Aplikacija za nadzor računalnih procesa i zauzeće računalnih procesa

Pečurlić, Dino

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:635079

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

C# APLIKACIJA ZA NADZOR RAČUNALNIH PROCESA I ZAUZEĆE RAČUNALNIH PROCESA

Završni rad

Dino Pečurlić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek,05.09.2017.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Dino Pečurlić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3685, 23.07.2014.
OIB studenta:	06445308140
Mentor:	Doc.dr.sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za nadzor računalnih procesa i zauzeće računalnih procesa
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	05.09.2017.
Datum potvrde ocjene Odbora:	11.09.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTIRADA

Osijek, 12.09.2017.

Ime i prezime studenta:	Dino Pečurlić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3685, 23.07.2014.
Ephorus podudaranje [%]:	1%

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za nadzor računalnih procesa i zauzeće računalnih procesa**

izrađen pod vodstvom mentora Doc.dr.sc. Ivica Lukić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD 1

1.1 Zadatak završnog rada 1

2. PROCESI 2

2.1 Definicija procesa 2

2.2 Pojam niti,skupina niti,vlakna,poslova i *user-mode* planiranja 2

2.2.1 Nit 2

2.2.2 Posao 2

2.2.3 Skupine niti 2

2.2.4 *User-mode* planiranje 2

2.2.5 Vlakna 3

3. BIBLIOTEKE KORIŠTENE ZA REALIZACIJU APLIKACIJE 4

3.1 Biblioteke korištene za manipulaciju procesima 4

3.1.1 Klasa 'Process' 4

3.1.2 iDisposable sučelje 4

3.1.3 PerformanceCounter klasa 4

3.1.4 Brojači performasi procesora 5

3.1.5 Memory Performance objekt 5

3.1.6 FileVersionInfo klasa 5

3.1.7 System.ComponentModel biblioteka 5

3.2 Ostale biblioteke korištene za realizaciju aplikacije 6

3.2.1 Microsoft.VisualBasic.Devices biblioteka 6

3.2.2 ComputerInfo klasa 6

3.2.3 Network klasa 6

3.2.4 System.Net i System.Net.NetworkInformation biblioteka 6

3.2.5 SharpPcap biblioteka 7

3.2.6 System.Threading biblioteka 7

3.2.7 System.Text.RegularExpressions biblioteka 7
3.2.8 System.Collections biblioteka 7
3.2.9 System.Drawing biblioteka 8
3.2.10 System.Linq i System.Management biblioteke 8
3.2.11 System.IO biblioteka 8
4. PROBLEMI PRILIKOM UPRAVLJANJA PROCESIMA 9
4.1 Korištenje TotalProcessorTime alata 9
4.2 Korištenje WorkingSet64 alata 9
4.3 Kašnjenje u odgovoru aplikacija 10
5. PRIMJENA U PROGRAMSKOM JEZIKU C# 11
5.1 Zadatak programa 11
5.2 Rad programa 12
5.3 Opis koda programa 22
5.3.1 Metode i okidači 22
5.3.2 Prikupljanje informacija o hardveru 23
5.3.3 Osvježavanje liste procesa i mrežnih adaptera 24
5.3.4 Osvježavanje liste procesa koji stvaraju mrežni promet 27
5.3.4 Ažuriranje informacija o odabranom procesu 36
5.3.5 Crtanje grafova ukupnog zauzeća ram memorije i vremena procesora 40
5.3.6 Crtanje grafova zauzeća ram memorije i vremena procesora odabranog procesa 42
5.3.7 Okidači promjene odabrane vrijednosti padajućih izbornika 45
5.3.8 Okidači postavljeni na prostor za pretragu 48
6. ZAKLJUČAK 50
LITERATURA 51
SAŽETAK 53
ŽIVOTOPIS 55
PRILOZI 56

1. UVOD

Tema ovog završnog rada je C# aplikacija za nadzor računalnih procesa i zauzeće računalnih procesa. Aplikacija treba prikazati sve pokrenute procese, te računalne resurse koje zauzima pojedini proces. Također, za svaki proces je potrebno pružiti kratak opis, te prikazati mrežni promet po procesu.

Glavni dio rada je podijeljen na četiri dijela: *Procesi, biblioteke korištene za realizaciju aplikacije, problemi prilikom rada s procesima i primjena u programskom jeziku c#.*

U poglavlju *Procesi* su objašnjene glavne razlike između procesa, niti, bazena niti i vlakana, te je objašnjen pojam *User-mode* planiranja. Također su objašnjena glavna svojstva procesa, biblioteke koje je potrebno uključiti za obavljanje određenog rada s procesima, te opisi korištenih alata, funkcija, metoda i slično.

U poglavlju *Biblioteke korištene za realizaciju aplikacije* su navedene i objašnjene biblioteke, te razne metode i alati korišteni za dobivanje potrebnih informacija i obavljanja željenih operacija s istima.

U poglavlju *Problemi prilikom upravljanja procesima* su objašnjeni problemi koji se pojavljuju prilikom korištenja funkcija i metoda iz korištenih biblioteka, opisani su nedostatci određenih metoda kao i razlike u rezultatima koje pokazuju aplikacije operacijskog sustava Windows i ove aplikacije.

U poglavlju *Primjena u programskom jeziku C#* se prikazuje kod koji omogućava željene rezultate, tj. omogućava izvršavanje radnji nad procesima.

1.1 Zadatak završnog rada

Zadatak ovog završnog rada je napraviti C# aplikaciju gdje se korisniku pruža uvid u pokrenute procese, njihovo zauzeće resursa, te opis procesa kako bi se korisniku olakšalo upravljanje istima. Aplikacija također daje osnove informacije o računalu, te mrežnim adapterima i njihovom prometu.

2. PROCESI

2.1 Definicija procesa

Aplikacija se sastoji od jednog ili više procesa. Proces, najjednostavnije rečeno, je program u izvođenju. Svaki proces osigurava resurse koji su potrebni za pokretanje određenog programa. Proces se sastoji od više dijelova; virtualne adrese, koda, dijela koji omogućuje upravljanje sa sustavskim objektima, jedinstvenog identifikacijskog broja, varijabli okoline, dijela koji osigurava sigurno izvođenje programa, razine prioriteta, minimalnih i maksimalnih resursa koji su potrebni za rad procesa, te barem jedne niti. Svaki proces se pokreće sa niti koja se naziva primarna nit koja omogućuje stvaranje dodatnih niti.

2.2 Pojam niti,skupina niti,vlakna,poslova i *user-mode* planiranja

2.2.1 Nit

Nit je dio procesa koja se planirano izvršava. Sve niti jednog procesa dijele sve računalne resurse s procesom. Svaka nit održava upravitelje iznimkama, memorijski prostor koji koristi, jedinstvenu identifikaciju niti, te se brine o prioritetu prilikom izvođenja i o strukturama koje koristi operacijski sustav. Navedene strukture operacijski sustav koristi kako bi spremio stog jezgre(*kernel*-a), registre računala, informacijski blok niti i korisnički stog, što se sve zajedno naziva kontekstom niti, u adresni prostor procesa kojem nit pripada.

2.2.2 Posao

Posao je grupa procesa koja kao takva omogućuje operacijskom sustavu da njome upravlja kao cjelinom. Poslovi (ili objekti posla) su cjeline koje se mogu imenovati, osiguravati i međusobno komunicirati. Upravljanjem poslovima operacijski sustav utječe na sve procese koji su dio posla, ili su u nekakvom odnosu s poslom.

2.2.3 Skupine niti

Skupine niti se koriste kako bi se smanjio ukupan broj niti i kako bi se omogućilo upravljanje radnim nitima(niti koje su stvorene i 'čekaju' na posao) koje obavljaju brojne zadatke u pozadini aplikacije. Na taj način primarna nit se oslobađa posla kako bi mogla nastaviti s radom.

2.2.4 *User-mode* planiranje

User-mode planiranje je mehanizam koji aplikacije koriste kako bi upravljale, tj. planirale izvršavanje svojih niti bez da moraju za to koristiti raspoređivač operacijskog sustava. Glavna

prednost takvog planiranja je ta što aplikacija ne mora koristiti raspoređivač operacijskog sustava čak i ako dođe do zastoja prilikom izvršavanja niti. Mogućnost prebacivanja između niti na korisničkoj razini predstavlja veliku prednost ako se radi o kraćim poslovima s malo sustavskih poziva.

2.2.5 Vlakna

Vlakna su jedinice izvršavanja kojima upravlja aplikacija, tj. aplikacija planira njihovo izvršavanje. Svaka nit može planirati izvršavanje više vlakana, a uglavnom korištenje vlakana ne daje nikakvu prednost nad dobro napravljenom višenitnom aplikacijom.

3. BIBLIOTEKE KORIŠTENE ZA REALIZACIJU APLIKACIJE

3.1 Biblioteke korištene za manipulaciju procesima

Elementi biblioteke System.Diagnostics su najčešće korišteni prilikom izvedbe ove aplikacije. Ta biblioteka sadrži tipove koji omogućuju interakciju sa procesima, bilješkama događaja i brojačima performansi(*performance counters*). Ostale biblioteke koje su izvedene iz biblioteke System.Diagnostics također imaju tipove koji im omogućuju pristup alatima za analizu koda, alatima za praćenje događaja i praćenje izvještaja o događajima za Windows sustav praćenja(ETW), te imaju mogućnost pisanja i čitanja izvještaja o događajima kao i mogućnost prikupljanja podataka o performansama određenih dijelova računala. Biblioteka '*System.ComponentModel*' korištena je za hvatanje iznimki prilikom upravljanja procesima.

3.1.1 Klasa 'Process'

Klasa '*Process'* omogućuje pristup lokalnim i udaljenim procesima, te omogućuje radnje s njima kao što su: pokretanje, ponovno pokretanje, zaustavljanje itd. Pomoću te klase moguće je stvoriti popis svih procesa, uključujući i procese operacijskog sustava. Nakon realizacije klase, tj. nakon stvaranja objekta klase '*Process'* moguće je dobiti razne informacije o procesu kao što su: zauzeće memorije, identifikacija procesa, broj niti vezanih za taj proces, učitane module itd. Na objekte klase '*Process'* moguće je utjecati opcijom 'odbaci' sučelja '*iDisposble*'.

3.1.2 iDisposable sučelje

Namjena ovog sučelja je oslobađanje resursa tako što oslobađa memoriju zauzetu od strane objekta koji se više ne koristi. Ukoliko se neki objekt više ne koristi, čistač otpada automatski oslobađa memoriju alociranu za taj objekt. Kako nije moguće predvidjeti kada će se točno dogoditi oslobađanje memorije poželjno je koristiti metodu '*Dispose()'* kako aplikacija ne bi bespotrebno zauzimala prostor.

3.1.3 PerformanceCounter klasa

Brojači performansi se koriste kako bi se dobile razne informacije o računalu kao što su npr. performanse neke aplikacije, operacijskog sustava ili upravljačkog programa. Takvi brojači su vrlo bitni jer omogućuju analizu rada željenog programa te s time i poboljšanje rada istog. Operacijski sustav i različiti uređaji računala održavaju podatke koje je moguće iskoristiti pomoći brojača performansi kako bi se pružio jednostavniji, često i grafički, uvid u performanse sustava. Aplikacije mogu koristiti podatke brojača kako bi odlučile koliko je memorije potrebno

alocirati, kada i koliko opteretiti internetsku konekciju, tj. pomoću tih podataka može se odlučiti koliko resursa je potrebno za izvršavanje određenog procesa.

3.1.4 Brojači performasi procesora

U ovoj skupini brojača nalaze se brojači sustava, procesora, procesa i niti. Svi ti brojači daju informacije o poslu koji obavlja procesor. Brojači iz ove skupine koji su se koristili u izradi aplikacije ovog rada su brojač vremena procesora objekta '*Processor'* i broj vremena procesora objekta '*Process'*. Brojač '*% Processor Time*' iz objekta '*Processor'* kao povratnu vrijednost daje postotak vremena tijekom kojeg je procesor bio zauzet izvršavajući instrukcije u nekom vremenskom intervalu. Taj brojač se koristi u programu Task Manager sustava Windows. Brojač '% Processor Time' iz objekta kao povratnu vrijednost daje postotak vremena koji je procesor uložio za izvršavanje danog procesa.

3.1.5 Memory Performance objekt

'*Memory Performance*' objekt sastoji se od brojača performansi koji opisuju ponašanje memorije, tj. opisuju promjene u zauzeću fizičke(instalirane) i virtualne memorije računala. Fizičku memoriju predstavlja količina RAM memorije dok virtualnu memoriju predstavlja kombinacija fizičke memorije i memorije na tvrdom tisku. Brojači performansi memorije nadgledaju i proces straničenja(proces premještanja straničnih datoteka, tj. *page file-a*,između tvrdog diska i RAM memorije). U kodu ovog programa korišten je brojač performansi imena '*% Committed Bytes In Use*'. Taj brojač pokazuje odnos memorije, dodijeljenih bajta memoriji i granice dodjeljivanja. Dodijeljena memorija predstavlja količinu RAM memorije u upotrebi za koju postoji rezerviran prostor u straničnoj datoteci(eng. *page file*), a granica dodjeljivanja(eng. *commit limit*) je određena veličinom stranične datoteke.

3.1.6 FileVersionInfo klasa

Klasa čiji objekti daju informacije o datotekama na tvrdom disku. Pomoću brojnih metoda mogu se dobiti razne informacije o odabranoj datoteci, tako npr. metodom '*GetVersionInfo*' se dobija informacija o verziji tražene datoteke. Ta metoda je korištena kako bi se dobio opis određenog procesa.

3.1.7 System.ComponentModel biblioteka

Navedena biblioteka sadrži klase koje služe za implementiranje ponašanja kontrola i komponenti prilikom dizajniranja i pokretanja aplikacije. Za ovu aplikaciju važna je klasa '*Win32Exception*' koja se koristi za hvatanje iznimki prilikom upravljanja procesima.

3.2 Ostale biblioteke korištene za realizaciju aplikacije

Biblioteke koje su korištene za dobivanje raznih informacija važnih za aplikaciju su '*Microsoft.VisualBasic.Devices*', *'System.Net*' i '*SharpPcap*' . Biblioteka '*Microsoft.VisualBasic.Devices*' je korištenja radi dobivanja i spremanja osnovnih informacija o računalnu, dok je 'System.Net.NetworkInformation' biblioteka korištena radi dobivanja informacija vezanih uz mrežu i način na koje je računalno spojeno na istu. Biblioteka '*SharpPcap*' omogućava praćenje i analiziranje mrežnog prometa računala. Ostale biblioteke koje su korištene pri izradi aplikacije su: '*System.Threading*', '*System.Text.RegularExpressions*', '*System.Collections'*, *'System.Drawing*', *'System.Linq*', *'System.Management*' i *'System.IO*'.

3.2.1 Microsoft.VisualBasic.Devices biblioteka

Navedena biblioteka ima 10 klasa od kojih realizacija svake može dati različite informacije o računalu. Klase koje se nalaze u biblioteci su: Audio, Clock, Computer, ComputerInfo, Keyboard, Mouse, Network, NetworkAvailableEventARg, Ports i ServerComputer. Za potrebe ove aplikacije korištena je klasa '*ComputerInfo'*.

3.2.2 ComputerInfo klasa

Navedena klasa sadrži pribor(eng. '*properties'*) koji daje informacije o memoriji i imenu računala, učitanim sklopovima i operacijskom sustavu. Ova klasa je važna jer je pomoću nje moguće dobiti informaciju o količini virtualne i fizičke memorije pa to omogućava računanje postotka zauzeća memorije pojedinog procesa i slično.

3.2.3 Network klasa

Navedena klasa sadrži pribor, događaje i razne metode koje omogućuju interakciju s mrežom ukoliko je računalno spojeno na istu. Metode ove klase omogućuju slanje ili skidanje podataka sa željene lokacije što je važno jer omogućuje skidanje ažuriranja o npr. opisu procesa ili slanje podataka o stabilnosti aplikacije.

3.2.4 System.Net i System.Net.NetworkInformation biblioteka

'*System.Net.NetworkInformation'* biblioteka omogućava pristup podatcima o podatkovnom, tj. mrežnom prometu i informacijama o mrežnim adresama i promjenama. Sadrži klase kao što je klasa '*Ping*' koja daje informaciju o dostupnosti računala na mreži. Klase i njihove metode su korištene za uvid u podatke kao što su imena i adrese mrežnih adaptera, njihova dostupnost, te mrežni promet po protokolima ICMPv4 i ICMPv6.

'*System.Net*' biblioteka sadrži klase koje pružaju jednostavno sučelje za razne internet protokole, za pristup i osvježavanje postavki unutar same biblioteke, za sastavljanje i slanje e-pošte, za mrežnu komunikaciju između klijenata itd.Ova biblioteka, uz '*System.Net.Sockets*' biblioteku,u aplikaciji se koristi za određivanje IP adresa svih adaptera, te za određivanje IP adrese adaptera preko kojeg se odvija mrežni promet.

3.2.5 SharpPcap biblioteka

'*SharpPcap*' biblioteka nije dio popisa biblioteka koje se dobivaju prilikom instalacije programa Visual Studio. Ova biblioteka omogućuje, na jednostavan način, snimanje, ubrizgavanje i analizu mrežnih paketa za .NET aplikacije, neovisno o platformi. Korištenje '*SharpPcap*' biblioteke zasniva se na '*WinPcap'* upravljačkom programu pa ga je potrebno imati instaliranog prije korištenja aplikacije. '*SharpPcap*' biblioteka se koristi zato što se na jednostavan način može postaviti filter za filtriranje mrežnih podataka, što omogućava hvatanje i analizu paketa poslanih ili primljenih od strane određenog procesa. '*SharpPcap'* biblioteka, između ostalog, omogućuje hvatanje paketa i na brzinama većim od 3 MB/s, omogućuje hvatanje paketa na niskoj razini(eng, *Low-level package capturing*) na određenom mrežnom adapteru, te upravljanje datotekama paketa, tj. pisanje ili čitanje.

3.2.6 System.Threading biblioteka

Navedena biblioteka sadrži elemente koji omogućuju višenitno programiranje. Kako bi se omogućila sinkronizirana aktivnost niti i sinkroniziran pristup podatcima ova biblioteka sadrži '*ThreadPool*' i '*Timer*' klasu. Elementi ove biblioteke su se koristili u izradi aplikacije kako bi 'oslobodila' glavna nit čime se optimizira rad aplikacije, i najvažnije, višenitno programiranje omogućuje praćenje mrežnog prometa više procesa odjednom.

3.2.7 System.Text.RegularExpressions biblioteka

Navedena biblioteka sadrži klase koje omogućuju pronalazak traženog niza znakova u određenom tekstu. *Regular Expression* izrazi služe kao predložak koji se koristi za traženje. Za izradu ove aplikacije takvi izrazi su se koristili za oblikovanje izlaza naredbe netstat u programu Command Prompt. Naredba netstat daje informacije o procesima koji sudjeluju u mrežnom prometu, pa se pomoću nje i regularnih izraza dobivaju informacije o adapteru i popisu portova koji pojedini proces koristi prilikom komunikacije.

3.2.8 System.Collections biblioteka

Ova biblioteka sadrži klase koje definiraju različite kolekcije objekata kao što su liste, rječnici, nizovi , redovi i hash tablice. Podbiblioteke koje su se koristile u aplikaciji su

'*System.Collections.Generic*' i '*System.Collections.Concurrent*'. '*System.Collections.Generic*' biblioteka sadrži klase i sučelja koja definiraju kolekcije koje omogućavaju korisnicima da stvaraju kolekcije čiji su članovi istog tipa. '*System.Collections.Concurrent*' biblioteka omogućuje stvaranje kolekcija koje omogućuju pristup svojim članovima iz više niti.

3.2.9 System.Drawing biblioteka

Navedena biblioteka omogućuje crtanje gdje se klasa '*Pen*' koristi za crtanje linija i raznih oblika, a klase koje su izvedene iz apstraktne klase '*Brush*' omogućuju ispunjavanje nacrtanih oblika. Za crtanje grafova u aplikaciji je korištena biblioteka '*System.Drawing.Drawing2D*' koja omogućuje dvodimenzionalno crtanje.

3.2.10 System.Linq i System.Management biblioteke

'*System.Linq*' biblioteka pruža klase i sučelja koja omogućuju korištenje upita, a '*System.Management*' biblioteka daje mogućnost pristupa velikom broju upravljačkih informacija o sustavu, uređajima i aplikacijama koje su spremljene u '*ManagementClass*' klasama. Različite aplikacije mogu slati upite, koje omogućuje '*Systen.Linq*' biblioteka, kako bi dobile željene informacije, kao što su npr. količina slobodne memorije na disku, trenutna brzina procesora itd.

3.2.11 System.IO biblioteka

Navedena biblioteka sadrži tipove koji omogućavaju pisanje i čitanje podataka iz datoteka, te upravljanje datotekama i direktorijima. Ova biblioteka je korištena u aplikaciji kako bi se iz tekstualnih datoteka pročitao kratki opis određenog procesa.

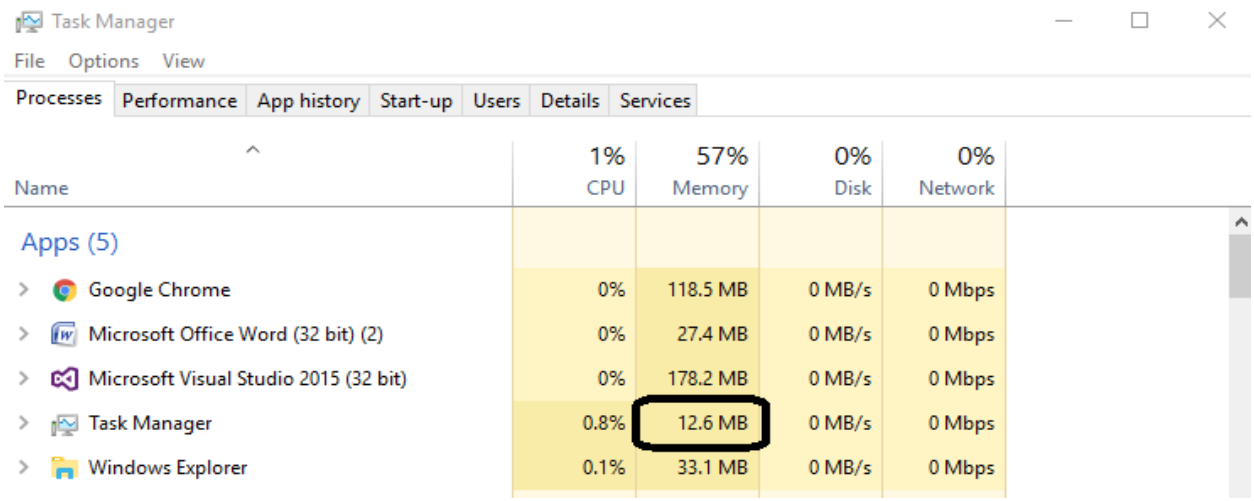
4. PROBLEMI PRILIKOM UPRAVLJANJA PROCESIMA

4.1 Korištenje TotalProcessorTime alata

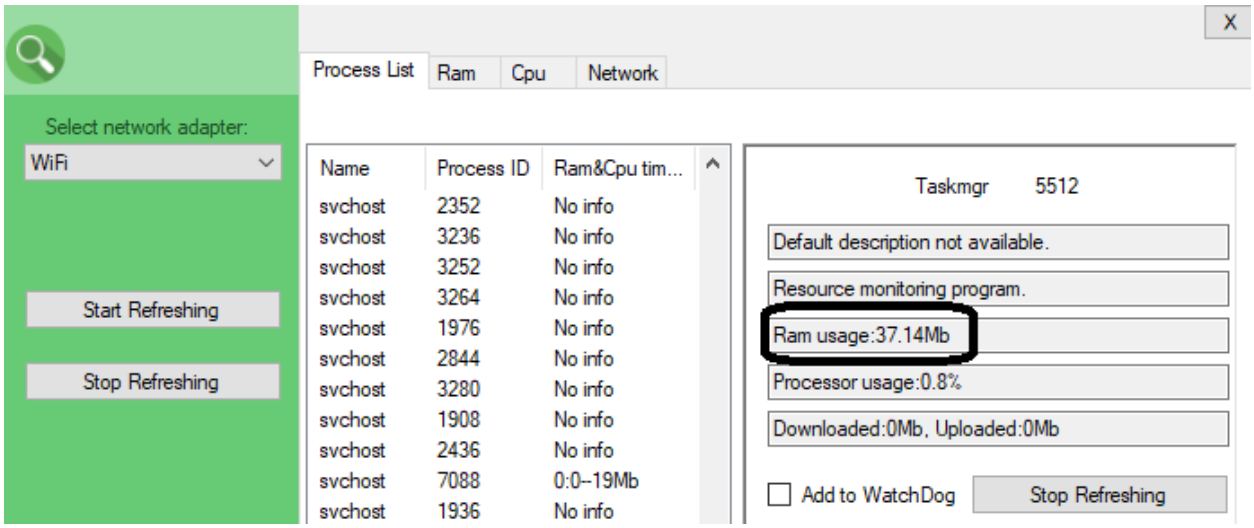
TotalProcessorTime je alat biblioteke System.Diagnostics koji za rezultat vraća ukupno vrijeme koje je procesor utrošio za obradu određenog procesa. Alat funkcionira ukoliko se radi o procesima koji nisu procesi sustava, tj. ako se radi o korisničkim procesima. Pri pokušaju korištenja ovog alata na sustavskom procesu, isto kao i korištenje '*MainModule.FileName*', za rezultat vraća iznimku jer operacijski sustav ne dozvoljava upravljanje takvim procesom čak ni administratoru računala. Za računanje vremena koji je procesor utrošio na bilo koji proces moguće je koristiti brojač performansi.

4.2 Korištenje WorkingSet64 alata

WorkingSet64 je alat klase '*Process*' koji daje informaciju o količini alocirane memorije za određeni proces, u bajtima. Rezultati korištenja alata WorkingSet64 razlikuju se od rezultata programa Task Manager za isti proces.



Sl. 4.1 Prikaz zauzeća ram memorije procesa Task Manager u istoimenom programu



Sl. 4.2 Prika zauzeća ram memorije procesa Task Manager pomoću izrađene aplikacije

Za shvaćanje problema potrebno je razlikovati radni set, virtualne i privatne bajte. Radni set je ukupna memorija alocirana za neki proces. Sve memorijske 'stranice' koje je koristio proces ubrajaju se kao radni set procesa što znači da proces ne mora koristi svu memoriju koja je za njega alocirana. Ukoliko se dogodi da u računalnu količina slobodne memorije padne ispod dozvoljene razine operacijski sustav na to reagira oduzimanjem memorijskih stranica koje proces ne koristi. Virtualni bajti predstavljaju trenutnu količinu virtualnog prostora kojeg proces koristi. Privatni bajti predstavljaju trenutnu količinu memorije koja je alocirana za određeni proces, te ne može biti dijeljena s drugim procesima.

4.3 Kašnjenje u odgovoru aplikacija

Kako bi aplikacija davala točne rezultate potrebno je ažurirati popis procesa te njihovo zauzeće resursa u pravilnim vremenskim razmacima. Pravilan rad takve aplikacije nije moguće ostvariti korištenjem samo primarne niti jer se za ažuriranje podataka u pravilnim vremenskim razmacima koriste brojači vremena, a to znači da se određene metode stalno pozivaju što uzrokuje zastajkivanje, odnosno sporu reakciju aplikacije. Takav problem se rješava korištenjem biblioteke '*System.Threading'* koja omogućava stvaranje dodatnih niti koje u pozadini izvršavaju željene naredbe bez blokiranja primarne niti.

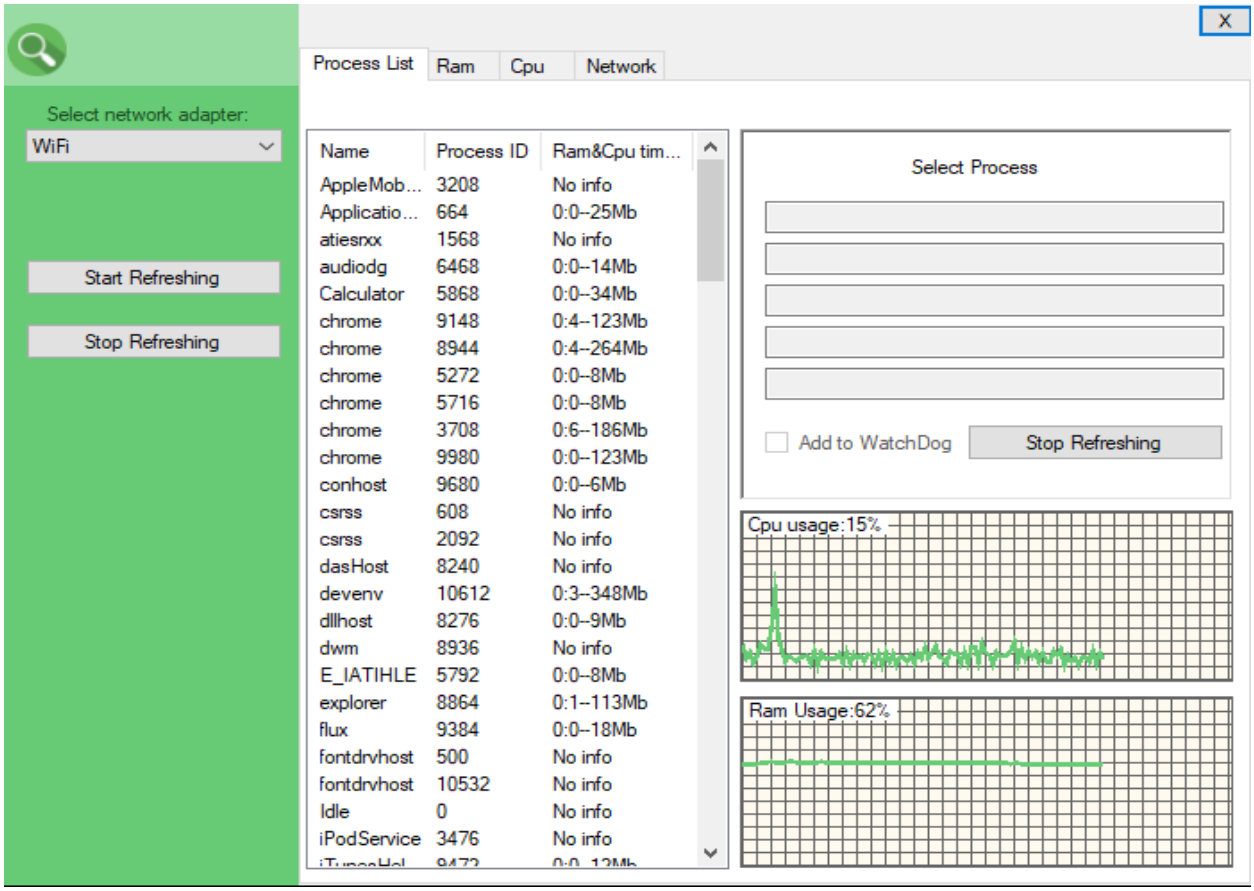
5. PRIMJENA U PROGRAMSKOM JEZIKU C#

5.1 Zadatak programa

Zadatak programa je da pri pokretanju prikaže popis pokrenutih procesa, da pruži osnovne informacije o njima, kao što su zauzeće ram memorije i korištenje vremena procesora, te da osigura točnost informacija, tj. da se informacije ažuriraju u pravilnim vremenskim razmacima. Program prilikom pokretanja mora u padajućem izborniku automatski odabrati adapter preko kojeg se odvija mrežni promet, te u kartici 'Network' mora ispisati listu procesa koji stvaraju mrežni promet, te za svaki proces ispisati brzinu preuzimanja i slanja podataka. Također u kartici 'Network' mora ispisati i sve mrežne adaptere kao i informacije o njima. Nakon odabira procesa s popisa moguće je vidjeli detaljnije informacije o procesu, kao što su kratki opis procesa, zauzeće ram memorije i vremena procesora, te količina poslanih i skinutih podataka. Stavljanjem kvačice na 'Add to WatchDog' omogućava se pristup karticama 'Ram' i 'Cpu' gdje se nalaze informacije o komponentama računala, kao i o odabranom procesu.

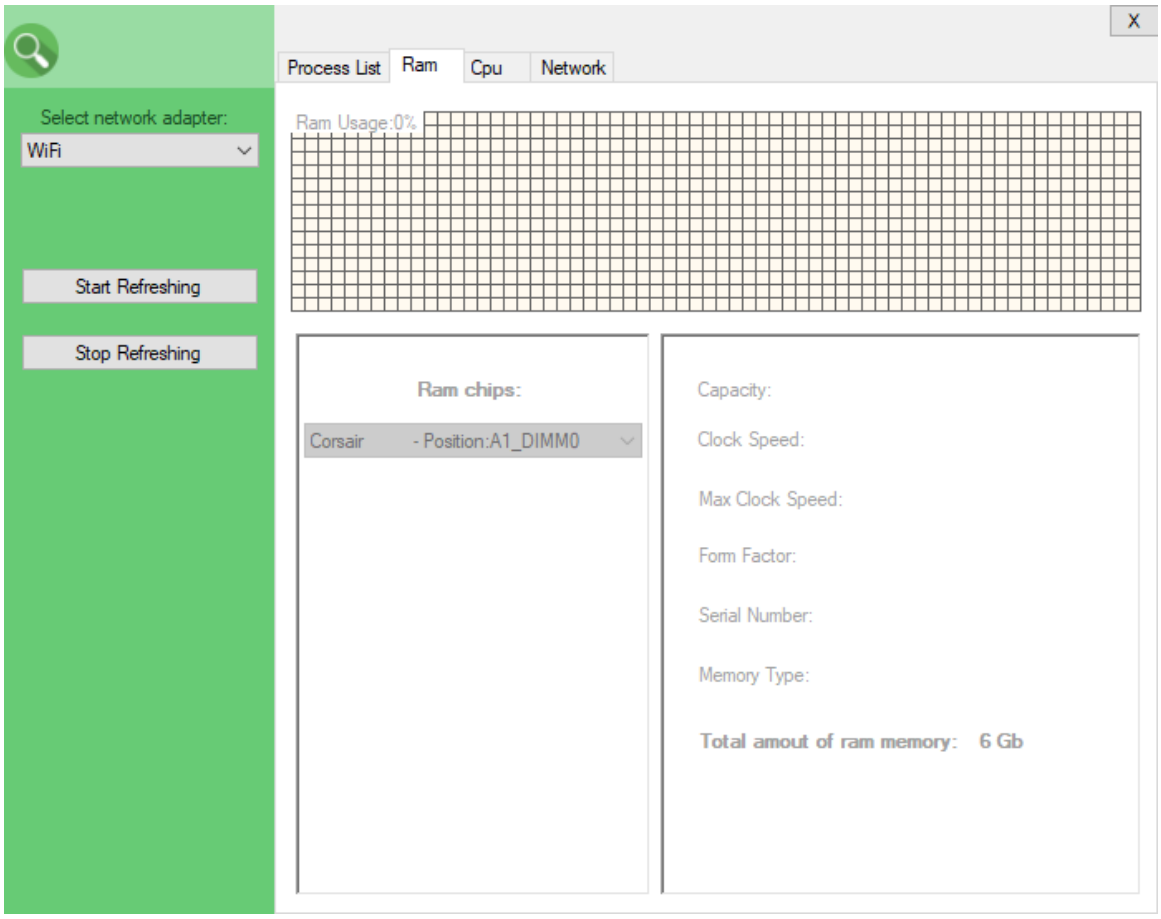
5.2 Rad programa

Ova aplikacija je napravljena pomoći Windows formi te pri pokretanju izgleda kao na sljedećoj slici.

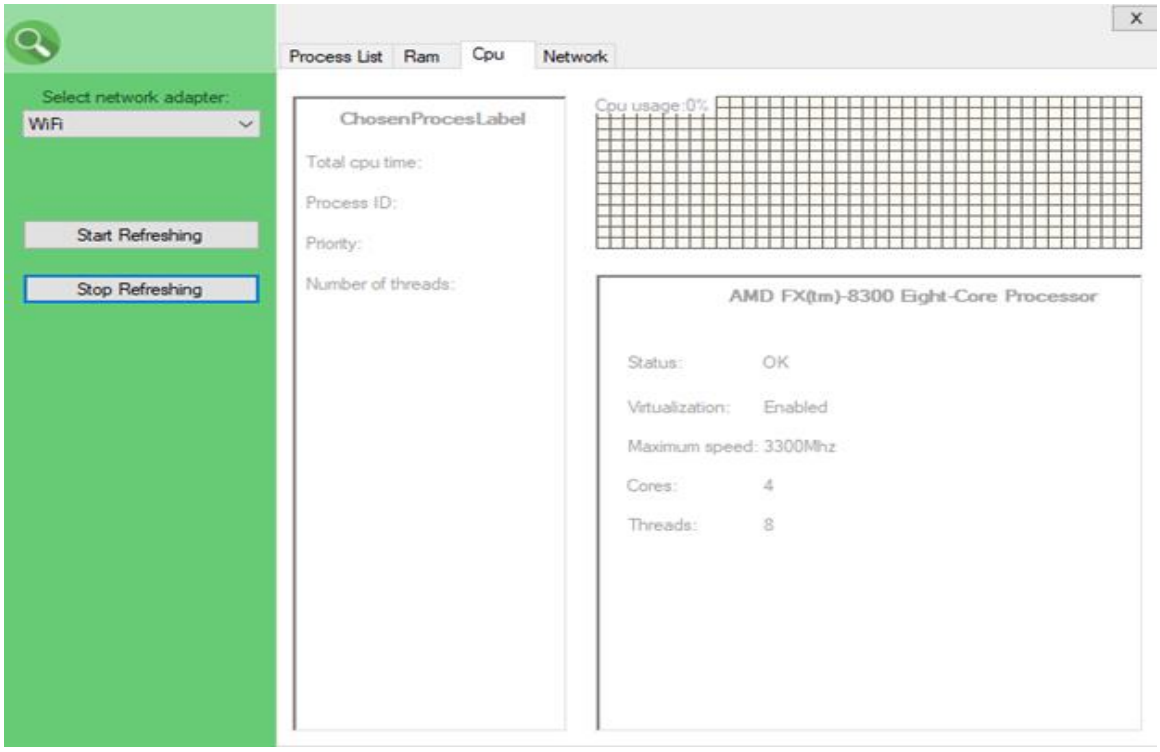


Sl. 5.1 Pokrenuta aplikacija

Program pri pokretanju u padajućem izborniku ispod natpisa 'Select network adapter:' ispisuje sve dostupne mrežne adaptere koji su žično ili bežično spojeni na ruter i automatski odabire adapter preko kojeg je računalno spojeno na internet. Lista procesa se ažurira svakih dvije sekunde dok se grafovi koji prikazuju ukupno zauzeće ram memorije i procesora ažuriraju svaku sekundu. Unutar liste procesa nalazi se ime procesa, identifikacijski broj procesa, te ukupna iskorištenost ram memorije i vremena procesora od strane pojedinog procesa. Na karticama 'Ram' i 'Cpu' nalaze se informacije o istoimenim komponentama, kao i o odabranom procesu. Na kartici 'Cpu' se odmah prikazuju podatci o procesoru, dok je na kartici 'Ram' potrebno odabrati jedan od izbora padajućeg izbornika kako bi se prikazale informacije o odabranom ram čipu. Kartice 'Ram' i 'Cpu' su onemogućene sve dok korisnik ne odabere proces s liste procesa i stavi kvačicu na 'Add to WatchDog'.

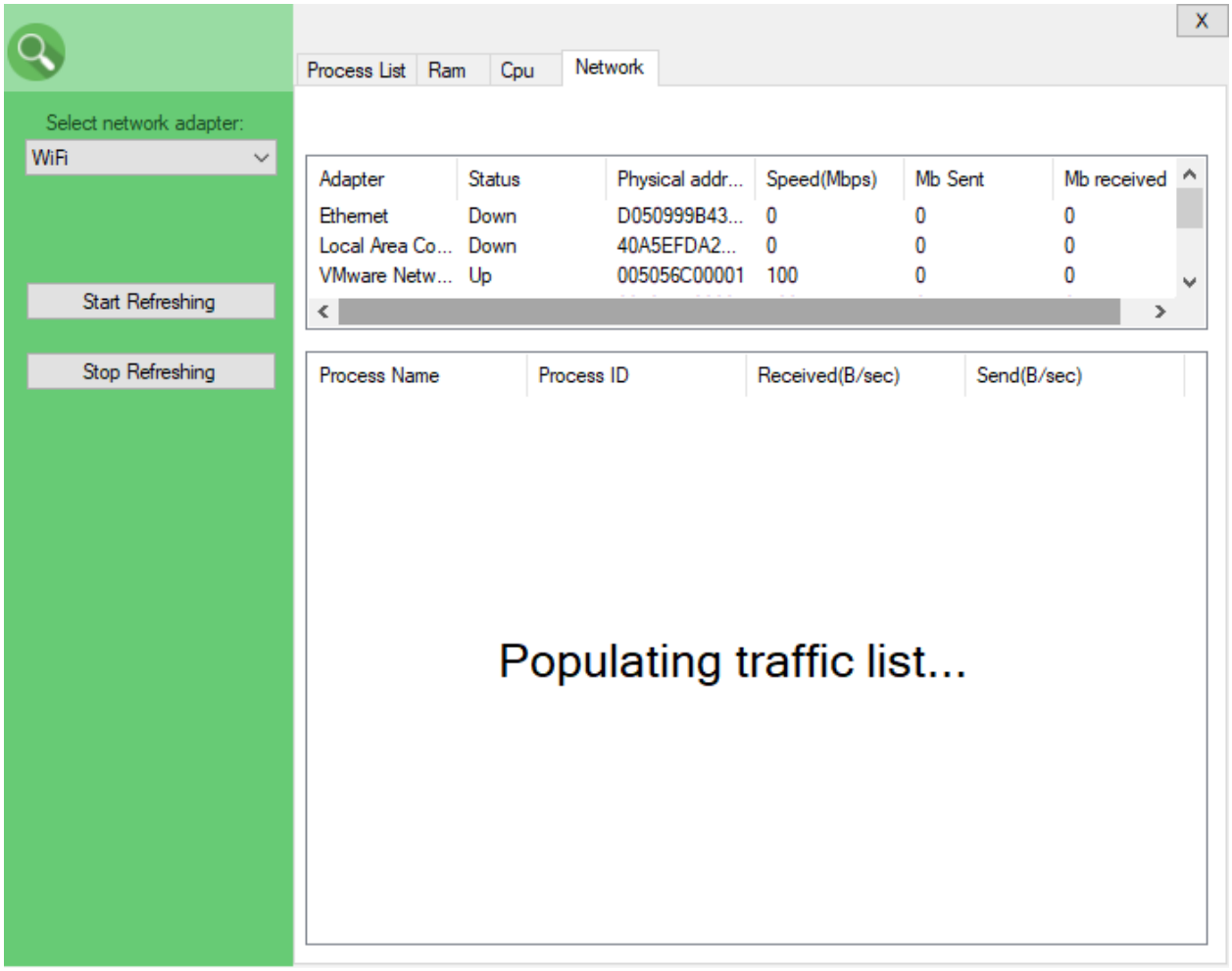


Sl. 5.2 Prikaz Ram kartice nakon pokretanja aplikacije

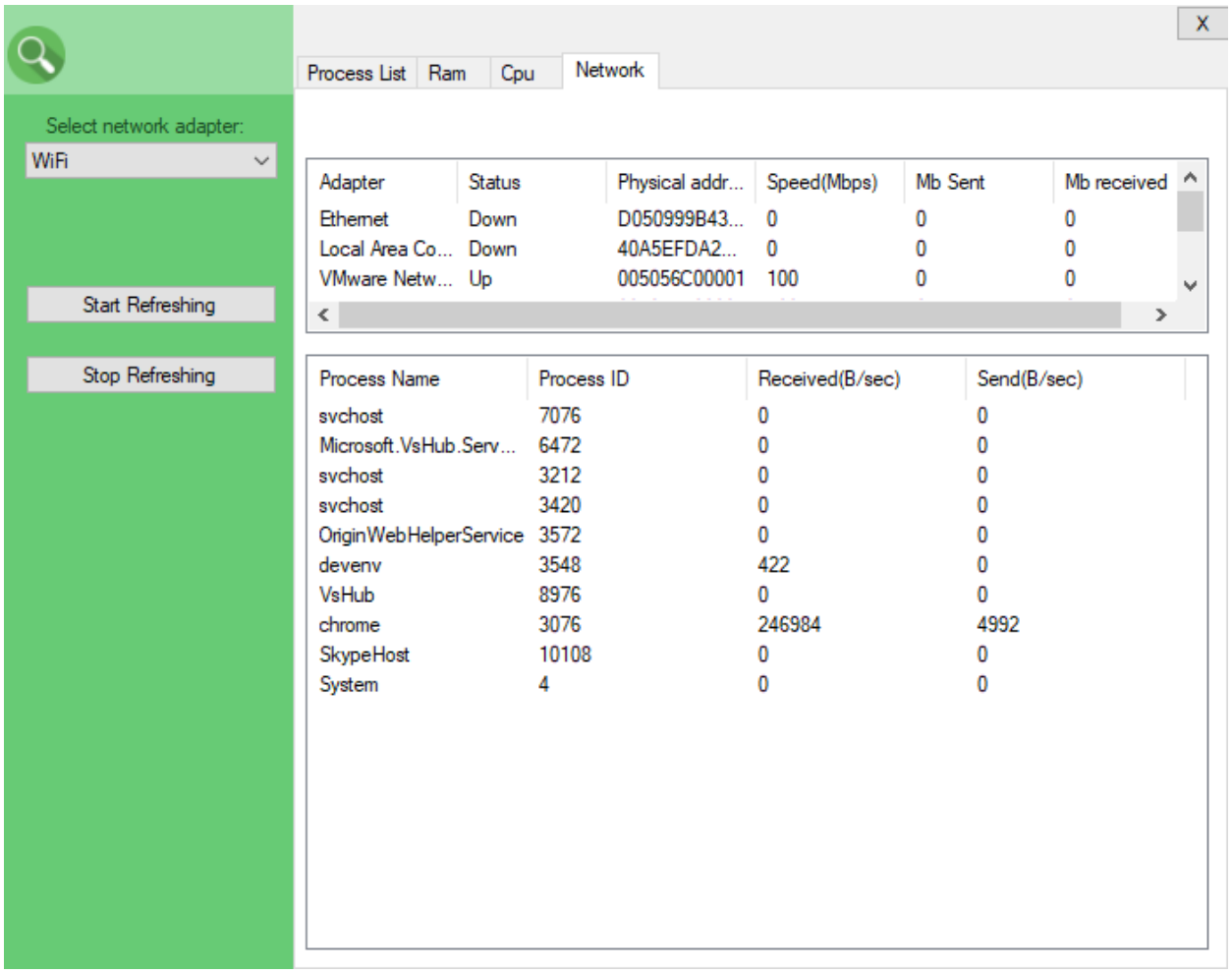


Sl. 5.3 Prikaz Cpu kartice nakon pokretanja aplikacije

Na kartici 'Network' prikazuju se svi dostupni mrežni adapteri, pa čak i testni(eng. *loopback*) adapteri, te informacije o njima kao što su status, MAC adresa, brzina i količina skinutih i poslanih podataka u megabajtima. Lista tih podataka također se ažurira svakih dvije sekunde. Lista ispod liste koja prikazuje mrežne adaptere prikazuje sve procese koji stvaraju mrežni promet na odabranom adapteru(kao što je i prije navedeno adapter se može odabrati u padajućem izborniku s lijeve strane programa), te brzinu kojom šalju i primaju podatke. Prije nego što se pribave podatci o mrežnom prometu procesa, kao i pri promjeni mrežnog adaptera u padajućem izborniku, u aplikaciji se pojavljuje labela s tekstom „Populating traffic list...".

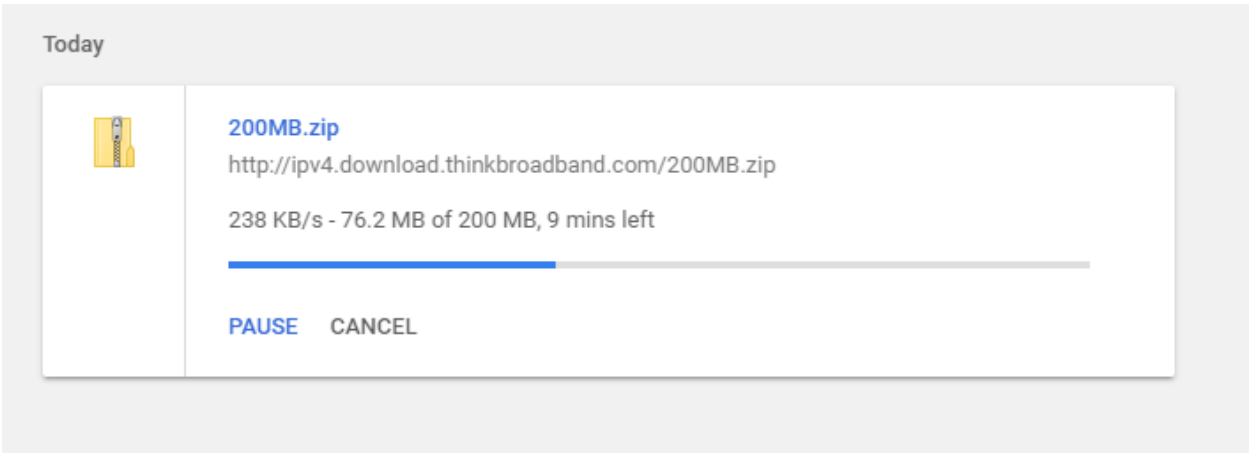


Sl. 5.4 Prikaz sadržaja 'Network' kartice prije popunjavanja liste s procesima

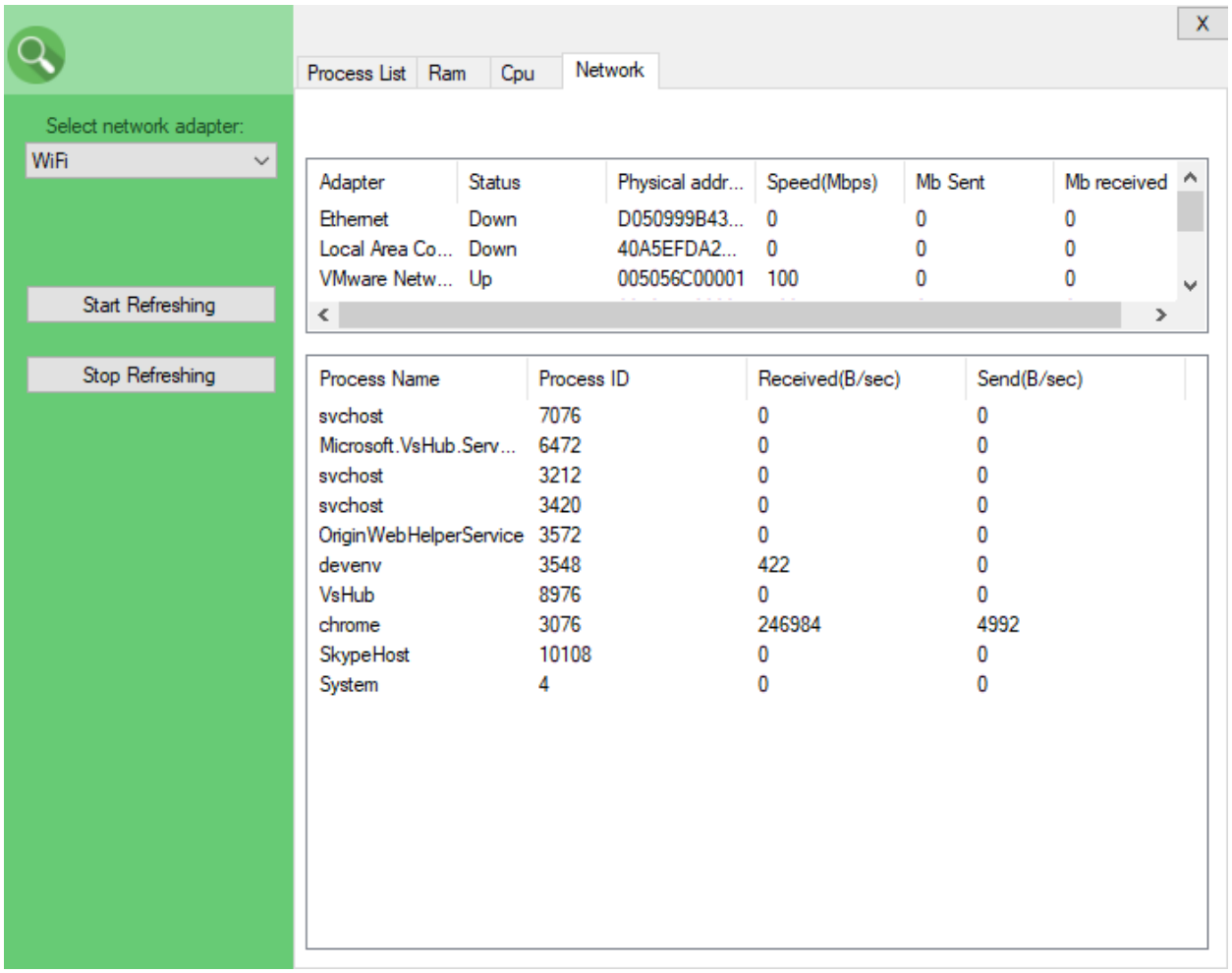


Sl. 5.5 Prikaz 'Network' kartice nakon popunjavanja liste s procesima

Radi testiranja ispravnosti aplikacije u programu Google Chrome stavljena je na preuzimanje testna datoteka. Google chrome program u danom trenutnu prikazao je brzinu preuzimanja od 238 KB/s dok je ova aplikacija prikazala 246984 B/s što je približno 242 KB/s.

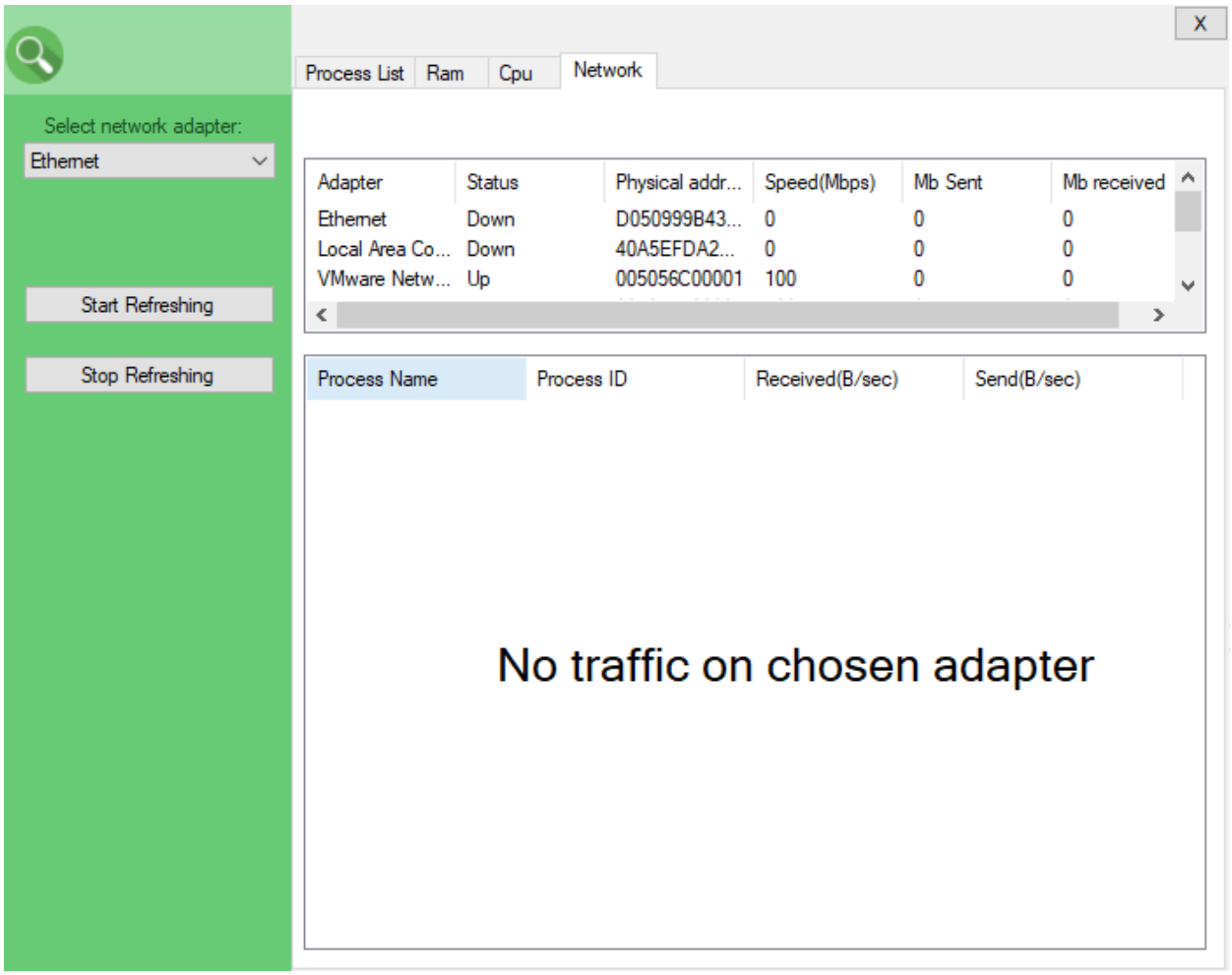


Sl. 5.6 Prikaz brzine preuzimanja testne datoteke u programu Google Chrome



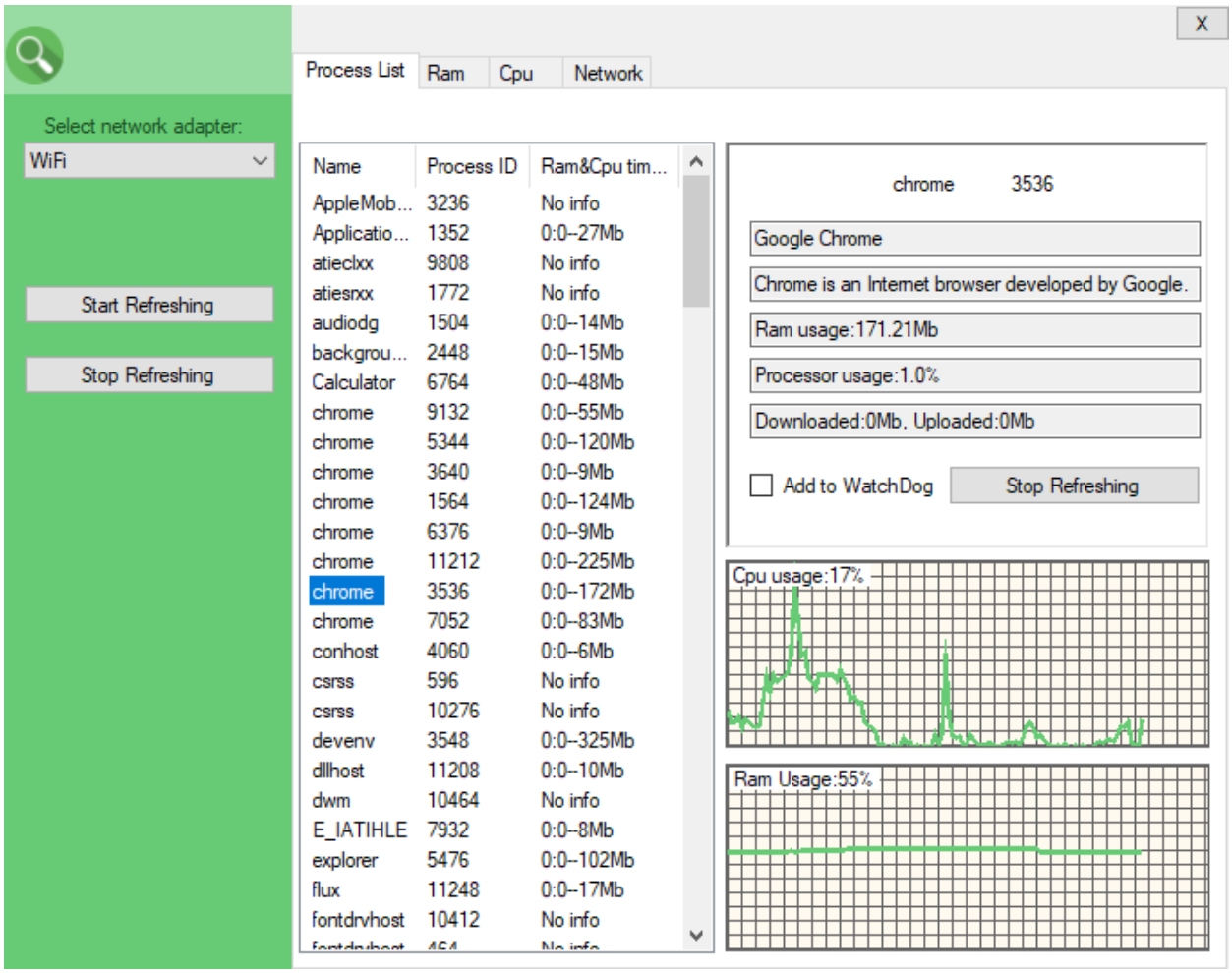
Sl. 5.7 Prikaz brzine preuzimanja testne datoteke u izrađenoj aplikaciji

Prilikom promjene mrežnog adaptera u padajućem izborniku u listi procesa pojavljuju se samo oni procesi koji stvaraju mrežni promet putem odabranog adaptera. Ukoliko nema procesa koji komuniciraju putem odabranog adaptera pojavljuje se tekst "No traffic on chosen adapter". Ako se nakon nekog vremena ipak pojave procesi koji stvaraju promet putem odabranog adaptera tekst "No traffic on chosen adapter" će se maknuti, te će takvi procesi biti prikazani u listi.



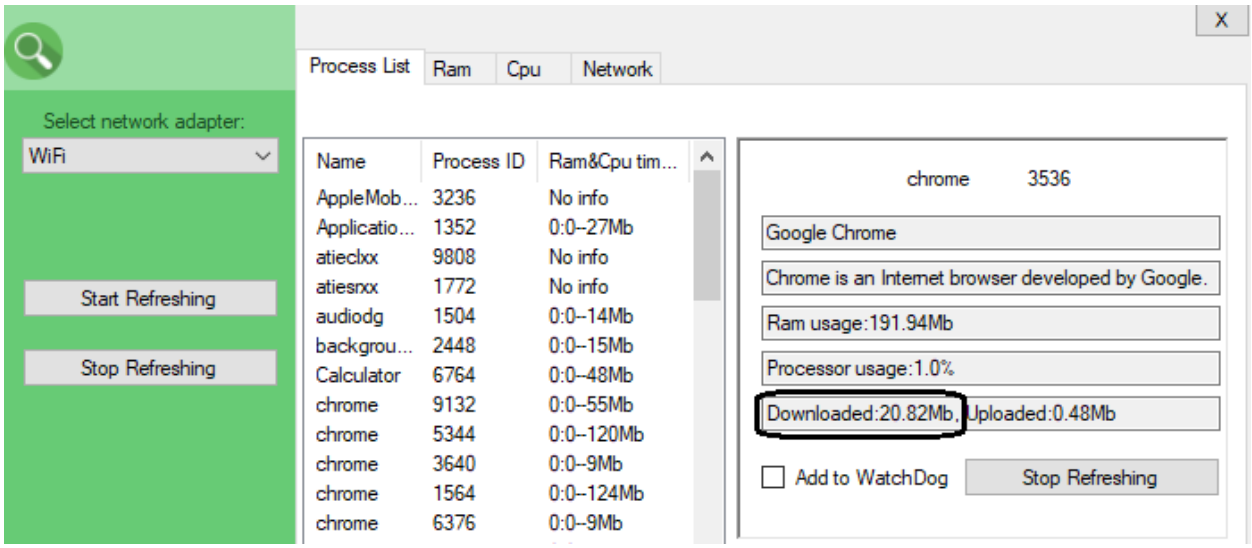
Sl. 5.8 Prikaz teksta koji se pojavljuje ukoliko nema procesa koji stvaraju promet na odabranom adapteru

Ukoliko korisnik klikne na neki od procesa s liste procesa na kartici 'Process List' s desne strane programa pojavljuju se informacije o odabranom procesu. Informacije koje se prikazuju su ime procesa, identifikacijski broj procesa, ugrađeni opis procesa, dodatni opis procesa, korištenje ram memorije i procesora, te količina poslanih i skinutih podataka od trenutka odabiranja procesa.



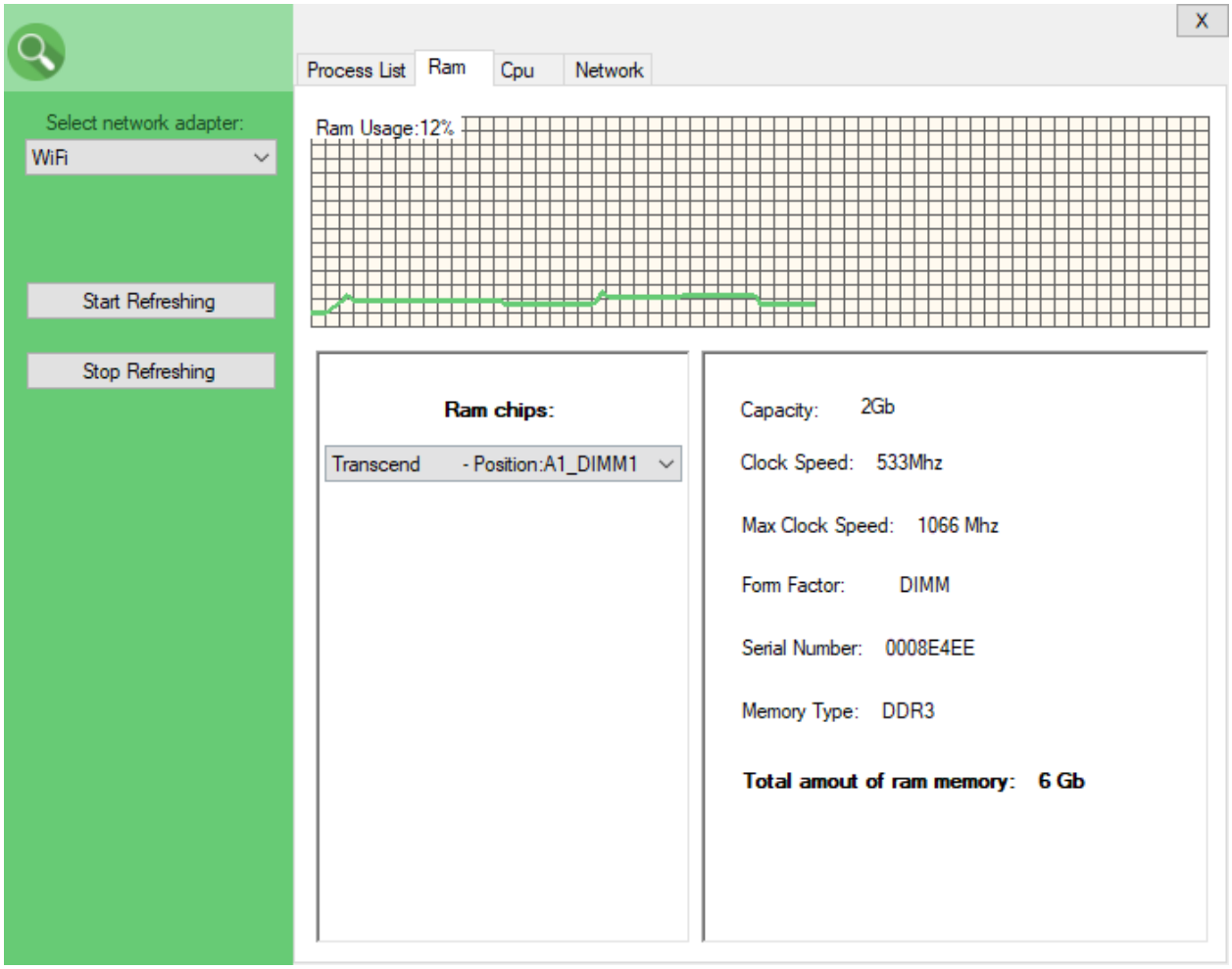
Sl. 5.9 Prikaz aplikacije nakon odabira procesa

Radi testiranja ispravnosti u programu Google Chrome stavljena je na preuzimanje testna datoteka od 20 megabajta, koliko je i približno prikazano kao preuzeto u izrađenoj aplikaciji.



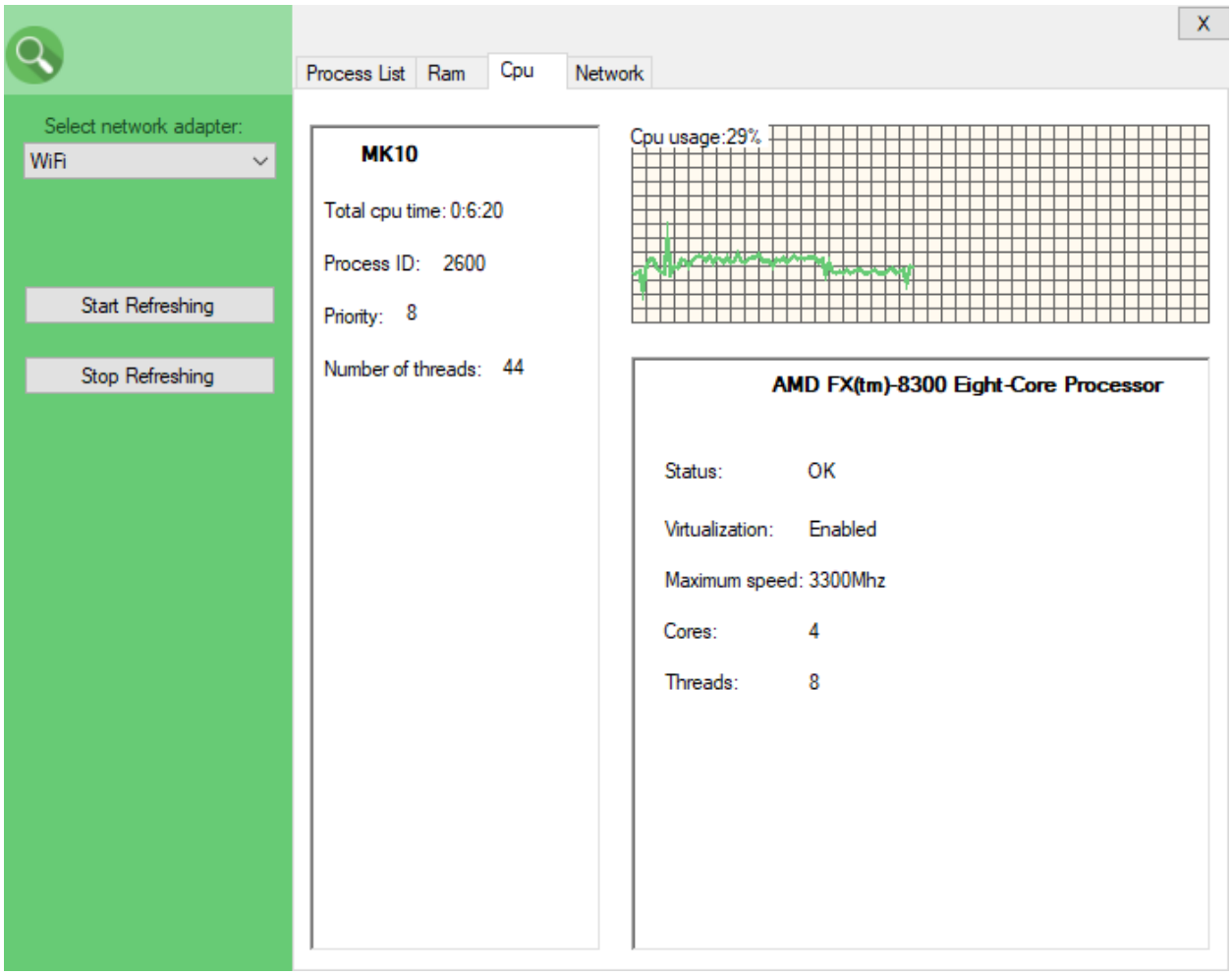
Sl. 5.10 Prikaz količine preuzetih podataka od strane izabranog procesa

Nakon odabira određenog procesa s liste procesa na kartici 'Process List' moguće je s desne strane staviti kvačicu na kvadratić s oznakom 'Add to WatchDog'. Nakon stavljanja kvačice omogućen je sadržaj na karticama 'Ram' i 'Cpu' gdje se prikazuju dodatne informacije o odabranom procesu. Ukoliko se klikne na neki drugi proces s liste procesa na 'Process List' kartici, a prethodno je kvačica stavljena na neki odabrani proces, kartice 'Ram' i 'Cpu' će ponovno onemogućiti i očistiti svoj sadržaj te će se kvačica maknuti. Isto se dogodi ukoliko se odabrani proces ugasi. Na kartici 'Ram' moguće je u padajućem izborniku odabrati jedan od prikazanih ram čipova nakon čega se pojavljuju informacije o kapacitetu, trenutnoj i maksimalnoj brzini, faktoru oblika, serijskom broju i vrsti memorije ram čipa. Na istoj kartici nacrtan je i graf koji prikazuje koliko odabrani proces zauzima ram memorije u postotku. Ukupna količina ram memorije uvijek je ispisana.



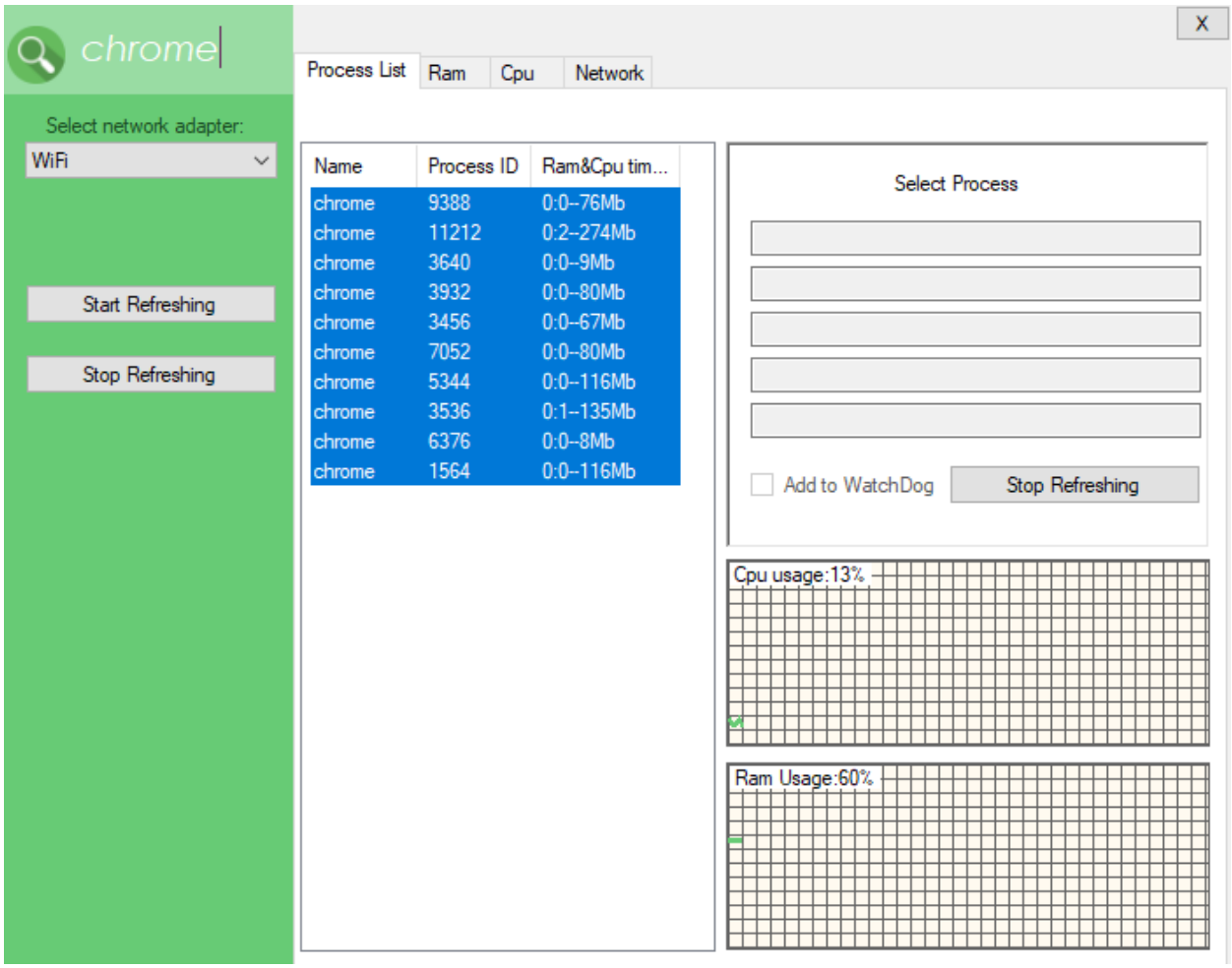
Sl. 5.11 Prikaz 'Ram' kartice nakon stavljanje kvačice kod odabranog procesa

Na kartici 'Cpu' prikazan je status procesora, maksimalna brzina, broj logičkih i fizičkih jezgri, te da li je virtualizacija omogućena ili ne. Također je uz ime procesa ispisan i identifikacijski broj procesa, prioritet i broj niti procesa, te koliko ukupno vremena procesora je dodijeljeno tom procesu. Na istoj kartici nacrtan je i graf koji prikazuje koliko vremena procesora, u postotku, unutar 0.9 sekundi odabrani proces iskoristi.



Sl. 5.12 Prikaz 'Cpu' kartice nakon stavljanje kvačice kod odabranog procesa

U lijevom gornjem kutu aplikacije, pored ikone povećala, moguće je upisati ime za pretragu liste procesa na kartici 'Process List'. Odmah nakon klikanja u prostor za upisivanje teksta osvježavanje liste procesa se zaustavlja. Ukoliko korisnik želi pretražiti listu potrebno je unijeti tekst za pretragu te pritisnuti enter tipku na tipkovnici. Nakon izlaska iz prostora za pretragu osvježavanje liste procesa ponovno započinje, a prostor za pretragu briše svoj sadržaj.



Sl. 5.8 Prikaz rezultata pretrage

Pomoću tipki 'Start Refreshing' i 'Stop Refreshing' s lijeve strane aplikacije korisnik može odabrati želi li da se lista procesa redovito ažurira ili ne. Klikom na 'Stop Refreshing' prestaje osvježavanje liste procesa na kartici 'Process List', ali isto tako prestaje osvježavanje liste procesa koji koriste određeni adapter na kartici 'Network'. Klikom na 'Start Refreshing' osvježavanje listi procesa na 'Process List' i ' Network' kartici ponovno započinje. Klikom na tipku 'Stop Refreshing' koja se nalazi s desne strane aplikacije prestaje osvježavanje podataka o odabranom procesu. Osvježavanje se može pokrenuti klikom na proces.

5.3 Opis koda programa

5.3.1 Metode i okidači

Kod programa sastoji se od više funkcija, metoda i okidača koji se izvršavaju ovisno o odabiru određenog gumba ili neke druge mogućnosti aplikacije. Metode koje se koriste odmah pri pokretanju aplikacije su: metoda za inicijalizaciju komponenti, metoda za pribavljanje informacija o hardveru, brojač koji upravlja osvježavanjem elemenata listi, metoda koja popunjava padajući izbornik s imenima adaptera, metoda koja onemogućuje određene kontrole aplikacije i metoda koja pokreće crtanje grafova.

Okidači koji okidaju na ponovno crtanje prostora za sliku(*PictureBox*) stu postavljeni na četiri prostora za sliku. Ti prostori koriste se za crtanje grafova na karticama 'Process List' , 'Ram' i 'Cpu' u aplikaciji. Postavljena su tri okidača na prostor za pretragu liste procesa(*TextBox*). Okidači se aktiviraju na ulazak u prostor za pretragu,napuštanje prostora za pretragu i na pritisak tipke s tipkovnice dok je prostor za pretragu u fokusu. Na oba padajuća izbornika(ComboBox) postavljeni su okidači koji se aktiviraju na promjenu izabranog elementa, a na padajući izbornik koji prikazuje dostupne adaptere postavljen je i okidač koji se aktivira kada se miš zadrži na jednom od elemenata. Prostor za kvačicu(CheckBox) je odmah onemogućen, jer je potrebno prvo odabrati proces kako bi se mogla staviti kvačica, te je postavljen okidač koji se aktivira na promjenu stanja prostora za kvačicu.

```
public Aa()
{
InitializeComponent();
GetInfo();
InitTimer();
NetworkDropBox();
WatchCheckBox.Enabled = false;
DisableControls();
CpuPictureBox.Paint += new PaintEventHandler(CpuRedraw);
RamPictureBox.Paint += new PaintEventHandler(RamRedraw);
ChosenProcCpuPB.Paint += new PaintEventHandler(ProcessCpuRedraw);
ChosenProcRamPB.Paint += new PaintEventHandler(ProcessRamRedraw);
NetworkComboBox.MouseHover += new EventHandler(mouseRest);
WatchCheckBox.CheckedChanged += new EventHandler(CheckBoxChange);
NetworkComboBox.SelectedValueChanged += new EventHandler(FocusItemChange);
RamComboBox.SelectedValueChanged += new EventHandler(RamBoxFocusChange);
SearchBox.Enter += new EventHandler(SearchBoxEnter);
SearchBox.Leave += new EventHandler(SearchBoxLeave);
SearchBox.KeyDown += new KeyEventHandler(SearchBoxChange);
GraphsStart();
}
```

5.3.2 Prikupljanje informacija o hardveru

Metoda GetInfo omogućuje prikaz informacija o hardveru računala uz pomoć biblioteke 'System.Management' gdje se uz pomoć upita pribavljaju informacije. Informacije o procesoru se postavljaju u odgovarajuće labele, a informacije o pojedinom čipu ram memorije dodaju se u odgovarajući padajući izbornik. Osim imena proizvođača u padajući izbornik dodaje se i pozicija ram čipa te se ta informacija koristi kao identifikator pojedinog ram čipa kako bi se dobile točne informacije.

```
private void GetInfo()
{
ulong totalRam=0;
ManagementObjectSearcher processor = new ManagementObjectSearcher("select * from
Win32_Processor");
ManagementObjectSearcher memory = new ManagementObjectSearcher("root\\CIMV2", "select *
from Win32_PhysicalMemory");
foreach (ManagementObject obj in processor.Get())
      {
        ProcessorNameLabel.Text = obj["Name"].ToString();
        StatusLabel.Text = obj["Status"].ToString();
        SpeedLabel.Text = obj["MaxClockSpeed"].ToString() + "Mhz";
        if (obj["VirtualizationFirmwareEnabled"].ToString() == "True")
                VirtLabel.Text = "Enabled";
        else
                VirtLabel.Text = "Disabled";
        CoresLabel.Text = obj["NumberOfCores"].ToString();
        ThreadsLabel.Text = obj["NumberOfLogicalProcessors"].ToString();
       }
foreach(ManagementObject obj in memory.Get())
       {
         string item = obj["Manufacturer"].ToString();
         item.Trim();
         item+=" - Position:"+obj["DeviceLocator"].ToString();
         RamComboBox.Items.Add(item);
         totalRam += (ulong)obj["Capacity"];
        }
RamComboBox.SelectedIndex = 0;
TotalRamLabel.Text = (totalRam/1024/1024/1024).ToString()+" Gb";
RamAmount.totalRam = totalRam;
}
```

5.3.3 Osvježavanje liste procesa i mrežnih adaptera

Metoda InitTimer je brojač koji omogućava pozivanje metode P_refresh za prikazivanje liste pokrenutih procesa i informacija o njima , te metode net liste mrežni adaptera i informacija o njima,svake dvije sekunde.

```
private System.Windows.Forms.Timer timer1;
        public void InitTimer()
        {
            timer1 = new System.Windows.Forms.Timer();
            timer1.Tick += new EventHandler(P_refresh);
            timer1.Tick += new EventHandler(net);
            timer1.Interval = 2000;
            timer1.Start();
        }
```

Metoda P_refresh služi za pozivanje metode 'Get_Process' koja popunjava popis pokrenutih procesa. Metoda 'Get_Process' se poziva preko metode 'P_refresh' zato što se metoda 'Get_process', osim putem brojača 'timer1', poziva na više mjesta u aplikaciji pa je jednostavnije pozvati metodu putem druge metode nego li preopterećivati metodu 'Get_Process' samo zbog različitih parametara koje prima.

```
public void P_refresh(object sender, EventArgs e)
{
       Get_Process(pc_info);
}
```

Metoda 'Get_Process' dohvaća sve pokrenute procese, sortira ih po imenu, te stvara objekte klase 'Procesi' gdje postoje metode za dohvaćanje svih važnih informacija. Prije dodavanja stavki na listu potrebno je očistiti sve prijašnje spremljene stavke. Za dodavanje informacija o procesu na listu koriste se objekti klase 'Procesi' koji imaju metode za dohvaćanje imena procesa, identifikacijskog broja procesa, te količine iskorištene ram memorije i vremena procesora. Također su obrađene i iznimke koje su rezultat toga što operacijski sustav ne dozvoljava pristupanje sistemskim procesima.

```
private void Get_Process()
{
       int i = 0;
       Process[] procesi_popis = Process.GetProcesses();
       Array.Sort(procesi_popis,
           delegate       (Process       x,       Process       y)       {       return
       x.ProcessName.CompareTo(y.ProcessName); });
       Procesi[] Svi_procesi = new Procesi[procesi_popis.Length];
       Lista_procesa_prikaz.Items.Clear();
       foreach (Process proces in procesi_popis)
            {
              Svi_procesi[i] = new Procesi(proces);
              ListViewItem item = new ListViewItem();
              item.Text = Svi_procesi[i].Vrati_ime();
              item.SubItems.Add(Svi_procesi[i].Vrati_id().ToString());
              item.SubItems.Add(Svi_procesi[i].Vrati_CpuRaminfo());
              Lista_procesa_prikaz.Items.Add(item);
              i++;
            }
}
public class Procesi
        {
            string ime;
            int id;
            long Ram_usage;
            Process proces;
            public Procesi(Process p)
            {
                proces = p;
                ime = proces.ProcessName;
                id = proces.Id;
                Ram_usage = proces.PeakWorkingSet64;
            }
            public string Vrati_ime()
            {
                return ime;
            }
            public int Vrati_id()
            {
                return id;
            }
```

```
            public long Vrati_zauzece_rama()
            {
                return Ram_usage;
            }
            public string Vrati_CpuRaminfo()
            {
                try
                {
                    string rez = proces.TotalProcessorTime.Hours.ToString() + ":" +
proces.TotalProcessorTime.Minutes.ToString();
                    rez = rez + "--" + proces.WorkingSet64 / 1024 / 1024 +"Mb";
                    return rez;
                }
                catch
                {
                    return "No info";
                }
            }
        }
```

Metoda 'net' obavlja sličan posao kao i metoda Get_Process, tj. pribavlja informacije o mrežnim adapterima i svim njihovim svojstvima te nakon toga popunjava pogled na listu.

```
private void net(object sender, EventArgs e)
{
    if(NetworkInterface.GetIsNetworkAvailable())
  {
     NetworkInterface[] sucelja = NetworkInterface.GetAllNetworkInterfaces();
     Adapters_listview.Items.Clear();
     foreach (NetworkInterface inf in sucelja)
     {
       ListViewItem item = new ListViewItem();
       item.Text = inf.Name;
       item.SubItems.Add(inf.OperationalStatus.ToString());
       item.SubItems.Add(inf.GetPhysicalAddress().ToString());
       int brzina = (int)inf.Speed;
       item.SubItems.Add(((int)inf.Speed/1000000).ToString());
       item.SubItems.Add((inf.GetIPv4Statistics().BytesSent/1024/1024).ToString());
       item.SubItems.Add((inf.GetIPv4Statistics().BytesReceived/1024/1024).ToString());
       Adapters_listview.Items.Add(item);
```

```
    }
  }
}
```

5.3.4 Osvježavanje liste procesa koji stvaraju mrežni promet

Metoda 'NetworkDropBox' određuje lokalnu IP adresu, tj. adresu adaptera putem kojeg se odvija mrežni promet. Metoda određuje lokalnu IP adresu tako što pokušava uspostaviti vezu s Google-om putem UDP protokola. Nije bitna adresa na koju se metoda pokušava spojiti, to se radi na taj način samo da se utvrdi putem kojeg adaptera(a time i koje IP adrese) računalo pokušava poslati podatke. Nakon dobivanja lokalne IP adrese dohvaćaju se dostupni adapteri koji se žično ili bežično spajaju na internet. Kako metode biblioteka 'SharpPcap' i 'System.Net.NetworkInformation' na različite načine numeriraju mrežne adaptere potrebno zabilježiti nekakav jedinstven identifikator, tj. MAC adresu adaptera kako bi se mogao pronaći indeks željenog adaptera unutar metoda ili objekata 'SharpPcap' biblioteke.

Nakon pronalaska željenih informacija imena svih adaptera se dodaju u padajući izbornik, a indeks koji će se koristiti za metode 'SharpPcap' biblioteke i lokalna IP adresa spremaju se u statičnu klasu imena 'AdapterInfo'.

```
private void NetworkDropBox()
        {
            int index=0,ListeningAdapterID=0;
            IPAddress hostIP;
            using (Socket socket = new
            Socket(AddressFamily.InterNetwork,SocketType.Dgram, ProtocolType.Udp))
            {
                socket.Connect("8.8.8.8", 4860);
                IPEndPoint endPoint = socket.LocalEndPoint as IPEndPoint;
                hostIP = endPoint.Address;
            }
            string mac="";
            CaptureDeviceList devs = CaptureDeviceList.Instance;
            NetworkInterface[] interfaces = NetworkInterface.GetAllNetworkInterfaces();
            if (interfaces.Count() != 0)
            {
                for (int i = 0; i < interfaces.Length; i++)
                {
                    if (interfaces[i].NetworkInterfaceType
```

```
==NetworkInterfaceType.Ethernet ||
interfaces[i].NetworkInterfaceType == NetworkInterfaceType.Wireless80211)
                    {
                        NetworkComboBox.Items.Add(interfaces[i].Name);
                        foreach (UnicastIPAddressInformation ip in
interfaces[i].GetIPProperties().UnicastAddresses)
                        {
                            if (ip.Address.Equals(hostIP))
                            {
                                index = i;
                                mac = interfaces[i].GetPhysicalAddress().ToString();
                            }
                        }
                    }
                }
                for (int i = 0; i < devs.Count; i++)
                {
                    devs[i].Open();
                    if (devs[i].MacAddress.ToString() == mac)
                        ListeningAdapterID = i;
                    devs[i].Close();
                }
                AdapterInfo.AdapterIndex = ListeningAdapterID;
                AdapterInfo.LocalIP = hostIP;
                NetworkComboBox.SelectedIndex = index;
                StartThread();
            }
            else
            {
                NetworkComboBox.Items.Add("No network adapter detected.");
                NetworkComboBox.SelectedIndex = 0;
                PopulatingLabel.Text = "No network adapter detected.";
            }
        }
```

Klasa 'AdapterInfo', koja se koristi pri kraju metode 'NetworkDropBox', je statična klasa koja ima dva javna tipa, a to su indeks adaptera i lokalna IP adresa.

```
static class AdapterInfo
 {
    public static IPAddress LocalIP { get; set; }
    public static int AdapterIndex { get; set; }
 }
```

Na kraju metode 'NetworkDropBox' poziva se metoda 'StartThread' koja na novoj, ranije definiranoj, niti poziva metodu 'NetList'.

```
private void StartThread()
  {
     ProcessListNetRefresh = new Thread(NetList);
     ProcessListNetRefresh.IsBackground = true;
     ProcessListNetRefresh.Start();
   }
```

Metoda 'NetList' je zadužena za osvježavanje podataka o listi koja se u aplikaciji nalazi na kartici 'Network', tj. ona se brine o osvježavanju podataka o procesima koji stvaraju mrežni promet. Ova metoda se ne poziva nikakvim brojačem jer se u njoj stvara objekt klase 'ProcessNetList' u kojoj se nalazi velik dio koda koji se ne može izvršiti za manje od 5 sekundi.

```
private void NetList()
        {
            LabelChange(PopulatingLabel, "Populating traffic list...",true);
            do
            {
                ProcessNetList test = new ProcessNetList();
                UpdateListView();
                if (ConcurrentList.DownloadBag.Count != 0)
                {
                    PopulatingLabel.Visible = false;
                    foreach (string s in ConcurrentList.DownloadBag)
                    {
                        string[] arrayD = s.Split(':');
```

```
                        foreach (string s1 in ConcurrentList.UploadBag)
                        {
                            string[] arrayU = s1.Split(':');
                            if (arrayD[1] == arrayU[0])
                            {
                                ListViewItem item = new ListViewItem();
                                item.Text = arrayD[0];
                                item.SubItems.Add(arrayD[1]);
                                item.SubItems.Add(arrayD[2]);
                                item.SubItems.Add(arrayU[1]);
                                UpdateListView(item);
                            }

                        }
                    }
                }
                else
                {
                    LabelChange(PopulatingLabel, "No traffic on chosen adapter",true);
                }
            }
            while (NetListRefresh);
}
```

Nakon stvaranja objekta klase 'ProcessNetList' metoda 'NetList' poziva metodu 'UpdateListView'. 'UpdateListView' metoda je preopterećena, kada se pozove bez argumenata onda briše sve predmete liste, a kada se pozove s jednim argumentom onda dodaje predmete na listu koja prikazuje procese koji koriste mrežu(kartica 'Network' u aplikaciji). Također ta metoda koristi objekt 'Invoke' kako bi se omogućilo dodavanje predmeta na listu s druge niti.

```
        public void UpdateListView(ListViewItem item)
        {
            if (ProcessNetUsageList.InvokeRequired)
            {
                ProcessNetUsageList.Invoke(new
Action<ListViewItem>(UpdateListView),item);
            }
            else
            {
                ProcessNetUsageList.Items.Add(item);
```

```
            }
        }
        public void UpdateListView()
        {
            ProcessNetUsageList.Items.Clear();
        }
```

Poslije poziva 'UpdateListView' metoda 'NetList' provjerava jesu li liste koje se nalaze u statičnoj klasi 'ConcurrentList' prazne. Razlog korištenja konkurentnih lista je to što je dalje u programu potrebno imati podatke iz više niti na jednom mjestu. Ti podatci su dodani posebnim redosljedom u varijable tekstualnog tipa te su odvajani znakom ':' koji se koristi kao separator. Podatci spremljeni u listu 'DownloadBag' su oblika: 'Ime procesa:Identifikacijski broj procesa:Brzina preuzimanja podataka', a podatci spremljeni u listu 'UploadBag' su oblika: 'Identifikacijski broj procesa:Brzina slanja podataka'. Metoda 'NetList' također brine o labeli koja ispisuje što se događa prilikom ispisa procesa koji koriste mrežni adapter. Petlja u metodi 'NetList' ovisi u varijabli 'NetListRefresh' koja je definirana na početku programa, a njeno stanje mijenjaju tipke 'Start Refresh' i 'Stop Refresh' s desne strane aplikacije. O podatcima u klasi 'ConcurrentList' brine se klasa 'ProcessNetList'.

Konstruktor klase 'ProcessNetList' poziva metodu 'FillLists' koja otvara naredbeni redak u novom procesu te upisuje naredbu 'netstat –ano' koja ispisuje sve procese koji stvaraju mrežni promet, te sve portove i IP adrese putem kojih se komunikacija odvija. Kako se nebi skoro identičan kod pisao dva puta, umjesto preopterećivanja metode 'FillLists', koristi se bool varijabla koja signalizira je li potrebno popunjavati listu procesa ili listu portova koje pojedini proces koristi. Prvo se popunjava lista procesa,zatim se čiste konkurentne liste, a zatim se za svaki proces iz liste ponovno poziva metoda 'FillLists' koja sada za dani proces nalazi portove koje proces koristi. Na kraju metode provjerava se stanje bool varijable koja signalizira radi li se o popunjavanju liste procesa ili portova. Ako se radi o popunjavanju liste portova onda se poziva metoda 'StartThreads' unutar klase 'ProcessNetList'.

```
public ProcessNetList()
        {
            FillLists();
            ProcIDmode = false;
            ConcurrentList.ClearDownload();
            ConcurrentList.ClearUpload();
            foreach (int pid in ProcessList)
            {
                processIDq = pid;
                PortList.Clear();
                FillLists();
            }
        }
```

```
void FillLists()
        {
            Process cmd = new Process();
            cmd.StartInfo.FileName = "cmd.exe";
            cmd.StartInfo.UseShellExecute = false;
            cmd.StartInfo.RedirectStandardInput = true;
            cmd.StartInfo.RedirectStandardOutput = true;
            cmd.StartInfo.RedirectStandardError = true;
            cmd.StartInfo.CreateNoWindow = true;
            cmd.Start();
            cmd.StandardInput.WriteLine("netstat -ano");
            cmd.StandardInput.WriteLine("exit");
            Regex regex1 = new Regex("\\s+", RegexOptions.Compiled);
            string niz = null;
            while ((niz = cmd.StandardOutput.ReadLine()) != null)
            {
                niz = niz.Trim();
                if (niz.StartsWith("TCP", StringComparison.OrdinalIgnoreCase))
                {
                    niz = regex1.Replace(niz, ",");
                    string[] array = niz.Split(',');
                    int pozicija = array[1].ToString().IndexOf(":",
StringComparison.Ordinal);
                    string IPonly = array[1].ToString().Substring(0, pozicija);
                    if (ProcIDmode)
                    {
                        int pid = int.Parse(array[4]);
                        if (!ProcessList.Contains(pid) && pid != 0 && IPonly ==
AdapterInfo.LocalIP.ToString())
                            ProcessList.Add(pid);
                    }
                    else
                    {

                        if (array[4] == processIDq.ToString())
                        {
                            string socket = array[1];
                            int pos = socket.LastIndexOf(':');
                            int port = int.Parse(socket.Substring(pos + 1));
                            if (!PortList.Contains(port))
                                PortList.Add(port);
                        }
                    }
                }
                if (niz.StartsWith("UDP", StringComparison.OrdinalIgnoreCase))
                {
                    niz = regex1.Replace(niz, ",");
                    string[] array = niz.Split(',');
                    int pozicija = array[1].ToString().IndexOf(":",
StringComparison.Ordinal);
                    string IPonly = array[1].ToString().Substring(0, pozicija);
                    if (ProcIDmode)
                    {
                        int pid = int.Parse(array[3]);
                        if (!ProcessList.Contains(pid) && pid != 0 && IPonly ==
AdapterInfo.LocalIP.ToString())
                            ProcessList.Add(pid);
                    }
                    else
                    {
                        if (array[3] == processIDq.ToString())
                        {
```

```
                            string socket = array[1];
                            int pos = socket.LastIndexOf(':');
                            int port = int.Parse(socket.Substring(pos + 1));
                            if (!PortList.Contains(port))
                                PortList.Add(port);
                        }
                    }
                }
            }
            cmd.Close();
            if (ProcIDmode == false)
                StartThreads();
        }
```

'StartThreads' metoda klase 'ProcessNetList' pravi filter prema portovima iz liste portova koje je popunila metoda 'FillLists' i prema podatcima o adapteru koji se koristi spremljenim u statičnoj klasi 'AdapterInfo'. Nakon što je filter napravljen stvaraju se dva nova objekta, objekt klase 'UploadForListview' i objek klase 'DownloadForListview'. Pri njihovom stvaranju poziva se i metoda 'ReturnDevice' koja pronalazi adapter prema zadanom indeksu te ga otvara kako bi se na njemu mogli nadgledati paketi. Metode dva nova objekta se pozivaju na novoj niti.

```
ICaptureDevice ReturnDevice()
        {
            ICaptureDevice device = CaptureDeviceList.New()[AdapterInfo.AdapterIndex];
            device.Open(DeviceMode.Promiscuous);
            return device;

        }
void StartThreads()
        {
            if (PortList.Count != 0)
            {
                string filter_partSent = "", filter_partReceived = "";
                for (int i = 0; i < PortList.Count; i++)
                {
                    string constant = " src port ";
                    if (i == 0)
                        filter_partSent = "(";
                    filter_partSent += constant + PortList[i];
                    if (i + 1 == PortList.Count)
                        filter_partSent += " )";
                    else
                        filter_partSent += " or";
                }
                for (int i = 0; i < PortList.Count; i++)
                {
                    string constant = " dst port ";
                    if (i == 0)
                        filter_partReceived = "(";
                    filter_partReceived += constant + PortList[i];
                    if (i + 1 == PortList.Count)
                        filter_partReceived += " )";
                    else
                        filter_partReceived += " or";
                }
                string filterSent = "src host " + AdapterInfo.LocalIP + " and " +
filter_partSent;
              string filterReceived = "dst host " + AdapterInfo.LocalIP + " and " +
```

```
filter_partReceived;
                UploadForListview procUpload = new UploadForListview(filterSent,
processIDq, ReturnDevice());
                DownloadForListview procDownload = new
DownloadForListview(filterReceived, processIDq, ReturnDevice());
                Thread t1 = new Thread(() => procUpload.SentPackets());
                Thread t2 = new Thread(() => procDownload.ReceivedPackets());
                t1.IsBackground = true;
                t2.IsBackground = true;
                t1.Start();
                t2.Start();
            }
        }
```

Metoda 'SentPackets' klase 'UploadForListview' postavlja okidač koji se aktivira na primanje paketa podataka, nakon toga postavlja odgovarajući filter(fitriranje poslanih podataka) na zadani mrežni adapter te započinje nadgledanje paketa podataka. Nakon dvije sekunde nadgledanje podataka prestaje te se podatci spremaju u odgovarajućem obliku u konkurentnu listu 'UploadBag' unutar statične klase 'ConcurrentList'.

```
class UploadForListview
    {
        string FilterUpload;
        string data;
        int adapterIndex;
        int ProcessID;
        long dataLenght;
        double dataPerSec;
        ICaptureDevice uredaj;
        public UploadForListview(string filter,int pid,ICaptureDevice device)
        {
            FilterUpload = filter;
            adapterIndex = AdapterInfo.AdapterIndex;
            ProcessID = pid;
            uredaj = device;
        }
        public void SentPackets()
        {
            uredaj.OnPacketArrival += new PacketArrivalEventHandler(device_OnPacketSent);
            uredaj.Filter = FilterUpload;
            uredaj.StartCapture();
            uredaj.StopCaptureTimeout = new TimeSpan(0,0,1);
            Thread.Sleep(2000);
            dataPerSec = Math.Round(dataLenght / 2d, 3);
            Process p = Process.GetProcessById(ProcessID);
            data = ProcessID  + ":" + dataPerSec;
            ConcurrentList.UploadBag.Add(data);
            uredaj.Close();
        }
        private void device_OnPacketSent(object sender, CaptureEventArgs e)
        {
            dataLenght += e.Packet.Data.Length;
        }
    }
```

Metoda 'ReceivedPackets' klase 'DownloadForListview' postavlja okidač koji se aktivira na primanje paketa podataka, nakon čega se postavlja odgovarajući filter(fitriranje primljenih podataka) na zadani mrežni adapter te započinje nadgledanje paketa podataka. Nakon dvije sekunde nadgledanje podataka prestaje te se podatci spremaju u odgovarajućem obliku u konkurentnu listu 'DownloadBag' unutar statične klase 'ConcurrentList'.

```
    class DownloadForListview
    {
        string FilterDownload;
        string data;
        int adapterIndex;
        int ProcessID;
        ICaptureDevice uredaj;
        long dataLenght;
        double dataPerSec;
        public DownloadForListview(string filter,int pid,ICaptureDevice d)
        {
            uredaj = d;
            FilterDownload = filter;
            adapterIndex = AdapterInfo.AdapterIndex;
            ProcessID = pid;
        }
        public void ReceivedPackets()
        {
            uredaj.OnPacketArrival += new
PacketArrivalEventHandler(device_OnPacketReceived);
            uredaj.Filter = FilterDownload;
            uredaj.StartCapture();
            Thread.Sleep(2000);
            uredaj.StopCapture();
            dataPerSec = Math.Round(dataLenght/2d,2);
            Process p =Process.GetProcessById(ProcessID);
            data = p.ProcessName + ":" + ProcessID + ":" + dataPerSec;
            ConcurrentList.DownloadBag.Add(data);
            uredaj.Close();
        }
        private void device_OnPacketReceived(object sender, CaptureEventArgs e)
        {
            dataLenght += e.Packet.Data.Length;
        }
    }
```

5.3.4 Ažuriranje informacija o odabranom procesu

Okidač koji se aktivira na klik na jedan o procesa s liste na 'Process List' kartici programa postavljen je dizajneru programa Visual Studio. Navedeni okidač poziva 'Lista_procesa_prikaz_MouseClick' metodu. Ta metoda prvo omogućuje stavljanje kvačice na prostor za kvačicu 'Add to WatchDog', zatim provjerava ako je kvačica već stavljena, te ju uklanja. Nakon toga očiste se svi prostori gdje se ispisuju podatci te se resetiraju vrijednosti varijabli statične klase 'Network_Proc_Stats'. Nakon toga metoda pokreće dvije nove niti. Metoda također prvo zaustavlja niti pokrenute pri prijašnjem klikanju na jedan od procesa s liste.

```
private void Lista_procesa_prikaz_MouseClick(object sender, MouseEventArgs e)
        {
            refresh = true;
            WatchCheckBox.Enabled = true;
            if(WatchCheckBox.Checked)
            {
                WatchCheckBox.Checked = false;
            }
            RamTextBox.Text = CpuTextBox.Text = Description_textbox.Text =
NetworkTextBox.Text=AddDescriptionTexBox.Text="";
            ProcIDlabel.Text = ImeProcesaLabel.Text = "";
            Network_Proc_Stats.Reset();
            try
            {
                var proc =
System.Diagnostics.Process.GetProcessById(Int32.Parse(Lista_procesa_prikaz.Items[Lista_pr
ocesa_prikaz.FocusedItem.Index].SubItems[1].Text));
                if (Clicked)
                {
                    ThreadRefresh.Abort();
                    if (NetworkRefresh.ThreadState ==
System.Threading.ThreadState.Background)
                        NetworkRefresh.Abort();
                }
                ThreadRefresh = new Thread(() => ProcesKlik(proc));
                ThreadRefresh.IsBackground = true;
                ThreadRefresh.Start();
                MainNetwork GetStats = new MainNetwork();
                NetworkRefresh = new Thread(() => GetStats.GetStatistics(proc.Id));
                NetworkRefresh.IsBackground = true;
                NetworkRefresh.Start();
            }
            catch(System.ArgumentException)
            {
                WatchCheckBox.Enabled = false;
                MessageBox.Show("Chosen process is not running.");
            }
        }
```

Jedna od dvije niti koje pokreće metoda 'Lista_procesa_prikaz_MouseClick' je nit koja osvježava informacije o procesu kao što su iskorištenost ram memorije i procesora te opisi procesa. Metoda koja se izvršava na toj niti je 'ProcesKlik'. Ova nit se u aplikaciji može završiti na 2 načina: Korištenjem naredbe 'ThreadRefresh.Abort()' ili postavljanjem bool varijable 'refresh' na neistinu.

```
private void ProcesKlik(Process proc)
        {
            Clicked = true;
            int cpucount = Environment.ProcessorCount;
            string path = Path.Combine(Environment.CurrentDirectory, @"DescriptionsLib",
proc.ProcessName+".txt");
            string DefaultOpis,opis;
            LabelChange(ImeProcesaLabel, proc.ProcessName);
            LabelChange(ProcIDlabel, proc.Id.ToString());
            try
            {
                string putanja = proc.MainModule.FileName;
                FileVersionInfo informacijeProcesa =
FileVersionInfo.GetVersionInfo(putanja);
                DefaultOpis = informacijeProcesa.FileDescription;
            }
            catch (Win32Exception)
            {
                DefaultOpis = "Default description not available.";
            }
            if (!File.Exists(path))
                opis = "No aditional info.";
            else
                opis = File.ReadAllText(path);
            TextBoxChange(Description_textbox, DefaultOpis);
            TextBoxChange(AddDescriptionTexBox,opis);
            do
            {
                try
                {
                    proc.Refresh();
                    PerformanceCounter CpuCounter = new PerformanceCounter("Process", "%
Processor Time", proc.ProcessName);
                    CpuCounter.NextValue();
                    Thread.Sleep(1000);
                    float postotak = (float)CpuCounter.NextValue() / cpucount;
                    TextBoxChange(CpuTextBox, "Processor usage:" +
postotak.ToString("n1") + "%");
                    bytesTo Mb = new bytesTo(proc.WorkingSet64);
                    TextBoxChange(RamTextBox, "Ram usage:" + Mb.ToMb().ToString() +
"Mb");
                    bytesTo Mb_rec = new bytesTo(Network_Proc_Stats.NetReceivedBytes);
                    bytesTo Mb_sent = new bytesTo(Network_Proc_Stats.NetSendBytes);
                    TextBoxChange(NetworkTextBox, "Downloaded:" + Mb_rec.ToMb() + "Mb" +
", Uploaded:" + Mb_sent.ToMb() + "Mb");
                }
                catch (InvalidOperationException)
                {
                    refresh = false;
                    WatchCheckBox.Enabled = false;
                    WatchCheckBox.Checked = false;
                    MessageBox.Show("Process is terminated.");
                }
            }
```

```
            while (refresh);
        }
```

'ProcesKlik' metoda za mjenjanje sadržaja labela i prostora za text koristi metode 'LabelChange' i 'TextBoxChange'. Te metode koriste 'Invoke' objekt koji omogućava popunjavanje labela i prostora za text(*textbox*) s drugih niti.

```
delegate void LabelChangeCallback(Label label, string text);
private void LabelChange(Label label, string text)
        {
            if (this.InvokeRequired)
            {
                LabelChangeCallback deg = new LabelChangeCallback(LabelChange);
                this.Invoke(deg, new object[] { label, text });
            }
            else
                label.Text = text;
        }
delegate void TextBoxChangeCallback(TextBox textbox, string text);
        private void TextBoxChange(TextBox textbox, string text)
        {
            if (this.InvokeRequired)
            {
                TextBoxChangeCallback deg = new TextBoxChangeCallback(TextBoxChange);
                this.Invoke(deg, new object[] { textbox, text });
            }
            else
                textbox.Text = text;
        }
```

Druga od dvije niti koje pokreće metoda 'Lista_procesa_prikaz_MouseClick' je nit koja osvježava informacije o mrežnom prometu odabranog procesa. Dobavljanje portova koje odabrani proces koristi obavlja se putem naredbe 'netstat –ano', koju izvršava klasa 'MainNetwork', na identičan način kao što to radi i prije opisana klasa 'ProcessNetList'. Jedina razlika je način funkcioniranja objekata koji bilježe količinu poslanih i primljenih podataka odabranog procesa. Za to se koriste klase 'Dowload' i 'Upload'. Metoda 'ReceivedPackets' klase 'Download' prvo stvara okidač koji se aktivira na primanje paketa podataka, zatim poslavlja filter te započinje nadgledanje paketa podataka. Nakon toga koristi se naredba 'SpinWait.SpinUntil(CheckDownload)' koja zadržava nit aktivnom sve dok se ne ispuni uvjet 'CheckDownload', odnosno sve dok metoda 'CheckDownload' ne vrati istinu kao rezultat. Metoda 'CheckDownload' provjerava varijablu 'shutdown' statične klase 'Network_Proc_Stats' koju na istinu postavlja metoda 'Reset' koja se poziva prilikom klika na proces s liste procesa. Varijablu 'Refresh' na neistinu postavlja tipka 'Stop Refreshing' s desne strane aplikacije.

```
class Download
    {
        string FilterDownload;
        ICaptureDevice device;
        public Download(string filter,ICaptureDevice dev)
        {
            FilterDownload = filter;
            device = dev;
        }
        public void ReceivedPackets()
        {
            device.OnPacketArrival += new
PacketArrivalEventHandler(device_OnPacketReceived);
            device.Open(DeviceMode.Promiscuous);
            device.Filter = FilterDownload;
            device.StartCapture();
            SpinWait.SpinUntil(CheckDownload);
            try
            {
                device.StopCapture();
                device.Close();
            }
            catch
            {
                device.Close();
            }
        }
        private static void device_OnPacketReceived(object sender,CaptureEventArgs e)
        {
            NetReceivedBytes += e.Packet.Data.Length;
        }
        private static bool CheckDownload()
        {
            if (shutdown)
                return true;
            else
                return false;
        }
    }
```

Klasa 'Upload' funkcionira na identičan način kao i klasa 'Download'.

```
class Upload
    {
        string FilterUpload;
        ICaptureDevice device;
        public Upload(string filter, ICaptureDevice dev)
        {
            FilterUpload = filter;
            device = dev;
        }
        public void SentPackets()
        {
            device.OnPacketArrival += new PacketArrivalEventHandler(device_OnPacketSent);
            device.Open(DeviceMode.Promiscuous);
            device.Filter = FilterUpload;
            device.StartCapture();
```

```
            SpinWait.SpinUntil(CheckUpload);
            try
            {
                device.StopCapture();
                device.Close();
            }
            catch
            {
                device.Close();
            }
        }
        public static void device_OnPacketSent(object sender,CaptureEventArgs e)
        {
            NetSendBytes += e.Packet.Data.Length;
        }
        private static bool CheckUpload()
        {
            if (shutdown)
                return true;
            else
                return false;
        }
    }
```

5.3.5 Crtanje grafova ukupnog zauzeća ram memorije i vremena procesora

Crtanje navedenih grafova započinje pozivanjem metode 'GraphsStart'. Ta metoda stvara objekt klase 'GraphsPoints' čijem se konstruktoru kao argumenti predaju dva prostora za crtanje grafova(*PictureBox*). To se odvija na odvojenoj niti. Metoda 'GraphsStart' također poziva metodu 'InitGraphTimer' koja svaku sekundu poziva drugu metodu koja uzrokuje aktivaciju ranije napravljenog okidača, tj. okidača koji se aktivira na ponovno crtanje izabrane kontrole(u ovom slučaju *PictureBox*).

```
private void GraphsStart()
        {
            GraphsPoints g;
            Graphs=new Thread(()=> g = new GraphsPoints(CpuPictureBox, RamPictureBox));
            Graphs.IsBackground = true;
            Graphs.Start();
            InitGraphTimer();
        }
private System.Windows.Forms.Timer GraphTimer;
        private void InitGraphTimer()
        {
            GraphTimer = new System.Windows.Forms.Timer();
            GraphTimer.Tick += new EventHandler(DrawGraph);
            GraphTimer.Interval = 1000;
            GraphTimer.Start();
        }
        private void DrawGraph(object sender, EventArgs e)
        {
                RamPictureBox.Invalidate();
                CpuPictureBox.Invalidate();
        }
```

Metode 'RamRedraw' i 'CpuRedraw' izvršavaju identičnu zadaću samo na različitim mjestima za crtanje. Te metode prvo ispunjuju prostor za crtanje bijelom bojom i iscrtavaju crne kvadratiće radi lakšeg snalaženja na grafu. Nakon toga na grafove se ucrtavaju linije uz pomoć liste točki koju na odvojenoj niti popunjava objekt klase 'GraphsPoints'.

```
private void CpuRedraw(object sender,PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(new HatchBrush(HatchStyle.Cross, Color.FromArgb(100, 100,
100), Color.FloralWhite), CpuPictureBox.ClientRectangle);
            if (GraphsPoints.CpuPoints.Count > 1)
                g.DrawLines(new Pen(new SolidBrush(Color.FromArgb(103, 203, 117)),
(float)2.5), GraphsPoints.CpuPoints.ToArray());
            CpuUtilization.Text ="Cpu usage:" + GraphsPoints.cpuUsage + "%";
        }

private void RamRedraw(object sender,PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(new HatchBrush(HatchStyle.Cross, Color.FromArgb(100, 100,
100), Color.FloralWhite), RamPictureBox.ClientRectangle);
            if (GraphsPoints.RamPoints.Count > 1)
                g.DrawLines(new Pen(new SolidBrush(Color.FromArgb(103, 203,
117)),(float)2.5), GraphsPoints.RamPoints.ToArray());
            RamUtilization.Text ="Ram Usage:" + GraphsPoints.ramUsage + "%";
        }
```

Objekt klase 'GraphsPoints' sadrži statične varijable koje imaju vrijednosti zauzeća ram memorije, vremena procesora, te varijablu koja predstavlja x os grafa. Također postoje i dvije statične liste u koje se upisuju točke za crtanje grafa. Konstruktor poziva metodu 'GetPoints' koja stvara dva brojača performansi, jedan za iskorištenost vremena procesora, a drugi za iskorištenost ram memorije, te se nakon toga popunjavaju liste.

```
class GraphsPoints
    {
        public static int graphX = 0, ramUsage, cpuUsage;
        PictureBox CpuPictureBox, RamPictureBox;
        public static List<Point> CpuPoints = new List<Point>();
        public static List<Point> RamPoints = new List<Point>();

        public GraphsPoints(PictureBox pb1, PictureBox pb2)
        {
            CpuPictureBox = pb1;
            RamPictureBox = pb2;
            GetPoints();
        }
        private void GetPoints()
        {
            while (Aa.MainGraphs)
            {
                PerformanceCounter CpuCounter = new PerformanceCounter("Processor", "%
Processor Time", "_Total");
                PerformanceCounter RamCounter = new PerformanceCounter("Memory", "%
```

```
Committed Bytes In Use");
                CpuCounter.NextValue();
                Thread.Sleep(900);
                cpuUsage = (int)Math.Round(CpuCounter.NextValue());
                ramUsage = (int)Math.Round(RamCounter.NextValue());
                int cpu = (CpuPictureBox.Height * cpuUsage) / 100;
                int ram = (RamPictureBox.Height * ramUsage) / 100;
                if (graphX > CpuPictureBox.Width)
                {
                    graphX = 0;
                    RamPoints.Clear();
                    CpuPoints.Clear();
                }
                RamPoints.Add(new Point(graphX, RamPictureBox.Height - ram));
                CpuPoints.Add(new Point(graphX, CpuPictureBox.Height - cpu));
                graphX += 2;
            }
        }
```

5.3.6 Crtanje grafova zauzeća ram memorije i vremena procesora odabranog procesa

Crtanje grafova za odabrani proces treba započeti tek kada se stavi kvačica u prostoru na kvačicu 'Add to WatchDog', pa je zato napravljen okidač kojeg aktivira promjena stanja prostora za kvačicu. Kada se stanje promjeni, ukoliko je kvačica stavljena, tada je potrebno omogućiti sadržaj kartica 'Cpu' i 'Ram' te pokrenuti crtanje grafova pozivanjem metode 'ProcessGraphsStart'. Ukoliko kvačica nije stavljena potrebno je onemogućiti sadržaj kartica 'Cpu' i 'Ram' , očistiti prostor za grafove i zaustaviti crtanje grafova.

```
private void CheckBoxChange(object sender,EventArgs e)
        {
            if (WatchCheckBox.Checked)
            {
                int pid = Int32.Parse(ProcIDlabel.Text);
                foreach (Control c in tabControl1.Controls)
                {
                    if (c.Text == "Cpu" || c.Text == "Ram")
                        c.Enabled = true;
                }
                GraphOn = true;
                ProcessGraphsStart(Process.GetProcessById(pid));
            }
            else
            {
                foreach (Control c in tabControl1.Controls)
                {
                    if (c.Text == "Cpu" || c.Text == "Ram")
                        c.Enabled = false;
                }
                GraphOn = false;
                ProcessGraphsTimer.Stop();
                DisableControls();
                ClearGraphs();
            }
        }
private void ProcessGraphsStart(Process p)
```

```
        {
            ClearGraphs();
            ProcessGraphsPoints g;
            Thread t = new Thread(() => g = new ProcessGraphsPoints(ChosenProcCpuPB,
ChosenProcRamPB, p));
            t.IsBackground = true;
            t.Start();
            InitProcGraphs();
        }
```

Crtanje grafova za odabrani proces funkcionira skoro identično i crtanje grafova za ukupno zauzeće ram memorije i vremena procesora. Poziv metode 'InitProcGraphs' uzrokuje pokretanje brojača koji svake sekunde pokreće metodu 'DrawProcessGraphs' koja uzorkuje aktiviranje okidača i metodu 'SelectedProcInfo' koja osvježava i ispisuje informacije o odabranom procesu. Metoda 'ProcInfoError' poziva se ukoliko dođe do iznimke, a sama metoda čisti labele.

```
private System.Windows.Forms.Timer ProcessGraphsTimer;
        private void InitProcGraphs()
        {
            ProcessGraphsTimer = new System.Windows.Forms.Timer();
            ProcessGraphsTimer.Tick += new EventHandler(DrawProcessGraphs);
            ProcessGraphsTimer.Tick += new EventHandler(SelectedProcInfo);
            ProcessGraphsTimer.Interval = 1000;
            ProcessGraphsTimer.Start();
        }
        private void SelectedProcInfo(object sender,EventArgs e)
        {
            try
            {
                ChosenProcName.Text = ProcessGraphsPoints.proc.ProcessName;
                string procTime=
ProcessGraphsPoints.proc.TotalProcessorTime.Hours.ToString()+":" +
ProcessGraphsPoints.proc.TotalProcessorTime.Minutes.ToString()+":";
                procTime +=
ProcessGraphsPoints.proc.TotalProcessorTime.Seconds.ToString();
                ProcTimeLabel.Text = procTime;
                PIDlabel.Text = ProcessGraphsPoints.proc.Id.ToString();
                PriorityLabel.Text = ProcessGraphsPoints.proc.BasePriority.ToString();
                ThreadsNumLabel.Text = ProcessGraphsPoints.proc.Threads.Count.ToString();
            }
            catch(Win32Exception)
            {
                ProcInfoError("No access on system processes.");
            }
            catch(InvalidOperationException)
            {
                ProcInfoError("Process terminated.");
            }
        }
        private void ProcInfoError(string a)
        {
            ProcTimeLabel.Text = a;
            PIDlabel.Text = a;
            PriorityLabel.Text = a;
```

```
            ThreadsNumLabel.Text = a;
        }
```

Metode koje poziva okidač izvršavaju identičan kod kao i metode koje poziva okidač napravljen za crtanje grafova ukupnog zauzeća rama i vremena procesora.

```
private void ProcessCpuRedraw(object sender,PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(new HatchBrush(HatchStyle.Cross, Color.FromArgb(100, 100,
100), Color.FloralWhite), ChosenProcCpuPB.ClientRectangle);
            if (ProcessGraphsPoints.CpuPoints.Count > 1)
                g.DrawLines(new Pen(new SolidBrush(Color.FromArgb(103, 203, 117)),
(float)2.5), ProcessGraphsPoints.CpuPoints.ToArray());
            ProcessCpuUsage.Text = "Cpu usage:" + ProcessGraphsPoints.cpuUsage + "%";
        }

private void ProcessRamRedraw(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(new HatchBrush(HatchStyle.Cross, Color.FromArgb(100, 100,
100), Color.FloralWhite), ChosenProcRamPB.ClientRectangle);
            if (ProcessGraphsPoints.RamPoints.Count > 1)
                g.DrawLines(new Pen(new SolidBrush(Color.FromArgb(103, 203, 117)),
(float)2.5), ProcessGraphsPoints.RamPoints.ToArray());
            ProcessRamUsage.Text = "Ram Usage:" + ProcessGraphsPoints.ramUsage + "%";
        }
```

Objekt klase 'ProcessGraphPoints' koji popunjava listu točaka koja se koristi za crtanje grafova također funkcionira skoro potpuno identično kao i objekt klase 'GraphPoints'. Razlika je u tome što se koristi drugi brojač performansi procesora, te se zauzeće rama dobiva pomoću naredbe 'proc.WorkingSet64'. Također popunjavanje liste točaka će se izvršavati sve dok je stanje varijable 'GraphOn' postavljeno na istinu. Crtanje ovih grafova zaustavlja promjena stanja prostora za kvačicu, klik na neki drugi proces s liste procesa na kartici 'Process List' ili gašenje samog procesa koji je odabran.

```
class ProcessGraphsPoints
    {
        public static int graphX = 0, ramUsage, cpuUsage;
        PictureBox CpuPictureBox, RamPictureBox;
        public static Process proc;
        public static List<Point> CpuPoints = new List<Point>();
        public static List<Point> RamPoints = new List<Point>();

        public ProcessGraphsPoints(PictureBox pb1, PictureBox pb2, Process p)
        {
            CpuPictureBox = pb1;
            RamPictureBox = pb2;
            proc = p;
            GetPoints();
        }
```

```
        private void GetPoints()
        {
            var myPC = new ComputerInfo();
            while (Aa.GraphOn)
            {
                try
                {
                    proc.Refresh();
                    PerformanceCounter CpuCounter = new PerformanceCounter("Process", "%
Processor Time", proc.ProcessName);
                    CpuCounter.NextValue();
                    Thread.Sleep(900);
                    cpuUsage = (int)Math.Round(CpuCounter.NextValue() /
Environment.ProcessorCount);
                    ramUsage = (int)((double)proc.WorkingSet64/RamAmount.totalRam*100);
                    int cpu = (CpuPictureBox.Height * cpuUsage) / 100;
                    int ram = (RamPictureBox.Height * ramUsage) / 100;
                    if (graphX > CpuPictureBox.Width)
                    {
                        graphX = 0;
                        RamPoints.Clear();
                        CpuPoints.Clear();
                    }
                    RamPoints.Add(new Point(graphX, RamPictureBox.Height - ram));
                    CpuPoints.Add(new Point(graphX, CpuPictureBox.Height - cpu));
                    graphX += 2;
                }
                catch(InvalidOperationException)
                {
                    Clear();
                }
            }
            Clear();
        }
        private void Clear()
        {
            RamPoints.Clear();
            CpuPoints.Clear();
            graphX = 0;
        }
    }
```

5.3.7 Okidači promjene odabrane vrijednosti padajućih izbornika

Okidač padajućeg izbornika za izbor mrežnog adaptera aktivira se na promjenu odabrane stavke. Metoda 'FocusItemChange' prvo provjerava postoje li dostupni mrežni adapteri za odabir, a nakon toga poziva metodu 'StopThread' koja zaustavlja nit koja osvježava listu procesa koji stvaraju mrežni promet. Nakon što metoda 'FocusItemChange' pronađe, po imenu, odgovarajući adapter iz liste adaptera koju daje metoda klase 'System.Net.NetworkInformation.NetworkInterface' , onda zapisuje MAC i IP adresu odabranog adaptera kako bi se isti adapter mogao pronaći u klasi 'CaptureDeviceList' biblioteke 'SharpPcap'. Razlog tomu je, prije navedena, razlika u numeriranju mrežni adaptera između biblioteka 'SharpPcap' i 'System.Net.NetworkInformation'. Naposljetku se indeks i IP adresa

adaptera spremaju u varijable statične klase 'AdapterInfo' te se ponovno pokreće nit na kojoj se izvršava osvježavanje liste procesa koji stvaraju mrežni promet.

```
private void FocusItemChange(object sender,EventArgs e)
        {
            if (NetworkComboBox.Items[0].ToString() != "No network adapter detected.")
            {
                StopThread();
                string SelectedAdapterName = NetworkComboBox.SelectedItem.ToString();
                string mac = "";
                foreach (NetworkInterface inf in
NetworkInterface.GetAllNetworkInterfaces())
                {
                    if ((inf.NetworkInterfaceType == NetworkInterfaceType.Ethernet ||
inf.NetworkInterfaceType == NetworkInterfaceType.Wireless80211) && inf.Name ==
SelectedAdapterName)
                    {
                        IPAddress ip = inf.GetIPProperties().UnicastAddresses[1].Address;
                        AdapterInfo.LocalIP = ip;
                        mac = inf.GetPhysicalAddress().ToString();
                    }
                }
                CaptureDeviceList list = CaptureDeviceList.Instance;
                for (int i = 0; i < list.Count; i++)
                {
                    list[i].Open();
                    if (list[i].MacAddress.ToString() == mac)
                        AdapterInfo.AdapterIndex = i;
                }
                StartThread();
            }
        }
```

Pri promjeni odabrane vrijednosti padajućeg izbornika koji sadrži ram čipove aktivira se okidač koji poziva metodu 'RamBoxFocusChange'. Ta metoda upitom u klasu 'Win32_PhysicalMemory dobavlja informacije o odabranom ram čipu.

```
private void RamBoxFocusChange(object sender,EventArgs e)
        {
            ManagementObjectSearcher memory = new ManagementObjectSearcher("Root\\CIMV2",
"select * from Win32_PhysicalMemory");
            foreach(ManagementObject obj in memory.Get())
            {
                string item = RamComboBox.SelectedItem.ToString();
                int position = item.LastIndexOf(":");
                string MBOPosition = item.Substring(position + 1);
                if(MBOPosition== obj["DeviceLocator"].ToString())
                {
                    CapacityLabel.Text =
((ulong)obj["Capacity"]/1024/1024/1024).ToString()+"Gb";
                    ClockSpeedLabel.Text = obj["ConfiguredClockSpeed"].ToString()+"Mhz";
                    MaxClockSpeedLabel.Text = obj["Speed"].ToString()+" Mhz";
                    FormFactorLabel.Text = obj["FormFactor"].ToString();
                    switch(obj["FormFactor"].ToString())
                    {
```

```
                        case "0":
                            FormFactorLabel.Text = "Unknown";
                            break;
                        case "1":
                            FormFactorLabel.Text = "Other";
                            break;
                        case "2":
                            FormFactorLabel.Text = "SIP";
                            break;
                        case "3":
                            FormFactorLabel.Text = "DIP";
                            break;
                        .
                        .
                        .
                        .
                        case "23":
                            FormFactorLabel.Text = "LGA";
                            break;
                    }
                    SerialNumberLabel.Text = obj["SerialNumber"].ToString();
                    MemoryTypeLabel.Text = obj["MemoryType"].ToString();
                    switch(obj["MemoryType"].ToString())
                    {
                        case "0":
                            MemoryTypeLabel.Text = "Unknown";
                            break;
                        case "1":
                            MemoryTypeLabel.Text = "Other";
                            break;
                        .
                        .
                        .
                        .
                        case "25":
                            MemoryTypeLabel.Text = "FBD2";
                            break;
                    }
                    break;
                }
            }
        }
```

U prethodnom kodu nisu ispisane sve 'case' mogućnosti jer se kod ponavlja. Faktor oblika(eng. *form factor*) u programu se dobija brojevima gdje: 0 predstavlja 'Unknown', 1 predstavlja 'Other', 2 predstavlja 'SIP', 3 predstavlja 'DIP', 4 predstavlja 'ZIP', 5 predstavlja 'SOJ', 6 predstavlja 'Proprietary', 7 predstavlja 'SIMM', 8 predstavlja 'DIMM', 9 predstavlja 'TSOP', 10 predstavlja 'PGA', 11 predstavlja 'RIMM', 12 predstavlja 'SODIMM', 13 predstavlja 'SRIMM', 14 predstavlja 'SMD', 15 predstavlja 'SSMP', 16 predstavlja 'QFP', 17 predstavlja 'TQFP', 18 predstavlja 'SOIC', 19 predstavlja 'LCC', 20 predstavlja 'PLCC', 21 predstavlja 'BGA', 22 predstavlja 'FPBGA', a 23 predstavlja 'LGA'.

Tip memorije također je predstavljen brojem gdje: 0 predstavlja 'Unknown', 1 predstavlja 'Other', 2 predstavlja 'DRAM', 3 predstavlja 'Synchronous 'DRAM', 4 predstavlja 'Cache DRAM', 5 predstavlja 'EDO', 6 predstavlja 'EDRAM', 7 predstavlja 'VRAM', 8 predstavlja 'SRAM', 9 predstavlja 'RAM', 10 predstavlja 'ROM', 11 predstavlja 'Flash', 12 predstavlja 'EEPROM', 13 predstavlja 'FEPROM'', 14 predstavlja 'EPROM', 15 predstavlja 'CDRAM', 16 predstavlja '3DRAM', 17 predstavlja 'SDRAM', 18 predstavlja 'SGRAM', 19 predstavlja 'RDRAM', 20 predstavlja 'DDR', 21 predstavlja 'DDR2', 22 predstavlja 'DDR2 FB-DIMM',24 predstavlja 'DDR3', a 25 predstavlja 'FBD2'.

Na padajući izbornik koji sadrži mrežne adaptere postavljen je i okidač koji se aktivira kada se kursor miša zadrži na odabranoj stavci. Taj okidač poziva metodu 'mouseRest' koja pruža dodatne informacije, tj. opis i MAC adresu odabranog adaptera.

```
private void mouseRest(object sender, EventArgs e)
        {

            string info = "";
            NetworkInterface[] devs = NetworkInterface.GetAllNetworkInterfaces();
            foreach(NetworkInterface inf in devs)
            {
                if (NetworkComboBox.SelectedItem.ToString() == inf.Name)
                    info = inf.Description + ",MAC address:"+inf.GetPhysicalAddress();
            }
            NetworkAdapterToolTip.SetToolTip(NetworkComboBox, info);
        }
```

5.3.8 Okidači postavljeni na prostor za pretragu

Postoje 3 okidača postavljena na prostor za unos teksta za pretragu(*TextBox).* Jedan se aktivira kada korisnik klikne unutar prostora za pretragu(poziva metodu 'SearchBoxEnter'), drugi se aktivira kada se pritisne tipka s tipkovnice dok je prostor za pretragu u fokusu(poziva metodu 'SearchBoxChange') i treći se aktivira kada se napusti prostor za pretragu(poziva metodu 'SearchBoxLeave'. Ulaženjem u prostor za pretragu, tj. klikom na njega, zaustavlja se brojač koji osvježava listu pokrenutih procesa. Pritiskom tipke provjerava se da li je pritisnuta tipka enter, ako je onda se obavlja pretraga i ispis nađenih rezultata. Napuštanjem prostora za pretragu, tj. klikanjem izvan njega, ponovno se pokreće brojač koji osvježava listu pokrenutih procesa.

```
private void SearchBoxLeave(object sender, EventArgs e)
        {
            SearchBox.Clear();
            Get_Process();
        }
private void SearchBoxChange(object sender,KeyEventArgs e)
        {
```

```
            if (e.KeyCode==Keys.Enter && SearchBox.Text!="")
            {
                for(int i=0;i<Lista_procesa_prikaz.Items.Count;i++)
                {

                    var item = Lista_procesa_prikaz.Items[i];
                    if (item.Text.ToLower().Contains(SearchBox.Text.ToLower()))
                    {
                        item.BackColor = SystemColors.Highlight;
                        item.ForeColor = SystemColors.HighlightText;
                    }
                    else
                    {
                        Lista_procesa_prikaz.Items.Remove(item);
                        i--;
                    }

                }
            }
        }
private void SearchBoxEnter(object sender,EventArgs e)
        {
            timer1.Stop();
        }
```

6. ZAKLJUČAK

Računalni proces je program u izvođenju koji za svoj rad zahtjeva određene resurse. U ovom završnom radu napravljena je aplikacija za nadzor računalnih procesa koja na jednostavan i jasan način prikazuje pokrenute procese i njihovo zauzeće računalnih resursa.

Aplikacija ovog završnog rada, radi svoje optimalne funkcionalnosti, koristi višenitnost pa je zbog toga, prije svega, jasno objašnjena razlika između procesa, niti, bazena niti i vlakana. Nakon toga su objašnjene biblioteke koje su se koristile za realizaciju aplikacije, te navedeni neki problemi koji su se javili prilikom izrade aplikacije.

Iako aplikacija funkcionira kako je i zamišljeno moguće je dodatno optimizirati njen rad što bi zahtijevalo više vremena za izradu. Glavni razlozi optimizacije bili bi dodatno ubrzavanje rada aplikacije kroz korištenje višenitnosti gdje god je to moguće. Također postoje ograničenja pri dobivanju informacija o procesima operacijskog sustava koje postavlja sam operacijski sustav.

Izrađena aplikacija uspješno izvršava svoj zadatak što je i testirano na više računala.

LITERATURA

[1] Microsoft API and reference catalog , https://msdn.microsoft.com/library , datum pristupa: 28.06.2017

[2] Process Class, https://msdn.microsoft.com/en-us/library/system.diagnostics.process(v=vs.110).aspx , datum pristupa: 03.09.2017

[3] System.Management Namespace, https://msdn.microsoft.com/en-us/library/system.management(v=vs.110).aspx , datum pristupa: 03.09.2017

[4] Processor Counters, https://technet.microsoft.com/en-us/library/cc938593.aspx , datum pristupa: 03.09.2017

[5] Processes and Threads, https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx , datum pristupa: 03.09.2017

[6] Computer System Hardware Classes, https://msdn.microsoft.com/en-us/library/aa389273.aspx , datum pristupa: 03.09.2017

[7] Netstat, https://technet.microsoft.com/en-us/library/bb490947.aspx, datum pristupa:03.09.2017

[8] PerformanceCounter Class, https://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter(v=vs.110).aspx, datum pristupa:03.09.2017

[9] Memory Performance Counters, https://msdn.microsoft.com/en-us/library/x2tyfybc(v=vs.71).aspx, datum pristupa:03.09.2017

[10] FileVersionInfo Class, https://msdn.microsoft.com/en-us/library/system.diagnostics.fileversioninfo(v=vs.110).aspx, datum pristupa:03.09.2017

[11] System.ComponentModel Namespace, https://msdn.microsoft.com/en-us/library/system.componentmodel(v=vs.110).aspx, datum pristupa:03.09.2017

[12] Microsoft.VisualBasic.Devices Namespace, https://msdn.microsoft.com/en-us/library/microsoft.visualbasic.devices(v=vs.110).aspx, datum pristupa:03.09.2017

[13] ComputerInfo Class, https://msdn.microsoft.com/en-us/library/microsoft.visualbasic.devices.computerinfo(v=vs.110).aspx, datum pristupa:03.09.2017

[14] System.Net.NetworkInformation Namespace, https://msdn.microsoft.com/en-us/library/system.net.networkinformation(v=vs.110).aspx, datum pristupa:03.09.2017

[15] SharpPcap, https://sourceforge.net/projects/sharppcap/, datum pristupa:03.09.2017

[16] System.Threading Namespace, https://msdn.microsoft.com/en-us/library/system.threading(v=vs.110).aspx, datum pristupa:03.09.2017

[17] Regex Class, https://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex(v=vs.110).aspx, datum pristupa:03.09.2017

[18] Collections (C#), https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections, datum pristupa:03.09.2017

[19] System.Drawing Namespace, https://msdn.microsoft.com/en-us/library/system.drawing(v=vs.110).aspx, datum pristupa:03.09.2017

[20] System.Linq Namespace, https://msdn.microsoft.com/en-us/library/system.linq(v=vs.111).aspx, datum pristupa:03.09.2017

[21] System.IO Namespace, https://msdn.microsoft.com/en-us/library/system.io(v=vs.110).aspx, datum pristupa:03.09.2017

SAŽETAK

Cilj ovog rada je bio napraviti C# aplikaciju koja će prikazati sve pokrenute procese na računalu, dati informacije o njima te prikazati njihovo zauzeće računalnih resursa. Uz pokrenute procese, aplikacija mora pružiti mogućnost odabira procesa, te ispisati detaljne informacije o odabranom procesu kao što su iskorištenost RAM memorije, iskorištenost vremena procesora, korištenje internet prometa itd. Također aplikacija mora ispisati procese koji stvaraju mrežni promet. Prikaz pokrenutih procesa i informacija o njima ostvaren je pomoću microsoft API-ja koje dolaze s instalacijom programa Visual Studio, dok je prikaz procesa koji stvaraju mrežni promet ostvaren pomoću SharpPcap API-ja. Funkcionalnost aplikacije omogućuje višenitno programiranje, a preglednost se postigla crtanjem grafova vrijednosti zauzeća RAM memorije i vremena procesora.

Ključne riječi: C#, višenitno programiranje, procesi, SharpPcap API, računalni resursi

ABSTRACT

Title: Application for monitoring computer processes and computer resources which processes use

The aim of this paper was to make C# application which will show every running process on computer, give information about them and show their usage of computer resources. Besides showing running processes on computer, application must provide ability to choose process and show detailed process information like cpu usage, ram usage, network usage etc. Also application has to show processes which create network traffic. Showing list of running processes and information about them is accomplished through usage of microsoft APIs which are contained in installation of Visual Studio program, while showing list of processes which create network traffic is archieved by using SharpPcap API. Functionality of this application is accomplished through usage of multithreading, and graphs of cpu and ram usage, shown in application, lead to clarity of what shown numbers mean.

Keywords: C#, multithreading, processes, SharpPcap API, computer resources

ŽIVOTOPIS

Dino Pečurlić je rođen 20. studenog 1995. godine u Osijeku. Završio je osnovnu školu Retfala u Osijeku, nakon čega upisuje Opću gimnaziju u Osijeku. Tijekom srednje škole sudjelovao je na natjecanjima iz informatike. 2014. godine, nakon maturiranja s vrlo dobrim uspjehom, upisuje Elektrotehnički fakultet u Osijeku, smjer računarstvo.

Vlastoručni potpis:

_____

PRILOZI

P1 SharpPcap tutorijal, https://www.codeproject.com/Articles/12458/SharpPcap-A-Packet-Capture-Framework-for-NET, datum pristupa: 03.09.2017