Analiza utjecaja potrošnje radne memorije na performanse procesa

Domagoj Klasić

Varaždin

14. veljače 2009

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Domagoj Klasić

Analiza utjecaja potrošnje radne memorije na performanse procesa

Varaždin, 2009.

Sadržaj

UVOD **1**

1. POTROŠNJA MEMORIJE I PERFORMANSE PROCESA **2**

2. ANALIZA U PRAKSI – FIREFOX ULTIMATE OPTIMIZER **4**

2.1 UVODNA OPAŽANJA 4

2.2 DETALJNIJA ANALIZA – PROCESS EXPLORER 5

ZAKLJUČAK **8**

LITERATURA **9**

Uvod

Radna memorija, jedna je od osnovnih komponenti računala i nerijetko nam služi kao indikator njegove brzine. Pa tako, za računalo koje ima više radne memorije kažemo da je brže. S druge strane, kada govorimo o aplikacijama koje koristimo na računalu onda više cijenimo one aplikacije koje koriste manje radne memorije, te za njih kažemo da su brže i efikasnije.

To bi značilo da je najbrže ono računalo koje ima puno neiskorištene memorije, budući da na njemu koristimo samo „brze“ aplikacije. Nije li to onda dodatni trošak?

Naravno, u praksi takvo razmišljanje ne prolazi, a obično se pojavljuje zbog nerazumijevanja mehanizama upravljanja i uporabe radne memorije računala u modernim operacijskim sustavima. Ovaj kratak rad u prvom dijelu ukratko opisuje način na koji aplikacije koriste radnu memoriju na Windows operacijskom sustavu. Drugi dio rada na praktičnom primjeru prikazuje primjenu opisane tehnike analize uporabe radne memorije. Osim toga, praktični primjer otkriva nerazumijevanje tog mehanizma koje se pojavljuje kod laika, a često i kod programera koji stoje iza različitih aplikacija.

1. Potrošnja memorije i performanse procesa

Kako bi mogli provesti analizu potreban nam je uvid u potrošnju memorijskih resursa promatranog procesa. Sam operacijski sustav detaljno bilježi sve resurse procesa koje on koristi, a radna memorija nije iznimka. Ta činjenica nam olakšava analizu budući da ne moramo izraditi vlastiti mehanizam praćenja potrošnje memorije. Uvid u dodijeljene resurse, operacijski sustav omogućuje preko niza različitih brojača. Brojači vezani uz potrošnju radne memorije koje je potrebno pratiti su:

Potrošnja fizičke memorije

- ***Working Set (WS)*** – Ukupna količina memorijskih stranica iz virtualne memorije koje je proces nedavno koristio. Te stranice se nalaze u fizičkoj memoriji te u skladu s time, taj brojač služi kao indikator potrošnje fizičke memorije.

Potrošnja virtualne memorije

- ***Private Bytes (PB)*** – Ukupna količina virtualnih memorijskih stranica koje je proces alocirao za vlastitu uporabu. Ove memorijske stranice ne mogu se dijeliti s drugim procesima na sustavu i pod direktnom su kontrolom pojedinog procesa.

Količina fizičke memorije (WS) koju proces ima na raspolaganju nije pod kontrolom procesa, već samog operacijskog sustava. Operacijski sustav procesu dodjeljuje maksimalnu količinu fizičke memorije koju on smije iskoristiti. Točan iznos odraz je ponašanja procesa koje operacijski sustav prati i količine virtualne memorije (PB) koju je proces alocirao. Koje stranice iz virtualne memorije će biti prisutne u fizičkoj memoriji, odluka je operacijskog sustava.

Ukoliko proces u svome radu iz virtualne memorije dohvati memorijsku adresu koja se trenutno ne nalazi u njemu raspoloživoj fizičkoj memoriji nastati će hardverski prekid čijom obradom će operacijski sustav, s diska, traženu memorijsku stranicu prebaciti u fizičku memoriju te potom nastaviti izvršavanje procesa.
Bitno je uočiti, da, ni u kojem slučaju, proces nema kontrolu nad fizičkom memorijom, što je odlika svih modernih operacijskih sustava.

Iz navedenog zaključujemo da na performanse procesa utječe razlika između virtualne memorije koju je proces alocirao i fizičke memorije koja mu je dostupna.

Ukoliko je razlika pozitivna, proces ima više virtualne memorije nego što mu je dostupno fizičke memorije te će u svojemu radu više puta generirati prekide u kojemu će operacijski sustav morati stranice iz virtualne memorije prebaciti u fizičku memoriju, što naravno zahtjeva dulje vrijeme obrade. Vjerojatnost generiranja prekida proporcionalna je veličini razlike.

Za razliku koja je blizu nule ili negativna, količina fizičke memorije podjednaka je ili veća od količine virtualne memorije te je vjerojatnost generiranja prekida mala ili jednaka nuli. Kako neće biti mnogo prekida, proces će pokazivati bolje performanse.

Kako bi pratili prekide koje proces generira prilikom rada s stranicama virtualne memorije koja nije u fizičkoj memoriji može nam poslužiti još jedan brojač. Riječ je o brojaču ***Page Fault (PF)***. On prikazuje ukupan broj prekida izazvanih radom s stranicama virtualne memorije koje se ne nalaze u fizičkoj memoriji.

Za proces koji ima znatno više alocirane virtualne memorije nego što mu je dostupno fizičke i aktivno ju koristi vrijednost tog brojača će biti iznimno visoka. Zbog toga nam taj brojač može poslužiti kao brzi način određivanja performansi procesa.

2. Analiza u praksi – Firefox Ultimate Optimizer

Za ilustraciju ovakve analize uzeti ćemo primjer iz prakse. Analizirati ćemo program pod nazivom *Firefox Ultimate Optimizer*[1] (u daljnjem tekstu FUO). Autor programa tvrdi da isti znatno ubrzava rad web preglednika *Firefox* smanjujući potrošnju memorije.

Prilikom analize koristiti ćemo sljedeće alate:

- **Windows Task Manager** – program koji omogućuje pregled aktivnih procesa i resursa koji oni koriste. Isporučuje se s instalacijom Windows operacijskog sustava.
- **Process Explorer**[2] – program iste namjene kao i Windows Task Manager samo s većom funkcionalnošću i mogućnošću praćenja većeg broja resursa.
- **IL Disassembler** – program koji omogućuje prikaz IL koda za aplikaciju pisanu u .NET razvojnom okruženju. Dio je instalacije razvojnog okruženja, odnosno Visual Studio integrirane okoline za razvoj.

Opisanom analizom pokušati ćemo utvrditi da li program ispunjava tvrdnje autora.

Mnogo toga rečeno je o resursima koje web preglednik *Firefox* troši za vrijeme svoga rada. Kao jedna od njegovih najvećih mana ističe se velika potrošnja memorije, koja se nekada mjeri u nekoliko stotina megabajta – po mnogima, za jedan web preglednik, previše. Zbog toga, mnogi traže najrazličitija rješenja kako bi smanjili potrošnju memorije u web pregledniku *Firefox*. Jedno od takvih rješenja je već spomenuti FUO.

2.1 Uvodna opažanja

Prvo ćemo pokrenuti web preglednik na uobičajen način, bez pokrenutog FUO-a. U web pregledniku otvoriti ćemo 7 web stranica, svaka u svojoj kratici. Od 7 web stranica dvije će biti „zahtjevne“ web 2.0 aplikacije. To su *Gmail* i *Facebook*. Odmah nakon otvaranja svih 7 web stranica pregledati ćemo listu aktivnih procesa te provjerili koliko memorije zauzima koji proces koristeći *Windows Task Manager*. Odmah na vrhu liste nalazi se web preglednik s približnom veličinom WS-a od 100 MB-a. Situacija je prikazana na slici 1.

[1] Program je dostupan s adrese: http://firefox-ultimate-optimizer.en.softonic.com/, učitano 9. veljače 2009.
[2] Program je dostupan s adrese: http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx, učitano 9. veljače 2009

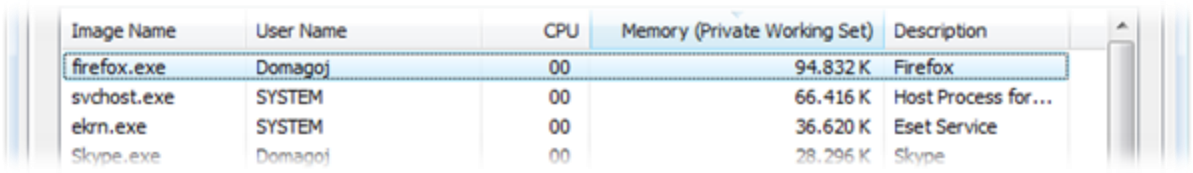
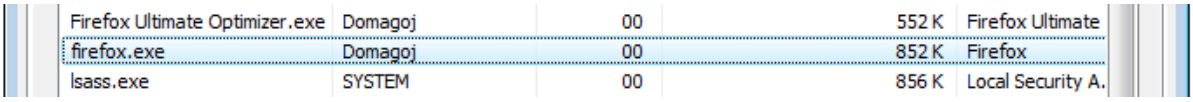


Image Name	User Name	CPU	Memory (Private Working Set)	Description
firefox.exe	Domagoj	00	94.832 K	Firefox
svchost.exe	SYSTEM	00	66.416 K	Host Process for...
ekrn.exe	SYSTEM	00	36.620 K	Eset Service
Skype.exe	Domagoj	00	28.296 K	Skype

Slika 1 - Firefox bez pokrenutnog FUO-a

Nakon toga, ne koristeći web preglednik pokrenuti ćemo FUO i pritom obratiti pozornost što se događa s veličinom WS-a. Kao što je prikazano na slici 2, vidimo znatan pad u potrošnji memorije kod web preglednika.



Firefox Ultimate Optimizer.exe	Domagoj	00	552 K	Firefox Ultimate
firefox.exe	Domagoj	00	852 K	Firefox
lsass.exe	SYSTEM	00	856 K	Local Security A.

Slika 2 - Firefox i FUO

Nakon kratke uporabe Firefoxa potrošnja memorije je lagano porasla, ali ostaje ispod 15 MB-a, što je znatno manje od standardnih uvjeta.

Odmah možemo uočiti i jednu zanimljivu činjenicu. Uporabom web preglednika potrošnja memorije počinje rasti ali u pravilnim vremenskim intervalima potrošnja memorije bilježi znatan pad. To nas upućuje da program vrlo vjerojatno koristi tajmer koji svojim otkucajima poziva funkciju programa koja smanji web pregledniku dostupnu memoriju.

2.2 Detaljnija analiza – Process Explorer

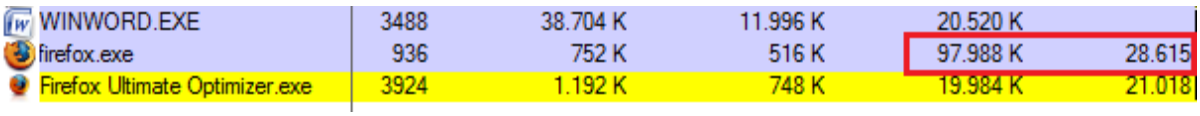
Kako bi mogli pratiti dodatne brojače koji su nam potrebni za analizu (PB i PF) koristiti ćemo alat *Process Explorer*. Slika 3 prikazuje listu procesa i pregled relevantnih brojača. Pokrenut je samo web preglednik *Firefox*, a ne i FUO.

Process	PID	Working Set	WS Private	Private Bytes	PF Delta	Private Bytes History
firefox.exe	936	110.984 K	88.392 K	97.928 K		
svchost.exe	1056	80.112 K	71.348 K	74.716 K	5	

Slika 3 - Firefox u Process Exploreru

Odnos između veličine WS-a i količine PB-a je podjednak. U skladu s time i brojač *PF Delta* (prikazuje broj PF-ova po sekundi) je na nuli.

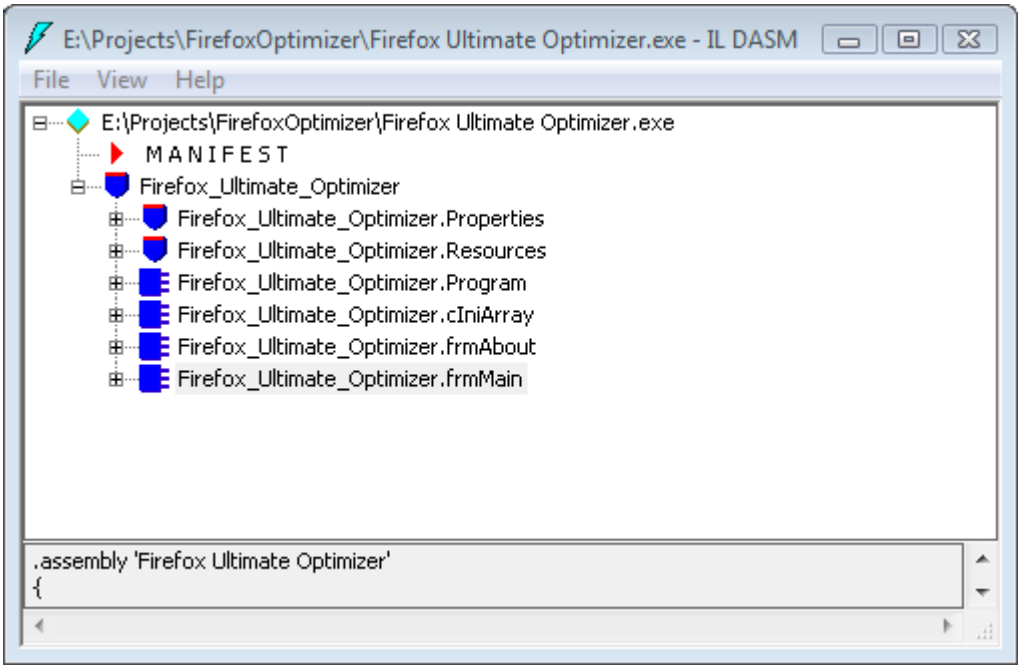
Nakon pokretanja FUO-a dolazi do znatnih promjena u količini zauzete fizičke memorije. No, količina alocirane virtualne memorije ostaje podjednaka (Brojač PB). Situacija je prikazana na slici 4.



Slika 4 - Firefox i FUO u Process Exploreru

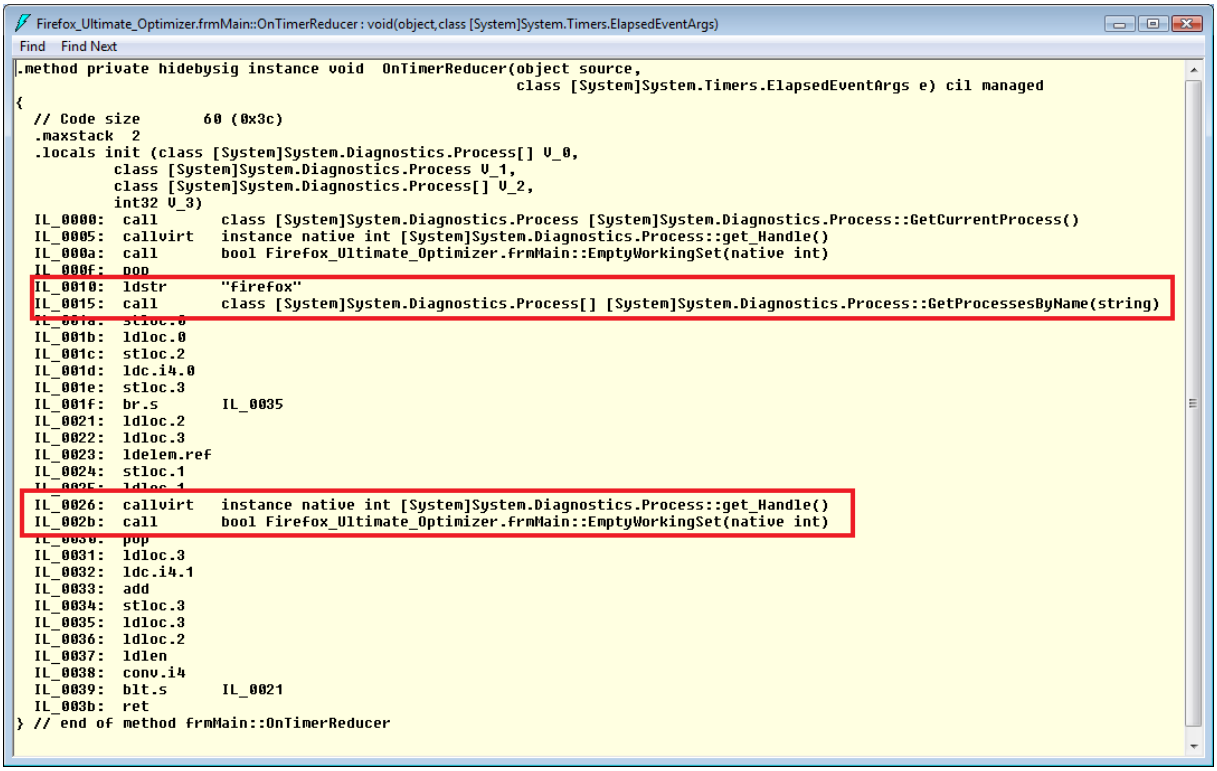
Crvenim pravokutnikom označen je i brojač *PF Delta* koji prikazuje broj PF-ova po sekundi. Vrijednost na slici se odnosi na situaciju kojoj ne koristimo web preglednik, za vrijeme korištenja brojač ima i znatno veću vrijednost. Takav broj prekida veliki je udar na performanse što se osjeti i u normalnom korištenju web preglednika.

FUO za svoj rad zahtjeva .NET izvršnu okolinu (CLR) i pisan je u .NET okruženju. Budući da je pisan u nekom od .NET jezika njegova slika na disku sadrži kod preveden u IL - intermediate language koji se tek prilikom izvođenja prevodi u strojne instrukcije. Koristeći program *IL Disassemlber* možemo zavirit u kod programa. Na sljedećoj slici je prikazan *IL Disassembler* s otvorenim FUO-om.



Slika 5 - Klase u FUO-u

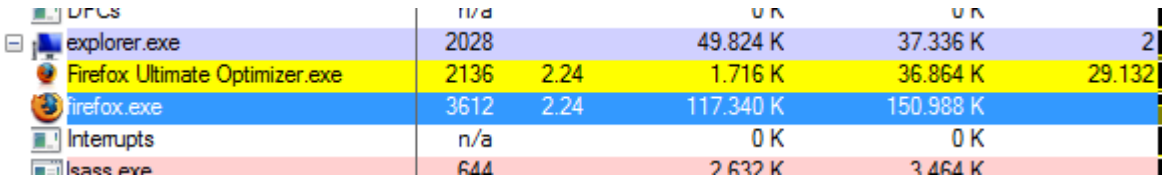
Vidimo da ima svega nekoliko klasa. Uz malo prebiranja po kodu vrlo brzo možemo vidjeti da u klasi *frmMain* postoji metoda *OnTimerReducer()*. To nam objašnjava činjenicu zašto se količina memorije koju Firefox koristi smanjuje u pravilnim vremenskim razmacima – program očito koristi programirani tajmer koji svojim otkucajima poziva spomenutu metodu, a ona pak smanjuje dostupnu memoriju web pregledniku. To nam pokazuje i kod te metode prikazan u IL jeziku na sljedećoj slici.



Slika 6 - Kod funkcije OnTimerReducer()

Crveno su označeni zanimljivi dijelovi. Prvo vidimo poziv metodi *GetProcessByName()* s argumentom *firefox* koja vraća listu objekata tipa *System.Diagnostics.Process* – za svaki proces koji odgovara tom imenu po jedan objekt. Kasnije vidimo dohvaćanje oznake procesa i poziv Win32 API funkcije *EmptyWorkingSet()*. Ta Windows API funkcija traži upravitelja memorijom (*memory manager*) da iz fizičke memorije dodijeljene procesu ukloni što je moguće više stranica. I to je razlog zašto web preglednik *Firefox* ima tako malo fizičke memorije na raspolaganju.

Kao dokaz ove činjenice možemo napraviti i mali eksperiment. *EmptyWorkingSet()* zahtjeva određene privilegije. Ukoliko FUO pokrenemo pod privilegijama standardnog korisnika, a Firefox proces pod administratorskim privilegijama on neće moći administratorskom procesu ograničiti raspoloživu količinu memorije. Takva situacija prikazana je na slici.



Slika 7 - FUO ne radi s smanjenim privilegijama

Zaključak

Nije moguće povećati performanse procesa smanjujući mu količinu dostupne memorije. Takvim postupkom dobivamo suprotan učinak. Zahtjev brzine za određenu aplikaciju ne znači da ona mora koristit što je manje moguće memorije. Upravo suprotno, radna memorija za nekoliko redova veličine je brža od diska ili nekog drugog sekundarnog spremišta podataka i poželjno je za aplikaciju da maksimalno iskoristi tu brzinu držeći što je više moguće podataka u radnoj memoriji. Osim toga, treba biti svjestan da korisnička aplikacija ne može proizvoljno utjecati na raspored i dodjeljivanje fizičke memorije na računalu – za to postoje operacijski sustavi. A performanse pojedine aplikacije (odnosno procesa) ne možemo povećati korištenjem krivih mehanizama kao što je Windows API funkcija *EmptyWorkingSet()*.

Literatura

1. Russinovich, M., Solomon D., Microsoft Windows Internals(4th Edition): Microsoft Windows Server 2003, Windows XP and Windows 2000, Microsoft Press, Washington 2005
2. http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx, učitano 9. veljače 2009
3. http://msdn.microsoft.com/en-us/library/ms682606(VS.85).aspx, učitano 9. veljače 2009