

Distributed Database and Tasks System

מגיש: איתי בן שלום

ת.ז. 209479161

אורט ע"ש נעמי שמר גן יבנה

מוגש ל: אנטולי פיימר

אפריל 2018

תוכן עניינים:

רקע 3

מבוא 4

תיאור המוצר 5

אופן פעולת המערכת 8

פעולת המערכת באופן כללי 8

חלוקה למודולים 10

Communication 13

Database 23

Decider 24

DriverCommunicator 27

WDM-DKOM 29

ControllerConnection 31

בכדי להפעיל את המודול הזה ה- Manager מריץ את הפקודה: 31

Manager 32

Controller 39

להוסיף קודים לאחר בדיקה סופית 40

DirectoryServer 40

דוגמאות הרצה 42

הצעות לשיפור 43

סיכום 43

# רקע

סביבת פיתוח:

Microsoft Visual Studio 2015 community

שפות תכנות:

Node: C, C++, C#, Python 2.6/2.7

Directory Server: Python 2.6/2.7

Controller: Python 2.6/2.7

דרישות נוספות:

Python packages:

psutil – package for retrieving information on processes and system.

netifaces – network interfaces data extraction package

scapy – packets manipulation package

הערה: פונקציות מסויימות אינן נתמכות במחשבים עם מערכת הפעלה של 64 סיביות (הפונקצייה של DKOM), הסבר נוסף בהמשך.

מבוא

היום עם התקדמות הטכנולוגיה יש דרישה בתחומים רבים לביצוע חישובים שצורכים זמן ומשאבים רבים במיוחד בארגונים טכנולוגיים גדולים. לשם כך ניתן לרכוש מחשבים עם יכולות חישוב גבוהות או שרתים עם זיכרון רב.

המערכת הזו מספקת פתרון לבעיה הזו ללא רכישת מחשבי על ושרתים בעלויות גבוהות אלא על ידי רתימת כוח חישוב וזיכרון לא מנוצל של מחשבים אישיים בחברה (כל מחשב שמשתמשים בו בחברה – מחשבים של מזכירים/ות, מתכנתים/ות, מעצבים/ות וכו').

קהל היעד המרכזי של המערכת הוא תאגידים טכנולוגיים גדולים שצריכים יכולות חישוב גבוהות ו/או זיכרון רב ויש ברשותם כמות נכבדת של מחשבים אישיים.

# תיאור המוצר

שם המוצר:

DDTS – Distributed Database and Tasks System

חלקים:

Node:

זהו החלק שמותקן בכל מחשב שמהווה חלק מהמערכת והוא משמש למספר דברים:

* אחראי על ביצוע המשימות שנשלחות אל המחשב ושליחת התוצאות של הביצוע בחזרה
* אחראי על ביצוע פעולות במסד הנתונים שבמחשב ושליחת התוצאות שמחזירות הפעולות בחזרה.
* אפשור חיבור של Controller ל-Node ובעצם זאת אפשור של שליחת משימות אל Node-ים אחרים וקבלת תוצאות בחזרה.

Directory server:

זהו החלק שנמצא בשרת אחר (לא על מחשב שמותקן עליו Node) והוא אחראי על אפשור התקשורת בין Node-ים שונים. הוא עושה זאת על ידי העברת הכתובת של כל Node ל- Node-ים האחרים.

כלומר המידע שנשלח מ-Node אחד ל-Node אחר לא עובר דרך ה-Directory server, המידע הזה נשלח ישירות לכתובות שכל Node מקבל מה-Directory server.

Controller:

זהו החלק שדרכו ניתן לשלוח משימות למערכת, החלק הזה מתחבר ל-Node שרץ על המחשב הנוכחי ושולח דרכו משימות ושאילתות ל-Node-ים אחרים.

את החלק הזה ניתן להפעיל בשתי דרכים:

* עם ממשק גרפי
* עם ממשק טקסטואלי

הממשק הטקסטואלי נועד בעיקר לשם אפשור של חיבור אפליקציות שונות למערכת בעזרת ניתוב מחדש של ה-stdin וה-stdout.

הסבר נוסף על פעולתו של כל חלק ועל אופן השימוש במערכת יוצג בהמשך.

# אופן פעולת המערכת

## פעולת המערכת באופן כללי

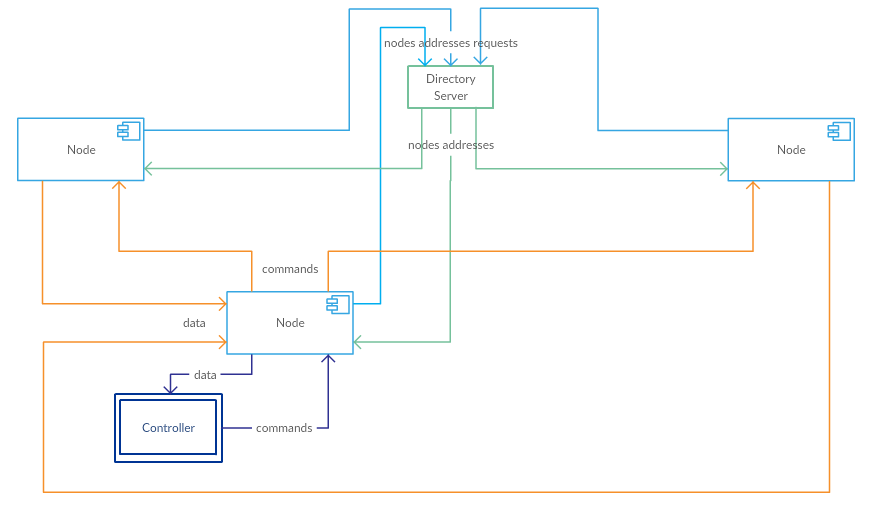
ה- DirectoryServer הוא שרת שמשתף את הכתובות של ה- Node-ים.

ה- Controller מתחבר ל- Node במחשב הנוכחי ושולח דרכו משימות ושאילתות לשאר ה- Node-ים.

ה- Node הוא החלק המרכזי שרץ על כל מחשב ומאפשר ביצוע של משימות והתחברות.

כאשר אין אף חיבור של Controller המערכת רק מתקשרת עם ה- DirectoryServer, אך לאחר התחברות של Controller והתחלת פעולה, ה- Node במחשב הנוכחי שולח משימות ל- Node-ים אחרים, והם מבצעים את המשימות ושולחים בחזרה מידע שהתקבל מביצוע הפעולות. לאחר מכן, ה- Controller מציג את המידע שהתקבל ושולח את המשימות הבאות וכך חוזר חלילה.

להלן שרטוט המציג את אופן הפעולה של המערכת באופן כללי:



## חלוקה למודולים

במערכת הזו ישנם מספר חלקים והם כאמור:

Node, Directory Server ו Controller.

החלקים Directory Server ו Controller בנויים כל אחד כרכיב יחיד והחלק Node בנוי ממספר רכיבים/מודולים והם:

Communication:

מערכת התקשורת של כל Node, היא כתובה ב- Python ובשימוש נרחב ב- Scapy. התקשורת היא בשימוש בפקטות UDP בכדי להימנע מחסימות של התקשורת ב- Firewall.

Database:

זהו המודול שאחראי על מסד הנתונים בכל Node. כתוב ב- Python ומשתמש ב- SQLite.

Decider:

למודול זה מועבר המידע מכל דבר שקורה בתוכנה והוא מעביר אותו לאן שנדרש. המודול הזה מתווך בין דברים שקורים בNode הנוכחי ומידע שמתקבל "מבחוץ". המודול כתוב ב- Python.

DriverCommunicator:

מטרת המודול הזה הוא לתקשר עם הדרייבר. נכתב ב- C++.

WDM-DKOM:

זהו הדרייבר המדובר, זהו Kernel Driver שמטרתו להחביא את המערכת (או לפחות את החלק המרכזי בה) מהמשתמש כדי שהוא לא יכבה את התוכנה. חלק זה פועל רק עבור מערכת הפעלה מבוססת 32 סיביות. הדרייבר נכתב ב- C.

ControllerConnection:

חלק זה מאפשר את החיבור של Controller במידת הצורך. נכתב ב- Python.

Manager:

חלק זה אחראי על העברת המידע בין כל המודולים והתחלת תהליכים חדשים (וניהולם) במידת הצורך. נכתב ב- C#.

**הערה:** המידע עובר בין המודלים בעזרת ניתוב מחדש של ה- stdin/stdout שה- Manager מבצע. בהסבר המורחב על המודולים, שיוצג בהמשך, לא אציג את אופן העברת המידע ואת שינוי הפורמט של המידע בדרך אלא רק את אופן פעולת המודול, אבל לשם שלמות ההסבר אציין כאן כי עם קבלת המידע, תהליכון בכל מודול שומר את המידע בתור באופן בטוח ותהליכון אחר מוציא את המידע באופן בטוח מהתור ומבצע את תפקידו של המודול.

## Communication

בכדי להפעיל את המודול הזה ה- Manager מריץ את הקובץ communicator.py, קובץ זה משנה את הפורמט של המידע ומטפל בהעברתו באופן הדרוש. בשביל התקשורת הוא משתמש במחלקה Communication שהיא מטפלת בתקשורת עם ה- Node-ים האחרים.

המחלקה Communication משתמשת בשני דברים מרכזיים בכדי לתקשר עם ה- Node-ים האחרים והם: Encoder ו- Lowlevelcommunicator עליהם יוסבר בקצרה בהמשך. להלן קוד המחלקה Communicator:

#!/usr/bin/python

**from** time **import** sleep

**import** communicationUtils

**from** encoder **import** Encoder

**from** lowLevelCommunicator **import** LowLevelCommunicator

**class** **Communication:**

**def** \_\_init\_\_**(**self**,** ID**,** communicationKey**,** holePunchingAddr **=** communicationUtils**.**GetDirServerAddr**()):**

self**.**port **=** communicationUtils**.**defaultPort**()**

**while** communicationUtils**.**IsPortTaken**(**self**.**port**):** # take over a port

self**.**port **+=** 1

**if** self**.**port **>=** 2**\*\***16**:**

self**.**port **=** 2000 # after saved ports

self**.**\_\_encoder **=** Encoder**(**communicationKey**)**

**with** open**(**"config.cfg"**,** "w"**)** **as** f**:**

f**.**write**(**self**.**\_\_encoder**.**encrypt**(**str**(**self**.**port**)))** # save the port for controller

self**.**\_\_communicator **=** LowLevelCommunicator**(**self**.**port**,** holePunchingAddr**,** ID**)**

self**.**\_\_communicator**.**start**()**

**def** send**(**self**,** qryOrTsk**,** toId**):** # FIN

self**.**\_\_communicator**.**log**.**write**(**"communication, sending: " **+** qryOrTsk **+** "\n"**)**

self**.**\_\_communicator**.**log**.**flush**()**

self**.**\_\_communicator**.**sendTo**(**self**.**\_\_encoder**.**encrypt**(**qryOrTsk**),** self**.**getAddrById**(**toId**))** # to = (host,port)

**def** getReceivedMessages**(**self**):**

QandT **=** self**.**\_\_communicator**.**getReceivedMessages**()** #getRecievedQuerriesAndTasks()

**if** len**(**QandT**)** **>** 0**:**

self**.**\_\_communicator**.**log**.**write**(**"communication, recieved messages: " **+** str**(**self**.**\_\_encoder**.**decrypt**(**QandT**))** **+** "\n"**)**

self**.**\_\_communicator**.**log**.**flush**()**

**return** self**.**\_\_encoder**.**decrypt**(**QandT**)**

**def** getAddrById**(**self**,** ID**):**

contacts **=** dict**(**self**.**\_\_communicator**.**getContacts**())**

**try:**

**return** contacts**[**str**(**ID**)]**

**except:**

**try:**

**return** self**.**\_\_communicator**.**getAllPossibleContacts**()[**str**(**ID**)]**

**except** Exception **as** e**:**

**raise** e

**def** refreshContacts**(**self**,** timeout **=** 0.2**):**

self**.**\_\_communicator**.**refreshContacts**()**

sleep**(**timeout**)**

**def** getContacts**(**self**):**

**return** **[**c**[**0**]** **for** c **in** self**.**\_\_communicator**.**getContacts**()]** # only id's

המחלקה Encoder:

מחלקה זו אחראית על ההצפנה, זוהי הצפנה יחסית פשוטה בעזרת פעולת xor עם מפתח בגודל 1024 סיביות שלאחריה התוצאה מקודדת בפורמט של Base64. להלן הקוד:

#!/usr/bin/python

**from** base64 **import** b64encode**,** b64decode

**class** **Encoder:** # encryption with xor (list of values) and base64

**def** \_\_init\_\_**(**self**,** key**):**

# key = 1024bit integer

self**.**\_\_keys **=** **[]**

**while** key **!=** 0**:**

self**.**\_\_keys**.**append**(**int**(**key**%**256**))**

key **/=** 256

**def** decrypt**(**self**,** msgs**):** # in format [(ID, msg)...]

**if(**type**(**msgs**)** **==** type**(**list**())):**

retLst **=** **[]**

**for** item **in** msgs**:**

retLst**.**append**((**item**[**0**],** self**.**\_\_decrypt**(**item**[**1**])))**

**return** retLst

**return** self**.**\_\_decrypt**(**msgs**)** # if single item

**def** \_\_decrypt**(**self**,** msg**):** # type(msg) = string

notBase64 **=** b64decode**(**msg**)**

newMsg **=** ""

i **=** 0

**for** ch **in** notBase64**:**

newMsg **+=** chr**(**ord**(**ch**)** **^** self**.**\_\_keys**[**i **%** len**(**self**.**\_\_keys**)])**

i **+=** 1

**return** newMsg

**def** encrypt**(**self**,** msg**):** # type(msg) = string

#return msg # debug

newMsg **=** ""

i **=** 0

**for** ch **in** msg**:**

newMsg **+=** chr**(**ord**(**ch**)** **^** self**.**\_\_keys**[**i **%** len**(**self**.**\_\_keys**)])**

i **+=** 1

**return** b64encode**(**newMsg**)** # base 64 to disable some spacial characters

אציג כאן את הקוד של LowLevelCommunicator ששולח ומקבל הודעות בפועל בעזרת Scapy אבל לא ארחיב עליו לעומק. באופן כללי החלק הזה מקבל את כל הפקטות שהמחשב מקבל ומסנן רק את אלה שנשלחו אל ה- Node ושומר אותם. תהליכון אחר מסנן את הפקטות לפי המטרה שלהן בעזרת מידע שנמצא בפקטה ומחלץ מהן מידע וברגע שנקראת הפונקציה getReceivedMessages מורכבות הודעות מהמידע שהתקבל קודם לכן וכל ההודעות השלמות (כלומר שהתקבלו במלואן) מוחזרות ברשימה. יש לשים לב שההודעות לא בהכרח יתקבלו באותו הסדר.

ברגע שפקטה מתקבלת נשלחת פקטת תשובה שמעידה על קבלת הפקטה, במידה ו- Node שולח הודעה ולא מקבל פקטת תשובה, הוא שולח את הפקטה שוב, כך אפשר לאמת שכל המידע התקבל. להלן הקוד המדובר:

#!/usr/bin/python

**from** scapy**.**all **import** **\***

**from** thread **import** start\_new\_thread

**from** socket **import** **\***

**from** time **import** sleep**,** time

**import** communicationUtils

**class** **LowLevelCommunicator:** # FIN

**def** \_\_init\_\_**(**self**,** port**,** holePunchingAddr**,** ID**):** # FIN

self**.**\_\_port **=** port

self**.**\_\_recvFromAddr **=** **{}** # store addresses that sended to this node by format {ID:ADDR}

self**.**\_\_sendedAndNotResponded **=** **[]**

self**.**\_\_sniffed **=** **[]**

self**.**\_\_rawMessages **=** **{}**

self**.**\_\_pacTimeout **=** 1 # 1 = default # in seconds

self**.**\_\_isPortProtectionServiceStarted **=** **False**

self**.**\_\_isRecievingThreadStarted **=** **False**

self**.**\_\_isSendedValidationThreadStarted **=** **False**

self**.**\_\_shutdown **=** **False**

self**.**\_\_holePunchingAddr **=** holePunchingAddr

self**.**\_\_dirSerAddr **=** communicationUtils**.**GetDirServerAddr**();**

self**.**\_\_ID **=** ID

self**.**\_\_seq **=** 0

self**.**\_\_recentMessagesIdSeq **=** **[]** # last 20 seconds msgs (id,seq)

self**.**\_\_recentMessagesTimeStamp **=** **[]** # last 500 msgs (id,seq)

self**.**otherNodes **=** **[]** # [(ID,ADDR)...]

self**.**allOtherNodes **=** **{}** # {ID:ADDR, ...}

self**.**EOM **=** "<EOF>" # end of message

self**.**SOM **=** "<SOM>" # start of message

self**.**MAX\_SEQ **=** 2**\*\***16

self**.**log **=** open**(**"lowLevelCom.log"**,** "w"**)**

# self.pcapWriter = PcapWriter("sniffLog" + str(ID) + ".pcap", append=True, sync=True) for debugging

**def** start**(**self**):** # FIN

self**.**startPortProtectionService**()**

self**.**startRecievingThread**()**

self**.**startSendedMsgsValidationThread**()**

**def** startPortProtectionService**(**self**):** # FIN

**if** self**.**\_\_isPortProtectionServiceStarted**:** **return** # already started

start\_new\_thread**(**self**.**\_\_portProtectionService**,** **())**

self**.**\_\_isPortProtectionServiceStarted **=** **True**

self**.**log**.**write**(**"port protection started\n"**)**

self**.**log**.**flush**()**

**def** \_\_portProtectionService**(**self**,** gapBetweenPunches**=**1**):** # FIN

i **=** 0

**while** **not** self**.**\_\_shutdown**:**

# optional: maybe block any connection that attemps to bind or use that port

i **+=** 1

**if** i **==** 2**:**

self**.**requestContacts**()**

i **=** 0

**else:**

self**.**log**.**write**(**"hole punching...\n"**)**

self**.**log**.**flush**()**

holePunchPac **=** IP**(**dst**=**self**.**\_\_holePunchingAddr**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**self**.**\_\_holePunchingAddr**[**1**])/**str**(**self**.**\_\_ID**)**

send**(**holePunchPac**,** verbose**=False)**

sleep**(**gapBetweenPunches**)**

**def** startSendedMsgsValidationThread**(**self**):** #FIN

**if** self**.**\_\_isSendedValidationThreadStarted**:** **return** # already started

start\_new\_thread**(**self**.**\_\_sendedValidationThread**,** **())**

self**.**\_\_isSendedValidationThreadStarted **=** **True**

self**.**log**.**write**(**"sended messages validation started\n"**)**

self**.**log**.**flush**()**

**def** startRecievingThread**(**self**):** # FIN

**if** self**.**\_\_isRecievingThreadStarted**:** **return** # already started

start\_new\_thread**(**self**.**\_\_recievingThread**,** **())**

self**.**\_\_isRecievingThreadStarted **=** **True**

self**.**log**.**write**(**"recieving started\n"**)**

self**.**log**.**flush**()**

**def** \_\_recievingThread**(**self**):** # FIN

start\_new\_thread**(**self**.**\_\_sniffingThread**,())**

**while** **not** self**.**\_\_shutdown**:**

sniffedCpy **=** self**.**\_\_sniffed**[**0**:]**

self**.**\_\_sniffed **=** **[]**

**for** pac **in** sniffedCpy**:** # self.\_\_sniffed:

# check if the packet is valid

origChecksum **=** pac**[**UDP**].**chksum

**del** pac**[**UDP**].**chksum

pac **=** pac**.**\_\_class\_\_**(**str**(**pac**))** # dump to string and rebuild the packet

**if** origChecksum **!=** pac**[**UDP**].**chksum**:** # packet is damaged

**continue** # ignore this packet

# then extract data to recData

**try:**

splt **=** pac**[**Raw**].**load**.**split**(**","**)**

self**.**\_\_recvFromAddr**[**splt**[**0**]]** **=** **(**pac**[**IP**].**src**,** pac**[**UDP**].**sport**)** # add to addresses by ID's dictionary

**if** splt**[**2**]** **==** "r"**:** # recieved response => remove from self.\_\_sendedAndNotResponded

# print "r received" # for debug

**for** i **in** self**.**\_\_sendedAndNotResponded**:**

**if** i**[**0**]** **==** int**(**splt**[**3**]):** # remove

# print "removed: " + str(i) # for debug

self**.**\_\_sendedAndNotResponded**.**remove**(**i**)**

**break**

**elif** splt**[**0**]** **==** "0"**:** # ID == 0 => its a directory server

# its the node connections data... use it, save it to list or ID

**for** i **in** eval**(**","**.**join**(**splt**[**3**:])):** # list

**if** i **not** **in** self**.**otherNodes**:**

self**.**otherNodes**.**append**(**i**)** # (ID, ADDR), ADDR = (IP, PORT)

self**.**allOtherNodes**.**update**(**dict**(**self**.**otherNodes**))**

**elif** splt**[**2**]** **==** "m"**:** # msg

**if** **(**splt**[**0**],** splt**[**1**])** **not** **in** self**.**\_\_recentMessagesIdSeq**:** # not recieved yet

self**.**\_\_recentMessagesIdSeq**.**append**((**splt**[**0**],** splt**[**1**]))**

self**.**\_\_recentMessagesTimeStamp**.**append**(**time**())**

self**.**\_\_rawMessages**[(**int**(**splt**[**0**]),** int**(**splt**[**1**]))]** **=** ","**.**join**(**splt**[**3**:])** # dict[ID, Seq] = data

self**.**\_\_sendRecievedResponse**(**splt**)**

**else:** # unimplemented packet type

**raise** Exception**(**"not implemented packet type (LLC reached to else statment)"**)**

**except** Exception **as** e**:**

**raise** Exception**(**"illegal packet, exception: " **+** str**(**e**))**

# remove messages that older than 20 seconds, by binary search -> fast

currTime **=** time**()** # faster

start **=** 0

end **=** len**(**self**.**\_\_recentMessagesTimeStamp**)** **-** 1

mid **=** 0

**while** start **<=** end**:**

mid **=** **(**start **+** end**)** **/** 2

**if** currTime **-** self**.**\_\_recentMessagesTimeStamp**[**mid**]** **>** 10**:**

start **=** mid **+** 1

**else:**

end **=** mid **-** 1

self**.**\_\_recentMessagesIdSeq **=** self**.**\_\_recentMessagesIdSeq**[**mid**:]**

self**.**\_\_recentMessagesTimeStamp **=** self**.**\_\_recentMessagesTimeStamp**[**mid**:]**

**if** **(**len**(**self**.**\_\_sniffed**)** **==** 0**):**

sleep**(**0.1**)**

**def** \_\_sniffingThread**(**self**):** # FIN

myIntIps **=** communicationUtils**.**GetMachineInternalIps**()**

myIntIps **=** **[**ip**.**encode**(**'ascii'**,** 'ignore'**)** **for** ip **in** myIntIps**]** # from unicode to str

pacFilter **=** **lambda** p**:** p**.**haslayer**(**UDP**)** **and** p**[**UDP**].**dport **==** self**.**\_\_port **and** p**.**haslayer**(**IP**)** **and** p**[**IP**].**dst **in** myIntIps

stopFilter **=** **lambda** x**:** self**.**\_\_shutdown

sniff**(**lfilter**=**pacFilter**,** prn**=**self**.**\_\_appendSniffedPac**,** stop\_filter**=**stopFilter**)**

**def** \_\_appendSniffedPac**(**self**,** p**):** # FIN

#print "appended: ",

#p.summary()

self**.**\_\_sniffed**.**append**(**p**)**

**def** sendTo**(**self**,** msg**,** to**):** # FIN

**if** type**(**to**)** **==** type**(**list**()):**

**for** node **in** to**:**

self**.**\_\_sendTo**(**msg**,** Node**)**

**else:** # to single node

self**.**\_\_sendTo**(**msg**,** to**)**

# returning recieved messages in format: [(ID1, msg1), (ID2, msg2), (ID3, msg3)...]

**def** getReceivedMessages**(**self**):** # FIN

**if** len**(**self**.**\_\_rawMessages**)** **==** 0**:** # if raw messages empty exit

**return** **[]**

messages **=** **[]**

sortedRawMessagesKeys **=** list**(**sorted**(**self**.**\_\_rawMessages**.**keys**()))** # sort by (primaryElement, secondaryElement)

isMsgValid **=** **True**

lastSeqId **=** sortedRawMessagesKeys**[**0**]** # first key[0] = first key id

lastKeys **=** **[**sortedRawMessagesKeys**[**0**]]** # initialize with the first key

msgParts **=** **[]** #[self.\_\_rawMessages[sortedRawMessagesKeys[0]]]

**for** i **in** sortedRawMessagesKeys**:** #[1:]:

#print "i: ", type(i), " last: ", type(lastSeqId)

**if** self**.**\_\_rawMessages**[**i**]** **==** self**.**SOM**:**

isMsgValid **=** **True** # reinitialize for the msg

lastKeys **=** **[]** # reinitialize for the msg

msgParts **=** **[]** # reinitialize for the msg

**elif** isMsgValid **and** lastSeqId**[**0**]** **==** i**[**0**]** **and** **(**lastSeqId**[**1**]** **+** 1**)** **%** self**.**MAX\_SEQ **==** i**[**1**]** **and** self**.**\_\_rawMessages**[**i**]** **==** self**.**EOM**:**

lastKeys**.**append**(**i**)**

**if** isMsgValid**:**

messages**.**append**((**i**[**0**],** ""**.**join**(**msgParts**)))** # ID, msg

**for** key **in** lastKeys**:** # remove "used" raw messages

self**.**\_\_rawMessages**.**pop**(**key**)**

isMsgValid **=** **False**

**elif** isMsgValid **and** lastSeqId**[**0**]** **==** i**[**0**]** **and** **(**lastSeqId**[**1**]** **+** 1**)** **%** self**.**MAX\_SEQ **==** i**[**1**]:** # same id and next seq

msgParts**.**append**(**self**.**\_\_rawMessages**[**i**])**

lastKeys**.**append**(**i**)**

**else:** # lastSeqId[0] == i[0] and lastSeqId[1] + 1 != i[1]

isMsgValid **=** **False**

lastSeqId **=** i

**return** messages # in the format (ID, msg)

**def** \_\_sendedValidationThread**(**self**):** # FIN

"""

re-send packets that havn't recieved or recieved incorrectly

"""

**while** **not** self**.**\_\_shutdown**:**

**for** i **in** self**.**\_\_sendedAndNotResponded**:** # i = (seq, time(), pac)

**if** time**()** **-** i**[**1**]** **>=** self**.**\_\_pacTimeout**:**

self**.**log**.**write**(**"re sended: " **+** str**(**i**[**2**])** **+** "\n"**)**

self**.**log**.**flush**()**

send**(**i**[**2**],** verbose **=** **False)**

i**[**1**]** **=** time**()**

# print "re-sent: " + str(i) # for debug

sleep**(**0.2**)**

**def** \_\_sendTo**(**self**,** msg**,** to**):** # FIN

# to = (ip, port)

self**.**log**.**write**(**"sending: " **+** msg **+** "\n"**)**

self**.**log**.**flush**()**

msgIndicatorLen **=** len**(**"m,"**)** # indicates thats a message

maxIdAndSeqLen **=** len**(**str**(**2**\*\***48**))** **+** len**(**str**(**self**.**MAX\_SEQ**))**

maxPortLength **=** len**(**str**(**2**\*\***16**))**

dataPerPac **=** **(**508 **-** maxIdAndSeqLen **-** maxPortLength **-** msgIndicatorLen**)** # 508 = for sure safe length

numToSend **=** int**(**len**(**msg**)** **/** dataPerPac**)** **+** 1 # round down + 1 =~ round up

toSend **=** **[]**

toSend**.**append**(**IP**(**dst**=**to**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**to**[**1**])/(**str**(**self**.**\_\_ID**)** **+** "," **+** str**(**self**.**\_\_seq**)** **+** ",m," **+** self**.**SOM**))** # start message

self**.**\_\_sendedAndNotResponded**.**append**([**self**.**\_\_seq**,** time**(),** toSend**[-**1**]])**

self**.**\_\_incSeq**()**

**for** i **in** xrange**(**numToSend **-** 1**):** # split data to packets with max len of "dataPerPac"

# ID,Seq,messageIndicator,data

# the response address would be found by the ID

raw **=** str**(**self**.**\_\_ID**)** **+** "," **+** str**(**self**.**\_\_seq**)** **+** ",m," **+** msg**[**i**\***dataPerPac**:(**i**+**1**)\***dataPerPac**]**

toSend**.**append**(**IP**(**dst**=**to**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**to**[**1**])/**raw**)**

self**.**\_\_sendedAndNotResponded**.**append**([**self**.**\_\_seq**,** time**(),** toSend**[-**1**]])**

self**.**\_\_incSeq**()**

toSend**.**append**(**IP**(**dst**=**to**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**to**[**1**])/**

**(**str**(**self**.**\_\_ID**)** **+** "," **+** str**(**self**.**\_\_seq**)** **+** ",m," **+** msg**[(**numToSend**-**1**)\***dataPerPac**:]))** # last data packet

self**.**\_\_sendedAndNotResponded**.**append**([**self**.**\_\_seq**,** time**(),** toSend**[-**1**]])**

self**.**\_\_incSeq**()**

toSend**.**append**(**IP**(**dst**=**to**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**to**[**1**])/(**str**(**self**.**\_\_ID**)** **+** "," **+** str**(**self**.**\_\_seq**)** **+** ",m," **+** self**.**EOM**))** # end message

self**.**\_\_sendedAndNotResponded**.**append**([**self**.**\_\_seq**,** time**(),** toSend**[-**1**]])**

self**.**\_\_incSeq**()**

#print "tosend: " + str(toSend) # fir debugging

send**(**toSend**,** verbose **=** **False)**

**def** \_\_sendRecievedResponse**(**self**,** pacDataSplt**):** # FIN

self**.**log**.**write**(**"sended recieved response to: " **+** str**(**pacDataSplt**)** **+** "\n"**)**

self**.**log**.**flush**()**

splt **=** pacDataSplt # pacData.split(",")

**try:**

to **=** self**.**\_\_recvFromAddr**[**splt**[**0**]]** # self.getAddrById(int(splt[0]))

**except** Exception **as** e**:**

**print** "send recv response err: " **+** str**(**e**)**

# ID,Seq,recvResponseIndicator,recPacSeq

raw **=** str**(**self**.**\_\_ID**)** **+** "," **+** str**(**self**.**\_\_seq**)** **+** ",r," **+** splt**[**1**]** #r=recieved , splt[1] = other node's seq

# print "sending resp: " + raw # for debugging

send**(**IP**(**dst**=**to**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**to**[**1**])/**raw**,** verbose **=** **False)**

**def** \_\_incSeq**(**self**):** # FIN

""" icrease sequence indicator """

self**.**\_\_seq **+=** 1

**if** self**.**\_\_seq **>=** self**.**MAX\_SEQ**:** self**.**\_\_seq **=** 0 # reset seq # like % but faster

**def** shutdown**(**self**):** # FIN

self**.**\_\_shutdown **=** **True**

sleep**(**1**)**

**def** refreshContacts**(**self**):**

self**.**log**.**write**(**"refreshed contacts\n"**)**

self**.**log**.**flush**()**

self**.**otherNodes **=** **[]**

self**.**requestContacts**()**

**def** requestContacts**(**self**):**

self**.**log**.**write**(**"requested for contacts\n"**)**

self**.**log**.**flush**()**

requestPac **=** IP**(**dst**=**self**.**\_\_dirSerAddr**[**0**])/**UDP**(**sport**=**self**.**\_\_port**,** dport**=**self**.**\_\_dirSerAddr**[**1**])/(**">" **+** str**(**self**.**\_\_ID**))**

send**(**requestPac**,** verbose**=False)**

**def** getContacts**(**self**):**

**return** self**.**otherNodes**[**0**:]** # copy

**def** getAllPossibleContacts**(**self**):** # return {ID:ADDR,...} for all nodes from all times

ret **=** self**.**\_\_recvFromAddr**.**copy**()**

ret**.**update**(**self**.**allOtherNodes**)**

**return** ret

## Database

זהו מודול שפותח קובץ SQLite ומריץ שאילתות שמועברות אליו, על מסד הנתונים ומחזיר את התוצאות של כל שאילתא. שם הקובץ הוא ה- ID של ה- Node בבסיס הקסדצימלי (שזוהי כתובת ה- mac של כרטיס הרשת שלו) כלומר לפי שם הקובץ ניתן לפצל את המידע ששומרים בכל שאילתא רק במספר מסויים של Node-ים או לחפש מידע מסויים רק ב- Node-ים מסויימים.

הקוד של המודול הזה הוא פשוט יחסית ולכן לא אכתוב אותו פה.

## Decider

בכדי להפעיל את המודול הזה ה- Manager מריץ את הקובץ decider.py שמשנה את הפורמט של המידע ומעביר אותו למחלקה בשם DataController, מחלקה זו בעיקר משנה את המידע בצורה הדרושה כדי לאפשר תקשורת בין הNode הנוכחי לאחרים, ניתן לראות במודול הזה כמו NAT בשביל משימות במקום כתובות IP. להלן קוד המחלקה DataController:

**import** os

**from** task **import** **\***

**class** **DataController:**

#region constants

SEND\_CMD **=** 's'

START\_PROCESS\_CMD **=** 'p'

QUERY\_CMD **=** 'q'

PROCESS\_ENDED\_CODE **=** 'e'

PROCESS\_DATA\_CODE **=** 'd'

QUERY\_RESPONSE\_CODE **=** 'Q'

TASK\_CODE **=** 't'

DISPLAY\_CODE **=** 'D' # display to admin <=> only if the admin is connected

WRITE\_FILE\_CMD **=** 'w'

CLIENTS\_LIST\_CODE **=** 'c'

#endregion

#region static vars

tasks **=** **[]**

#endregion

#region "events"

@staticmethod

**def** OnCommunicationDataRecieved**(**data**):** # check what command recieved

data **=** eval**(**data**)**

**if** data**[**0**]** **==** DataController**.**CLIENTS\_LIST\_CODE**:**

**print** DataController**.**DISPLAY\_CODE **+** repr**(**data**)**

**elif** data**[**0**]** **==** DataController**.**DISPLAY\_CODE**:** # start an admin session request

**print** DataController**.**DISPLAY\_CODE **+** data**[**1**:].**split**(**","**)[**1**]** # data[1:] = -1,portNum,

**elif** data**[**0**]** **==** DataController**.**START\_PROCESS\_CMD**:** # start process, to c#, not repered

DataController**.**StartProcess**(**data**[**1**:])**

**elif** data**[**0**]** **==** DataController**.**QUERY\_CMD**:** # start query

DataController**.**Query**(**data**[**1**:])**

**elif** data**[**0**]** **==** DataController**.**PROCESS\_DATA\_CODE**:** # data recieved from task

**print** DataController**.**DISPLAY\_CODE **+** repr**(**data**)** # keep the CMD code and send to the controller

**elif** data**[**0**]** **==** DataController**.**PROCESS\_ENDED\_CODE**:** # a sended task ended

**print** DataController**.**DISPLAY\_CODE **+** repr**(**data**)** # keep the CMD code

**elif** data**[**0**]** **==** DataController**.**QUERY\_RESPONSE\_CODE**:** # a sended query response

**print** DataController**.**DISPLAY\_CODE **+** repr**(**data**)** # keep the CMD code

**elif** data**[**0**]** **==** DataController**.**WRITE\_FILE\_CMD**:**

splt **=** data**[**1**:].**split**(**","**)**

**try:** # make sure the dir exists

os**.**makedirs**(**"temp/" **+** os**.**path**.**dirname**(**splt**[**2**]))** # take only dir

**except** OSError**:** # dir exists

**pass**

**with** open**(**"temp/" **+** splt**[**2**],** "w"**)** **as** f**:**

f**.**write**(**","**.**join**(**data**.**split**(**","**)[**3**:]))** # write to file

DataController**.**SendData**(**splt**[**0**],** splt**[**1**],** DataController**.**PROCESS\_ENDED\_CODE**)**

#SendData(sourceNodeId, sourceNodeTaskId, PROCESS\_ENDED\_CODE)

**else:**

**raise** Exception**(**"unknown command: '" **+** data**[**0**]** **+** "' full msg: " **+** data**)**

@staticmethod

**def** OnProcessDataReceived**(**data**):**

tsk **=** DataController**.**\_FindTaskByData**(**data**)**

DataController**.**SendData**(**tsk**.**sourceNodeId**,** tsk**.**sourceNodeTaskId**,** DataController**.**PROCESS\_DATA\_CODE **+** ","**.**join**(**data**.**split**(**","**)[**1**:]))**

@staticmethod

**def** OnProcessEnded**(**data**):**

tsk **=** DataController**.**\_FindTaskByData**(**data**)**

DataController**.**SendData**(**tsk**.**sourceNodeId**,** tsk**.**sourceNodeTaskId**,** DataController**.**PROCESS\_ENDED\_CODE**)**

DataController**.**tasks**.**remove**(**tsk**)**

@staticmethod

**def** OnQueryReceived**(**data**):**

tsk **=** DataController**.**\_FindTaskByData**(**data**)**

DataController**.**SendData**(**tsk**.**sourceNodeId**,** tsk**.**sourceNodeTaskId**,** DataController**.**PROCESS\_DATA\_CODE **+** ","**.**join**(**data**.**split**(**","**)[**1**:]))**

DataController**.**tasks**.**remove**(**tsk**)**

#endregion

@staticmethod

**def** SendData**(**toId**,** nodeTaskId**,** data**,** toRepr**=True):**

**if** toRepr**:**

**print** DataController**.**SEND\_CMD **+** toId **+** "," **+** nodeTaskId **+** "," **+** repr**(**data**)** **+** "\n"**,**

**else:**

**print** DataController**.**SEND\_CMD **+** toId **+** "," **+** nodeTaskId **+** "," **+** data **+** "\n"**,**# if the data is already formatted

@staticmethod

**def** StartProcess**(**data**):**

sourceNodeId**,** sourceNodeTaskId **=** data**.**split**(**","**)[**0**:**2**]**

tsk **=** Task**(**TaskType**.**TASK**,** sourceNodeId**,** sourceNodeTaskId**)**

DataController**.**tasks**.**append**(**tsk**)**

**print** DataController**.**START\_PROCESS\_CMD **+** str**(**tsk**.**id**)** **+** "," **+** ","**.**join**(**data**.**split**(**","**)[**2**:])** **+** "\n"**,** # do not repr (c# is starting the process)

@staticmethod

**def** Query**(**data**):**

fromId**,** nodeTaskId **=** data**.**split**(**","**)[**0**:**2**]** # take the first 2 vals (fromId, nodeTaskId, data)

tsk **=** Task**(**TaskType**.**QUERY**,** fromId**,** nodeTaskId**)**

DataController**.**tasks**.**append**(**tsk**)**

**print** DataController**.**QUERY\_CMD **+** str**(**tsk**.**id**)** **+** "," **+** repr**(**","**.**join**(**data**.**split**(**","**)[**2**:]))** **+** "\n"**,**

@staticmethod

**def** \_FindTaskByData**(**data**):** # if the complexity will be too high then turn it to dictionary

**for** t **in** DataController**.**tasks**:**

**if** t**.**id **==** int**(**data**.**split**(**","**)[**0**]):**

**return** t

המחלקה DataController משתמשת במחלקה Task שתפקידה להחזיק במידע עבור משימות שהתקבלו ויצירת id חדש עבור כל משימה, קוד המחלקה Task:

**class** **Task:**

nextId **=** 0 # static var

**def** \_\_init\_\_**(**self**,** taskType**,** sourceNodeId**,** sourceNodeTaskId**):**

self**.**type **=** taskType

self**.**id **=** Task**.**nextId

self**.**sourceNodeId **=** sourceNodeId

self**.**sourceNodeTaskId **=** sourceNodeTaskId

Task**.**nextId **+=** 1

**if** Task**.**nextId **>=** 2**\*\***16**:**

Task**.**nextId **=** 0

## DriverCommunicator

מודול זה טוען את הדרייבר WDM-DKOM לזיכרון ומתקשר איתו.

ישנן 2 פונקציות מרכזיות למודול הזה והן פונקציית הטעינה ופונקציית ההחבאה של תהליך (קריאה לדרייבר להחביא תהליך).

פונקציית הטעינה משתמשת ב Service Control Manager כדי לטעון את הדרייבר ולכן דרושות הרשאות גבוהות. להלן הקוד של הפונקציות:

פונקציית הטעינה:

bool loadSysFileSCM**(**char**\*** driverName**,** char**\*** displayName**,** char**\*** serviceName**)** // load manually with scm

**{**

// Open a handle to the SCM

SC\_HANDLE scmHandle **=** OpenSCManager**(NULL,** **NULL,** SC\_MANAGER\_ALL\_ACCESS**);**

**if** **(!**scmHandle**)**

**{**

**return** FALSE**;**

**}**

char currDir**[**515**];**

GetCurrentDirectory**(**512**,** currDir**);** // get the .sys file location

char path**[**1000**];**

snprintf**(**path**,** 998**,** "%s\\%s.sys"**,** currDir**,** driverName**);** // string formatting

SC\_HANDLE driverHandle **=** CreateService**(**scmHandle**,** // Handle to SCManager

serviceName**,** // Service Name

displayName**,** // Display Name

SERVICE\_ALL\_ACCESS**,** // Desired Access

SERVICE\_KERNEL\_DRIVER**,** // Service Type

SERVICE\_DEMAND\_START**,** // Start Type

SERVICE\_ERROR\_NORMAL**,** // Error Controle

path**,** // Binary Path Name

**NULL,** // Load OrderGroup

**NULL,** // Tag Id

**NULL,** // Dependencies

**NULL,** // Service Start Name

**NULL);** // Password

**if** **(!**driverHandle**)**

**{**

**if** **(**GetLastError**()** **==** ERROR\_SERVICE\_EXISTS**)**

**{** // Service exists

driverHandle **=** OpenService**(**scmHandle**,** serviceName**,** SERVICE\_ALL\_ACCESS**);** // get a handle to the existing service handle

**if** **(!**driverHandle**)**

**{**

CloseServiceHandle**(**scmHandle**);**

**return** FALSE**;**

**}**

**}**

**else**

**{**

CloseServiceHandle**(**scmHandle**);**

**return** FALSE**;**

**}**

**}**

// start the Driver

**if** **(**0 **==** StartService**(**driverHandle**,** 0**,** **NULL)** **&&** GetLastError**()** **!=** ERROR\_SERVICE\_ALREADY\_RUNNING**)**

**{**

CloseServiceHandle**(**scmHandle**);**

CloseServiceHandle**(**driverHandle**);**

**return** FALSE**;**

**}**

CloseServiceHandle**(**scmHandle**);**

CloseServiceHandle**(**driverHandle**);**

**return** TRUE**;**

**}**

פונקציית ההחבאה:

bool hideProcess**(**char **\*** driverName**,** int pid**)**

**{**

char**\*** driverToOpen **=** **new** char**[**100**];**

sprintf**(**driverToOpen**,** "\\\\.\\%s"**,** driverName**);** // "\\\\.\\DKOM" for example

HANDLE hFile **=** CreateFile**(**driverToOpen**,** GENERIC\_WRITE**,** FILE\_SHARE\_READ **|** FILE\_SHARE\_WRITE**,** **NULL,** OPEN\_EXISTING**,** 0**,** **NULL);** // open driver

**if** **(**hFile **==** INVALID\_HANDLE\_VALUE**)**

**{**

printf**(**"Error: Unable to connect to the driver (%d)\nMake sure the driver is loaded.\n"**,** GetLastError**());**

**return** FALSE**;**

**}**

DWORD write**;**

**if** **(!**WriteFile**(**hFile**,** **&**pid**,** **sizeof(**DWORD**),** **&**write**,** **NULL))** // hide

**{**

printf**(**"\nError: Unable to hide process (%d)\n"**,** GetLastError**());**

**return** FALSE**;**

**}**

**return** TRUE**;**

**}**

## WDM-DKOM

תפקיד הדרייבר הזה הוא להחביא תהליך מהמשתמש, הוא עושה זאת על ידי הסרת התהליך מרשימת התהליכים ב- Kernel כלומר רשימת הEPROCESS-ים השיטה הזו נקראת:

DKOM – Direct Kernel Object Manipulation

והיא אינה עובדת במערכות הפעלה מבוססות 64 סיביות. ה Id של התהליך מועבר בעזרת ה- irp (IO Request Packet). הפונקצייה שמבצעת זאת מובאת להלן:

// this is the "MJ\_WRITE" method:

NTSTATUS HideProcess**(**PDEVICE\_OBJECT pDeviceObj**,** PIRP irp**)** // pointer to device object and pointer to irp (IO request packet)

**{**

UNREFERENCED\_PARAMETER**(**pDeviceObj**);**

irp**->**IoStatus**.**Information **=** 0**;** // if not successful

#pragma region get passed data

PVOID buffer **=** MmGetSystemAddressForMdlSafe**(**irp**->**MdlAddress**,** NormalPagePriority**);** // get non paged memory => this is the data passed with the request (pid)

**if(!**buffer**)**

**{**

irp**->**IoStatus**.**Status **=** STATUS\_INSUFFICIENT\_RESOURCES**;**

IoCompleteRequest**(**irp**,** IO\_NO\_INCREMENT**);**

**return** STATUS\_INSUFFICIENT\_RESOURCES**;**

**}**

DbgPrint**(**"Process ID: %d"**,** **\*(**PHANDLE**)**buffer**);**

#pragma endregion

#pragma region try to access process

PEPROCESS Process**;**

NTSTATUS status **=** PsLookupProcessByProcessId**(\*(**PHANDLE**)**buffer**,** **&**Process**);**

**if(!**NT\_SUCCESS**(**status**))** // if fails

**{**

DbgPrint**(**"Error: Unable to open process object (%#x)"**,** status**);**

irp**->**IoStatus**.**Status **=** status**;**

IoCompleteRequest**(**irp**,** IO\_NO\_INCREMENT**);**

**return** STATUS\_INSUFFICIENT\_RESOURCES**;**

**}**

#pragma endregion

DbgPrint**(**"EPROCESS address: %#x"**,** Process**);**

PULONG ptr **=** **(**PULONG**)**Process**;**

#pragma region scan EPROCESS for pid

// Scan the EPROCESS structure for the PID

ULONG offset **=** 0**;**

PLIST\_ENTRY CurrListEntry**;**

**for(**short i **=** 0**;** i **<** **(**short**)**512**;** i**++)**

**{**

**if(**ptr**[**i**]** **==** **\*((**PULONG**)**buffer**))**

**{**

offset **=** **(**ULONG**)&**ptr**[**i **+** 1**]** **-** **(**ULONG**)**Process**;** // ActiveProcessLinks is located next to the PID

CurrListEntry **=** **(**PLIST\_ENTRY**)((**PUCHAR**)&**ptr**[**i **+** 1**]);**

DbgPrint**(**"ActiveProcessLinks offset: %#x"**,** offset**);**

**break;**

**}**

**}**

**if(!**offset**)** // not found

**{**

irp**->**IoStatus**.**Status **=** STATUS\_UNSUCCESSFUL**;**

IoCompleteRequest**(**irp**,** IO\_NO\_INCREMENT**);**

**return** STATUS\_UNSUCCESSFUL**;**

**}**

#pragma endregion

CurrListEntry **=** **(**PLIST\_ENTRY**)((**PUCHAR**)**Process **+** offset**);** // Get the ActiveProcessLinks address

#pragma region remove EPROCESS object

// Unlink the target process from other processes (unlink from list):

CurrListEntry**->**Blink**->**Flink **=** CurrListEntry**->**Flink**;** // point prevEPROCESS Flink to nextEPROCESS

CurrListEntry**->**Flink**->**Blink **=** CurrListEntry**->**Blink**;** // point nextEPROCESS Blink to prevEPROCESS

CurrListEntry**->**Flink **=** CurrListEntry**;** // Point Flink and Blink to self to prevent BSOD

CurrListEntry**->**Blink **=** CurrListEntry**;**

ObDereferenceObject**(**Process**);** // Dereference the target process -> decrease the reference counter

#pragma endregion

irp**->**IoStatus**.**Information **=** **sizeof(**HANDLE**);** // if successful there is information

irp**->**IoStatus**.**Status **=** STATUS\_SUCCESS**;**

IoCompleteRequest**(**irp**,** IO\_NO\_INCREMENT**);**

**return** STATUS\_SUCCESS**;**

**}**

## ControllerConnection

## בכדי להפעיל את המודול הזה ה- Manager מריץ את הפקודה:

controllerConnection.py [port]

או ליתר דיוק:

python –u controllerConnection.py [port]

מידע על הארגומנט –u ניתן למצוא בחלק של Manager.

ControllerConnection פותח שרת TCP בפורט שהועבר, שאפשר להתחבר אליו רק מתוך המחשב וכך ה- Controller יכול להתחבר.

הדבר פשוט יחסית ולכן לא אביא כאן קטע קוד.

## Manager

זהו הקובץ שמופעל בכדי להריץ את התוכנה (ה- Node). המודול הזה מריץ את שאר המודולים. המודול ברובו הוא מבוסס אירועים כלומר כאשר קורה משהו (לדוגמא קבלת מידע מתהליך) מתקבלת הודעה ומתבצעת הפעולה המתאימה. הדבר היחיד שבו הוא לא לגמרי מבוסס אירועים הוא מסד הנתונים וזאת משום שמסד הנתונים יכול לבצע רק פעולה אחת ברצף ולכן קיים תהליכון שמחכה להוראות לביצוע במסד הנתונים.

קל לשים לב כי כאשר ה Manager מריץ קבצי Python הוא כותב לפניהם: python –u, הסיבה לכך היא שהארגומנט –u גורם למידע המודפס במהלך הריצה להיות מודפס ישר (ולא להיכנס ל- Buffer) בלי צורך לקרוא ל- sys.stdout.flush() (שירוקן את ה- Buffer), זוהי תכונה של Python ולכן לא אתעכב עליה.

במחלקה ProcessManager יש שימוש במחלקות ProcessHandler ו- ProcessHider, אני לא אתייחס אליהם משום שהקוד שם לא מיוחד אך באופן כללי:

ProcessHider משתמש במודול DriverCommunicator ומחביא תהליכים אם מערכת ההפעלה מאפשרת זאת.

ProcessManager משתמש במחלקה Process של מייקרוסופט ובמחלקה ProcessHider וכך יוצרת API נוח לשימוש להתחלת תהליכים וניהולם.

עם הרצת הקובץ, מתבצעות כמה פעולות ואז נבנה ProcessManager ונקראת הפונקצייה Run, להלן הקוד של ProcessManager:

#define CHECK\_OS

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Diagnostics**;**

**using** System**.**IO**;**

**using** System**.**Linq**;**

**using** System**.**Net**.**NetworkInformation**;**

**using** System**.**Threading**;**

**namespace** Manager

**{**

class ProcessManager

**{**

#region constants

**private** const int COMMUNICATION\_PROCESS\_IND **=** 0**;**

**private** const int DECISIONS\_PROCESS\_IND **=** 1**;**

**private** const int DATABASE\_PROCESS\_IND **=** 2**;**

**private** const int MAIN\_PROCESSES\_NUM **=** 3**;**

**private** const char SEND\_CMD **=** 's'**;**

**private** const char START\_PROCESS\_CMD **=** 'p'**;**

**private** const char QUERY\_CMD **=** 'q'**;**

**private** const char PROCESS\_ENDED\_CODE **=** 'e'**;**

**public** const char PROCESS\_DATA\_CODE **=** 'd'**;**

**private** const char QUERY\_RESPONSE\_CODE **=** 'Q'**;**

**private** const char DISPLAY\_CODE **=** 'D'**;** // display to admin <=> only if the admin is connected

**private** **readonly** string databaseFile**;**//@"db\db.db";

#endregion

#region fields

**private** ProcessHider procHider**;**

**private** ProcessHandler**[]** mainProcesses**;**

**private** ProcessHandler controllerProcess**;**

**private** List**<**ProcessHandler**>** secondaryProcesses**;**

**private** Queue**<**string**>** Queries**;**

**private** bool shutdown**;**

**private** bool isAdminConnected**;**

**private** int mainThreadId**;**

#endregion

**public** ProcessManager**()**

**{**

procHider **=** **new** ProcessHider**();**

mainProcesses **=** **new** ProcessHandler**[**MAIN\_PROCESSES\_NUM**];**

secondaryProcesses **=** **new** List**<**ProcessHandler**>();**

Queries **=** **new** Queue**<**string**>();**

shutdown **=** **false;**

isAdminConnected **=** **false;**

mainThreadId **=** Thread**.**CurrentThread**.**ManagedThreadId**;**

Directory**.**CreateDirectory**(**"temp"**);** // for output files or executables i.e. python code or batch files

#if (CHECK\_OS)

**if** **(!**Environment**.**Is64BitOperatingSystem**)** // 32 bit system

#endif

procHider**.**HideProc**(**Process**.**GetCurrentProcess**());**

databaseFile **=** NetworkInterface**.**GetAllNetworkInterfaces**().**Where**(**nic **=>** nic**.**OperationalStatus **==** OperationalStatus**.**Up**)**

**.**Select**(**nic **=>** nic**.**GetPhysicalAddress**().**ToString**()).**Last**();**// to know the difference between different node's databases

**}**

/// <summary>

/// call to shutdown the manager

/// </summary>

**public** void Shutdown**()**

**{**

**this.**shutdown **=** **true;**

**lock** **(this.**mainProcesses**)** **lock** **(this.**secondaryProcesses**)**

**{**

**foreach** **(**var procH **in** **this.**secondaryProcesses**)** // kill all

procH**.**Kill**();**

**foreach** **(**var procH **in** **this.**mainProcesses**)**

procH**.**Kill**();**

**this.**mainProcesses **=** **new** ProcessHandler**[**MAIN\_PROCESSES\_NUM**];**

**this.**secondaryProcesses **=** **new** List**<**ProcessHandler**>();**

**}**

**}**

/// <summary>

/// manager main loop

/// </summary>

**public** void Run**()**

**{**

**for** **(**int i **=** 0**;** i **<** **this.**mainProcesses**.**Length**;** i**++)**

**this.**mainProcesses**[**i**]** **=** **new** ProcessHandler**(this.**procHider**);**

// python -u = python unbuffered -> disable the need for sys.stdout.flush() after every print (like doing it automatically)

**if** **(!this.**mainProcesses**[**COMMUNICATION\_PROCESS\_IND**].**StartProcess**(**"python"**,** @"-u Communication\communicator.py"**,** **this.**OnCommunicationReceived**)** **||**

**!this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**].**StartProcess**(**"python" **,** @"-u Decider\decider.py"**,** **this.**OnDecisionRecieved**)** **||**

**!this.**mainProcesses**[**DATABASE\_PROCESS\_IND**].**StartProcess**(**"python"**,** @"-u Database\database.py " **+** databaseFile**))**

Environment**.**Exit**(-**1**);** // could not start main processes

DatabaseThread**();** // starts the thread if it is in the main thread

**this.**Shutdown**();**

Thread**.**Sleep**(**500**);**

**}**

**private** void DatabaseThread**()**

**{**

// the next code is made to start this thread as a seperate thread

//if (Thread.CurrentThread.ManagedThreadId == mainThreadId) // not started as a seperate thread

//{

// Thread dbThread = new Thread(DatabaseThread);

// dbThread.Start();

// return;

//}

**while** **(!this.**shutdown**)**

**{**

// lock is not needed because Count returns a variable and not counts the queries

**while** **(!this.**shutdown **&&** **this.**Queries**.**Count **>** 0**)** // while not empty

**{**

bool IsBusy **=** **false;**

string data**;**

**lock** **(this.**Queries**)**

**{**

data **=** **this.**Queries**.**Dequeue**();**

**}**

**lock** **(this.**mainProcesses**[**DATABASE\_PROCESS\_IND**])**

**{**

string query **=** data**.**Substring**(**data**.**IndexOf**(**','**));**

string taskId **=** data**.**Substring**(**0**,** data**.**IndexOf**(**','**));**

DataReceivedEventHandler resultHandler **=** **null;**

resultHandler **=** **(**s**,** e**)** **=>**

**{**

**if** **(**String**.**IsNullOrEmpty**(**e**.**Data**))**

**return;**

**lock** **(this.**mainProcesses**[**DATABASE\_PROCESS\_IND**])** **lock** **(this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**])**

**{**

**this.**mainProcesses**[**DATABASE\_PROCESS\_IND**].**RemoveOutputHandler**(**resultHandler**);** // it's working (removing the ptr and not null)

**this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**].**SendData**(**QUERY\_RESPONSE\_CODE **+** taskId **+** "," **+** e**.**Data**);**

**}**

IsBusy **=** **false;**

**};**

IsBusy **=** **true;**

**this.**mainProcesses**[**DATABASE\_PROCESS\_IND**].**AddOutputHandler**(**resultHandler**);**

**this.**mainProcesses**[**DATABASE\_PROCESS\_IND**].**SendData**(**query**);**

int sleepCounter **=** 1**;**

**while** **(**IsBusy**)**

**{**

Thread**.**Sleep**(**sleepCounter**);**

sleepCounter **=** sleepCounter **\*** 2 **<** 10000 **?** sleepCounter **\*** 2 **:** 10000**;** // preventing it from reaching to too big numbers

**}**

**}**

**}**

Thread**.**Sleep**(**100**);**

**}**

**}**

**private** void OnCommunicationReceived**(object** sender**,** DataReceivedEventArgs e**)**

**{**

**lock** **(this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**])**

**this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**].**SendData**(**SEND\_CMD **+** e**.**Data**);**

**}**

**private** void OnDecisionRecieved**(object** sender**,** DataReceivedEventArgs e**)**

**{**

// execute command

**if** **(**String**.**IsNullOrEmpty**(**e**.**Data**))**

**{**

Console**.**WriteLine**(**"command was empty"**);** // temp

**return;**

**}**

string trimData **=** e**.**Data**.**TrimStart**();** // stdin/out might add some spaces in the start

**switch** **(**trimData**[**0**])**

**{**

**case** SEND\_CMD**:**

**lock** **(this.**mainProcesses**[**COMMUNICATION\_PROCESS\_IND**])**

**this.**mainProcesses**[**COMMUNICATION\_PROCESS\_IND**].**SendData**(**trimData**.**Substring**(**1**));** // pass data without command

// NOTICE: the data is a filename with the data

**break;**

**case** START\_PROCESS\_CMD**:** // the data should be: <command type char><proccess identification string>,

ProcessHandler newProc **=** **new** ProcessHandler**(this.**procHider**);**

**lock** **(this.**secondaryProcesses**)**

**this.**secondaryProcesses**.**Add**(**newProc**);**

**if(**newProc**.**StartProcess**(**trimData**.**Substring**(**1**),** **this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**]))** // remove command character)

**{**

EventHandler exitHandler **=** **(**s**,** e2**)** **=>**

**{**

**lock** **(this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**])**

**this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**].**SendData**(**ProcessManager**.**PROCESS\_ENDED\_CODE **+** trimData**.**Split**(**','**)[**0**].**Substring**(**1**));**

// equals to START\_PROCESS\_CMD + trimData.Split(',')[0].Substring(1)

**lock** **(this.**secondaryProcesses**)**

**this.**secondaryProcesses**.**Remove**(**newProc**);**

**};**

newProc**.**AddExitHandler**(**exitHandler**);**

**}**

**else** // start process failed

**{**

**lock** **(this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**])**

**this.**mainProcesses**[**DECISIONS\_PROCESS\_IND**].**SendData**(**ProcessManager**.**PROCESS\_ENDED\_CODE **+** trimData**.**Split**(**','**)[**0**].**Substring**(**1**));**

// equals to START\_PROCESS\_CMD + trimData.Split(',')[0].Substring(1)

**lock** **(this.**secondaryProcesses**)**

**this.**secondaryProcesses**.**Remove**(**newProc**);**

**}**

**break;**

**case** QUERY\_CMD**:**

**lock(**Queries**)**

**{**

Queries**.**Enqueue**(**trimData**.**Substring**(**1**));** // the queued commands will be executed in another thread

**}**

**break;**

**case** DISPLAY\_CODE**:**

**if** **(!this.**isAdminConnected**)**

**{**

**if** **(**trimData**[**1**]** **!=** '\'' **&&** trimData**[**1**]** **!=** '"'**)** // not repr'd => try to connect from another controller

**{**

isAdminConnected **=** **true;**

**this.**controllerProcess **=** **new** ProcessHandler**(this.**procHider**);**

**this.**controllerProcess**.**StartProcess**(**"python"**,** @"-u ControllerConnection\controllerConnection.py " **+** trimData**.**Substring**(**1**),** OnControllerRecieved**);**

**this.**controllerProcess**.**AddExitHandler**((**s**,** e2**)** **=>** **{** isAdminConnected **=** **false;** Console**.**WriteLine**(**"admin session ended"**);** **});**

**}**

//else drop the message

**}**

**else**

**{**

**if** **(**trimData**[**1**]** **!=** '\'' **&&** trimData**[**1**]** **!=** '"'**)** // not repr'd => try to connect from another controller

**{**

Thread t **=** **new** Thread**(()** **=>**

**{**

System**.**Windows**.**Forms**.**MessageBox**.**Show**(**"an admin is already connected\nonly one admin allowed per node!"**,** "Error"**,**

System**.**Windows**.**Forms**.**MessageBoxButtons**.**OK**,** System**.**Windows**.**Forms**.**MessageBoxIcon**.**Error**,**

System**.**Windows**.**Forms**.**MessageBoxDefaultButton**.**Button1**,** System**.**Windows**.**Forms**.**MessageBoxOptions**.**ServiceNotification**);**

**});**

t**.**Start**();**

**}**

**else**

**lock** **(this.**controllerProcess**)**

**this.**controllerProcess**.**SendData**(**trimData**.**Substring**(**1**));**

**}**

**break;**

**default:**

Console**.**WriteLine**(**"unknown cmd data: " **+** e**.**Data **+** "\n\n trimmed: " **+** trimData**);**

StreamWriter lsw **=** **new** StreamWriter**(**"unknown cmd.hex"**);**

lsw**.**Write**(**e**.**Data **+** "\n" **+** trimData**);**

lsw**.**Close**();**

**throw** **new** InvalidOperationException**(**"Unknown command: " **+** e**.**Data **+** "\n\n trimmed: " **+** trimData**);**

**}**

**}**

**private** void OnControllerRecieved**(object** sender**,** DataReceivedEventArgs e**)**

**{**

**if** **(!**String**.**IsNullOrEmpty**(**e**.**Data**))**

**lock** **(this.**mainProcesses**[**COMMUNICATION\_PROCESS\_IND**])**

**this.**mainProcesses**[**COMMUNICATION\_PROCESS\_IND**].**SendData**(**e**.**Data**);** // pass data to send

**}**

**private** void KillSecondaryProcess**(**ProcessHandler p**)**

**{**

**lock(this.**secondaryProcesses**)**

**this.**secondaryProcesses**.**Remove**(**p**);**

p**.**Kill**();**

**}**

**}**

**}**

## Controller

זהו כאמור החלק שמתחבר ל- Node שרץ על המחשב הנוכחי, כדי לשלוח משימות ושאילתות לשאר ה- Node-ים.

לממשק המשתמש ישנן שתי אפשרויות: ממשק גרפי וממשק טקסטואלי. בכדי להפעיל את הממשק הגרפי יש להריץ את הקובץ controller.py וכדי להריץ את הממשק הטקסטואלי יש להריץ את הקובץ bash.py.

לאחר שאחד מהקבצים הורץ, הוא שולח הודעה ל- Node שעל המחשב באיזה פורט לפתוח שרת להתחברות וכך ניתן להתחבר ל- Node. בקובץ config.cfg כתוב הפורט שבו מקשיב ה- Node (מוצפן) וכך ניתן לשלוח את ההודעה הראשונית.

לאחר שה- Node פותח שרת TCP בפורט המבוקש, ה- Controller מתחבר לשרת וניתן להריץ פקודות.

המחלקה TasksManager אחראית על ניהול המשימות והיא נעזרת ב- Task וב- TaskType (יש לשים לב שאלו אינן אותן מחלקות של Decider).

## להוסיף קודים לאחר בדיקה סופית

## DirectoryServer

זהו השרת שמשתף את כתובות ה- Node-ים הפעילים ביניהם. כתובת השרת הזה יכולה להשתנות ואפילו יכולים להיות מספר שרתים כאלה (כל שרת כזה יוצר רשת אחרת של Node-ים) ולכן צריכה להיות דרך לגלות את הכתובת של השרת(ים) כל פעם מחדש, ולכן בעת הרצת הקובץ directoryServer.py, מועלה הכתובת הנוכחית של השרת לדף אינטרנט בכתובת: <http://dirser.atwebpages.com/dirSer/status.php>

כך כל Node יכול לגשת לדף ולתקשר עם השרת.

כדי שהשרת ידע איזה Node-ים פעילים, כל Node שולח הודעה לשרת פעם בכמה שניות, השרת מקבל את ההודעה ויודע כי ה- Node פעיל וכאשר לאחר זמן מסויים (קבוע מראש) שבו השרת לא מקבל אף הודעה מ- Node מסויים, הוא מסמן שהוא לא פעיל יותר.

עם התחלת השרת נקראת הפונקצייה start, להלן קוד השרת:

**class** **Server:**

**def** \_\_init\_\_**(**self**):**

self**.**port **=** 13013

self**.**s **=** socket**(**AF\_INET**,** SOCK\_DGRAM**)**

self**.**s**.**bind**((**"0.0.0.0"**,** self**.**port**))**

self**.**isShutdown **=** **False**

self**.**clients **=** **[]** # (ID, address)

self**.**clientsLastCommunication **=** **{}** # class

self**.**MAX\_NODES\_NUM\_TO\_SEND **=** 20 # to fit to one packet

self**.**CLIENT\_TIMEOUT **=** 3 **\*** 4

self**.**SERVER **=** "dirser.atwebpages.com"

self**.**EOM **=** "<EOF>" # end of message

**def** uploadAddr**(**self**):**

# get current status

upToDateStatus **=** urllib**.**urlopen**(**"http://" **+** self**.**SERVER **+** "/dirSer/status.php"**).**read**().**replace**(**"\r"**,** ""**)** # turn \r\n to \n

# edit status file

f **=** open**(**"upload/status.php"**,** "w"**)**

f**.**write**(**upToDateStatus **+** "\n" **+** self**.**getMyIp**()** **+** "," **+** str**(**self**.**port**))**

f**.**close**()**

# then upload status

**if** **not** upload**(True):** #True): # upload and not verbose

**for** i **in** range**(**2**):** # try 2 more times

**if** upload**(True):** # break

**break**

**if** i **==** 1**:** # third try failed

**return** **False** #raise Exception("upload failed")

**return** **True**

**def** start**(**self**):**

**if** **not** self**.**uploadAddr**():** # upload the address

**return** **False** # fail

start\_new\_thread**(**self**.**run**,** **())** # then start the server

**return** **True** # success

**def** run**(**self**):**

start\_new\_thread**(**self**.**checkConnectionThread**,** **())**

#start\_new\_thread(self.recvThread, ())

self**.**recvThread**()**

**def** recvThread**(**self**):**

**while** **not** self**.**isShutdown**:**

data**,** addr **=** self**.**s**.**recvfrom**(**20**)** # data = [>]ID

**print** str**(**addr**)** **+** ": " **+** data

**if** data**[**0**]** **==** ">"**:** # request for nodes list

# can be also regular notification (usually hole punching) -> that's what the if is for

contacts **=** self**.**getContacts**(**data**[**1**:])**

**print** "sending contacts: " **+**str**(**contacts**)** **+** " to: " **+** data**[**1**:]**

contacts **=** **[**contacts**[**i**:**i **+** 10**]** **for** i **in** xrange**(**0**,** len**(**contacts**),** 10**)]** # split to force it to send each list in the same packet

**for** i **in** xrange**(**len**(**contacts**)):**

self**.**s**.**sendto**(**"0," **+** str**(**i**)** **+** ",m," **+** repr**(**contacts**[**i**]),** addr**)** # directory servers ID's are all 0

# regular notification (usually hole punching)

data **=** data**.**replace**(**">"**,** ""**)**

**if** **(**data**,** addr**)** **not** **in** self**.**clients**:**

self**.**clients**.**append**((**data**,** addr**))** # ID, ADDR

self**.**clientsLastCommunication**[(**data**,** addr**)]** **=** time**()**

**def** checkConnectionThread**(**self**):**

**while** **not** self**.**isShutdown**:**

i**=**0

**while** i **<** len**(**self**.**clients**):**

**if** time**()** **-** self**.**clientsLastCommunication**[**self**.**clients**[**i**]]** **>** self**.**CLIENT\_TIMEOUT**:** # remove if connection timed out

**del** self**.**clientsLastCommunication**[**self**.**clients**[**i**]]**

self**.**clients**.**remove**(**self**.**clients**[**i**])**

**else:**

i **+=** 1

sleep**(**0.2**)**

**def** getMyIp**(**self**):** # external ip

checkIpSock **=** socket**(**AF\_INET**,** SOCK\_DGRAM**)**

checkIpSock**.**connect**((**'8.8.8.8'**,** 0**))** # connecting to a UDP address doesn't send packets

**return** checkIpSock**.**getsockname**()[**0**]**

**def** getContacts**(**self**,** ID**):**

# send list of ID's and addresses of nodes that this node[ID] can send to

**return** **[**c **for** c **in** self**.**clients **if** c**[**0**]** **!=** ID**]**

**def** shutdown**(**self**):**

self**.**isShutdown **=** **True**

upToDateStatus **=** urllib**.**urlopen**(**"http://" **+** self**.**SERVER **+** "/dirSer/status.php"**).**read**()**

upToDateStatus **=** upToDateStatus**.**replace**(**"\n" **+** self**.**getMyIp**()** **+** "," **+** str**(**self**.**port**),** ""**)** # remove this addr

f **=** open**(**"upload/status.php"**,** "w"**)**

f**.**write**(**upToDateStatus**)**

f**.**close**()**

**if** **not** upload**(True):** # upload and not verbose

**for** i **in** range**(**2**):** # try 2 more times

**if** upload**(True):** # break

**break**

**if** i **==** 1**:**

**raise** Exception**(**"upload failed"**)**

sleep**(**0.5**)**

# דוגמאות הרצה

# הצעות לשיפור

# סיכום