Interview Assignment – (Air Conditioner System)

AmirAbbas Danesh

در ابتدای فایل ، توابعی که در ادامه از آنها استفاده کردیم را مشخص میکنیم .

```
int Decoder(int *RX_Data,
            void (*powerOn)(),
            void (*powerOff)(),
            int (*checkPacket)(int*, int),
            int* temperature,
            int *sleepTimer);

void powerOn();
void powerOff();
int checkPacket(int *RX_Data, int size);
```

در تابع main() ، از یک packet آماده و از پیش مشخص شده استفاده کردیم . (فرض میکنیم packet ها به صورت سریالی دریافت شده و در یک آرایه بایتی ذخیره شده ، در این مرحله قصد داریم روی آن پردازشی انجام دهیم تا دستور های کد شده را دیکود کنیم .)

همچنین متغیر های Temperature و sleepTimer را تعریف کردیم .

در ادامه ، برای دیکود کردن packet ، تابع Decoder را فراخوانی کرده و ورودی های لازم را به آن پاس دادیم .

```
int main() {

    int RX_Data[] = {0x3A, 0x0F, 0x33, 0x44 , 0x1a , 0x22 ,0x33 , 0x01 , 0x44 , 0x1b , 0x55 , 0x01 , 0xaa ,0x44 ,0x1B, 0x22  , 0xd6} ;
    int Temperature;
    int sleepTimer;
    Decoder(RX_Data, powerOn, powerOff, checkPacket, &Temperature, &sleepTimer);

    return 1;
}
```

در ادامه تابع Decoder را معرفی کرده و نحوه عملکرد آن را توضیح میدهیم .

ورودی هایی که تابع لازم دارد تا با آنها کار کند عبارت است از :

- آرایه بایتی packet یا همان RX_Data
- تابع powerOn که لازم است آن را به صورت پوینتر پاس دهیم .
- تابع powerOff که لازم است آن را به صورت پوینتر پاس دهیم .
- تابع checkPacket که مسئول چک کردن وجود یا عدم وجود نویز روی packet است و همچنین آن را به صورت پوینتر پاس دهیم .
- متغیر temperature که چون مقدار آن را میخواهیم تغییر دهیم ، این متغیر را نیز به صورت پوینتر پاس میدهیم .
- متغیر sleepTimer که چون مقدار آن را میخواهیم تغییر دهیم ، این متغیر را نیز به صورت پوینتر پاس میدهیم .

```
int Decoder(int *RX_Data,
            void (*powerOn)(),
            void (*powerOff)(),
            int (*checkPacket)(int*, int),
            int* temperature,
            int *sleepTimer)
{
```

در ابتدای تابع ، بررسی میکنیم که آیا packet با هدر درستی دریافت شده است یا خیر ، به همین منظور قطعه کد زیر را مینویسیم :

```
if (RX_Data[0] != 0x3A) {
    printf("\n  !! Invalid Packet Header !!\n");
    return 0;
}else
{
    printf("\n    ~ Header is Valid ~ --> byte[0] = 0x3A\n");
}
```

در مرحله بعدی بیت اول پکت را میخوانیم تا Command length یا به عبارتی تعداد دستورات بعدی را در متغیر ذخیره کنیم .

```
int dataPacketSize = RX_Data[1];
printf("    ~ Data size: %d ~ --> byte[1] = 0x%x \n\n", dataPacketSize , RX_Data[1]);
```

پیش از اجرای دستورات باید بررسی کنیم که آیا پکت دریافت شده دارای نویز است یا خیر ، بنابر این از خروجی تابع checkPacket استفاده میکنیم ، درصورتی که مقدار 0 برگردانده شود ، یعنی بیت آخر پکت ، با جمع بایت های 1 تا یکی قبل از LSB (least significant byte) برابر نبوده بنابر این در قسمتی از پکت ، نویز اثر داشته و دستورات بعدی نباید اجرا شوند .

```
int checkFlag = checkPacket(RX_Data, dataPacketSize);
if (!checkFlag) {
    printf("\n!! Invalid Packet !!\n");
    return 0;
}
```

پس از تعریف تابع checkPacket ، پکت و مقداری بایت هایی که باید بررسی شوند را به تابع پاس میدهیم.

بایت آخر که مسئول چک کردن نویز است را در متغیر LSB ذخیره میکنیم.

سپس با تعریف متغیر sumVal در هر مرحله iteration روی درایه های آرایه بایتی ، مقدار آن خانه از حافظه را با sumVal جمع میکنیم . و درنهایت (طبق گفته ی صورت مسئله) با عدد 0xFF ، And میکنیم .

```
int checkPacket(int *RX_Data, int size) {

    int LSB = RX_Data[size + 1] ;
    int sumVal = 0;
    for (int i = 1; i <= size; i++)
    {

        sumVal = sumVal + RX_Data[i] ;
    }
    sumVal = sumVal & 0xFF ;
```

به بدنه تابع Decoder برمیگردیم ، جایی که پس از بررسی عدم وجود نویز ، باید درایه های هر خانه از RX_Data را بررسی کرده و متناسب با دستور عمل کنیم .

```
int flag = 0 ;
for (int i = 2; i <= dataPacketSize ; i++)
{
    // printf("iterate on index %d \n" , i) ;
    if (flag > 0 ) {
        // printf("    pass index %d\n\n" , i ) ;
        flag -- ;
        continue;
    }
    switch (RX_Data[i]) {
        case 0x22:
            // printf("\n  ----PowerOff ------ :RX_Data[%d] --> 0x%x", i , RX_Data[i] );
            powerOff();
            break;
        case 0x33:
            // printf("\n  ----PowerOn ------  :RX_Data[%d] --> 0x%x \n", i , RX_Data[i] );
            powerOn();
        break;
        case 0x44:
            // printf("\n  ----Set Temp ------ :RX_Data[%d] --> 0x%x ", i , RX_Data[i] );
            *temperature = RX_Data[i];
            printf("\n-Temperature Set to: %d\n\n", RX_Data[i+1]);
            flag = 1 ;
            break;
        case 0x55:
            // printf("\n  ----Set Timer ------:RX_Data[%d] --> 0x%x ", i , RX_Data[i] );
            *sleepTimer = (RX_Data[i+1] << 8) | RX_Data[i+2];
            printf("\n-Sleep Timer Set to: %d\n\n", *sleepTimer);
            flag = 2 ;
            break;
        default:
            printf("\n  !! Unknown Command !!\n");
            break;
    }
}

return 1;
```

متغیر flag را تعریف میکنیم (در ادامه توضیح میدهیم چه نیازی به تعریف این متغیر است)

از index شماره دوم شروع به شمارش میکنیم .

از ساختار switch – case استفاده میکنیم تا بررسی کنیم پکت های بعدی شامل چه دستوری هستند .

همانطور که مشخص است دستور 0x22 مربوط به خاموش شدن دستگاه و دستور 0x33 مربوط به روشن شدن دستگاه است .

با رسیدن به این دستورات ، تابع مربوطه اجرا میشود .

```
void powerOn() {
    printf("~ System Powered On ~\n\n");
}

void powerOff() {
    printf("\n~ System Powered Off ~\n\n");
}
```

با رسیدن به دستور 0x44 باید دما را متناسب با بایت بعدی تنظیم کنیم . بنابراین بایت بعدی نباید به عنوان دستور درنظر گرفته شود ، از همین رو از یک متغیر به نام flag استفاده میکنیم تا در اینگونه مواقع به تعداد بایت مورد نظر پس از دستوراتی مانند 0x44 و 0x55 جهش داشته باشیم .

در ابتدای for loop مشاهده میکنید که ابتدا مقدار متغیر flag بررسی میشود که درصورتی که این متغیر مقداری بیش از 0 داشت ، از حلقه ای که در آن هستیم رد شده و مقدار flag را decrement کنیم .

به هنگام دستور تنظیم دما ، flag را مقدار 1 (چون بایت بعدی مربوط به دما است) و به هنگام دستور تنظیم تایمر ، مقدار این متغیر را 2 قرار میدهیم (زیرا 2 بایت بعدی حاوی دستور نبوده و زمان تایمر را مشخص میکنند) .

همچنین ممکن است به دلایلی نویز روی بایت آخر و یکی از دستورات اثر گذاشته و پکتی که از مرحله checkPacket عبور میکند شامل دستوری تعریف نشده باشد . در این شرایط متن Unknown Command چاپ میشود .

در آخر نیز یک پکت به عنوان تست نوشته شده که خروجی آن را بررسی میکنیم

```
int RX_Data[] = {0x3A, 0x0F, 0x33, 0x44 , 0x1a , 0x22 ,0x33 , 0x01 , 0x44 , 0x1b , 0x55 , 0x01 , 0xaa ,0x44 ,0x1D, 0x22  , 0xd8} ;
```

دستور ها به ترتیب شامل :

روشن شدن – تنظیم دما – خاموش شدن – روشن شدن – دستور ناشناخته – تنظیم دما – تنظیم تایمر – تنظیم دما و خاموش شدن است .

خروجی چاپ شده را مشاهده میکنیم .

```
    ~ Header is Valid ~ --> byte[0] = 0x3A
    ~ Data size: 15 ~ --> byte[1] = 0xf


 No Packet Loss

~ System Powered On ~


-Temperature Set to: 26


~ System Powered Off ~

~ System Powered On ~


  !! Unknown Command !!

-Temperature Set to: 27


-Sleep Timer Set to: 426


-Temperature Set to: 29


~ System Powered Off ~
```

با اعمال تغییر در بایت آخر : دستورات اجرا نخواهند شد و داریم :

```
int RX_Data[] = {0x3A, 0x0F, 0x33, 0x44 , 0x1a , 0x22 ,0x33 , 0x01 , 0x44 , 0x1b , 0x55 , 0x01 , 0xaa ,0x44 ,0x1D, 0x22  , 0xd9} ;
```

```
    ~ Header is Valid ~ --> byte[0] = 0x3A
    ~ Data size: 15 ~ --> byte[1] = 0xf


 Noise Detected

!! Invalid Packet !!
```

همچنین اگر header به صورت نادرست وارد شود داریم :

```
int RX_Data[] = {0x4A, 0x0F, 0x33, 0x44 , 0x1a , 0x22 ,0x33 , 0x01 , 0x44 , 0x1b , 0x55 , 0x01 , 0xaa ,0x44 ,0x1D, 0x22  , 0xd9} ;
```

```
    !! Invalid Packet Header !!
```