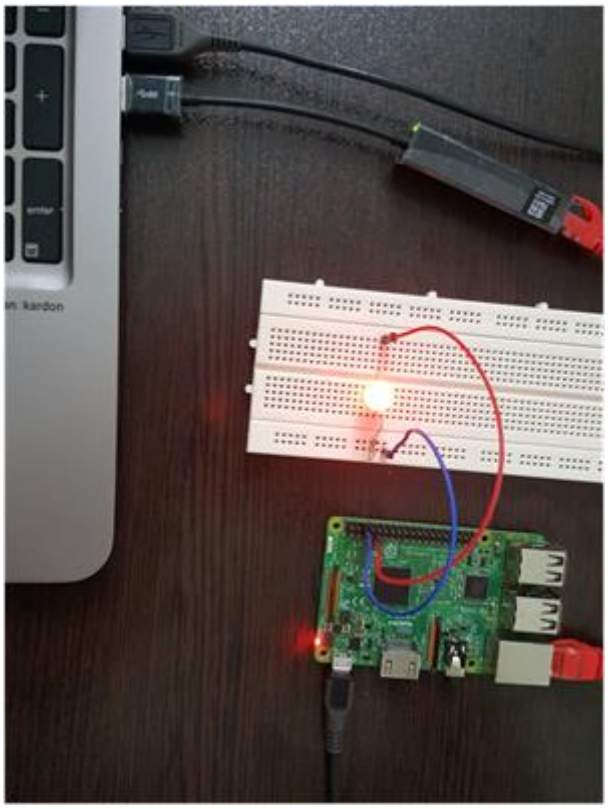
فصل چهارم

شرح پروژه

۴-۱ سخت‌افزار

در این پروژه از یک Raspberry pi 3 متصل به یک breadboard استفاده شده است. بر روی این breadboard یک LED قرار گرفته است که به یکی از پین‌های GPIO رزبری پای متصل است. اتصال اینترنت رزبری پای از طریق یک کابل ethernet متصل به لپ‌تاپ شخصی تأمین شده است. سیستم‌عامل رزبری‌پای مورد استفاده raspbian است که از طریق کارت Micro SD در برد بارگذاری می‌شود. برای ارتباط با برد از اتصال ssh استفاده می‌کنیم. در نهایت برای روشن کردن رزبری‌پای آن را با کابل Micro USB به کامپیوتر شخصی متصل می‌کنیم. در شکل ۲۰ سخت‌افزار پروژه قابل مشاهده است:



شکل ۲۰: سخت‌افزار پروژه

۴-۲ نرم‌افزار

برنامه پیاده سازی شده در این پروژه از دو قطعه Server و Client تشکیل شده است. برنامه Server در واقع مدیر اشیاء است که درخواست‌ها را دریافت و پردازش، تغییرات سخت‌افزاری را ایجاد و پاسخ مناسب

را ار سال می‌کند. برنامه Client در ابتدا مالک د ستگاه رزبری‌پای شده و سپس درخوا ست‌های مورد نظر خود را ارسال می‌کند.

ساختار این پروژه بر مبنای مدل FCAPS که در فصل اول شرح داده شد است. برای نمایش نحوه پیاده‌سازی هر کدام از چهار حوزه مطلوب، به کمک IoTivity و استاندارد OCF، نمونه‌هایی برای هرکدام در نظر گرفته شده است. نحوه کار هر کدام در بخش بعدی شرح داده خواهد شد:

۴-۲-۱ ساختار کد

ساختار برنامه کارساز با راه‌اندازی و تنظیمات اولیه شروع می‌شود. مهم‌ترین عمل‌کرد این قسمت آغازسازی پشته IoTivity با OCInit است. پس از تمام این اقدام‌های اولیه وارد حلقه اصلی برنامه می‌شویم. در این حلقه OCProcess مدام صـدا زده می‌شـود و تمام پردازش‌های سـطح پایین مورد نیاز بدین ترتیب انجام می‌گیرد.

در بخش راه‌اندازی منبع LED را نیز می‌سـازیم. API مورد اسـتفاده این کار یک تابع callback نیز برای مدیریت درخوا ست‌ها می‌گیرد که بخش ا سا سی تابع کار ساز ا ست. هرگاه IoTivity درخوا ستی برای این منبع دریافت کند، این تابع را صـدا می‌زند. این OCEntityHandlerCallback با توجه به flag دریافتی از پشته تابع مناسب آن درخواست را صدا می‌زند. آن تابع نیز به کمک createResponsePayload قالب پا سخ را تنظیم می‌کند و به پشته تحویل می‌دهد. کار ساز این پا سخ را از پشته دریافت می‌کند و پردازش می‌کند.

```
createResponsePayload() {
     // Format response
}

processRequest() {

}

OCEntityHandlerCallback() {
    // How to process each request and send response
```

```
    // Call proper processRequest function based on incoming
request flag
}

main() {
     ////////////////////////////////////////////////////
     //Initial Setup
     //Set Device Info
     //Set Platform Info
     //Initialize persistent storage
     //Initialize IoTivity stack
     //Create Resource
     ////////////////////////////////////////////////////
     //Main loop
     while(!STOP) {
        OCProcess()
     }

     OCStop()
}
```

عمل‌کرد اصلی کارخواه مشابه کارساز است. وجه تمایز اصلی آن در پردازش درخواست‌هاست. کارخواه دو وظیفه ارسال درخواست و پردازش پاسخ را بر عهده دارد. سازوکار کلی درخواست‌ها در بخش بعد شرح داده شده است. قبل از شروع فرایند پشته، یک فهرست نمایش داده می‌شود و با انتخاب گزینه مورد نظر تابع آغازگر آن درخواست فراخوانی می‌شود و درخواست بدین ترتیب به کارساز ارسال می‌شود.

```
RequestCallback() {
    //Called by stack whenever a response is sent
}

createPayload() {
    //For put and post requests to set proper values
}
```

```
Initiator() {
    //Send request to the server
    OCDoRequest();
}

menu() {
    switch() {
        ...
        case n:
            Initiator()
        ...
    }
}

main() {
     ////////////////////////////////////////////////////
     //Initial setup
     //Setup persistent storage
     //Setup DB for provisioning
     //Initiate Stack
     //
     ////////////////////////////////////////////////////
     //Main loop
     while(!STOP) {
       OCProcess()
     }
     OCStop()
}
```

۴-۲-۲ سازوکار درخواست‌ها

۴-۲-۲-۱ مدیریت پیکربندی

برای بخش مدیریت پیکربندی این پروژه دو بخش «انتقال مالکیت دستگاه به کاربر»، «درخواست Discovery منابع» و «ارسال درخواست‌های CRUDN» در نظر گرفته شده است.

انتقال مالکیت دستگاه به کاربر

در یک شبکه OCF برای اینکه کاربر بتواند به گره اینترنت ا شیاء د ستر سی دا شته با شد، باید مالک آن با شد. انتقال مالکیت یک عملکرد سمت کارخواه ا ست. بعد از این عمل ا ست که این کارخواه قادر خواهد بود به دستگاه دسترسی پیدا کند. برای این کار از سرایند[1] ocprovisioningmanager.h استفاده می‌کنیم تا از توابع و APIهای خود پشته برای این عملکردها استفاده کنیم. [۱۷] در اقدام اول می‌توانیم لیست دستگاه‌هایی را که در زیرشبکه هستند و کارخواه ما مالک آن‌ها نیستند را بیابیم:

مشاهده دستگاه‌های بدون مالک

برای دریافت لیست دستگاه‌های unowned از تابع OCDiscoverUnownedDevices استفاده می‌کنیم. این تابع دو ورودی می‌گیرد. یکی مدت زمانی که باید تا دریافت جواب کار ساز باید منتظر بماند و بعدی اشاره‌گر[2] به متغیر نگه‌داری لیست دستگاه‌های یافت‌شده بدون مالک.

```
if(OC_STACK_OK != OCDiscoverUnownedDevices(DISCOVERY_TIMEOUT, &g_unown_list))
    {
        OIC_LOG(ERROR, TAG, "OCDiscoverUnownedDevices API error");
        return -1;
    }
```

ثبت دستگاه و انتقال مالکیت

پس از یافتن و ذخیره‌سازی دستگاه‌های بدون مالک می‌توان به کمک تابع OCDoOwnershipTransfer انتقال مالکیت د ستگاه را انجام داد. این تابع لی ست د ستگاه‌های بدون مالک و یک Callback function می‌گیرد و فرایند را انجام می‌دهد. تابع callback را خود پشته فراخوانی می‌کند و به آن ورودی می‌دهد. طبق این ورودی‌ها می‌توان موفقیت یا عدم‌موفقیت فرایند را تشخیص داد و پیغام مناسب را نمایش داد.

[1] Header

[2] Pointer

در صـــورت موفقیت عملکرد، می‌توان به کمک OCDiscoverOwnedDevices این بار دســـتگاه را در لیست دستگاه‌های owned مشاهده کرد.

یافتن منابع دستگاه

در شبکه اینترنت ا شیاء که با ا ستاندارد OCF تو سعه یافته ا ست منابع متعددی وجود دارد. برخی از این منابع، منابع فیزیکی و برخی مجازی ه ستند. منابع فیزیکی همان ا شیاء در اینترنت ا شیاء ه ستند و منابع مجازی منابع خود IoTivity برای مدیریت اجزای مختلف سامانه. با د ستور Discovery می‌توان محتوای تمام این منابع را مشـــاهده کرد. دســـتور Discovery به کمک تابع OCDoRequest انجام می‌گیرد. این تابع، تابعی عمومی برای انواع درخواســـت‌ها در IoTivity اســـت. به کمک ورودی‌های آن ماهیت این تابع مشخص می‌شود.

```
OCDoRequest(NULL, OC_REST_DISCOVER, queryUri, 0, 0, CT_DEFAULT,
                        OC_LOW_QOS, &cbData, NULL, 0);
```

در این تابع چند ورودی مهم وجود دارد. ابتدا باید نوع متد درخواســـت را مشـــخص کرد. که در اینجا OC_REST_DISCOVER ا ست. سپس باید مسیر منابع را مشخص کرد. این مسیر داخل queryUri ذخیره شـده اسـت و مقدار آن "oic/res" اسـت. این تابع نیز به callback function نیاز دارد تا پشـته هنگام اجرای درخوا ست آن را فرابخواند. این تابع نیز در &cbData وارد شده ا ست. پ شته پس از اجرای درخواســـت نتیجـه را کـه clientresponse نـام دارد را در اختیـار تـابع callback قرار می‌دهـد. Clientresponse یک سـاختار تعریف شـده در IoTivity اسـت که شـامل تمام اطلاعات مربوط به منبع نظیر آدرس، پایه‌بار[1] و نتیجه درخواسـت اسـت. سـپس در callback که خودمان نوشـته‌ایم این پاسـخ را تجزیه[2] کرده و بخش‌های مدنظر را در ترمینال نمایش می‌دهیم. [۱۹]

ارسال درخواست‌های CRUD

درخوا ست‌های CRUD بخش مهمی از هر سامانه کارخواه-کار ساز را ت شکیل می‌دهند. در این پروژه نیز برای نشان دادن نمونه‌ای از هرکدام یک درخواست برای کار با منبع LED در نظر گرفته شده است.

[1] Payload

[2] Parse

سازوکار درخواست‌های این چنینی در در IoTivity مشابه هم است. آدرس دستگاه مقصد، متد مورد نظر و تابع callback را در کارخواه مشخص کرده و درخواست OCDoRequest را اجرا می‌کنیم. پشته درخواست را به کارساز ارسال می‌کند. کارساز هنگام ساختن منبع LED یک EntityHandlerCallback به آن اختصاص می‌دهد که وظیفه پردازش درخواست‌های وارده را دارد. این تابع callback نوع متد را تشخیص داده، پایه‌بار را دریافت می‌کند. اگر متد از نوع CREATE، UPDATE یا DELETE باشد، آنها را پردازش می‌کند. سپس پاسخ مناسب را ساخته و به پشته تحویل می‌دهد. کارخواه این پاسخ را دریافت کرده و در تابع callback مربوطه تجزیه کرده و موارد مورد نیاز را در ترمینال نمایش می‌دهد. [۲۰]

۴-۲-۳ نصب و راه‌اندازی

پس از نصب افزونه‌های مورد نیاز، iotivity را از repository رسمی آن دریافت کرده و در کامپیوتر شخصی و رزبری‌پای نصب می‌کنیم.

پس از نصب سایر کتابخانه‌های مورد نیاز، پین شماره ۷ در رزبری پای را یک خروجی تعریف می‌کنیم. (این پین به LED متصل است). سپس در کامپیوتر شخصی برنامه server را اجرا می‌کنیم و در رزبری‌پای برنامه client را. حال می‌توانیم وضعیت منابع مختلف را مشاهده کنیم و وضعیت LED را کنترل کنیم.

در شکل ۲۱ کارسازی را می‌بینیم که در حال اجراست و پس از تنظیمات اولیه محیط iot، یک درخواست POST را پردازش می‌کند:

```
yasamin@yasamin-UX510UWK:~/iotivity/out/linux/x86_64/release/examples/OCFSecure$ ./server
46:26.326 DEBUG: SERVER_APP: [main] Initializing and registering persistentstorage
46:26.326 DEBUG: SERVER_APP: [main] Initializing IoTivity stack for server
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.350 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: wb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: examples_OCFSecure_ocf_svr_db_server_RFOTM.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
46:26.354 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
46:26.354 INFO: SERVER_APP: [SetPlatformInfo] Set platform ID successfully to 12345678-1234-1234-1234-123456789012
46:26.354 INFO: SERVER_APP: [SetPlatformInfo] Set manufacturer name successfully to Vprime
46:26.354 INFO: SERVER_APP: [main] Platform set successfully
46:26.354 INFO: SERVER_APP: [main] Device set successfully
46:26.354 INFO: SERVER_APP: [main] Created resource successfully
46:26.354 INFO: SERVER_APP: [main] Server is running, press ctrl+c to stop...
```

شکل ۲۱: کارساز در حال اجرا و پردازش

طبق شکل ۲۲ برنامه کارخواه با نمایش فهرستی از دستورات موجود شروع به کار می‌کند:

```
yasamin@yasamin-UX510UWK:~/iotivity/out/linux/x86_64/release/examples/OCFSecure$ ./client
50:09.444 DEBUG: CLIENT_APP: [main] Initializing and registering persistent storage
50:09.444 DEBUG: CLIENT_APP: [main] Initializing IoTivity stack for client_server
**********************************************************
***********Provisioning DB file already exists.***********
**********************************************************

1. Provision
2. Discover Resources and Send Requests
0. Exit

>> Enter Menu Number:
```

شکل ۲۲: فهرست اصلی کارخواه

گزینه اول provision ا ست که برای یافتن و ثبت د ستگاه‌های جدید به کار می‌رود. منظور از د ستگاه در اینجا همان برد رزبری‌پای است که کارساز برنامه را اجرا می‌کند. اگر در ابتدا دستگاه در حالت unowned باشد، با انتخاب گزینه اول طبق شکل ۲۳ این فهرست میتوان uuid دستگاه را پیدا کرد:

```
1. Provision
2. Discover Resources and Send Requests
0. Exit

>> Enter Menu Number: 1


1. Discover unowned devices
2. Discover owned devices
3. Register unowned devices
9. Initial Menu
0. Exit

>> Enter Menu Number: 1

   Discovering Only Unowned Devices on Network..
   > Discovered Unowned Devices
     [1] 12345678-1234-1234-1234-123456789012   fe80::3e97:460e:523c:d5a0%wlp2s0:44007   ocf.1.0.0
```

شکل ۲۳: فهرست Provision کارخواه

با انتخاب گزینه ۳ سازوکار انتقال مالکیت دستگاه انجام می‌گیرد. مطابق شکل ۲۴ در صورت موفقیت پیغام مناسب نمایش داده می‌شود. و دستگاه در حالت «آماده عملیات عادی» قرار می‌گیرد. این بار با انتخاب گزینه ۲ دستگاه در نتیجه این لیست نمایش داده خواهد شد.

```
1. Discover unowned devices
2. Discover owned devices
3. Register unowned devices
9. Initial Menu
0. Exit

>> Enter Menu Number: 3

   Registering All Discovered Unowned Devices..
   > Registered Discovered Unowned Devices
   > Please Discover Owned Devices for the Registered Result
57:13.839 INFO: CLIENT_APP: Ownership Transfer SUCCEEDED - ctx: Provision Manager Client Application Context
```

شکل ۲۴: دستگاه آماده عملیات عادی

حال می‌توانیم تمام منابع تعریف شده در شبکه را به کمک گزینه دوم فهرست اصلی بیابیم. نمونه‌ای از این منابع که همان منبع LED ما می‌باشد به صورت زیر در شکل ۲۵ نمایش داده می‌شود:

```
59:20.535 INFO: PayloadLog:     Link#12
59:20.535 INFO: PayloadLog:     URI:/switch
59:20.535 INFO: PayloadLog:     Anchor:ocf://12345678-1234-1234-1234-123456789012
59:20.535 INFO: PayloadLog:     Resource Types:
59:20.535 INFO: PayloadLog:             oic.r.switch.binary
59:20.535 INFO: PayloadLog:     Interfaces:
59:20.535 INFO: PayloadLog:             oic.if.baseline
59:20.535 INFO: PayloadLog:             oic.if.a
59:20.535 INFO: PayloadLog:     Bitmap: 3
59:20.535 INFO: PayloadLog:     Secure?: false
59:20.535 INFO: PayloadLog:     Port: 0
59:20.535 INFO: PayloadLog:     Endpoint #1
59:20.535 INFO: PayloadLog:             tps: coaps
59:20.535 INFO: PayloadLog:             addr: 192.168.1.105
59:20.535 INFO: PayloadLog:             port: 43833
59:20.535 INFO: PayloadLog:             pri: 1
59:20.535 INFO: PayloadLog:     Endpoint #2
59:20.535 INFO: PayloadLog:             tps: coaps
59:20.535 INFO: PayloadLog:             addr: fe80::3e97:460e:523c:d5a0%25wlp2s0
59:20.535 INFO: PayloadLog:             port: 60059
59:20.535 INFO: PayloadLog:             pri: 1
```

شکل ۲۵: منابع موجود در شبکه

ار سال درخوا ست منوط به د ست‌یابی به تمام منابع می‌با شد. پس هرگاه این د ستور با موفقیت انجام شد، فهرست درخواست‌ها نمایش داده خواهد شد. در این فهرست طبق شکل ۲۶ می‌توانیم ۴ دستور CREATE، RETRIEVE، UPDATE، DELETE را اعمال کرد.

```
1. Create (New LED)
2. Retrieve (LED resource)
3. Update (LED value)
4. Update (URI)
5. Delete (Second LED)

6. Observe CPU

9. Initial Menu
0. Exit

>> Enter Menu Number:
```

شکل ۲۶: عملیات CRUDN کارخواه

با انتخاب دستور CREATE یک نمونه جدید LED ساخته می‌شود و اطلاعات مربوط به آن نمایش داده می‌شود (شکل ۲۷):

```
17:28.421 INFO: CLIENT_APP: Callback Context for PUT recvd successfully
17:28.421 INFO: CLIENT_APP: StackResult: UNKNOWN
17:28.421 INFO: PayloadLog: Payload Type: Representation
17:28.421 INFO: PayloadLog:     Resource #1
17:28.421 INFO: PayloadLog:     URI:/switch
17:28.421 INFO: PayloadLog:     Values:
17:28.421 INFO: PayloadLog:             createduri(string):/switch/1
17:28.421 INFO: CLIENT_APP: ==============> Put Response
```

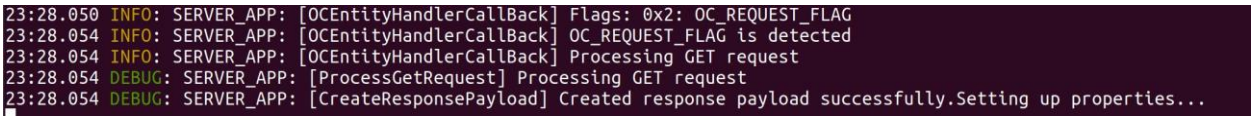
شکل ۲۷: نتیجه دستور CREATE

با انتخاب دستور RETRIEVE اطلاعات مربوط به منبع LED نمایش داده می‌شود (شکل ۲۸):

```
23:28.054 INFO: CLIENT_APP: Callback Context for GET query recvd successfully
23:28.054 INFO: CLIENT_APP: StackResult: OC_STACK_OK
23:28.054 INFO: CLIENT_APP: SEQUENCE NUMBER: 16777216
23:28.054 INFO: PayloadLog: Payload Type: Representation
23:28.054 INFO: PayloadLog:     Resource #1
23:28.054 INFO: PayloadLog:     URI:/switch
23:28.054 INFO: PayloadLog:     Resource Types:
23:28.054 INFO: PayloadLog:             oic.r.switch.binary
23:28.054 INFO: PayloadLog:     Interfaces:
23:28.054 INFO: PayloadLog:             oic.if.baseline
23:28.054 INFO: PayloadLog:             oic.if.a
23:28.054 INFO: PayloadLog:     Values:
23:28.054 INFO: PayloadLog:             value(bool):false
23:28.054 INFO: CLIENT_APP: ==============> Get Response
```

شکل ۲۸: نتیجه دستور RETRIEVE

هنگام تمام درخواست‌های CRUD، پیغام‌های مرحله به مرحله در سمت کارساز نیز ظاهر می‌شود (شکل ۲۹):



شکل ۲۹: نتیجه دستورات CRUDN در کارساز

۳-۴ جمع‌بندی

در این فصل، نتایج پیاده سازی پروژه و ابزار سخت‌افزاری و نرم‌افزاری استفاده شده، معرفی و نمایش داده شد. سخت‌افزار پروژه یک برد رزبری‌پای به همراه LED متصل به GPIO آن بود. نرم‌افزار پروژه کتابخانه‌ها وAPIهای IoTivity به همراه برخی کتابخانه‌های مورد نیاز جزئی دیگر بود که در نهایت منجر به تولید دو برنامه تحت خط فرمان لینوکس با زبان برنامه‌نویسی c شد که کارخواه و کارساز پروژه را تشکیل می‌دادند. نیازمندی‌های مطلوب پروژه از طریق این دو برنامه اجرا شد و نتیجه اجرا در این فصل نمایش داده شد.