

```

/* -----
   (c) Riduan M ABID, eMail: R.Abid@auburn.edu
   This code was implemented as a partial requirement for
   a Ph.D degree in Computer Science at Auburn University
   Supervisor: Dr. Biaz - eMail: biazsaa@auburn.edu,
   Shelby Center for Engineering Technology- Auburn Unveristy - 2009,
   ----- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <asm/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MP_COUNT 10
#define KEY_ETX 130508
#define KEY_AIRTIME 131313
#define ETX_SHARED_MEMORY_SIZE MP_COUNT * sizeof(struct ETX)

struct ETX {
    __u8 mac_addr[6];
    double df, dr, etx;
};

struct ETX ETXi[MP_COUNT];

void print_ETXi (struct ETX *ETXi);
int comp_mac_addr(__u8 * mac1, __u8 * mac2);
void print_mac_addr(__u8 * mac);

int main() {
    FILE* fp;
    __u8 ch, *shm_ETX, *shm_AIRTIME;
    __u8 NULL_mac[6] = {0}, temp_mac[6] = {0}, temp, found, neighbors;
    int shm_id_ETX, shm_id_AIRTIME, cnt = 0, i;

    key_t key_ETX, key_AIRTIME;
    double rate, error_rate;

    fp = fopen("/proc/net/madwifi/ath0/ratestats_250", "r");
    if (fp == NULL) {
        printf("\n COULD NOT OPEN FILE ");
        return 0;
    }

    // Getting the Neighbors MACs from the ETX process,
    // Locating Shared Memory for ETXi[] entries,
    key_ETX = KEY_ETX;
    if ((shm_id_ETX = shmget(key_ETX, ETX_SHARED_MEMORY_SIZE, IPC_CREAT)) < 0) {
        perror("shmget");
        exit(1);
    }
    // Reading ETXi[] entries,

```

```

if ((shm_ETX = shmat(shmid_ETX, NULL, 0)) == NULL) {
    perror("shmat");
    exit(1);
}
memcpy(ETXi, shm_ETX, ETX_SHARED_MEMORY_SIZE);
print_ETXi(ETXi);

// AIRTIME entries Shared Memory,
key_AIRTIME = KEY_AIRTIME;

if ((shmid_AIRTIME = shmget(key_AIRTIME, ETX_SHARED_MEMORY_SIZE, IPC_CREAT)) <
0) {
    perror("shmget");
    exit(1);
}

if ((shm_AIRTIME = shmat(shmid_AIRTIME, NULL, 0)) == NULL) {
    perror("shmat");
    exit(1);
}

while (1) {
    // Getting the numer of neighbors,
    memcpy(ETXi, shm_ETX, ETX_SHARED_MEMORY_SIZE);

    for (neighbors = 0, i=0; !comp_mac_addr(ETXi[i].mac_addr, NULL_mac);i++)
    {
        if (ETXi[i].etx != 100)
            neighbors++; // Still an Active Neighbor,
    }
    // Feedback,
    printf("\n ----- %d ----- ", cnt++);
    printf("\n -*- There are: %d ETX Negihbors", neighbors);

    fp = fopen("/proc/net/madwifi/ath0/ratestats_250", "r");
    for (found = 0;found < neighbors; ) { // checking that all ETX Neighbors
have been checked                               in the proc file,
        // Find a Colon :
        for(fscanf(fp, "%c", &ch); ch != ':';fscanf(fp, "%c", &ch));
        // Read the MAC addr,
        for(i = 0; i < 5; i++) {
            fscanf(fp, "%x", &temp_mac[i+1]); fgetc(fp);
        }
        // Read the rate;
        for(fscanf(fp, "%c", &ch); ch != '*';fscanf(fp, "%c", &ch));
        fscanf(fp, "%lf", &rate);

        // Feedback,

        // Look up the read mac in the ETX Table,
        for(i = 0; i < MP_COUNT; i++) {
            if (comp_mac_addr(ETXi[i].mac_addr, temp_mac)) {
                printf("\n -- MAC Found: "),

```

```

print_mac_addr(temp_mac);

printf("\n Rate: %.1f", rate);
// Compute the corresponding Airtime and store it in
the dr field of ETX,

if (ETXi[i].etx == 1000) {
    printf("\n Unreachable ..... ");
    ETXi[i].dr = 100;
} else {
    ETXi[i].dr = ETXi[i].etx / rate;
    printf("\n ETX: %.2f", ETXi[i].etx);
    printf("\n Airtime: %.3f", ETXi[i].dr);
}
found++;
break;
    }
}
}
fclose(fp);
// Writing to shared memory,
print_ETXi(ETXi);
memcpy(shm_AIRTIME, ETXi, ETX_SHARED_MEMORY_SIZE);
// ETX were written back as they contain the airtime value in the dr
field!
sleep(2);
}
return 0;
}

void print_ETXi (struct ETX *ETXi) {
    int i;
    for (i = 0; i < MP_COUNT; i++) {
        printf("\n * ");
        print_mac_addr(ETXi[i].mac_addr);
        if (ETXi[i].etx == 100) {
            printf(" ----- ");
            continue;
        }
        printf(" - df: %.2f", ETXi[i].df);
        printf(" - Airtime: %.3f", ETXi[i].dr);
        printf(" - ETX: %.2f", ETXi[i].etx);
    }
    printf("\n");
}

int comp_mac_addr(__u8 * mac1, __u8 * mac2) {
    int i;
    for (i = 0; i < 6; i++)
        if (mac1[i] != mac2[i])
            return 0;
    return 1;
}

void print_mac_addr(__u8 * mac) {
    int i;

```

```
__u8 temp[20];  
for (i = 0; i < 6; i++)  
    printf("%2x:", mac[i]);
```

```
}
```