МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

на тему

УВОД У IOT ИНЖЕЊЕРСТВО И ПРИМЕНА У КРЕИРАЊУ ЕДУКАТИВНИХ ИГРАЧКИ

Ученик:
Стефан Ђорђевић, IV_d

Ментор:
Јелена Хаџи-Пурић

Београд, мај 2017.

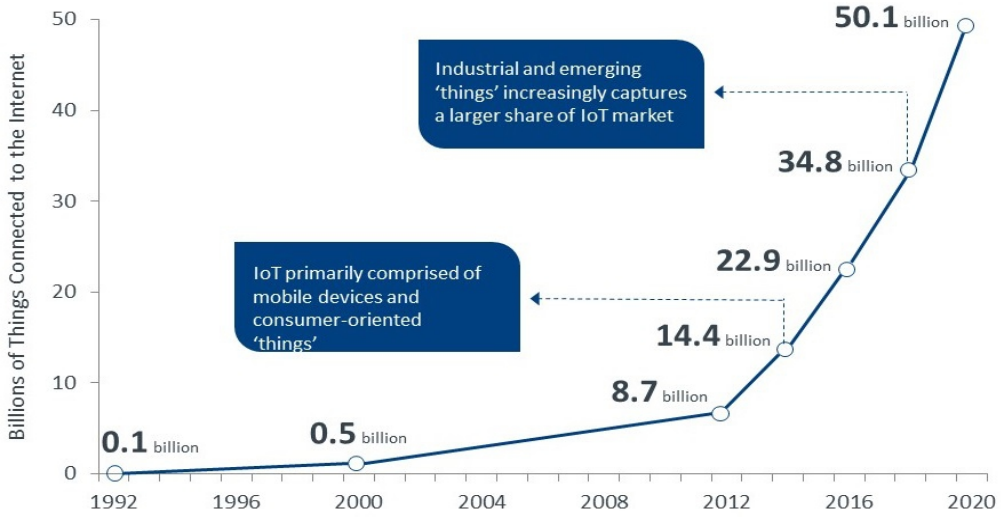
Садржај

Глава 1

Увод

Од појаве програмираних (embedded) уређаја седамдесетих година XX века па до данас, број примена активних производа уређаја непрестано расте. Појам embedded (уграђен) уређај се односи на минирачунаре дизајниране за тачно једну функцију, или неколико њих. Њихова главна употребна вредност лежи у масовној производњи, далеко јефтинијој од производње мултифункционалних рачунара, уштеди простора за складиштење и лаком преношењу. Мане су ограничена функционалност и немогућност репрограмирања, али се, као услужни елементи, могу наћи у свакој сфери индустрије, од терминала, преносивих геџета до контролних уређаја, као што је рутер, и нове класе embedded уређаја се непрестано појављују.

Приступачна цена и међународна сцена на којој се налазе разноразни примери и објашњења чине ову област популарном и омогућује стицање знања и почетничке вештине ка остварењу сопствених пројеката и побољшању постојећих решења у будућности. Циљ овог рада је опис развојног пута једног IoT пројекта и препрека, како би будуће генерације програмера могле ефикасније да прођу те прве кораке, и да се притом изворна заинтересованост и привлачност овој области сачува и негује.



Сл. 1: Експоненцијални раст уграђених уређаја од 1990. године и даља предвиђања [1]

Ипак, Internet of Things нису сви embedded уређаји, већ они који имају функционалност да комуницирају преко комуникационих модула (WiFi, Bluetooth, серијална веза) са другим компонентама и да у том процесу размењују информације и податке од кључног значаја за рад. Мрежа микроконтролера-џојстика и клијент-рачунара који сакупљају информације и размењују их је заснована на моделу Internet of Things.

У овом раду ће бити описана интеракција једног микроконтролера и рачунара, преко серијалне везе. Микроконтролер ће бити у саставу играчке-џојстика, а рачунар ће комуницирати преко управљачког програма драјвера. Током имплементације користио сам: јавно доступну литературу коју је објавио Microsoft, савете искусних програмера на форумима и јавно доступан репозиторијум андроид-џојстика DroidPad аутора Вила Шеклтона. Играчке-аутомобили као компјутерски мишеви одавно постоје на тржишту, а концепт играчке-џојстика треба да реши дететову жељу да седи за рачунаром како би играо видео игрице, и да, уместо тога, дете удаљи од екрана и функционално контролише збивања у игрици, без дуготрајног седења у столици и кривења кичме.

[2] Разумети како Интернет ствари технологија ће утицати на децу је тешко проценити. У полемикама, акценат дискусија је углавном сведен на негативан утицај технологије на децу, као што је социјална изолација. Међутим, људи углавном заборављају улогу родитеља, а он је најбитнији фактор који одређује како ће технологија утицати на дете.

Пошто се микроконтролер понаша као корисник у односу на главни ресурс – рачунар, широк обим упознавања са техником повезивања ће бити усмерен управо на рачунар, односно, драјвер, јер он треба правилно да препозна информације и проследи их оперативном систему као стање џојстика.

Драјвер је као управљачки програм један од најзначајнијих делова у систему, и његова главна функција јесте да производ прилагоди раду постојећих апликација. У ужем смислу, драјвер дефинише основне операције над производом, које се касније могу логички повезивати и надограђивати са другим деловима оперативног система. Будући да на нивоу драјвера раде уско специјализоване библиотеке и механизми (frameworks), секција везана за опис имплементације

драјвера је богато испуњена свим релевантним информацијама, јер и најмања грешка везана за програмирање Kernel Mode Driver Framework драјвера може довести до потпуног замрзавања стања оперативног система, и да се сатима не може отклонити. Због свега наведеног, посебна пажња је дата успостављању везе са стране драјвера, као посредника између играчке-џојстика и оперативног система рачунара.

Глава 2

Микропроцесор

У овој глави су описани основни концепти функционисања микроконтролера, кратка историја развоја до данашњих микроконтролера, припрема за рад и шема слања података кроз серијалну везу.

2.1 Микропроцесор и микроконтролер – архитектура и кратка историја

Микроконтролер [3] је дигитална електронска направа у облику интегрисаног кола. Намена микроконтролера је управљање уређајима и процесима, те у себи има интегрисан микропроцесор, меморију, дигиталне и аналогне улазе и излазе, дигиталне сатове („тајмере“), бројаче („каунтере“), осцилаторе, комуникационе склопове („интерфејсе“) и друге додатке за које је некада био потребан низ посебних интегралних кола („чипова“). Микроконтролер уобичајено ради у контролној петљи, дакле, очитава улазе и затим подешава излазе у складу са својим програмом. Петља се стално понавља док траје контрола процеса. Брзина микроконтролера се мери у посебним јединицама – Флопсовима (енг. FLOPS).

Главна разлика између модерних микропроцесора и микроконтролера је да су први оптимизовани за брзину и перформансе код рачунарских програма, док су микроконтролери оптимизовани у правцу интеграције већег броја кола, управљања процесима у стварном времену (real-time control), масовну производњу, ниску цену, и малу потрошњу струје. Микроконтролери су отпорнији и на варијације напона, температуре, влажности, вибрације и друге спољашње утицаје.

Са појавом првих микропроцесора 1971. године, почела је и њихова употреба у контролне сврхе. Међутим типичан систем је захтевао велики број додатних кола за рад, као што су били АД претварачи (енг. A/D converters), бројачи, осцилатори и друго. Временом је дошло до интегрисања потребних компоненти у једно коло, и тако је створен модерни микроконтролер.
Један од првих је био Моторола 6801 микроконтролер, развијен од 6800 микропроцесора.

Касније, 1985. године је од 6801 створен популарни 68HC11 са тад новом HCMOS технологијом, која је омогућила мању потрошњу, мању осетљивост на сметње и бржи рад. Данас многи модели могу да раде и врше контролу мањих уређаја без икаквих спољних делова, или са минималним бројем истих. Производња микроконтролера износи неколико милијарди годишње и знатно премашује производњу микропроцесора, типичних за личне компјутере (personal computers).

2.2 Припрема за рад

Конструкцију решења ћемо објаснити на примеру микроконтролера Ардуино (eng. Arduino).

Arduino [4] је физичко-рачунарска платформа (развојни систем) отвореног кода. Arduino плочу чине 8-битни Атмел AVR микроконтролер са припадајућим компонентама које омогућавају програмирање и повезивање са другом електроником. Битан аспект Arduino пројекта је стандардизован распоред конектора који омогућава лако повезивање са додатним модулима, познатијим као штитови. Ове додатне модуле, штитове, поизводе разни произвођачи широм света. Званичне Arduino плоче углавном користе megaAvr серију чипова, конкретно ATmega8, ATmega168, ATmega328, ATmega1280 и ATmega2560. Већина плоча поседује 5V линеарни напонски регулатор и 16MHz кристални осцилатор (или керамички резонатор у неким верзијама). Ардуино микроконтролери се испоручују са програмираним bootloader-ом који поједностављује поступак пребацивања преведеног кода у флеш меморију на чипу. Други микроконтролери обично захтевају засебан програматор.

Програмско окружење коришћено за програмирање Arduino микроплоче је „Arduino IDE”. Инсталација је стандардна и не захтева компликовано ангажовање корисника. Повезивање развојне плоче и рачунара је омогућено USB каблом, након чега је потребно унети тип Arduino развојне плоче и број порта на ком се налази на рачунару: Tools > Board: “Arduino/Genuino Uno” (конкретно име развојне плоче).

На развојној плочи се за потребе имплементације налазе повезане типке. Типка [5] (eng. Pushbutton) је врста прекидача електричних кола који прекида или спаја два или више проводника (умножак 2^k) само када је притиснут од стране корисника. Иначе се аутоматски, путем опруге или другог решења, враћа у стални-почетни положај.

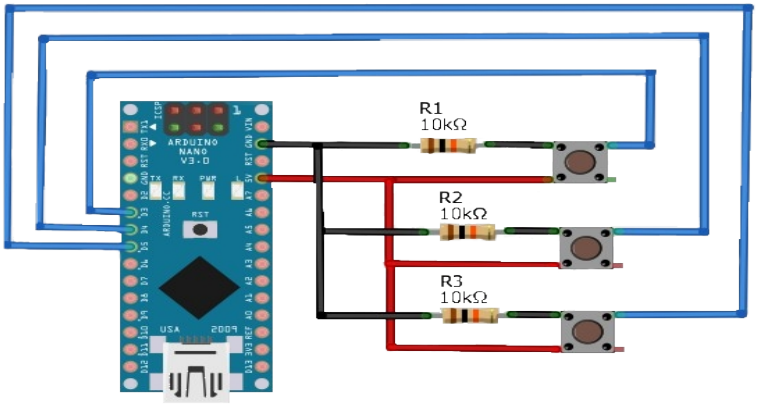
2.3 Шема слања података кроз серијалну везу

Ардуино плоча има atmega328p микроконтролер, и на њему се налазе 13 дигиталних pin-ова и 5 аналогних. Прва 2 pin-а се користе за серијски интерфејс (rs232) док ће се остали у имплементацији користити за тастере. На плочи се налази и FTDI (**F**uture **T**echnology **D**evices **I**nternational) чип, који серијску везу претвара у USB. Логички нивои су 5v.

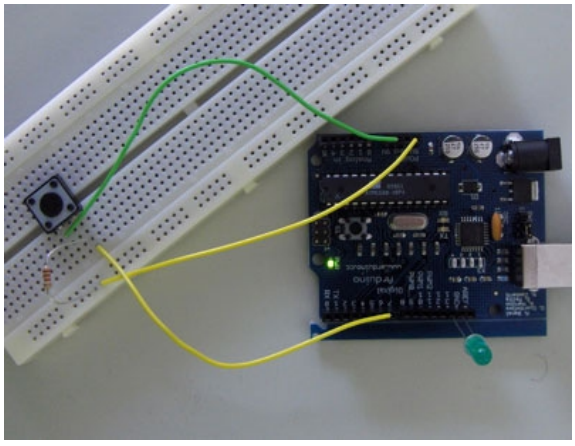
На шеми pushbutton-а, једна „ногица" (eng. Leg) је повезана на извор напона 5v [6], док је друга повезана на уземљење. Још једна жица је повезана на дигитални I/O pin.

У стању непритиснутог дугмета, лева и десна ногица нису повезане, и на pin-у се очитава вредност HIGH.

У стању притиснутог дугмета, лева и десна ногица су повезане, те је и pin повезан на уземљење, и на пину се очитава вредност LOW.



Сл 2: Шема повезивања компоненти



Сл 3: Повезивање pushbutton-а на Arduino развојну плочу

sketch_may25a §

```
int inPin = 7;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  Serial.begin(9600);       // set up Serial library at 9600 bps
  pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin);  // read input value
  if (val == HIGH) {         // check if the input is HIGH (button released)
    Serial.println(0);       // send input for released button
  } else {
    Serial.println(1);       // send input for closed button
  }
}
```

Сл 4: Скица читања стања на pin-у 7 и слање податка путем серијалне мреже [7]

На рачунару се очитавање битова [8] може обавити једноставном Node.js апликацијом. Arduino плоча је на рачунару обично препозната као ttyACM0 (на UNIX базираним оперативним системима).

```
var SerialPort = require("serialport").SerialPort
var serialPort = new SerialPort("/dev/ttyACM0", {
  baudrate: 9600
});
// it opens the connection and register an event 'data'
serialPort.on("open", function () {
  console.log('Communication is on!');

  // when your app receives data, this event is fired
  // so you can capture the data and do what you need
  serialPort.on('data', function(data) {
    console.log('data received: ' + data);
  });
});
```

Сл 5: Пример читања са серијалног порта и приказ података

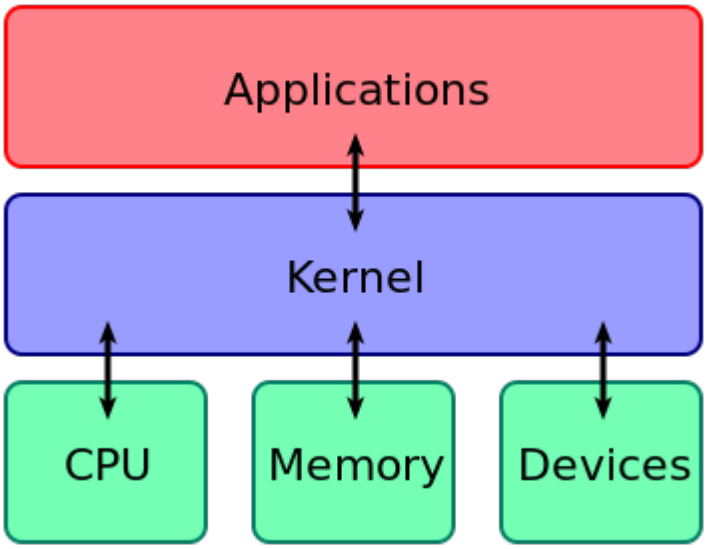
Глава 3

Драјвер

Главна грана имплементације решења је везана за пројектовање драјвера, управљачког програма који преводи надолазеће информације у облик доступан оперативном систему. У овој глави је описана архитектура модерних оперативних система, удео драјвера у њима, развој библиотека драјвера до данас, шема функционисања HID уређаја и опис најновијег шаблона за писање HID драјвера – Virtual HID Framework, као и опис припреме окружења за развој драјвера и читања података из корисничке апликације.

3.1 Архитектура модерних оперативних система

Са појавом модерних оперативних система и кернела [9] који посредује између корисничких апликација и компјутерских ресурса, створени су услови за додавање нових управљачких модула.



Сл. 6: Шема функционисања модерних оперативних система

Основна подела програмских процеса, делова корисничких апликација или управљачких јединица, је на нивоу приоритета који захтевају у раду рачунара. Ниво приоритета захтева рада

процеса се назива IRQL (**I**nterrupt **Req**uest **L**evel). IRQL означава да слојеви архитектуре најближи језгру оперативног система имају највећи приоритет извршавања, јер се даљи слојеви надовезују на њих, долазећи до корисничких апликација. Може се рећи да, као што човек мора да намири основне потребе (храна, пиће) пре него што може да се бави умним послом, који захтева стрпљење, а, пре свега тога, мора стално да дише, тим размишљањем и IRQL дефинише нивое приоритета, тако да се процеси највиших приоритета морају први извршити. Windows платформа дефинише распон IRQL од 0 до 31, где IRQL = 31 (HIGH_LEVEL) означава машинске провере и откривање катастрофалних (eng. fatal) грешки на хардверу. Највећи број управљачких програма се извршава на прва четири нивоа: PASSIVE_LEVEL, APC_LEVEL (1), DISPATCH_LEVEL (2) и DIRQL (3-26). Програми који раде на овим нивоима имају следећа значења/својства:

1. PASSIVE_LEVEL (0) – user mode (кориснички режим) и већина kernel mode операција се врши на овом нивоу. Организација меморије кроз странице (Pagable memory) је доступна.
2. APC_LEVEL (Asynchronous procedure calls, 1) – Ниво на коме се асинхрони позиви процедура и грешке у приступу страницама меморије (Page faults) дешавају. Драјвер може самостално да подеси IRQL тако да се асинхроне процедуре синхронизују са другим асинхроним процедурама, јер повећање нивоа са PASSIVE_LEVEL на APC_LEVEL онемогућава да се друге асинхроне процедуре извршавају. Неки апликациони програмски интерфејси (API) се не могу позвати на овом нивоу јер су позиви појединачним асинхроним процедурама онемогућени, што онемогућава рад неких улазно-излазних (IO) извршних асинхроних процедура (APC). Организација меморије кроз странице (Pagable memory) је и даље доступна.
3. DISPATCH_LEVEL (2) – на овом нивоу раде организатори нити (Thread Schedulers) и одложени позиви процедура (Deferred Procedure Calls). Организација меморије кроз странице (Pagable memory) није више доступна. Број апликативних програмских интерфејса је знатно умањен због немогућности приступа меморији кроз странице.
4. DIRQL (3-26) – прекиди у раду које генеришу уређаји.

3.2 Технологија драјвера и основне структуре

Драјвери [10] се, према разлици између коришћења апликативних програмских интерфејса само на PASSIVE_LEVEL и преласку на више IRQL деле на User Mode и Kernel Mode драјвере. Шаблони по којима се пишу драјвери су User Mode Driver Framework (у даљем тексту UMDF) Kernel Mode Driver Framework (у даљем тексту KMDF). Иако KMDF драјвери могу имати јако сужен избор библиотека, они могу да директно приступају меморији [11] и врше измене над њом, као и да инструментализују рад виших апликативних програмских интерфејса. KMDF подржава и наслеђује застарели Windows Driver Model (у даљем тексту WDM), објектно је оријентисан и пружа објектно-оријентисану перспективу WDM.

UMDF и KMDF су део Windows Driver Foundation (у даљем тексту WDF), чију употребу Microsoft заговара од раних двехиљадитих па надаље. Драјвер у овој имплементацији је програмиран на основу KMDF.

Почетна метода након иницијализације драјвера је DriverEntry метода, која се укључује на PASSIVE_LEVEL, и у којој се подешавају основне глобалне променљиве и атрибути драјвера.

Основна јединица драјвера је његова инстанца уређај (Device). Уређаји се према присуству хардвера деле на Physical Device Objects (у даљем тексту PDO) и Functional Device Objects (у даљем тексту FDO). Улога FDO се огледа у превођењу информација од стране PDO у облик карактеристичан уређајима другог драјвера. Уређаји су језгро функционисања драјвера јер је њихова смисао у чекању на одговарајући догађај и давању одговора на њега.

Сви захтеви над уређајима се уносе у структуре редове. Ред (у даљем тексту Queue) је структура која описује основне догађаје уређаја када доспу на ред:

- EVT_WDF_IO_QUEUE_IO_CANCELED_ON_QUEUE
- EVT_WDF_IO_QUEUE_IO_DEFAULT
- EVT_WDF_IO_QUEUE_IO_DEVICE_CONTROL
- EVT_WDF_IO_QUEUE_IO_INTERNAL_DEVICE_CONTROL
- EVT_WDF_IO_QUEUE_IO_READ
- EVT_WDF_IO_QUEUE_IO_RESUME

- EVT_WDF_IO_QUEUE_IO_STOP
- EVT_WDF_IO_QUEUE_IO_WRITE

IoDeviceControl и IoInternalDeviceControl су од посебног значаја, јер се у информацијама о тим догађајима (параметрима) преноси и IoControlCode – информација која објашњава какав уређај треба дати одговор на пристиглу поруку из спољашњости.

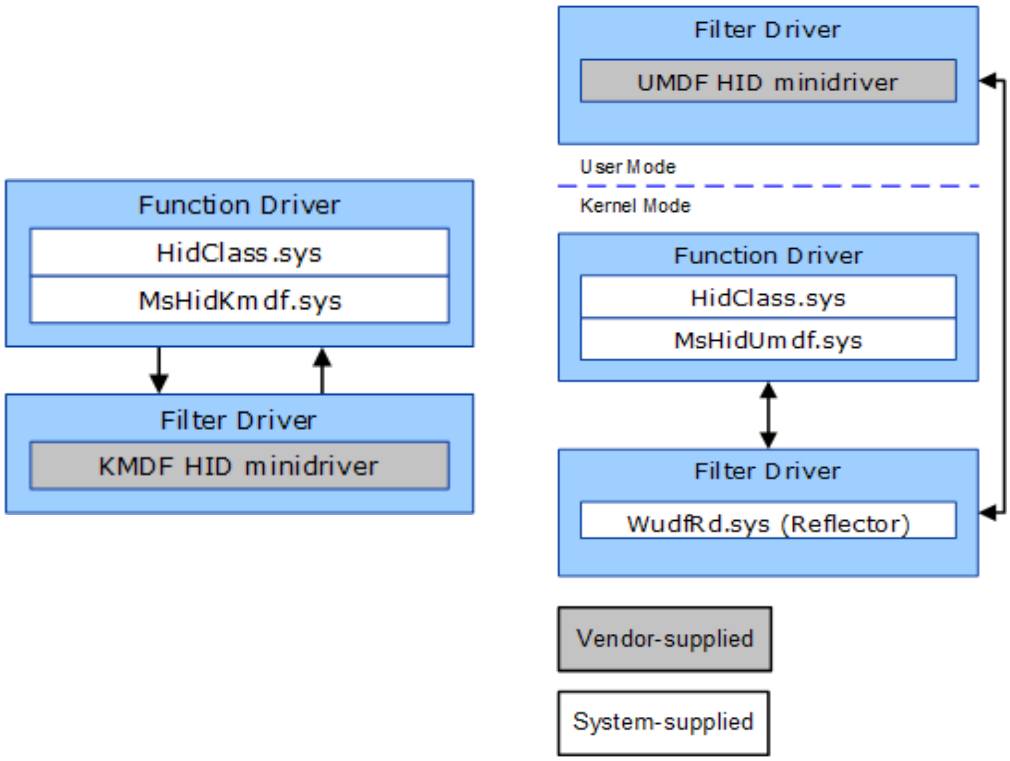
Ове структуре су део наследница WDM dispatch рутина, које процесуирају надолазеће I/O request пакете (IRP):

- IRP_MJ_CLEANUP
- IRP_MJ_CLOSE
- IRP_MJ_CREATE
- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_FILE_SYSTEM_CONTROL
- IRP_MJ_INTERNAL_DEVICE_CONTROL
- IRP_MJ_PNP
- IRP_MJ_POWER
- IRP_MJ_QUERY_INFORMATION
- IRP_MJ_READ
- IRP_MJ_SET_INFORMATION
- IRP_MJ_SHUTDOWN
- IRP_MJ_SYSTEM_CONTROL
- IRP_MJ_WRITE

3.3 HID уређаји и проток података

Џојстици, тастатуре и мишеви су део класе уређаја које се заједнички зову Human Input Devices (у даљем тексту HID). Њихово присуство се у систему препознаје од стране драјвера HidClass, који на основу шеме њиховог контекста одређује тип и све улазне компоненте уређаја. Према томе, KMDF драјвер за ручно прављени џојстик, као што је аутомобил, ће се регистровати као драјвер чији уређаји комуницирају са уређајима HidClass и преводе доспели улаз у облик разумљив њему. Драјвер чији уређаји препознају и преводе сигнале за потребе уређаја других драјвера назива се filter driver, а шема контекста HID уређаја HID Report Descriptor. У случају filter драјвера за HID, драјвер се још зове и HID-minidriver.

За превођење у облик карактеристичан HidClass уређајима, инстанцира се FDO, а уређај који ће примати са улаза информације о стању џојстика ће бити PDO. Међутим, развојни драјвер и HidClass не комуницирају директно, већ постоји драјвер/и који од одговарајућег типа развојног драјвера преводи информације HidClass драјверу. Ти драјвери се називају pass through драјвери.



Сл. 7 и сл. 8: Шеме комуникације HID драјвера за типове KMDF и UMDF, респективно [12]

Pass through драјвери решавају проблеме конфликта WDM Dispatch рутина између HidClass и framework драјвера за извршавање улазно-излазних захтева као што су Plug and Play (PnP) и power management захтеви. Као што је речено, framework драјвери су надодати на врх WDM структуре, те све структуре у старијим моделима постоје и у framework драјверима, и извршавају се у позадини.

HID Descriptor [13] је структура која описује ког је типа HID уређај (џојстик у примеру имплементације пројекта), за коју платформу је намењен, које су улазне компоненте и које вредности могу да заузимају. Ови подаци су нарочито значајни да би се интерпретирао садржај улазне buffer меморије:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Useless	Useless	Useless	Useless	Useless	Left Button	Middle Button	Right Button
Byte 1	X Axis Relative Movement as Signed Integer							
Byte 2	Y Axis Relative Movement as Signed Integer							

сл. 9 - илустрована шема HID дескриптора

Да би се омогућила комбинација више регистрованих улаза, у сваком тренутку HID уређај шаље шаблон вредности свих својих улазних компонената. На тај начин шаље у пакетима од 1 бајта битове који, рецимо, за дугмиће логички би требало да буду само 0 или 1, док нпр. за вектор којим се помера миш треба да се прошири спектар вредности, те је за довољну прецизност потребно узети више од 2 вредности.

Разлог зашто у примеру стоје пет неискоришћених битова је чиста техничка природа – ови битови се тренутно не користе, а структуре у којима се чувају подаци су облика 8, 16, 32 или 64 бита ((unsigned) char, (unsigned) integer).

```
typedef struct _HIDINJECTOR_INPUT_REPORT {
    unsigned char ReportId;
    union {
        struct {
            UCHAR Buttons;
            SHORT X;
            SHORT Y;
        } JoystickReport;
    } Report;
} HIDINJECTOR_INPUT_REPORT, *PHIDINJECTOR_INPUT_REPORT;
```

сл. 10: Шема како изгледа улаз у коду

Уместо да ручно програмер уноси бит по бит, постојеће библиотеке имплицитно извршавају тај посао, без великог трошка (ако се број frame-ова per second (ажурирања екрана у игрици у секунди) апроксимира на 60, 60 * 5 некоришћених битова неприметно оптерећује перформанс рачунара. Ради илустровања колико је то frame-ова, људско око детектује ~35 frame-ова per second).

3.3 Virtual HID Framework

Virtual HID Framework [14] (у даљем тексту VHF) је најновији метод који има за циљ да пружи једноставнији интерфејс програмирања за пројектанта HID уређаја. Овај шаблон програмирања је тренутно доступан за рад на KMDF драјверима и на платформи Windows 10.

Као што смо рекли, за комуникацију са HidClass драјвером је било потребно пројектовати и pass-through драјвер, и целокупан посао израде више пројеката није нимало једноставан за почетника у изради драјвера. Са тим на уму, Windows је објавио VHF који има интегрисан pass-through драјвер vhf.sys који комуницира са HidClass драјвером.

Основна подешавања се конфигуришу помоћу структуре VhfConfig, која је параметар у креирању VhfHandle, показивача на везу са vhf.sys драјвером. У овој структури је потребно навести HID дескриптор и опционалне асинхроне структуре, као што су:

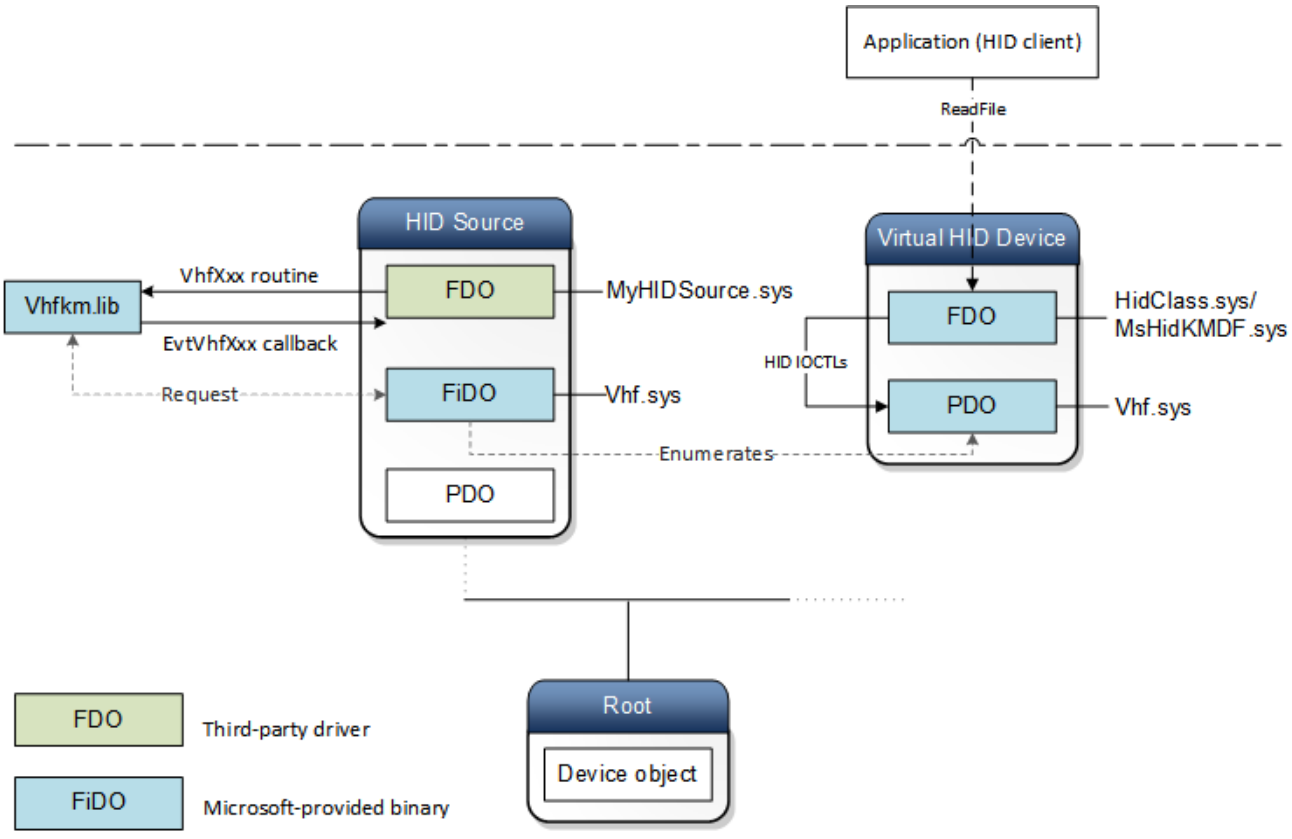
- EvtVhfReadyForNextReadReport
- EvtVhfAsyncOperation

EvtVhfReadyForNextReadReport оптимизује слање стања HID уређаја не дозвољавајући

позивање методе VhfSubmitReadReport док се претходни позив не заврши, тиме искључујући могућност дуплицирања позива у једном истом тренутку и њиховог преплетања.

```
deviceContext = DeviceGetContext(device);

VHF_CONFIG_INIT(&vhfConfig,
    WdfDeviceWdmGetDeviceObject(device),
    sizeof(VhfHeadSetReportDescriptor),
    VhfHeadSetReportDescriptor);

status = VhfCreate(&vhfConfig, &deviceContext->VhfHandle);

if (!NT_SUCCESS(status)) {
    TraceEvents(TRACE_LEVEL_ERROR, TRACE_DEVICE, "VhfCreate failed %!STATUS!", status);
    goto Error;
}

status = VhfStart(deviceContext->VhfHandle);
if (!NT_SUCCESS(status)) {
    TraceEvents(TRACE_LEVEL_ERROR, TRACE_DEVICE, "VhfStart failed %!STATUS!", status);
    goto Error;
}
```

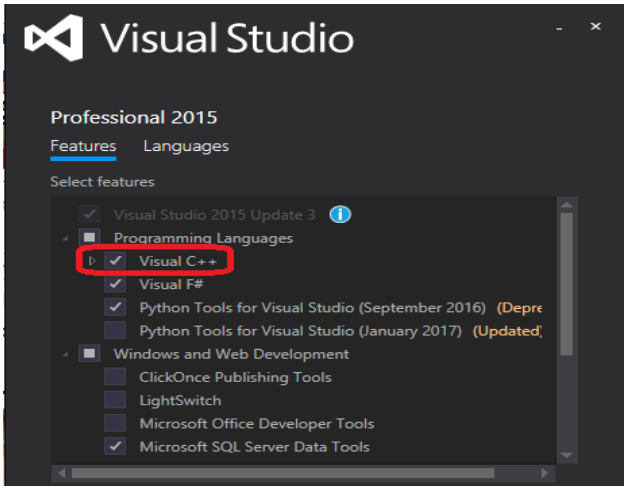
Сл. 11: Пример кода за креирање везе са vhf.sys



Сл. 12: Шема функционисања HID драјвера са VHF

3.4 Припрема развојног окружења драјвера

Програмско окружење коришћено за писање драјвера је „Visual Studio 2015 Professional” (у даљем тексту VS). При инсталацији потребно је одабрати необавезну (eng. optional) инсталацију за рад са програмским језиком C++.



сл. 13: Дијалог одабира инсталационих пакета

Након инсталације VS, потребно је инсталирати и додатне две компоненте: најновији Software Development Kit 10 (у даљем тексту SDK 10) и Windows Driver Kit 10 (у даљем тексту WDK 10).

Аутоматско тестирање развојних драјвера је одавно онемогућено на матичном рачунару. Наведена су два разлога:

1) Могуће изненадно замрзавање екрана (и стања оперативног система) услед неправилног програмирања драјвера.
2) Шира и неометана могућност праћења понашања драјвера (и оперативног система) на тест-рачунару.

Због наведених разлога, потребно је конфигурисати тест-рачунар за приступ од стране другог рачунара и извршавање драјвера у развоју. Пожељно је да тест-рачунар има активан Windows 10 оперативни систем, или барем Windows 8, јер се развој драјвера битно разликовао на ранијим платформама, те је Windows 10, као најновији, најопремљенији за брзу и ефикасну припрему тест-рачунара за развој програма. Аутори овог пројекта су спровели тестирање на 64-битној платформи Windows 10.

На тест-рачунару [15] је потребно инсталирати WDK 10 (није потребно инсталирати SDK 10 и VS) и у директоријуму инсталираног програма покренути WDK Test Target Setup MSI који одговара платформи на којој је укључен тест-рачунар. На пример, ако је рачунар 64-битни, онда ће путања датотеке изгледати овако:
C:\Program Files (x86)\Windows Kits\10\Remote\x64\WDK Test Target Setup x64-x64_en-us.msi

У заштитном зиду тест-рачунара (Windows Firewall) је потребно дозволити да се рачунар са ког се развија драјвер (у даљем тексту Host рачунар) повеже са њим и шаље податке. Да би омогућили пренос и повезивање, треба отворити Windows Firewall > Advanced Settings > Inbound Rules. Сва правила група Network Discovery и File and Printer Sharing, која се односе на тип активне мреже на којој се врши тест драјвера (Public, Private или Domain), треба дупло кликнути и у Properties > Scope поставити вредност Any IP Address за Remote IP Address.

Последњи корак припреме за извршавање драјвера се назива снабдевање (Provisioning) тест-рачунара. На тест-рачунару (или target computer) се инсталира посебан корисник (WDKRemoteUser) и подешава се debugger. За потребе ове имплементације, довољно је изабрати подразумеване вредности. У инстанци VS, Driver > Test > Configure Devices... отвара се нови прозор у ком су приказани постојећи тест-рачунари. Притиском на дугме Add New Device отвара се мени где се уносе редом име рачунара, IP адреса и тип снабдевања (Provision device and choose debugger settings или Manually configure debuggers and do not provision). Избором прве опције снабдевања, у следећем кораку setup-а бира се тип повезаности (Network, Serial, USB, Firewire) и IP Address host рачунара. Bus параметри нису потребни да се унесу за овај пример.

Након креирања новог пројекта (VS > File > New > Project > Kernel Mode Driver), потребно је конфигурисати га за аутоматско тестирање (deploy) на тест-рачунару. У Project Properties, потребно је поставити следеће вредности:

- Driver Settings > Driver Model вредности KMDF Version Major = 1 и KMDF Version Minor = 19 (у новијим верзијама ће се ови бројеви инкрементирати).
- Driver Install > Deployment вредност Target Device Name = претходно унесено име тест-рачунара, штиклирати Remove previous driver versions before deployment и изабрати Install/Reinstall and Verify

– Configuration Manager вредност Platform за именовани пројекат поставити на x32 уколико је OS тест-рачунара 32-битни или x64 за 64-битни оперативни систем.

Једнако важна функционалност, необавезна, али значајна, јесте укључивање појављивања дефинисаних порука [16] при тестирању драјвера, због праћења његових стања. Пошто KMDF драјвери нису user mode апликације, једини начин јесте исписивање порука у прозору VS званом Debugger Immediate Window. На тест-рачунару се такође могу пратити поруке из кернел програма коришћењем помоћног програма DebugView (потребна су администраторска права за праћење порука кернел нивоа и додатно у инстанци програма штиклирати Capture > Capture Kernel).

Међутим, приказ порука овог типа нису од Windows Vista-е више подразумеване, и да би их видели, потребно је отворити системски програм RegEdit и пратити путању HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Debug Print Filter (уколико не постоји, креирати) и додати DWORD вредност DEFAULT (не мењати постојећу вредност (Default), већ креирати нову) 0xff (ова вредност је минимално потребна за приказ стандардних порука са тест-рачунара).

Описаним процесима су припреме за почетак израде задатка завршене.

3.4 Комуникација уређаја драјвера са корисничким апликацијама

Основне библиотеке потребне за израду KMDF драјвера су ntddk.h и wdf.h. За техничку природу овог пројекта коришћене су још hidport.h, vhf.h и initguid.h.

За генерисање HID Descriptor може се користити HID Descriptor Tool [17]. Аутомобил-џојстик ће имати ***x*** и ***y*** компоненту и три дугмета:

```
CONST  HID_REPORT_DESCRIPTOR       G_DefaultReportDescriptor[] = {
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
    0x09, 0x04,                    // USAGE (Joystick)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x09, 0x01,                    //   USAGE (Pointer)
    0xa1, 0x00,                    //   COLLECTION (Physical)
    0x85, 0x01,                    //     REPORT_ID(1)
    0x05, 0x09,                    //     USAGE_PAGE (Button)
    0x19, 0x01,                    //     USAGE_MINIMUM (Button 1)
    0x29, 0x03,                    //     USAGE_MAXIMUM (Button 3)
    0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //     LOGICAL_MAXIMUM (1)
    0x95, 0x03,                    //     REPORT_COUNT (3)
    0x75, 0x01,                    //     REPORT_SIZE (1)
    0x81, 0x02,                    //     INPUT (Data,Var,Abs)
    0x95, 0x01,                    //     REPORT_COUNT (1)
    0x75, 0x05,                    //     REPORT_SIZE (5)
    0x81, 0x03,                    //     INPUT (Cnst,Var,Abs)
    0x05, 0x01,                    //     USAGE_PAGE (Generic Desktop)
    0x09, 0x30,                    //     USAGE (X)
    0x09, 0x31,                    //     USAGE (Y)
    0x15, 0x81,                    //     LOGICAL_MINIMUM (-127)
    0x25, 0x7f,                    //     LOGICAL_MAXIMUM (127)
    0x75, 0x08,                    //     REPORT_SIZE (8)
    0x95, 0x02,                    //     REPORT_COUNT (2)
    0x81, 0x06,                    //     INPUT (Data,Var,Rel)
    0xc0,                          //     END_COLLECTION
    0xc0                           // END_COLLECTION
};
```

Сл. 14: HID дескриптор

PDO мора да региструје интерфејс за комуникацију са user mode апликацијама. Тај интерфејс се из корисничке апликације посматра као File, и креиран се може наћи на путањи \\Device\\name. Да би драјвер могао да ради и на старијим верзијама, потребно је додати и линк, односно, креирати интерфејс \\DosDevices\\link:

```
if (NT_SUCCESS(WdfDeviceInitAssignName(mRawPDOInit, &mRawPDOName)))
    DbgPrint("WdfDeviceInitAssignName success\n");
WDF_OBJECT_ATTRIBUTES_INIT_CONTEXT_TYPE(&mRawPDOControlAttributes, CONTROL_DEVICE_EXTENSION);
if (NT_SUCCESS(WdfDeviceCreate(&mRawPDOInit, &mRawPDOControlAttributes, &mRawPDODevice)))
    DbgPrint("WdfDeviceCreate success\n");
if (NT_SUCCESS(WdfDeviceCreateSymbolicLink(mRawPDODevice, &mRawPDOLink)))
    DbgPrint("WdfDeviceCreateSymbolicLink success\n");

WDF_IO_QUEUE_CONFIG_INIT_DEFAULT_QUEUE(&mIOQueueConfig, WdfIoQueueDispatchSequential);
mIOQueueConfig.EvtIoWrite = EvtRawPDOIoWrite;
if (NT_SUCCESS(WdfIoQueueCreate(mRawPDODevice, &mIOQueueConfig, WDF_NO_OBJECT_ATTRIBUTES, &mIOQueue)))
    DbgPrint("WdfIOQueueCreate success\n");
else DbgPrint("WdfIOQueueCreate unsuccessful\n");
```

Сл. 15: Издвојен код из методе инстанцирања и подешавања PDO

```c
int main()
{
    HANDLE hDevice;
    DWORD nb;
    printf("IoTController\n");
    hDevice = CreateFile(TEXT("\\\\.\\IoTController"),
        GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
    if (hDevice == INVALID_HANDLE_VALUE) {
        int d = GetLastError();
    printf("%d\n", d);
    }
    HIDINJECTOR_INPUT_REPORT mState = { 0 };
    mState.ReportId = 1;
    mState.Report.JoystickReport.AbsoluteX = 1000;
    mState.Report.JoystickReport.AbsoluteY = -1000;
    mState.Report.JoystickReport.Buttons &= ~0x01;
    DWORD written = 0;
    if (!WriteFile(hDevice, &mState, sizeof(mState), &written, NULL)) {
        int d = GetLastError();
        printf("%d\n", d);
    }
    CloseHandle(hDevice);
    system("PAUSE");
    return 0;
}
```

Сл. 16: Скица комуникације са уређајем драјвера из корисничке апликације [18]

Глава 4

Дискусија истраживања и закључак

Протокол комуникације између микроконтролера и драјвера је врло јасно дефинисан и показује да се на исти начин могу и други уређаји интерпретирати као HID, нпр. лаптоп се може интерпретирати као тастатура или телефон као џојстик.

Алтернатива KMDF драјверу у овој имплементацији је UMDF, који би због стандардне библиотеке (STL) представљао много лакше решење, а појаве као Blue Screen of Death (BSOD) би биле много ређе. Најважнији разлог зашто је алтернатива боља је што је једина функција драјвера филтер, а ту функцију може и пожељно је да ради управо драјвер типа UMDF. Међутим, коментатори на интернету скоро увек препоручују KMDF као стартну тачку израде [19].

Иако активна и зрела грана програмирања, на интернету се тешко налазе туторијали за одређене типове драјвера за Windows, изузев Мајкрософтових јавно доступних примера. Информације које MSDN нуди концизно објашњавају своје поље, али су са становишта почетника без трунке искуства у програмирању драјвера јако конфузне, и неретко прескачу битне делове кода за процес који описују. Кроз објављивање VHF, назире се жеља Мајкрософта у популаризацији писања драјвера за њихове оперативне системе, али без појединачних корака и могућих проблема у изради описане шеме, велики део времена се троши на решавање тих проблема кроз претрагу интернета („гуглање”) или преглед великих пројеката, примера израђених од стране Мајкрософта, који у себи неретко носе велики број процедура везаних за израду шеме које нису објашњене. Велики је недостатак што не постоје снимљена упутства, и што нпр. Мајкрософтов код за израду VHF није референциран на сајту MSDN код артикла о писању драјвера који раде на принципу VHF, већ се може (само срећом) наћи „гуглањем”.

Кроз анализу примера који је овде дефинисан, једно од могућих решења је било да се направи Windows Service, који би требало да се покреће у позадини и да обавља посао који драјвер обавља, али је убрзо одбачен због недостатка материјала са којим треба почети, и јер је аутор текста желео да се упути у развој програмских модула изван корисничког интерфејса. Иако

неистражен, на основу форума на интернету, Windows Service доста личи на драјвер, изузев тога што не може да мења приоритет захтева. За тржишне производе је бољи драјвер, јер гарантује функционисање које одговара IRQL.

Кроз развој џојстика и сусретање са програмским изазовима је садржано знање и искуство које представља врата ка даљем бављењу комуникацијом између уређаја, од микроконтролера повезаних Bluetooth, Wi-Fi или кабловском везом, до микропроцесора повезаних PCI и другим далеко примитивнијим методима кроз које се постављају темељи за развој будућих апликативних програмских интерфејса.

Гране програмирања драјвера и микропроцесора су јако значајне у свим индустријским гранама које производе рачунарске компоненте. Тајне кода су јако чуване, као што је то донедавно било са HyperThreading и Virtuelisation технологијама које је Интел чувао и којима је осигуравао монопол на тржишту, после чега су на суду изгубили од АМДа за потплаћивање одржавања монопола и уз новчану одштету дали и све патенте својих чувених x86 процесора!

Основе IoT инжењеринга и комплетно описан процес израде примера постоје као јавно доступан документ на три блога, где се још увек претежно користи застарели WDM, а на српском језику уопште не постоје. У наредним издањима би овај истраживачки рад могао да се бави и вишим кернелским процедурама, уз прелазак на асемблеру ближим микропроцесорима.

Интернет ствари пружају велике могућности за когнитивни развој деце. Упоредо са развојем технологије се развијају и производи који омогућавају деци да уче и схватају. Деца већ у првим разредима основне школе могу програмирати на врло повољним дигиталним платформама као што је Raspberry Pi, играјући се са разним жицама и батеријама: „Kids don’t just create rocket ships; they can make apps that drive rocket ships” (Hannah Augur). Интернет је богат примерима које деца могу да пробају, а процес тражења решења и конструисања се показао да доводи до великог повећања способности детета да прими и схвати информацију. Осим што уче, деца уживају у раду који их окупира!

Мој будући рад биће усмерен ка изради концепта кућног бицикла као џојстика јер овим начином, осим избегавања седења за рачунарем, корисник стимулисано користи покретачке мишиће целог тела. Истраживања су показала да је довољно седење четири сата дневно у укоченом положају да доведе до повећања шанси за развој више болести, од којих су најзначајније дијабетес и кардиоваскуларне болести.

Захвалност за учешће у раду упућујем:

- **Вилу Шеклтону** (eng. Will Shackleton, University of Cambridge, Computer Sciences, Facebook)
  - Инспираија за израду џојстика – DroidPad
  - Open-source sample кода DroidPad – GitHub
  - Подршка приликом анализе проблема и функционисања драјвера
- Осталим повременим учесницима анализе и обраде решења:
  - **Михајло Милосављевић** и **Милан Сурла** (Teleskin)
  - **Каталин Ђорђу** (rom. Catalin Gheorghiu, CoderDojo Timisoara, Microsoft)
  - Ауторима онлајн туторијала о изради драјвера на сајтовима CodeProject, blogs.microsoft, репозиторијума ms-iot/samples (GitHub) и другим анонимним коментаторима са форума OSR Online
  - Професорима Математичке гимназије за изузетан допринос у саветовању поводом бирања ове теме и обезбеђивању лиценце Visual Studio 2015, без којег дубља анализа теме овог рада не би била могућа

ЛИТЕРАТУРА

[1.] Раст броја Интернет ствари

https://www.forbes.com/sites/gilpress/2016/09/02/internet-of-things-by-the-numbers-what-new-surveys-found/#153376c016a0

Последњи приступ сајту: 25.05.2017.

[2.] Утицај IoT на децу

http://dataconomy.com/2016/02/the-iot-for-kids-how-technology-affects-our-children/

Последњи приступ сајту: 25.05.2017.

[3.] Микроконтролер

https://sr.wikipedia.org/sr-el/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80

Последњи приступ сајту: 25.05.2017.

[4.] О Arduinu:

https://sr.wikipedia.org/wiki/Arduino

Последњи приступ сајту: 25.05.2017.

[5.] О pushbutton-има

https://sr.wikipedia.org/wiki/%D0%A2%D0%B0%D1%81%D1%82%D0%B5%D1%80

Последњи приступ сајту: 25.05.2017.

[6.] Читање сигнала са pushbutton-а

https://www.arduino.cc/en/tutorial/pushbutton

Последњи приступ сајту: 25.05.2017.

[7.] Слање података преко серијалне везе

http://www.ladyada.net/learn/arduino/lesson4.html

Последњи приступ сајту: 25.05.2017.

[8.] Читање сигнала са Arduino плоче

https://arduino.stackexchange.com/questions/11637/how-to-transfer-data-from-arduino-to-some-software-in-computer

Последњи приступ сајту: 25.05.2017.

[9.] Архитектура кернела

https://en.wikipedia.org/wiki/Kernel_(operating_system)

Последњи приступ сајту: 25.05.2017.

[10.] Увод у израду драјвера

https://www.codeproject.com/Articles/9504/Driver-Development-Part-Introduction-to-Drivers

Последњи приступ сајту: 25.05.2017.

[11.] Разлика између User и Kernel mode програмирања

https://blog.codinghorror.com/understanding-user-and-kernel-mode/

Последњи приступ сајту: 25.05.2017.

[12.] Креирање HID minidriver-a

https://msdn.microsoft.com/en-us/library/windows/hardware/ff540774(v=vs.85).aspx

Последњи приступ сајту: 25.05.2017.

[13.] О HID Descriptor-има

http://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/

Последњи приступ сајту: 25.05.2017.

[14.] Писање драјвера који функционишу на принципу VHF

https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/virtual-hid-framework--vhf-

Последњи приступ сајту: 25.05.2017.

[15.] Припрема тест рачунара

https://msdn.microsoft.com/en-us/windows/hardware/drivers/gettingstarted/provision-a-target-computer-wdk-8-1

Последњи приступ сајту: 25.05.2017.

[16.] Укључивање DbgPrint у тестирању

http://www.osronline.com/article.cfm?article=295

Последњи приступ сајту: 25.05.2017.

[17.] HID Descriptor Tool

http://www.usb.org/developers/hidpage#HID%20Descriptor%20Tool

Последњи приступ сајту: 25.05.2017.

[18.] Комуникација корисничке апликације са уређајем драјвера

https://msdn.microsoft.com/en-us/library/windows/hardware/dn613924(v=vs.85).aspx

Последњи приступ сајту: 25.05.2017.

[19.] Custom joystick driver компарација имплементација

http://osronline.com/showThread.CFM?link=205844

Последњи приступ сајту: 25.05.2017.