zero[8]; // zero this if you want to
13 };
14
15 struct in_addr {
16     unsigned long s_addr; // load with inet_aton()
17 };
18
19
20
21
22 serv_addr.sin_family = AF_INET;
```

provided by inet  
t, sizeof array) ->  
0 zeros.

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256):
4  # 1 // sockaddr_in
5  2
6  3 #include <netinet/in.h>
7  4
8  5 struct sockaddr_in {
9  6     short      sin_family;    // e.g. AF_INET
10  7     unsigned short sin_port;   // e.g. htons(3490)
11  8     struct in_addr sin_addr;    // see struct in_addr, below
12  9     char        sin_zero[8];   // zero this if you want to
13 10 };
14 11
15 12 struct in_addr {
16 13     unsigned long s_addr; // load with
17 14 };
18 15
19 16
20 17
21 18
22 19
23 20
24 21
25 22
26 23
27 24
28 25
29 26
30 27
31 28
32 29
33 30
34 31
35 32
36 33
37 34
38 35
39 36
40 37
41 38
42 39
43 40
44 41
45 42
46 43
47 44
48 45
49 46
50 47
51 48
52 49
53 50
54 51
55 52
56 53
57 54
58 55
59 56
60 57
61 58
62 59
63 60
64 61
65 62
66 63
67 64
68 65
69 66
70 67
71 68
72 69
73 70
74 71
75 72
76 73
77 74
78 75
79 76
80 77
81 78
82 79
83 80
84 81
85 82
86 83
87 84
88 85
89 86
90 87
91 88
92 89
93 90
94 91
95 92
96 93
97 94
98 95
99 96
100 97
101 98
102 99
103 100
104 101
105 102
106 103
107 104
108 105
109 106
110 107
111 108
112 109
113 110
114 111
115 112
116 113
117 114
118 115
119 116
120 117
121 118
122 119
123 120
124 121
125 122
126 123
127 124
128 125
129 126
130 127
131 128
132 129
133 130
134 131
135 132
136 133
137 134
138 135
139 136
140 137
141 138
142 139
143 140
144 141
145 142
146 143
147 144
148 145
149 146
150 147
151 148
152 149
153 150
154 151
155 152
156 153
157 154
158 155
159 156
160 157
161 158
162 159
163 160
164 161
165 162
166 163
167 164
168 165
169 166
170 167
171 168
172 169
173 170
174 171
175 172
176 173
177 174
178 175
179 176
180 177
181 178
182 179
183 180
184 181
185 182
186 183
187 184
188 185
189 186
190 187
191 188
192 189
193 190
194 191
195 192
196 193
197 194
198 195
199 196
200 197
201 198
202 199
203 200
204 201
205 202
206 203
207 204
208 205
209 206
210 207
211 208
212 209
213 210
214 211
215 212
216 213
217 214
218 215
219 216
220 217
221 218
222 219
223 220
224 221
225 222
226 223
227 224
228 225
229 226
230 227
231 228
232 229
233 230
234 231
235 232
236 233
237 234
238 235
239 236
240 237
241 238
242 239
243 240
244 241
245 242
246 243
247 244
248 245
249 246
250 247
251 248
252 249
253 250
254 251
255 252
256 253
257 254
258 255
259 256
260 257
261 258
262 259
263 260
264 261
265 262
266 263
267 264
268 265
269 266
270 267
271 268
272 269
273 270
274 271
275 272
276 273
277 274
278 275
279 276
280 277
281 278
282 279
283 280
284 281
285 282
286 283
287 284
288 285
289 286
290 287
291 288
292 289
293 290
294 291
295 292
296 293
297 294
298 295
299 296
300 297
301 298
302 299
303 300
304 301
305 302
306 303
307 304
308 305
309 306
310 307
311 308
312 309
313 310
314 311
315 312
316 313
317 314
318 315
319 316
320 317
321 318
322 319
323 320
324 321
325 322
326 323
327 324
328 325
329 326
330 327
331 328
332 329
333 330
334 331
335 332
336 333
337 334
338 335
339 336
340 337
341 338
342 339
343 340
344 341
345 342
346 343
347 344
348 345
349 346
350 347
351 348
352 349
353 350
354 351
355 352
356 353
357 354
358 355
359 356
360 357
361 358
362 359
363 360
364 361
365 362
366 363
367 364
368 365
369 366
370 367
371 368
372 369
373 370
374 371
375 372
376 373
377 374
378 375
379 376
380 377
381 378
382 379
383 380
384 381
385 382
386 383
387 384
388 385
389 386
390 387
391 388
392 389
393 390
394 391
395 392
396 393
397 394
398 395
399 396
400 397
401 398
402 399
403 400
404 401
405 402
406 403
407 404
408 405
409 406
410 407
411 408
412 409
413 410
414 411
415 412
416 413
417 414
418 415
419 416
420 417
421 418
422 419
423 420
424 421
425 422
426 423
427 424
428 425
429 426
430 427
431 428
432 429
433 430
434 431
435 432
436 433
437 434
438 435
439 436
440 437
441 438
442 439
443 440
444 441
445 442
446 443
447 444
448 445
449 446
450 447
451 448
452 449
453 450
454 451
455 452
456 453
457 454
458 455
459 456
460 457
461 458
462 459
463 460
464 461
465 462
466 463
467 464
468 465
469 466
470 467
471 468
472 469
473 470
474 471
475 472
476 473
477 474
478 475
479 476
480 477
481 478
482 479
483 480
484 481
485 482
486 483
487 484
488 485
489 486
490 487
491 488
492 489
493 490
494 491
495 492
496 493
497 494
498 495
499 496
500 497
501 498
502 499
503 500
504 501
505 502
506 503
507 504
508 505
509 506
510 507
511 508
512 509
513 510
514 511
515 512
516 513
517 514
518 515
519 516
520 517
521 518
522 519
523 520
524 521
525 522
526 523
527 524
528 525
529 526
530 527
531 528
532 529
533 530
534 531
535 532
536 533
537 534
538 535
539 536
540 537
541 538
542 539
543 540
544 541
545 542
546 543
547 544
548 545
549 546
550 547
551 548
552 549
553 550
554 551
555 552
556 553
557 554
558 555
559 556
560 557
561 558
562 559
563 560
564 561
565 562
566 563
567 564
568 565
569 566
570 567
571 568
572 569
573 570
574 571
575 572
576 573
577 574
578 575
579 576
580 577
581 578
582 579
583 580
584 581
585 582
586 583
587 584
588 585
589 586
590 587
591 588
592 589
593 590
594 591
595 592
596 593
597 594
598 595
599 596
600 597
601 598
602 599
603 600
604 601
605 602
606 603
607 604
608 605
609 606
610 607
611 608
612 609
613 610
614 611
615 612
616 613
617 614
618 615
619 616
620 617
621 618
622 619
623 620
624 621
625 622
626 623
627 624
628 625
629 626
630 627
631 628
632 629
633 630
634 631
635 632
636 633
637 634
638 635
639 636
640 637
641 638
642 639
643 640
644 641
645 642
646 643
647 644
648 645
649 646
650 647
651 648
652 649
653 650
654 651
655 652
656 653
657 654
658 655
659 656
660 657
661 658
662 659
663 660
664 661
665 662
666 663
667 664
668 665
669 666
670 667
671 668
672 669
673 670
674 671
675 672
676 673
677 674
678 675
679 676
680 677
681 678
682 679
683 680
684 681
685 682
686 683
687 684
688 685
689 686
690 687
691 688
692 689
693 690
694 691
695 692
696 693
697 694
698 695
699 696
700 697
701 698
702 699
703 700
704 701
705 702
706 703
707 704
708 705
709 706
710 707
711 708
712 709
713 710
714 711
715 712
716 713
717 714
718 715
719 716
720 717
721 718
722 719
723 720
724 721
725 722
726 723
727 724
728 725
729 726
730 727
731 728
732 729
733 730
734 731
735 732
736 733
737 734
738 735
739 736
740 737
741 738
742 739
743 740
744 741
745 742
746 743
747 744
748 745
749 746
750 747
751 748
752 749
753 750
754 751
755 752
756 753
757 754
758 755
759 756
760 757
761 758
762 759
763 760
764 761
765 762
766 763
767 764
768 765
769 766
770 767
771 768
772 769
773 770
774 771
775 772
776 773
777 774
778 775
779 776
780 777
781 778
782 779
783 780
784 781
785 782
786 783
787 784
788 785
789 786
790 787
791 788
792 789
793 790
794 791
795 792
796 793
797 794
798 795
799 796
800 797
801 798
802 799
803 800
804 801
805 802
806 803
807 804
808 805
809 806
810 807
811 808
812 809
813 810
814 811
815 812
816 813
817 814
818 815
819 816
820 817
821 818
822 819
823 820
824 821
825 822
826 823
827 824
828 825
829 826
830 827
831 828
832 829
833 830
834 831
835 832
836 833
837 834
838 835
839 836
840 837
841 838
842 839
843 840
844 841
845 842
846 843
847 844
848 845
849 846
850 847
851 848
852 849
853 850
854 851
855 852
856 853
857 854
858 855
859 856
860 857
861 858
862 859
863 860
864 861
865 862
866 863
867 864
868 865
869 866
870 867
871 868
872 869
873 870
874 871
875 872
876 873
877 874
878 875
879 876
880 877
881 878
882 879
883 880
884 881
885 882
886 883
887 884
888 885
889 886
890 887
891 888
892 889
893 890
894 891
895 892
896 893
897 894
898 895
899 896
900 897
901 898
902 899
903 900
904 901
905 902
906 903
907 904
908 905
909 906
910 907
911 908
912 909
913 910
914 911
915 912
916 913
917 914
918 915
919 916
920 917
921 918
922 919
923 920
924 921
925 922
926 923
927 924
928 925
929 926
930 927
931 928
932 929
933 930
934 931
935 932
936 933
937 934
938 935
939 936
940 937
941 938
942 939
943 940
944 941
945 942
946 943
947 944
948 945
949 946
950 947
951 948
952 949
953 950
954 951
955 952
956 953
957 954
958 955
959 956
960 957
961 958
962 959
963 960
964 961
965 962
966 963
967 964
968 965
969 966
970 967
971 968
972 969
973 970
974 971
975 972
976 973
977 974
978 975
979 976
980 977
981 978
982 979
983 980
984 981
985 982
986 983
987 984
988 985
989 986
990 987
991 988
992 989
993 990
994 991
995 992
996 993
997 994
998 995
999 996
1000 997
1001 998
1002 999
1003 1000
1004 1001
1005 1002
1006 1003
1007 1004
1008 1005
1009 1006
1010 1007
1011 1008
1012 1009
1013 1010
1014 1011
1015 1012
1016 1013
1017 1014
1018 1015
1019 1016
1020 1017
1021 1018
1022 1019
1023 1020
1024 1021
1025 1022
1026 1023
1027 1024
1028 1025
1029 1026
1030 1027
1031 1028
1032 1029
1033 1030
1034 1031
1035 1032
1036 1033
1037 1034
1038 1035
1039 1036
1040 1037
1041 1038
1042 1039
1043 1040
1044 1041
1045 1042
1046 1043
1047 1044
1048 1045
1049 1046
1050 1047
1051 1048
1052 1049
1053 1050
1054 1051
1055 1052
1056 1053
1057 1054
1058 1055
1059 1056
1060 1057
1061 1058
1062 1059
1063 1060
1064 1061
1065 1062
1066 1063
1067 1064
1068 1065
1069 1066
1070 1067
1071 1068
1072 1069
1073 1070
1074 1071
1075 1072
1076 1073
1077 1074
1078 1075
1079 1076
1080 1077
1081 1078
1082 1079
1083 1080
1084 1081
1085 1082
1086 1083
1087 1084
1088 1085
1089 1086
1090 1087
1091 1088
1092 1089
1093 1090
1094 1091
1095 1092
1096 1093
1097 1094
1098 1095
1099 1096
1100 1097
1101 1098
1102 1099
1103 1100
1104 1101
1105 1102
1106 1103
1107 1104
1108 1105
1109 1106
1110 1107
1111 1108
1112 1109
1113 1110
1114 1111
1115 1112
1116 1113
1117 1114
1118 1115
1119 1116
1120 1117
1121 1118
1122 1119
1123 1120
1124 1121
1125 1122
1126 1123
1127 1124
1128 1125
1129 1126
1130 1127
1131 1128
1132 1129
1133 1130
1134 1131
1135 1132
1136 1133
1137 1134
1138 1135
1139 1136
1140 1137
1141 1138
1142 1139
1143 1140
1144 1141
1145 1142
1146 1143
1147 1144
1148 1145
1149 1146
1150 1147
1151 1148
1152 1149
1153 1150
1154 1151
1155 1152
1156 1153
1157 1154
1158 1155
1159 1156
1160 1157
1161 1158
1162 1159
1163 1160
1164 1161
1165 1162
1166 1163
1167 1164
1168 1165
1169 1166
1170 1167
1171 1168
1172 1169
1173 1170
1174 1171
1175 1172
1176 1173
1177 1174
1178 1175
1179 1176
1180 1177
1181 1178
1182 1179
1183 1180
1184 1181
1185 1182
1186 1183
1187 1184
1188 1185
1189 1186
1190 1187
1191 1188
1192 1189
1193 1190
1194 1191
1195 1192
1196 1193
1197 1194
1198 1195
1199 1196
1200 1197
1201 1198
1202 1199
1203 1200
1204 1201
1205 1202
1206 1203
1207 1204
1208 1205
1209 1206
1210 1207
1211 1208
1212 1209
1213 1210
1214 1211
1215 1212
1216 1213
1217 1214
1218 1215
1219 1216
1220 1217
1221 1218
1222 1219
1223 1220
1224 1221
1225 1222
1226 1223
1227 1224
1228 1225
1229 1226
1230 1227
1231 1228
1232 1229
1233 1230
1234 1231
1235 1232
1236 1233
1237 1234
1238 1235
1239 1236
1240 1237
1241 1238
1242 1239
1243 1240
1244 1241
1245 1242
1246 1243
1247 1244
1248 1245
1249 1246
1250 1247
1251 1248
1252 1249
1253 1250
1254 1251
1255 1252
1256 1253
1257 1254
1258 1255
1259 1256
1260 1257
1261 1258
1262 1259
1263 1260
1264 1261
1265 1262
1266 1263
1267 1264
1268 1265
1269 1266
1270 1267
1271 1268
1272 1269
1273 1270
1274 1271
1275 1272
1276 1273
1277 1274
1278 1275
1279 1276
1280 1277
1281 1278
1282 1279
1283 1280
1284 1281
1285 1282
1286 1283
1287 1284
1288 1285
1289 1286
1290 1287
1291 1288
1292 1289
1293 1290
1294 1291
1295 1292
1296 1293
1297 1294
1298 1295
1299 1296
1300 1297
1301 1298
1302 1299
1303 1300
1304 1301
1305 1302
1306 1303
1307 1304
1308 1305
1309 1306
1310 1307
1311 1308
1312 1309
1313 1310
1314 1311
1315 1312
1316 1313
1317 1314
1318 1315
1319 1316
1320 1317
1321 1318
1322 1319
1323 1320
1324 1321
1325 1322
1326 1323
1327 1324
1328 1325
1329 1326
1330 1327
1331 1328
1332 1329
1333 1330
1334 1331
1335 1332
1336 1333
1337 1334
1338 1335
1339 1336
1340 1337
1341 1338
1342 1339
1343 1340
1344 1341
1345 1342
1346 1343
1347 1344
1348 1345
1349 1346
1350 1347
1351 1348
1352 1349
1353 1350
1354 1351
1355 1352
1356 1353
1357 1354
1358 1355
1359 1356
1360 1357
1361 1358
1362 1359
1363 1360
1364 1361
1365 1362
1366 1363
1367 1364
1368 1365
1369 1366
1370 1367
1371 1368
1372 1369
1373 1370
1374 1371
1375 1372
1376 1373
1377 1374
1378 1375
1379 1376
1380 1377
1381 1378
1382 1379
1383 1380
1384 1381
1385 1382
1386 1383
1387 1384
1388 1385
1389 1386
1390 1387
1391 1388
1392 1389
1393 1390
1394 1391
1395 1392
1396 1393
1397 1394
1398 1395
1399 1396
1400 1397
1401 1398
1402 1399
1403 1400
1404 1401
1405 1402
1406 1403
1407 1404
1408 1405
1409 1406
1410 1407
1411 1408
1412 1409
1413 1410
1414 1411
1415 1412
1416 1413
1417 1414
1418 1415
1419 1416
1420 1417
1421 1418
1422 1419
1423 1420
1424 1421
1425 1422
1426 1423
1427 1424
1428 1425
1429 1426
1430 1427
1431 1428
1432 1429
1433 1430
1434 1431
1435 1432
1436 1433
1437 1434
1438 1435
1439 1436
1440 1437
```

# Code Explanation (Cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <unistd.h> // read(sockfd, fname, 256);
4  #include <arpa/inet.h> // inet_xxx, ntohs
5
6  int main(int argc, char *argv[]) {
7
8      int sockfd = 0;
9      int bytesReceived = 0;
10     char recvBuff[1024];
11
12     memset(recvBuff, '0', sizeof(recvBuff));
13     struct sockaddr_in serv_addr;
14
15     /* Create a socket first */
16     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\nError : Could not create socket \n");
18         return 1;
19     }
20
21     /* Initialize sockaddr_in data structure */
```

**socket**(int domain, int type, int protocol) [3]

- Creates an endpoint for communication
- Returns a file descriptor
- domain: **AF\_INET** (Constant IPV4)
  - Specifies the communications domain in which a socket is to be created.
- type: **SOCK\_STREAM** (Provides sequenced, reliable, two-way, connection-based byte streams.)
  - Specifies the type of socket to be created.
- protocol: **0**
  - Specifies a particular protocol to be used with the socket. Specifying a protocol of 0 causes socket() to use an unspecified default protocol appropriate for the requested socket type.

# Code Explanation (Cont.)

```
12  memset(recvBuff, '0', sizeof(recvBuff));
13  struct sockaddr_in serv_addr;
14
15  /* Create a socket first */
16  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17      printf("\nError : Could not create socket \n");
18      return 1;
19  }
20
21  /* Initialize sockaddr_in data structure */
22  serv_addr.sin_family = AF_INET;
23  serv_addr.sin_port = htons(5000); // port
24  char ip[50];
25
26  /* Ask IP address from user if not entered */
27  if (argc < 2) {
28      printf("Enter IP address to connect: ");
29      scanf("%s", ip);
30  }
31  else {
32      strcpy(ip, argv[1]);
33  }
```

**htons()** -> function converts the unsigned short integer **hostshort** from **host byte** order to **network byte** order.

On the i386 the host byte order is Least Significant Byte first, whereas the network byte order, as used on the Internet, is Most Significant Byte first.

# Code Explanation (Cont.)

```
21  /* Initialize sockaddr_in data structure */
22  serv_addr.sin_family = AF_INET;
23  serv_addr.sin_port = htons(5000); // port
24  char ip[50];
25
26  /* Ask IP address from user if not entered in command line arguments */
27  if (argc < 2) {
28      printf("Enter IP address to connect to: ");
29      scanf("%s", ip);
30  }
31  else {
32      strcpy(ip, argv[1]);
33  }
34
35  serv_addr.sin_addr.s_addr = inet_addr(ip);
36
37  /* Attempt a connection */
38  if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
39      printf("\n Error : Connect Failed \n");
40      return 1;
41  }
```

**inet\_addr()** -> converts IP Address in char to unsigned long





# Code Explanation (Cont.)

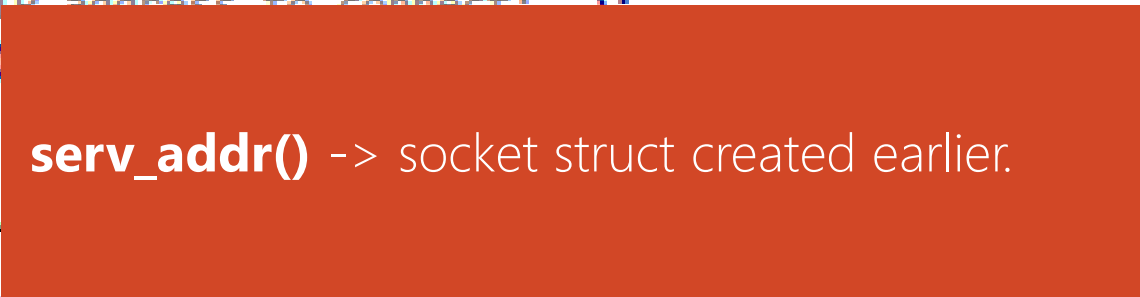
```
21  /* Initialize sockaddr_in data structure */
22  serv_addr.sin_family = AF_INET;
23  serv_addr.sin_port = htons(5000); // port
24  char ip[50];
25
26  /* Ask IP address from user if not entered in command line arguments */
27  if (argc < 2) {
28      printf("Enter IP address to connect: ");
29      scanf("%s", ip);
30  }
31  else {
32      strcpy(ip, argv[1]);
33  }
34
35  serv_addr.sin_addr.s_addr = inet_addr(ip);
36
37  /* Attempt a connection */
38  if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
39      printf("\n Error : Connect Failed \n");
40      return 1;
41  }
```

**connect()** -> System call connects the socket referred to by the file descriptor **sockfd** to the address specified by **addr**. The **addrlen** argument specifies the size of **addr**.

# Code Explanation (Cont.)

```
21  /* Initialize sockaddr_in data structure */
22  serv_addr.sin_family = AF_INET;
23  serv_addr.sin_port = htons(5000); // port
24  char ip[50];
25
26  /* Ask IP address from user if not entered in command line arguments */
27  if (argc < 2) {
28      printf("Enter IP address to connect: ");
29      scanf("%s", ip);
30  }
31  else {
32      strcpy(ip, argv[1]);
33  }
34
35  serv_addr.sin_addr.s_addr = inet_addr(ip);
36
37  /* Attempt a connection */
38  if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
39      printf("\n Error : Connect Failed \n");
40      return 1;
41  }
```

**serv\_addr()** -> socket struct created earlier.



# Code Explanation (Cont.)

```
41 }
42
43 printf("Connected to ip: %s : %d\n", inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port));
44
45 /* Create file where data will be stored */
46 FILE *fp;
47 char fname[100];
48 read(sockfd, fname, 256);
49 printf("File Name: %s\n", fname);
50 printf("Receiving file...");
51 fp = fopen(fname, "ab");
52 if (NULL == fp) {
53     printf("Error opening file");
54     return 1;
55 }
56 long double sz = 1;
57
58 /* Receive data in chunks of 1024 bytes */
59 while ((bytesReceived = read(sockfd, recvBuff, 1024)) > 0) {
60     sz++;
61     printf("Received: %Le Mb", (sz / 1024));
62     fflush(stdout);
63     fwrite(recvBuff, 1, bytesReceived, fp);
64 }
```

**inet\_ntoa()** -> function converts the IP address to **const char**

**ntohs()** -> convert port number from network byte order to host order

# Code Explanation (Cont.)

```
41 }
42
43 printf("Connected to ip: %s : %d\n", inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port));
44
45 /* Create file where data will be stored */
46 FILE *fp;
47 char fname[100];
48 read(sockfd, fname, 256);
49 printf("File Name: %s\n", fname);
50 printf("Receiving file...");
51 fp = fopen(fname, "ab");
52 if (NULL == fp) {
53     printf("Error opening file");
54     return 1;
55 }
56 long double sz = 1;
57
58 /* Receive data in chunks of 1024 bytes */
59 while ((bytesReceived = read(sockfd, recvBuff, 1024)) > 0) {
60     sz++;
61     printf("Received: %Le Mb", (sz / 1024));
62     fflush(stdout);
63     fwrite(recvBuff, 1, bytesReceived, fp);
64 }
65
```

**read()** -> read from a file or socket

# Code Explanation (Cont.)

```
41 }
42
43 printf("Connected to ip: %s : %d\n", inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port));
44
45 /* Create file where data will be stored */
46 FILE *fp;
47 char fname[100];
48 read(sockfd, fname, 256);
49 printf("File Name: %s\n", fname);
50 printf("Receiving file...");
51 fp = fopen(fname, "ab");
52 if (NULL == fp) {
53     printf("Error opening file");
54     return 1;
55 }
56 long double sz = 1;
57
58 /* Receive data in chunks of 1024 bytes */
59 while ((bytesReceived = read(sockfd, recvBuff, 1024)) > 0) {
60     sz++;
61     printf("Received: %Le Mb", (sz / 1024));
62     fflush(stdout);
63     fwrite(recvBuff, 1, bytesReceived, fp);
64 }
65
```

# Code Explanation (Cont.)

```
41 }
42
43 printf("Connected to ip: %s : %d\n", inet_ntoa(serv_addr.sin_addr), ntohs(serv_addr.sin_port));
44
45 /* Create file where data will be stored */
46 FILE *fp;
47 char fname[100];
48 read(sockfd, fname, 256);
49 printf("File Name: %s\n", fname);
50 printf("Receiving file...");
51 fp = fopen(fname, "ab");
52 if (NULL == fp) {
53     printf("Error opening file");
54     return 1;
55 }
56 long double sz = 1;
57
58 /* Receive data in chunks of 1024 bytes */
59 while ((bytesReceived = read(sockfd, recvBuff, 1024)) > 0) {
60     sz++;
61     printf("Received: %Le Mb", (sz / 1024));
62     fflush(stdout);
63     fwrite(recvBuff, 1, bytesReceived, fp);
64 }
65
```

**read()** -> read from a file or socket



# Code Explanation (Cont.)

---

```
52  if (NULL == fp) {  
53      printf("Error opening file");  
54      return 1;  
55  }  
56  long double sz = 1;  
57  
58  /* Receive data in chunks of 1024 bytes */  
59  while ((bytesReceived = read(sockfd, recvBuff, 1024)) > 0) {  
60      sz++;  
61      printf("Received: %Le Mb", (sz / 1024));  
62      fflush(stdout);  
63      fwrite(recvBuff, 1, bytesReceived, fp);  
64  }  
65  
66  if (bytesReceived < 0) {  
67      printf("\n Read Error \n");  
68  }  
69  
70  printf("\nFile OK....Completed\n");  
71  return 0;  
72  }  
73
```

# Code Explanation (Cont.) - Server

```
52 pthread_t tid;
53 struct sockaddr_in serv_addr;
54 int listenfd = 0, ret;
55 char sendBuff[1024];
56 int numrv;
57 size_t clen = 0, slen = 0;
58
59 listenfd = socket(AF_INET, SOCK_STREAM, 0);
60 if (listenfd < 0) {
61     printf("Error in socket creation\n");
62     return -1;
63 }
64
65 printf("Socket retrieve success\n");
66
67 serv_addr.sin_family = AF_INET;
68 serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
69 serv_addr.sin_port = htons(5000);
70 slen = sizeof(serv_addr);
71 ret = bind(listenfd, (struct sockaddr *) &serv_addr, slen);
72 if (ret < 0) {
73     printf("Error in bind\n");
74     return -1;
75 }
76
77 if (listen(listenfd, 10) == -1) {
78     printf("Failed to listen\n");
79     return -1;
80 }
```

**bind()** -> assigns a local protocol address to a socket.





# Code Explanation (Cont.) - Server

```
52 pthread_t tid;
53 struct sockaddr_in serv_addr;
54 int listenfd = 0, ret;
55 char sendBuff[1024];
56 int numrv;
57 size_t clen = 0, slen = 0;
58
59 listenfd = socket(AF_INET, SOCK_STREAM, 0);
60 if (listenfd < 0) {
61     printf("Error in socket creation\n");
62     return -1;
63 }
64
65 printf("Socket retrieve success\n");
66
67 serv_addr.sin_family = AF_INET;
68 serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
69 serv_addr.sin_port = htons(5000);
70 slen = sizeof(serv_addr);
71 ret = bind(listenfd, (struct sockaddr *) &serv_addr, slen);
72 if (ret < 0) {
73     printf("Error in bind\n");
74     return -1;
75 }
76
77 if (listen(listenfd, 10) == -1) {
78     printf("Failed to listen\n");
79     return -1;
80 }
```

**listen()** -> file descriptor and number of connections to allow

# Code Explanation (Cont.) - Server

```
87
88 while (1) {
89     clen = sizeof(c_addr);
90     printf("Waiting...\n");
91     connfd = accept(listenfd, (struct sockaddr *) &c_addr, (socklen_t *) &clen);
92     if (connfd < 0) {
93         printf("Error in accept\n");
94         continue;
95     }
96     err = pthread_create( &tid, NULL, &SendFileToClient, &connfd);
97     if (err != 0)
98         printf("\ncan't create thread :[%s]", strerror(err));
99 }
100 close(connfd);
101 return 0;
102 }
103
```



**pthread\_create**(process id, attributes,  
function, function argument)

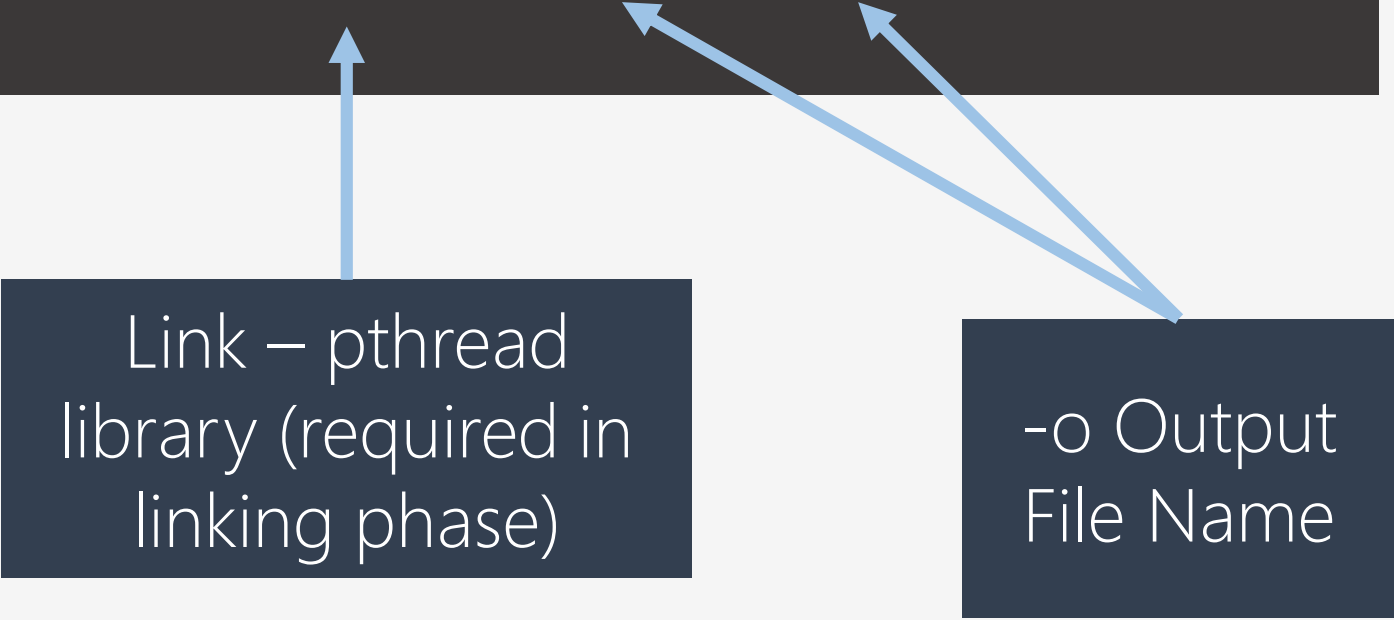
# Code Explanation (Cont.) - Server

```
9 void SendFileToClient(int *arg) {
10     int connfd = (int) *arg;
11     printf("Connection accepted and id: %d\n", connfd);
12     printf("Connected to Client: %s:%d\n", inet_ntoa(c_addr.sin_addr), ntohs(c_addr.sin_port));
13     write(connfd, fname, 256);
14
15     FILE *fp = fopen(fname, "rb");
16     if (fp == NULL) {
17         printf("File open error");
18     }
19
20     /*Read data from file and send it */
21     while (1) {
22         /*First read file in chunks of 256 bytes */
23         unsigned char buff[1024] = {
24             0
25         };
26         int nread = fread(buff, 1, 1024, fp);
27         printf("Bytes read %d \n", nread);
28
29         /*If read was success, send data. */
30         if (nread > 0) {
31             printf("Sending \n");
32             write(connfd, buff, nread);
33         }
34         if (nread < 1024) {
35             if (feof(fp)) {
36                 printf("End of file\n");
37                 printf("File transfer completed for id: %d\n", connfd);
38             }
39             if (ferror(fp))
40                 printf("Error reading\n");
41             break;
42         }
43     }
44     printf("Closing Connection for id: %d\n", connfd);
45     close(connfd);
}
```

# Compiling the program

---

```
$> gcc server.c -lpthread -o server
```



Link – pthread  
library (required in  
linking phase)

-o Output  
File Name

# Compiling the program

---

```
$> ./server bird.jpg
```

# Compiling the program

---

```
$> gcc client.c -o client
```



-o Output  
File Name

# Compiling the program

---

```
$> ./client 192.168.15.131
```

# Output - Server

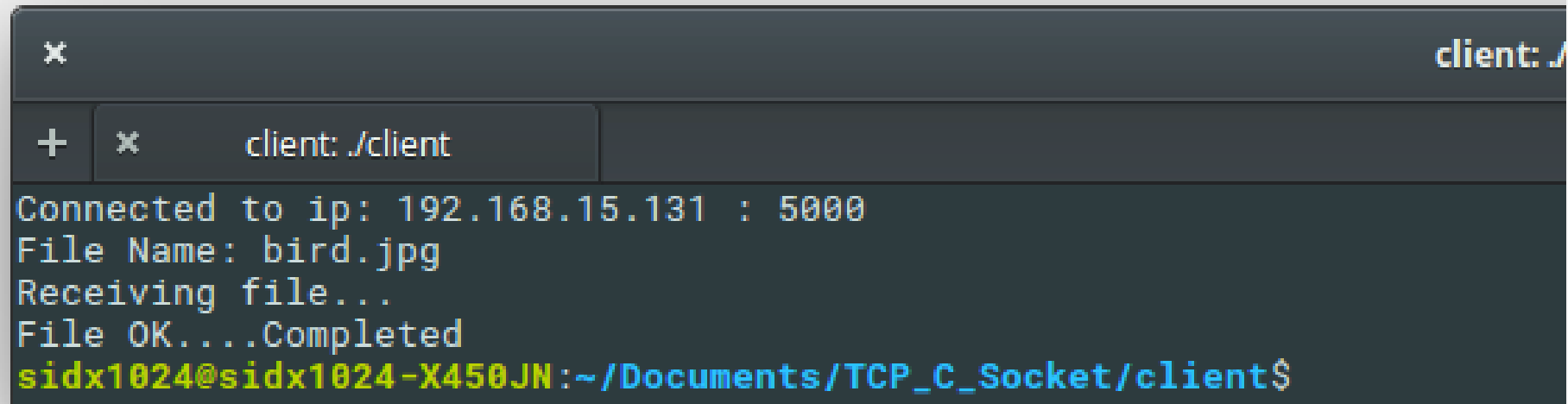
---

```
sidx1024@sidx1024-X450JN:~/Documents/TCP_C_Socket/server$ ./server
Socket retrieve success
Enter file name to send: bird.jpg
Waiting...
Waiting...
Connection accepted and id: 4
Connected to Clent: 192.168.15.131:44286
Segmentation fault (core dumped)
sidx1024@sidx1024-X450JN:~/Documents/TCP_C_Socket/server$
```



# Output - Client

---

A terminal window with a dark background. The title bar shows a close button (x) and the text 'client: ./client'. The terminal content shows a successful connection to 192.168.15.131:5000, file name 'bird.jpg', and successful receipt of the file. The prompt is 'sidx1024@sidx1024-X450JN:~/Documents/TCP\_C\_Socket/client\$'.

```
x client: ./client
+ x client: ./client
Connected to ip: 192.168.15.131 : 5000
File Name: bird.jpg
Receiving file...
File OK....Completed
sidx1024@sidx1024-X450JN:~/Documents/TCP_C_Socket/client$
```

# Some useful Linux commands - 1

---

```
$> sudo iw dev wlp3s0 scan | grep SSID
```

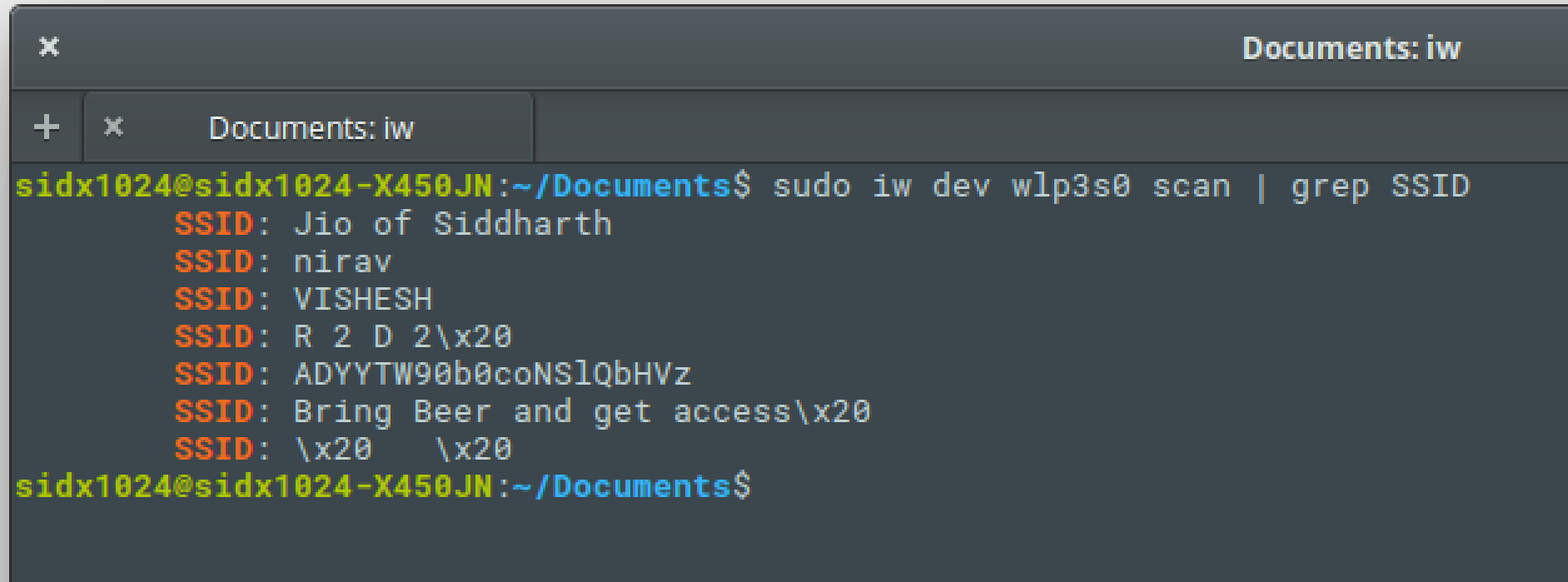
*iw* is a 802.11  
configuration utility  
for wireless devices.

OBJECT  
name  
(dev / phy)

*Global  
regular  
expression  
print*

# Output – iw

---



A terminal window titled "Documents: iw" showing the execution of the command `sudo iw dev wlp3s0 scan | grep SSID`. The output lists several SSIDs in orange text: "Jio of Siddharth", "nirav", "VISHESH", "R 2 D 2", "ADYYTW90b0coNSlQbHVz", "Bring Beer and get access", and two empty entries represented by `\x20`. The prompt indicates the user is `sidx1024` on machine `sidx1024-X450JN` in the `~/Documents` directory.

```
sidx1024@sidx1024-X450JN:~/Documents$ sudo iw dev wlp3s0 scan | grep SSID
SSID: Jio of Siddharth
SSID: nirav
SSID: VISHESH
SSID: R 2 D 2\x20
SSID: ADYYTW90b0coNSlQbHVz
SSID: Bring Beer and get access\x20
SSID: \x20 \x20
sidx1024@sidx1024-X450JN:~/Documents$
```

# Some useful Linux commands - 2

---

```
$> nmcli dev wifi
```

nmcli –  
NetworkManager  
CLI



OBJECT  
name  
(dev / phy)

# Output – nmcli

```
Documents: nmcli
+ Documents: nmcli
sidx1024@sidx1024-X450JN:~/Documents$ nmcli dev wifi
* SSID                MODE  CHAN  RATE        SIGNAL  BARS  SECURITY
  R 2 D 2              Infra 5      54 Mbit/s   100     ████████ WPA1 WPA2
  nirav                Infra 11     54 Mbit/s   92      ████████ WPA1 WPA2
  ADYYTW90b0coNSlQbHVz Infra 5      54 Mbit/s   69      ████████ WPA1 WPA2
* Jio of Siddharth     Infra 1      54 Mbit/s   66      ████████ WPA2
  Don't ask for wifi   Infra 8      54 Mbit/s   30      ████ WPA1
  Prohibited           Infra 2      54 Mbit/s   27      ████ WPA1 WPA2
                        Infra 9      54 Mbit/s   27      ████ WPA1 WPA2
  Bring Beer and get access Infra 6      54 Mbit/s   25      ████ WPA1 WPA2
  VISHESH              Infra 1      54 Mbit/s   19      ████ WPA2
sidx1024@sidx1024-X450JN:~/Documents$
```

# References

---

[1] Network Interface Names, <https://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/>

[2] [systemd/src/udev/udev-builtin-net\\_id.c](#)

[https://github.com/systemd/systemd/blob/master/src/udev/udev-builtin-net\\_id.c#L20](https://github.com/systemd/systemd/blob/master/src/udev/udev-builtin-net_id.c#L20)

[3] Socket Documentation

<http://man7.org/linux/man-pages/man2/socket.2.html>

[4] htons documentation

<https://linux.die.net/man/3/htons>

# Libraries used

---

[1] C/C++ library for monitoring signal strength of WiFi networks,

<https://github.com/bmegli/wifi-scan>

Code for both programs can be downloaded at :

<https://github.com/siddharth1024/MCWC-WiFi-MiniProject>

12<sup>th</sup> October 2017



Thank you