

Pe langa scrierea de cod, un aspect important in dezvoltarea de software este sa stim sa ne debuggam cat mai eficient programele noastre. In laboratorul de astazi vom discuta despre metode de debugging.

Linux:

Linux are un tool foarte puternic built in pentru debugging si anume gdb. Pe siteul cursului se afla o arhiva elemente_linux (parola linux)care contine printre altele fisierele pers.cpp, pers.h, gdb.cpp. Va rog sa le extrageti si sa rulati comanda:

```
gcc -o mytest gdb.cpp pers.cpp -g
```

A se observa -g care face compilarea impreuna cu simbolurile de debug, care ne vor permite sa navigam prin program pe masura ce se executa si de asemenea sa putem vizualiza variabilele.

Va rog sa rulati :

```
gdb mytest
```

Dupa rularea comenzii de mai sus, programul mytest e gata de rulare in gdb. Comenzi utile in gdb:

break fisier:linie – va pune un breakpoint la linia linie din fisierul fisier – aceasta inseamna ca atunci cand programul va ajunge la linia linie din fisier se va opri si vom fi lasati sa il analizam

step – o data programul oprit la un breakpoint, step va trece la urmatoarea instructiune, chiar incercand sa intre intr-o functie, daca urmatoarea instructiune este o functie. Daca exista simboluri de debug si instructiunea urmatoare este o functie se va intra in functie si executia se va opri la prima instructiune din functie

next - o data programul oprit la un breakpoint, next va trece la urmatoarea instructiune, dar nu va intra in functie, daca urmatoarea instructiune este o functie.

finish – produce iesirea din functie si afisarea valorii returnate din functie.

list – afiseaza codul din zona in care se afla programul oprit in acel moment, de asemenea list fisier:linie va afisa codul din fisierul fisier la linia linie.

bt – afiseaza stiva de apeluri pana in acel moment

Daca dorim sa vedem variabilele aflate in functiile care au dus la apelul functiei in care suntem se poate naviga pe stiva cu frame frame_number si afisa datele de acolo

print variabila – va afisa continutul variabilei.

run – face ca programul sa inceapa sa ruleze pana la primul breakpoint sau la final

continue – face ca programul sa ruleze de unde a fost oprit pana la urmatorul breakpoint sau pana la final.

Practic. Porniti cu gdb-ul programul proaspat compilat mytest. Puneti un breakpoint la linia 18 in fisierul gdb.cpp. Tastati comanda run pentru ca programul sa ruleze pana la primul breakpoint.

Afisati pers, pers.a, pers.age, pers.numa, t, errno. A se observa ca errno de fapt e un macrou care afiseaza continutul din memorie de la adresa intoarsa de __errno_location().

Intrati in functia insert_pers. Afisati din nou pe pers, pers.a, pers.age, pers.numa, t. Puteti afisa pe t? Afisati stiva si navigati in frameul superior, unde afisati pe t.

Intorceti-va la frameul curent si iesiti din functie.

Lasati programul sa ruleze pana la final.

Atasarea la un proces cu gdb.

Se poate intampla ca sa nu putem rula un program direct din gdb. De exemplu avem un daemon care s-a blocat undeva si nu fusese pornit din gdb. Putem sa ne atasam la un proces care ruleaza deja folosind comenzile :

```
ps -A | grep numele procesului
```

asa aflam pid-ul procesului la care dorim sa ne conectam.

Pornim gdb si o data pornit tastam attach numarul_pidului.

Practic

Fie programul de mai jos – salvat in attach.cpp:

```
1.#include <stdio.h>
2.#include <unistd.h>
3.int main()
4.{
5.    int t = 0;
6.    while(1)
7.        sleep(1);
8.    t++;
9.    return 0;
10.}
```

Compilati programul:

```
gcc -o att attach.cpp -g
```

Rulati att in background:

```
./att &
```

O data rulata comanda de mai sus ar trebui sa vedeti si numarul pidului cu care a fost lansat procesul de mai sus.

Porniti gdb ca root (sudo gdb)

O data pornit gdb, tastati attach numarul_pidului

Incercati sa puneti urmatoarele breakpointuri:

```
break attach.cpp:7
```

```
break attach.cpp:8
```

De ce credeti ca nu ati putut seta al doilea breakpoint?

Opriti debuggingul si modificati programul de mai sus astfel:

```
1.#include <stdio.h>
```

```
2.#include <unistd.h>
```

```
3.int main()
```

```
4.{
```

```
5.    int t = 0;
```

```
6.    while(t++<1000000)
```

```
7.        sleep(1);
```

```
8.    t++;
```

```
9.    return 0;
```

```
10.}
```

Compilati programul:

```
gcc -o att attach.cpp -g
```

Rulati att in background:

```
./att &
```

O data rulata comanda de mai sus ar trebui sa vedeti si numarul pidului cu care a fost lansat procesul de mai sus.

Porniti gdb ca root (sudo gdb)

O data pornit gdb, tastati attach numarul_pidului

Incercati sa puneti urmatoarele breakpointuri:

```
break attach.cpp:7
```

```
break attach.cpp:8
```

dati continue. Ar trebui ca acum programul sa fie oprit la linia 7. Sa iesim din bucla folosind:

```
jump attach.cpp:8
```

din acest moment putem rula restul programului , apucand sa ne conectam. Evident,atasari asemanatoare putem realiza si fara sa mai fie nevoie sleep-ul pe care l-am adaugat pentru a avea timp sa ne atasam si pentru a nu se incheia programul inainte de a apuca sa ne atasam.

Debugging memory leaks

Cel mai des utilizat tool pentru a detecta memory leakurile pe linux este valgrind. Daca avem un program pe care il rulam ./mytest, atunci pentru a vedea memory leakurile din el il putem rula cu :

```
valgrind --leak-check=yes ./mytest
```

La sfarsitul rularii veti avea afisate cate leakuri exista in program. Pentru a obtine si linia de cod unde s-a facut alocarea ar trebui ca programul nostru sa fi fost compilat cu simboluri de debugging -g.

Practic:

Scrieti un program in care sa faceti intentionat un memory leak. Rulati-l cu valgrind si vedeti daca este identificat memory leakul.

File descriptor leaks

Pe langa leakurile de memorie, o alta problema importanta sunt leakurile de file descriptori. Sa consideram codul:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
    int fd = 0;
```

```

while(1)
{
    fd = open("a.txt", O_CREAT);
    sleep(1);
}
close(fd);
return 0;
}

```

Codul este simplu, asa ca handle leakul este evident. Inchipuiti-va un program mai complex care ruleaza pe un server si care din cand in cand mai deschide cate un fisier pe care uita sa il inchida. La un moment dat sistemul va crasha programul pentru ca nu vor mai fi alti file descriptori disponibili. Cum putem analiza asa ceva?

Salvati programul de mai sus in fisierul fd_leaks.cpp.

Compilati-l:

```
gcc -o fdleak fd_leaks.cpp -g
```

Rulati-l in background:

```
./fdleak &
```

O sa aveti afisat pe ecran numarul pidului cu care a fost pornit programul.

Ca sa vedeti cati file descriptori are deschisi puteti afla ruland comanda:

```
ls /proc/fd/nr_pid/
```

O sa vedeti la fiecare rerulare ca numarul de file descriptori creste, ceea ce probabil inseamna ca avem un handle leak. Acum, ca stim ca avem un handle leak cum putem afla ce il genereaza?

O sa folosim comanda strace care ne va afisa toate apelurile sistem facute programul analizat.

```
strace ./fdleak
```

vom observa ca la fiecare secunda se deschide din nou fisierul a.txt, informatie pe care o vom folosi pentru a identifica locul de unde se face apelul.

Analiza unui program care nu merge.

Inca un utilitar interesant este strings care va spune stringurile hardcodate dintr-un executabil.

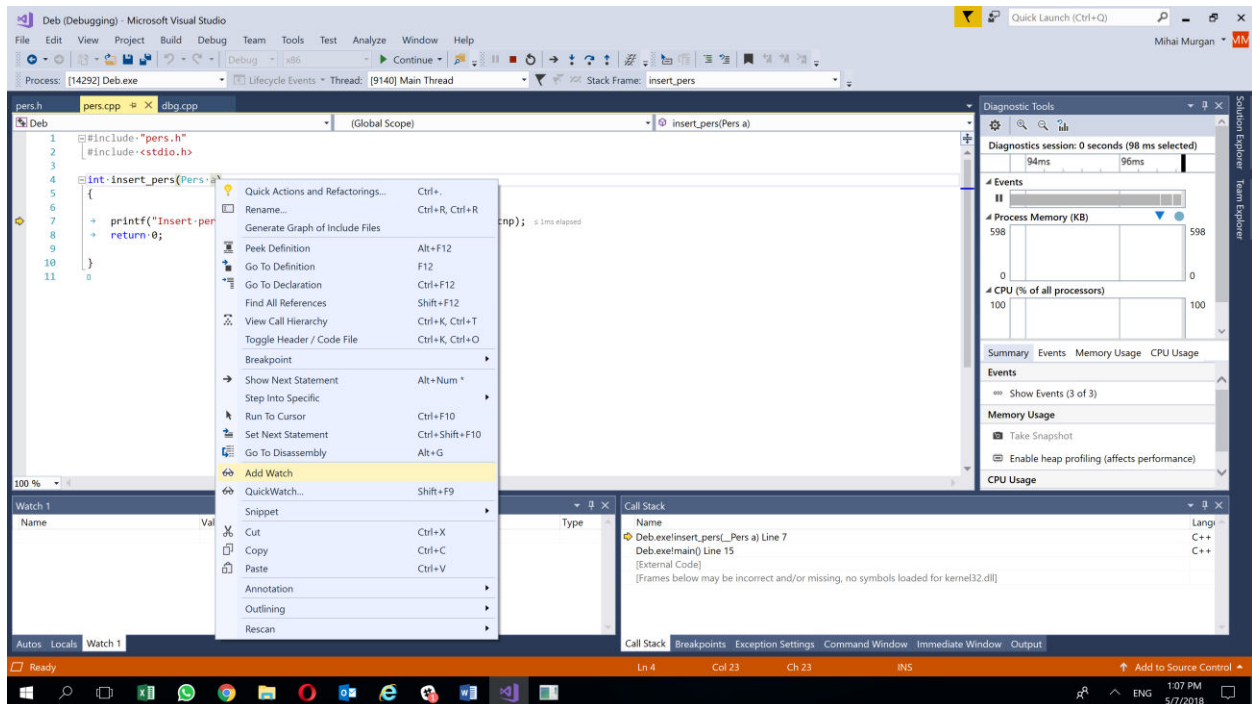
Practic. Folosind strace si strings depanati programul hunter aflat in arhiva elemente_linux si faceti-l sa mearga, evident, daca nu va descurcati puteti cere hinturi pe parcurs.

Windows:

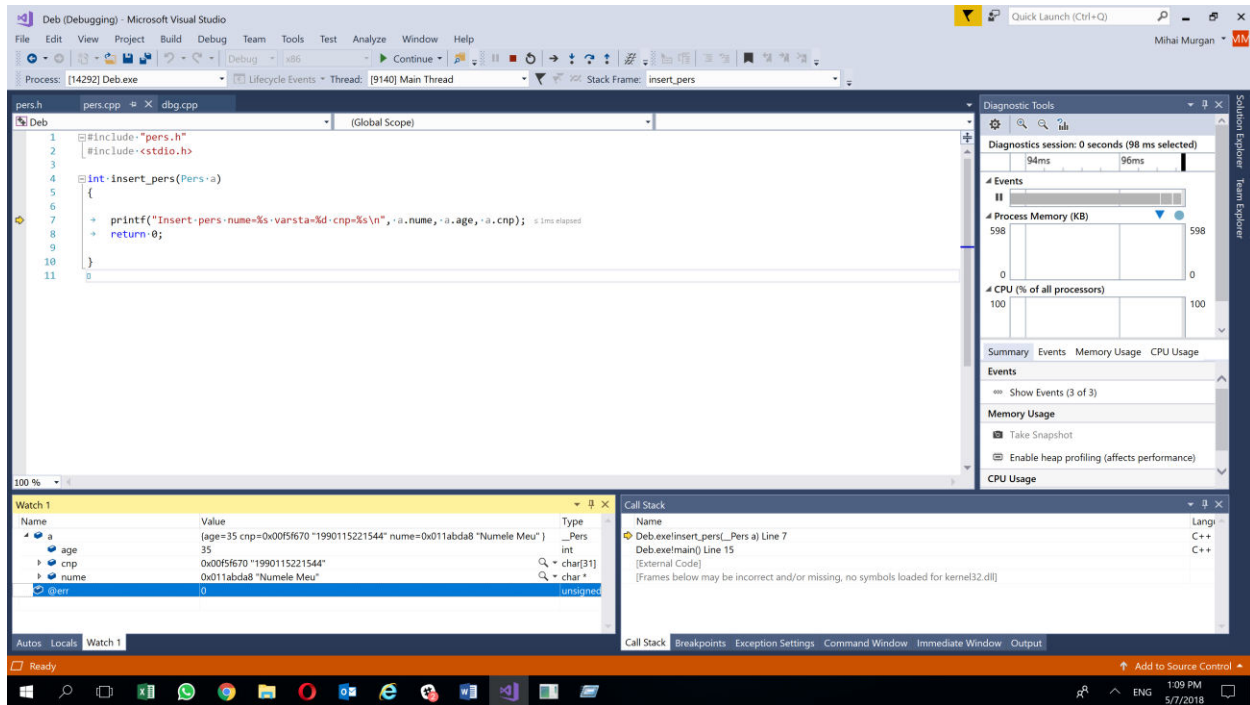
Visual Studio ofera modalitati puternice de debugging, avand debuggerul built in.

Breakpointurile in visual studio se adauga prin click in partea stanga a liniei unde se doreste adaugat break pointul sau prin punerea cursorului in locul in care se doreste adaugarea breakpointului si apasarea tastei F9.

O data pornita aplicatia din debugger si oprita intr-un breakpoint se poate naviga la instructiunea urmatoare cu F10, se poate intra intr-o functie cu F11, se poate iesi din functia curenta cu shift+F11. Vizualizarea unei variabile se poate face prin selectarea ei cu mouseul, click dreapta, add watch.



Vizualizarea valorilor se poate observa dupa adaugarea watch-ului in fereastra watch de jos



De asemenea, pentru a putea vedea in permanenta GetLastError, putem adauga intr-un camp watch (doar click stanga pe un camp liber) @err care indica last error.

De asemenea in partea de jos ar trebui sa aveti si fereastra call stack, cu ajutorul careia puteti naviga pe stiva de calluri.

Pentru windows exista pe site arhiva `elemente_windows` care contine resursele mentionate mai jos. Parola este windows

Practic. Puneti un breakpoint la linia 15 in fisierul `dbg.cpp`. Porniti programul in debugger.

Afisati `pers`, `pers.a`, `pers.age`, `pers.nume`, `t`, last error.

Intrati in functia `insert_pers`. Afisati din nou pe `pers`, `pers.a`, `pers.age`, `pers.nume`, `t`. Puteti afisa pe `t`? Folositi fereastra cu stiva de apeluri si navigati in frameul superior, unde afisati pe `t`.

Intorceti-va la frameul curent si iesiti din functie.

Lasati programul sa ruleze pana la final.

Atasarea la un proces:

Am modificat programul anterior ca mai jos, adaugand un `Sleep` la inceput.:

```
#include <stdio.h>
#include "pers.h"
#include <windows.h>

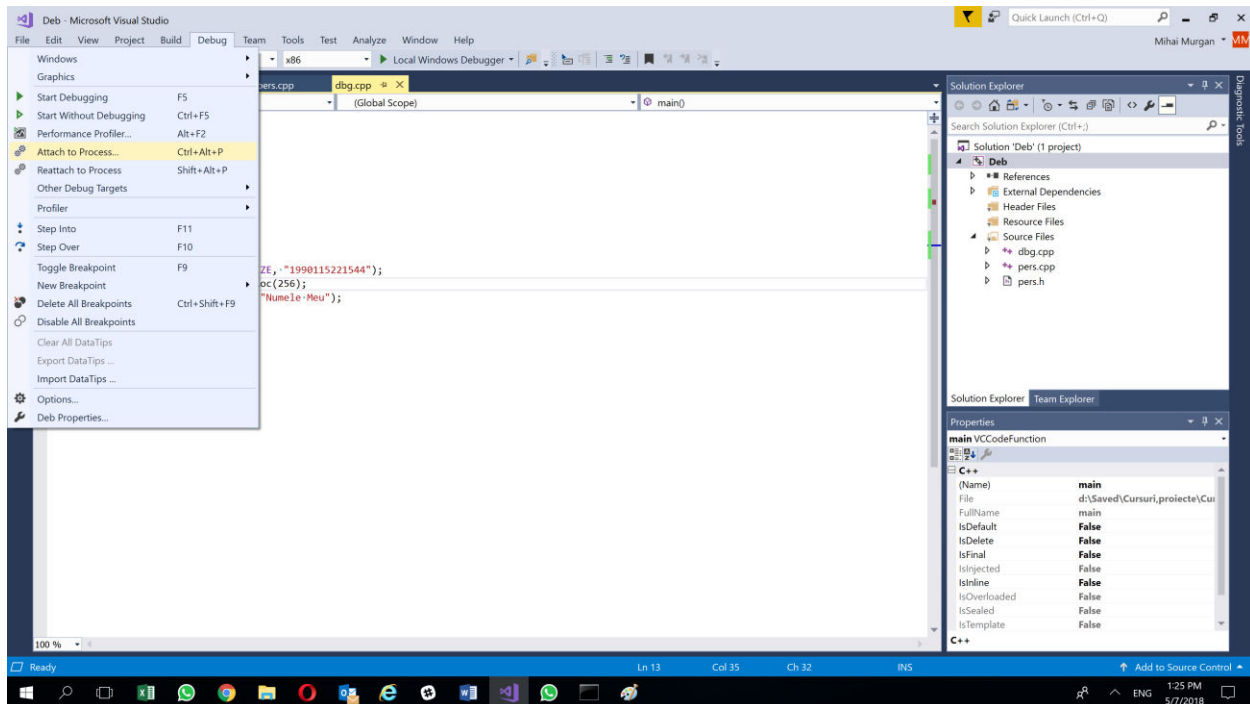
int main()
{
    while (1)
        Sleep(1000);
}
```

```

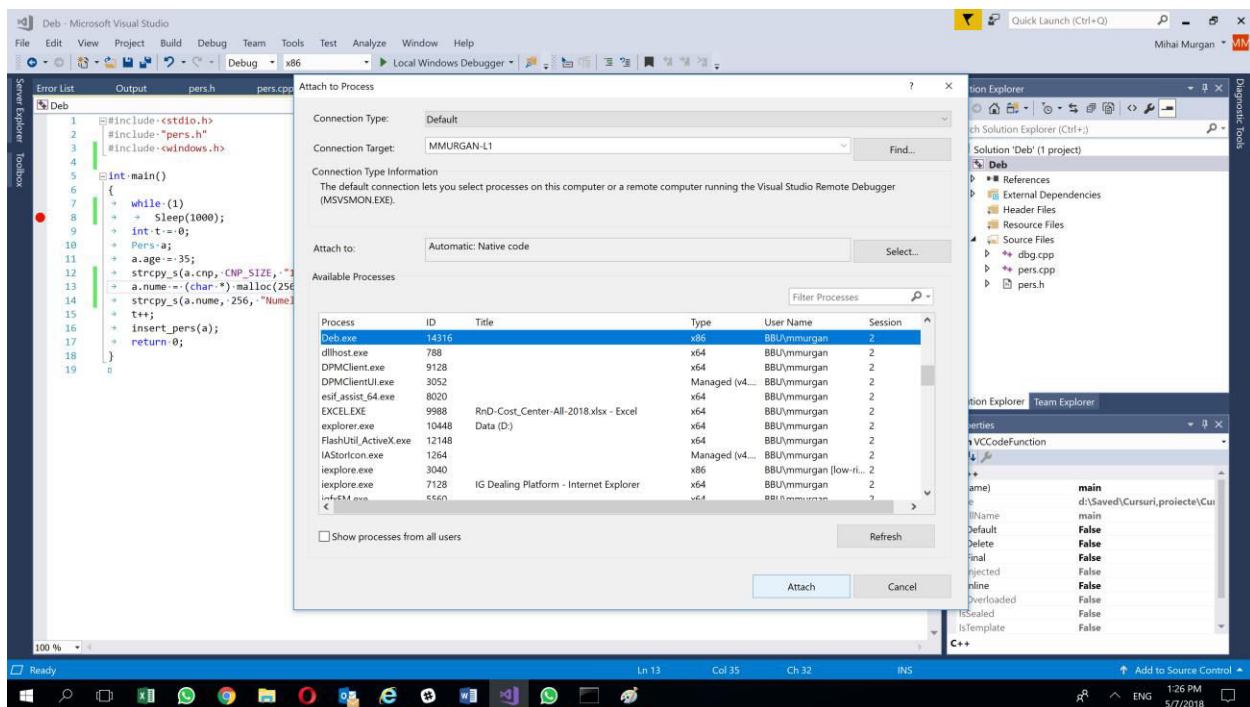
    int t = 0;
    Pers a;
    a.age = 35;
    strcpy_s(a.cnp, CNP_SIZE, "1990115221544");
    a.num = (char *) malloc(256);
    strcpy_s(a.num, 256, "Numele Meu");
    t++;
    insert_pers(a);
    return 0;
}

```

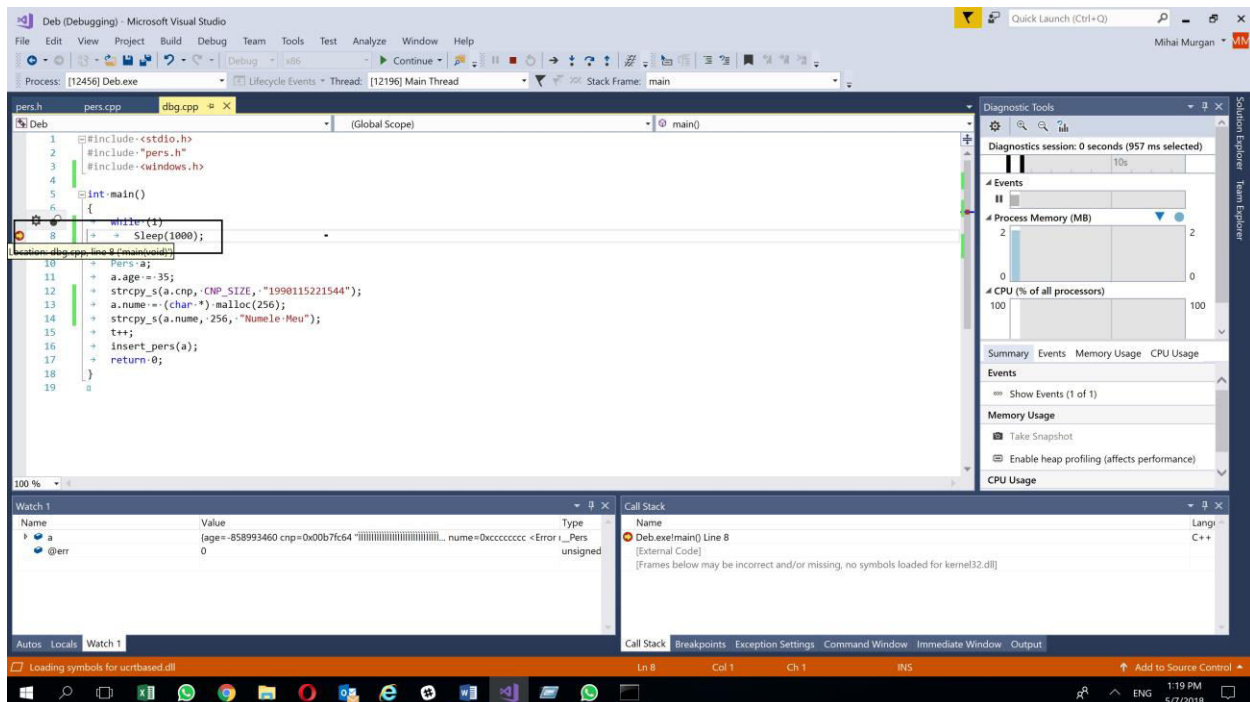
Am recompilat programul si l-am rulat din linia de comanda. Din Visual Studio ma atasez la el ca in figura de mai jos:



Si selectez procesul care ma intereseaza din lista de procese:



Pun un breakpoint in dreptul sleepului si trag efectiv cursorul la linia urmatoare, continuand executia:



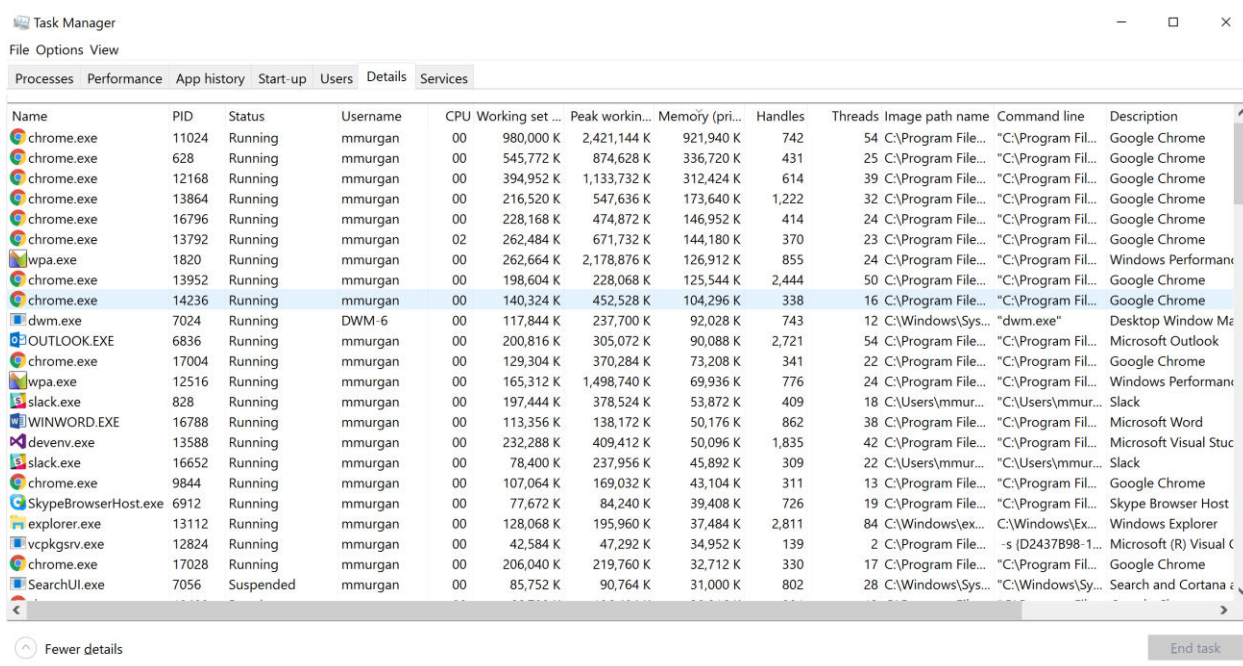
Evident,atasari asemanatoare putem realiza si fara sa mai fie nevoie sleep-ul pe care l-am adaugat pentru a avea timp sa ne atasam, aceste atasari fiind utile pentru a debuga un deadlock, un serviciu aflat intr-o stare inconsistenta, etc.

Practic

Modificati programul deb ca mai sus si refaceti pasii descrisi.

Investigare memory leak

Folosirea memoriei de catre un proces e un lucru dinamic. Astfel, la un moment dat e posibil sa fie un spike de alocare, dupa care sa revina la normal. Daca ne intereseaza un singur process cata memorie a avut alocata, probabil ca task manager e suficient ca sa ne raspunda la intrebare, uitandu-ne in zona details la Peak Working Set



Name	PID	Status	Username	CPU	Working set	Peak workin...	Memory (pri...	Handles	Threads	Image path name	Command line	Description
chrome.exe	11024	Running	mmurgan	00	980,000 K	2,421,144 K	921,940 K	742	54	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	628	Running	mmurgan	00	545,772 K	874,628 K	336,720 K	431	25	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	12168	Running	mmurgan	00	394,952 K	1,133,732 K	312,424 K	614	39	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	13864	Running	mmurgan	00	216,520 K	547,636 K	173,640 K	1,222	32	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	16796	Running	mmurgan	00	228,168 K	474,872 K	146,952 K	414	24	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	13792	Running	mmurgan	02	262,484 K	671,732 K	144,180 K	370	23	C:\Program File...	"C:\Program Fil...	Google Chrome
wpa.exe	1820	Running	mmurgan	00	262,664 K	2,178,876 K	126,912 K	855	24	C:\Program File...	"C:\Program Fil...	Windows Performanc
chrome.exe	13952	Running	mmurgan	00	198,604 K	228,068 K	125,544 K	2,444	50	C:\Program File...	"C:\Program Fil...	Google Chrome
chrome.exe	14236	Running	mmurgan	00	140,324 K	452,528 K	104,296 K	338	16	C:\Program File...	"C:\Program Fil...	Google Chrome
dwm.exe	7024	Running	DWM-6	00	117,844 K	237,700 K	92,028 K	743	12	C:\Windows\Sys...	"dwm.exe"	Desktop Window Ma
OUTLOOK.EXE	6836	Running	mmurgan	00	200,816 K	305,072 K	90,088 K	2,721	54	C:\Program File...	"C:\Program Fil...	Microsoft Outlook
chrome.exe	17004	Running	mmurgan	00	129,304 K	370,284 K	73,208 K	341	22	C:\Program File...	"C:\Program Fil...	Google Chrome
wpa.exe	12516	Running	mmurgan	00	165,312 K	1,498,740 K	69,936 K	776	24	C:\Program File...	"C:\Program Fil...	Windows Performanc
slack.exe	828	Running	mmurgan	00	197,444 K	378,524 K	53,872 K	409	18	C:\Users\mmur...	"C:\Users\mmur...	Slack
WINWORD.EXE	16788	Running	mmurgan	00	113,356 K	138,172 K	50,176 K	862	38	C:\Program File...	"C:\Program Fil...	Microsoft Word
devenv.exe	13588	Running	mmurgan	00	232,288 K	409,412 K	50,096 K	1,835	42	C:\Program File...	"C:\Program Fil...	Microsoft Visual Stuc
slack.exe	16652	Running	mmurgan	00	78,400 K	237,956 K	45,892 K	309	22	C:\Users\mmur...	"C:\Users\mmur...	Slack
chrome.exe	9844	Running	mmurgan	00	107,064 K	169,032 K	43,104 K	311	13	C:\Program File...	"C:\Program Fil...	Google Chrome
SkypeBrowserHost.exe	6912	Running	mmurgan	00	77,672 K	84,240 K	39,408 K	726	19	C:\Program File...	"C:\Program Fil...	Skype Browser Host
explorer.exe	13112	Running	mmurgan	00	128,068 K	195,960 K	37,484 K	2,811	84	C:\Windows\ex...	C:\Windows\Ex...	Windows Explorer
vcpgsrv.exe	12824	Running	mmurgan	00	42,584 K	47,292 K	34,952 K	139	2	C:\Program File...	-s (D2437B98-1...	Microsoft (R) Visual C
chrome.exe	17028	Running	mmurgan	00	206,040 K	219,760 K	32,712 K	330	17	C:\Program File...	"C:\Program Fil...	Google Chrome
SearchUI.exe	7056	Suspended	mmurgan	00	85,752 K	90,764 K	31,000 K	802	28	C:\Windows\Sys...	"C:\Windows\Sy...	Search and Cortana

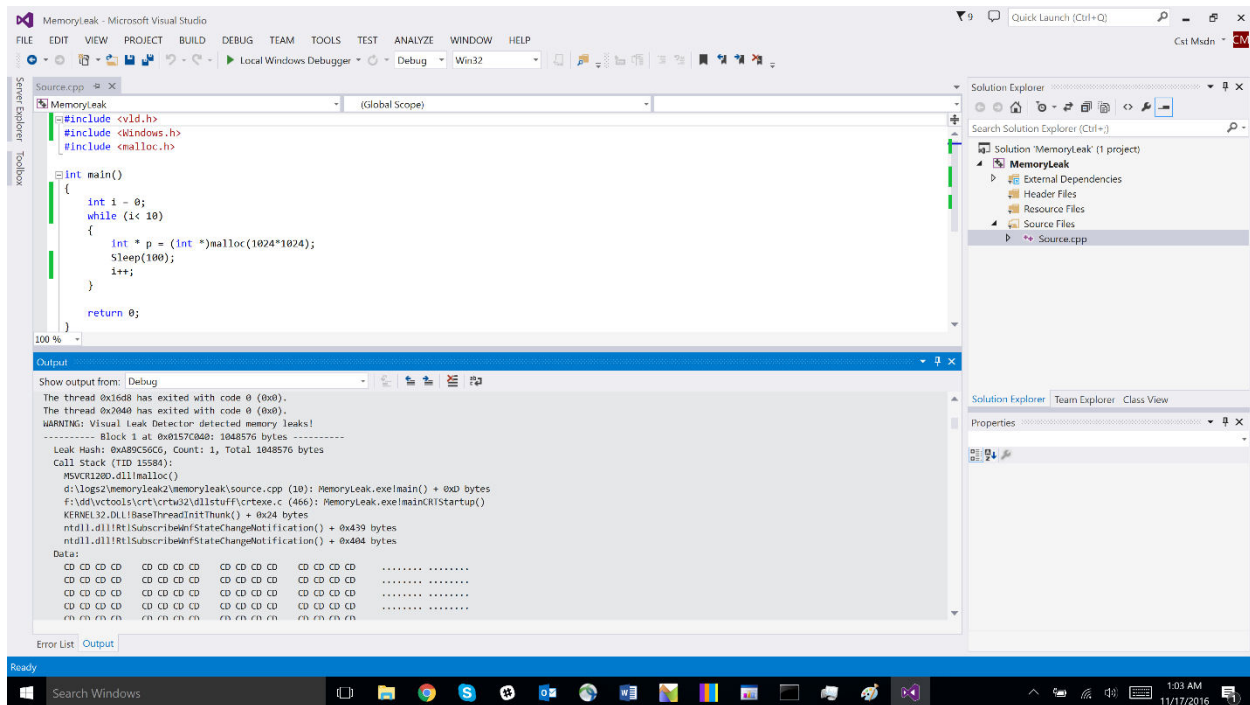
Putem observa din imaginea de mai sus ca Peak Working Set poate fi uneori semnificativ mai mare decat Working Set. (in cazul in care nu e afisat peak working set puteti adauga mai multe coloane in zona details a task managerului dand click dreapta pe zona de titlu(unde e afisat Nume, Pid, Status, etc) si selectand campurile suplimentare din fereastra care se deschide).

Ce facem cand stim ca avem, o aplicatie cu probleme, care genereaza leakuri de memorie?

Pentru identificarea memory leakurilor exista un tool foarte puternic care poate fi descarcat de pe:

<https://vld.codeplex.com/>

Se instaleaza si prin includerea fisierului vld.h in crearea programelor se vor suprascrie functiile malloc, free, new, delete, astfel fiecare alocare si dezalocare va fi traceuita si daca o alocare nu este urmata de o dezalocare ni se va da la sfarsitul programului un log cu toate leakurile detectate.



Se poate observa includerea `#include <vld.h>` (care trebuie sa fie primul include din fisierul care contine functia main) si dupa rularea programului in partea de jos se poate observa stiva de unde s-a facut alocarea fara dezalocare. In cazul in care nu va apare stiva completa cu vld, atunci modificati in felul urmator proiectul din visual studio:

Click pe proiect, in project properties click to Linker - All Options - Generate Debug Info, si de acolo selectati "Generate Debug Information optimized for sharing and publishing (/DEBUG:FULL)" instead of default "Generate Debug Information for faster links".

Practic. Scrieti un program in care sa faceti intentionat un memory leak. Rulati-l cu vld si vedeti daca este identificat memory leakul.

Handle leaks

Pe Windows, compania Sysinternals a dezvoltat mai multe tooluri foarte utile de debugging, incat pana la urma, Microsoft-ul i-a cumparat cu totul. Suita sysinternals o puteti downloada de aici:


<https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite>

Astazi vom vorbi de Process Monitor si Process Explorer. Process Monitor ne poate arata real time activitatea pe fisiere, networking, procese (incarcare de dll-uri, pornire de procese, pornire de threaduri) si registry. Process Explorer e un fel de task manager mai evoluat.

Sa pornim Process Monitor:

Process Monitor - Sysinternals: www.sysinternals.com

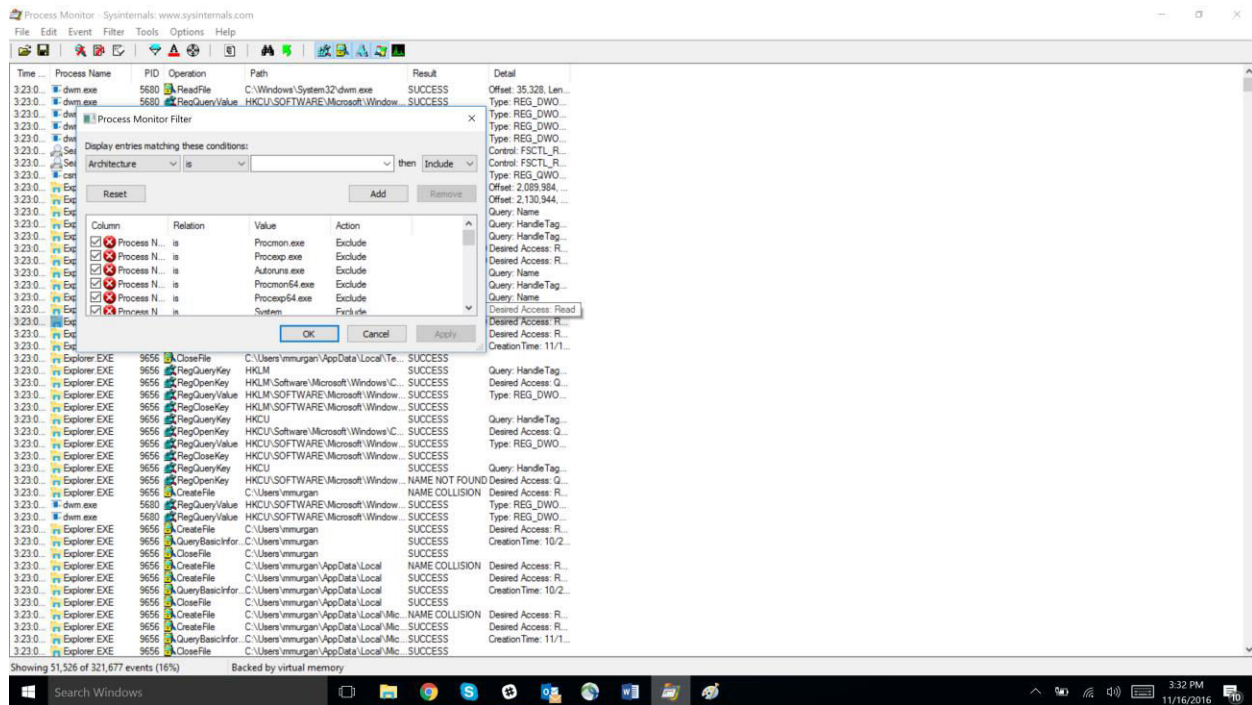
File Edit Event Filter Tools Options Help



Time ...	Process Name	PID	Operation	Path	Result	Detail
3:23:0...	dwm.exe	5680	ReadFile	C:\Windows\System32\dwm.exe	SUCCESS	Offset: 35,328, Len...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	SearchIndexer...	9424	FileSystemControlC:		SUCCESS	Control: FSCTL_R...
3:23:0...	SearchIndexer...	9424	FileSystemControlC:		SUCCESS	Control: FSCTL_R...
3:23:0...	csrss.exe	7976	RegQueryValue	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_QWO...
3:23:0...	Explorer.EXE	9656	ReadFile	C:\Windows\explorer.exe	SUCCESS	Offset: 2,089,984, ...
3:23:0...	Explorer.EXE	9656	ReadFile	C:\Windows\explorer.exe	SUCCESS	Offset: 2,130,944, ...
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCU\Software\Classes\Applications\...	NAME NOT FOUND	Desired Access: R...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCR\Applications\Procmon64.exe	NAME NOT FOUND	Desired Access: R...
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCU\Software\Classes\Applications\...	NAME NOT FOUND	Desired Access: R...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCR\Applications\Procmon64.exe	NAME NOT FOUND	Desired Access: R...
3:23:0...	Explorer.EXE	9656	CreateFile	C:\Users\vmurgan\AppData\Local\Te...	SUCCESS	Desired Access: R...
3:23:0...	Explorer.EXE	9656	QueryBasicInfor...	C:\Users\vmurgan\AppData\Local\Te...	SUCCESS	CreationTime: 11/1...
3:23:0...	Explorer.EXE	9656	CloseFile	C:\Users\vmurgan\AppData\Local\Te...	SUCCESS	
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKLM	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKLM\Software\Microsoft\Windows\C...	SUCCESS	Desired Access: Q...
3:23:0...	Explorer.EXE	9656	RegQueryValue	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	Explorer.EXE	9656	RegCloseKey	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCU\Software\Microsoft\Windows\C...	SUCCESS	Desired Access: Q...
3:23:0...	Explorer.EXE	9656	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	Explorer.EXE	9656	RegCloseKey	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	
3:23:0...	Explorer.EXE	9656	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
3:23:0...	Explorer.EXE	9656	RegOpenKey	HKCU\SOFTWARE\Microsoft\Window...	NAME NOT FOUND	Desired Access: Q...
3:23:0...	Explorer.EXE	9656	CreateFile	C:\Users\vmurgan	NAME COLLISION	Desired Access: R...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
3:23:0...	dwm.exe	5680	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...

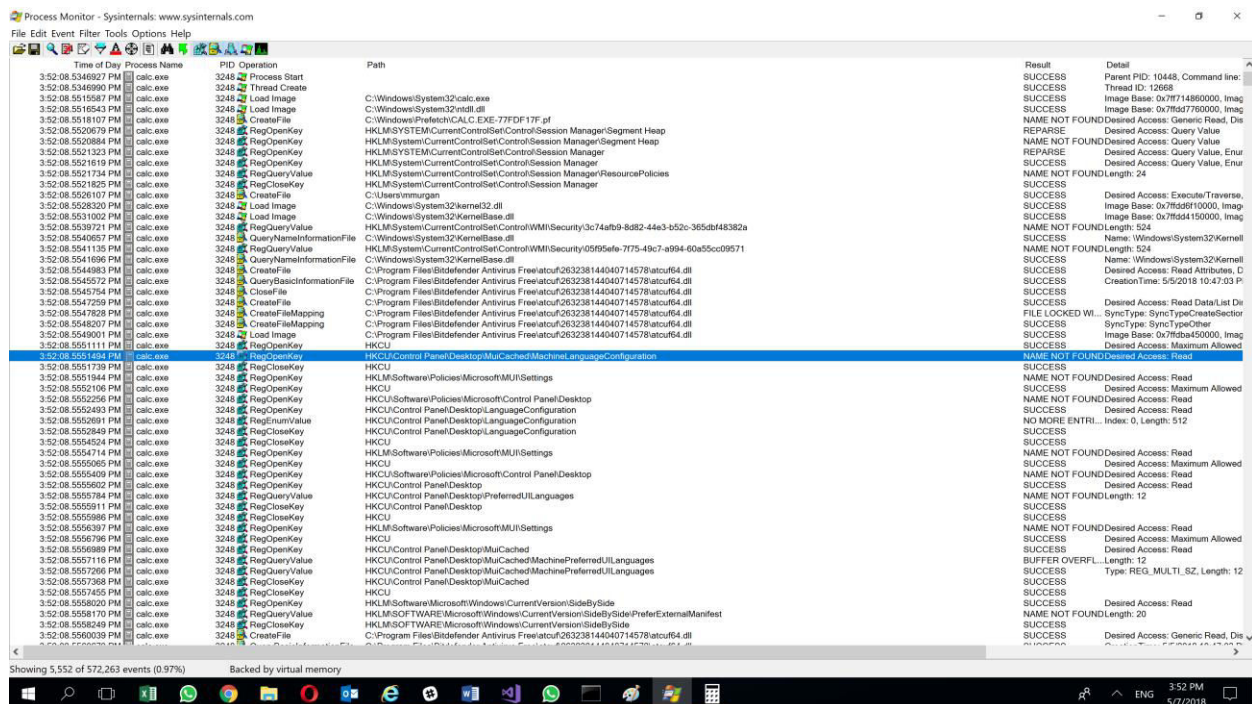
Observati va rog in zona de sus selectata cu negru – 4 butoane – daca toate sunt selectate, process monitor va afisa activitate pentru in ordine – registry, fisiere, networking, process and thread activity – incarcare de dll-uri, pornire de threaduri. Deselectand din ele, acele evenimente corespunzatoare butonului deselectat nu mai sunt afisate.

In bara de meniu se afla campul Filter, pe care daca il selectati va va apare un dropdown meniu, care mai contine inca un camp Filter, va rog sa il selectati, ar trebui sa se deschida urmatoarea casuta de dialog:

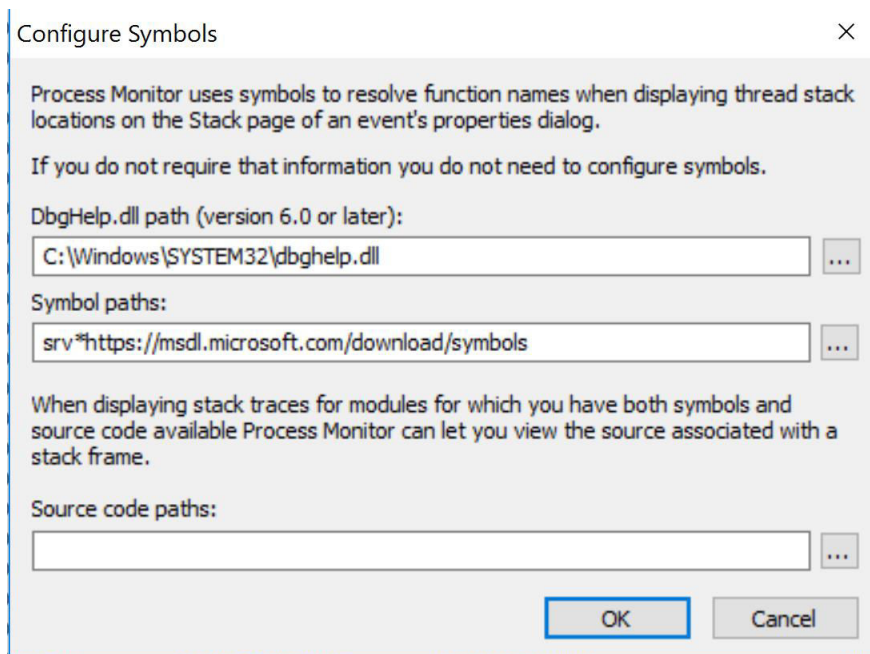


Diin cele doua dropdown boxuri de sus selectati in loc de Architecture -> Process Name si in loc de is -> contains, treceti in campul de text calc.exe si apasati butonul Add si apoi butonul OK.

Porniti executabilul calc.exe (este calculatorul de windows) si o sa vedeti urmatorul output in Process Monitor (in fine, unul similar ☺):



Cand compilati cu Visual Studio un executabil, langa el ati observat ca se creaza de fiecare data si un fisier .pdb – acesta contine un fel de harta care face legatura intre cod si offseti rezultati in binar. Pentru o aplicatie scrisa de voi, pe care o rulati si o monitorizati cu Process Monitor, la fiecare lucru pe care il face puteti sa ii vedeti stiva din cod si codul care a generat acel apel. Acest lucru il putem face spunandu-i Process Monitor unde sa gaseasca simbolurile, si anume: ne ducem la Options->Configure Symbols si obtinem fereastra:



In zona Symbol Paths exista o cale de genul:

srv*https://msdl.microsoft.com/download/symbols asta e calea spre serverele Microsoft de unde se pot incarca simbolurile pentru binarele Microsoft. Sa explicam – orice aplicatie care ruleaza pe o masina , pe langa executabilul ei si bibliotecile ei, incarca si biblioteci de baza Microsoft kernel32.dll, ntdll.dll, etc. Pentru a reconstrui stiva apelurilor pentru un anumit eveniment (e similar si in cazul crash dumpurilor), ProcessMonitor are nevoie de pdb-urile create la compilarea binarelor, care reprezinta o baza de date cu adresa de memorie unde se afla fiecare functie incarcata. In cazul Microsoft, aceste simboluri se pot lua din calea mentionata, in cazul nostru, ele se pot lua de unde am luat binarele, acolo ar trebui sa fie si pdb-urile – de exemplu sa zicem ca avem aplicatia compilata in : d:\Aplicatie\, astfel ca voi modifica calea in :

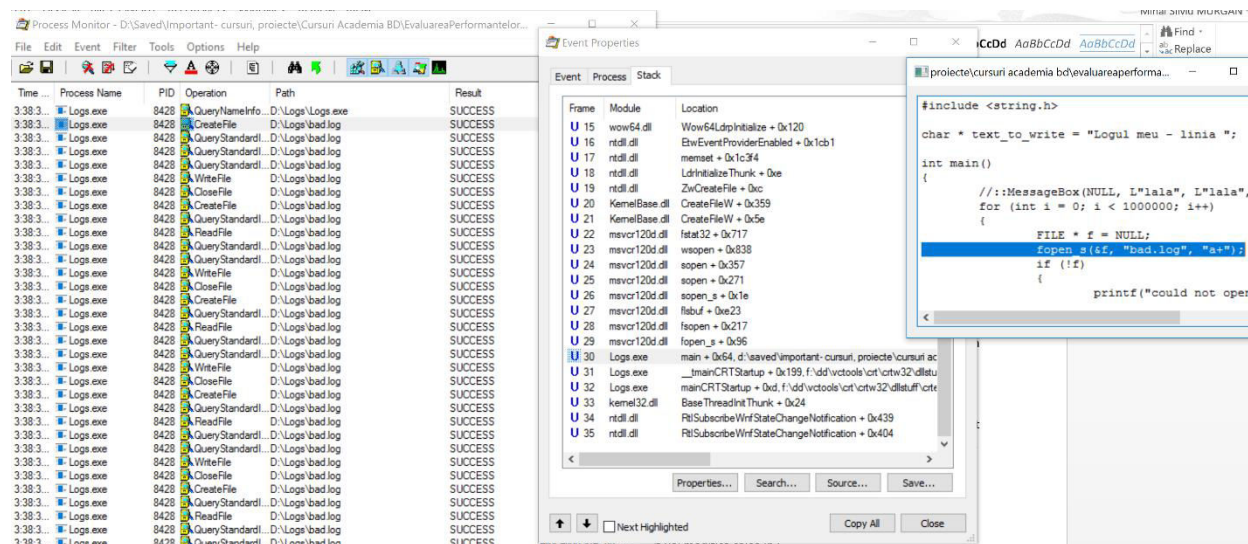
```
srv*https://msdl.microsoft.com/download/symbols;d:\Aplicatie\
```

atentie in d:\Aplicatie trebuie sa fie pdb-ul asociat executabilului rulat.

In campul Source code paths adaugam calea catre .sln-ul proiectului pe care il analizam : in cazul mentionat mai sus tot d:\Aplicatie dupa care apasam pe Ok.

Ne ducem in log pe instructiunea care ne intereseaza, de exemplu un CreateFile f, dam dublu click, se deschide fereastra cu Event Properties, alegem tabul Stack si acolo ar trebui sa vedem apeluri din

aplicatia noastra. Daca dam click pe acel apel si dam show source code, ar trebui sa ne arate efectiv bucata de cod care a generat acel apel. Vedeti exemplul de mai jos:



Ok, deci am vazut cum putem monitoriza activitatea pe disk, am vazut cum putem identifica cine o face si cum putem sa ne dam seama daca avem acces la pdb-uri si cod unde este problema in sine. Haideti sa vedem cum investigam o noua problema, un leak de handle-uri, adica un process care deschide fisiere si nu le mai inchide, daca face asta de cateva mii de ori, pe calculatoarele moderne probabil vorbim de milioane de ori sistemul poate deveni neresponsiv si fie o sa mearga mai greu fie o sa crape aplicatia care are problema. O sa spuneti ca milioane de handleuri sunt imposibil de atins asa ca nu merita sa acordam atentie. Si totusi, inchipuiti-va ca exista servicii pe servere care merg ani de zile, astfel la un leak de un handle la 2 secunde ajungem la peste 10 milioane pe an. Cum investigam astfel de probleme?

Practic

Fie codul de mai jos:

```
#include <windows.h>

int main()
{
    HANDLE hFile = INVALID_HANDLE_VALUE;
    while (1)
    {
        hFile = CreateFile(L"leak.txt", GENERIC_READ, FILE_SHARE_READ |
FILE_SHARE_WRITE | FILE_SHARE_DELETE, NULL,
OPEN_EXISTING, 0, NULL);
        Sleep(1000);
    }
    CloseHandle(hFile);
}
```


Sa il luam si sa compilam un executabil HandleLeak.exe – rulati-l dintr-o consola si uitati-va in Task Manager – sectiunea details, unde sa adaugati coloana handles (ca sa adaugati inca o coloana in plus in Task Manager dati click dreapta pe bara de titlu si o sa vi se deschida o fereastră cu ce coloane doriti sa mai adaugati):

Name	PID	Status	Username	CPU	Memory (pri...)	Handles	Threads	Image path name	Command line	Description
HandleLeak.exe	396	Running	mmurgan	12	320 K	69,838	1	D:\Logs\Handle...	HandleLeak.exe	HandleLeak.exe
igfxCUIService.exe	1992	Running	SYSTEM	00	24 K	205	12	C:\Windows\Sys...	C:\Windows\sys...	igfxCUIService Module
IntelCpHeciSvc.exe	5312	Running	SYSTEM	00	24 K	159	6	C:\Windows\Sys...	C:\Windows\sys...	IntelCpHeciSvc Executable
IpOverUsbSvc.exe	4496	Running	SYSTEM	00	24 K	238	5	C:\Program File...	"C:\Program Fil...	Windows IP Over USB PC Service
isa.exe	7504	Running	SYSTEM	00	2,036 K	440	11	C:\Program File...	"C:\Program Fil...	Intel(R) Security Assist
jhi_service.exe	6576	Running	SYSTEM	00	32 K	121	2	C:\Program File...	"C:\Program Fil...	Intel(R) Dynamic Application Loader Host Interface
LMS.exe	8856	Running	SYSTEM	00	2,488 K	401	11	C:\Program File...	"C:\Program Fil...	Intel(R) Local Management Service
lsass.exe	716	Running	SYSTEM	00	6,040 K	1,861	10	C:\Windows\Sys...	C:\Windows\sys...	Local Security Authority Process
MSBuild.exe	14680	Running	mmurgan	00	12,224 K	313	8	C:\Program File...	C:\Program File...	MSBuild.exe
mspaint.exe	14860	Running	mmurgan	00	23,140 K	394	13	C:\Windows\Sys...	"C:\Windows\sys...	Paint
notepad++.exe	2536	Running	mmurgan	00	856 K	182	3	C:\Program File...	"C:\Program Fil...	Notepad++ : a free (GNU) source code editor
NvBackend.exe	9840	Running	mmurgan	00	428 K	198	5	C:\Program File...	"C:\Program Fil...	NVIDIA Update Backend
nvSCPAPISvr.exe	1528	Running	SYSTEM	01	1,260 K	179	4	C:\Program File...	"C:\Program Fil...	Stereo Vision Control Panel API Server
nvtray.exe	17360	Running	mmurgan	00	1,656 K	227	3	C:\Program File...	"C:\Program Fil...	NVIDIA Settings
nvsvsc.exe	1520	Running	SYSTEM	00	1,356 K	178	4	C:\Windows\Sys...	"C:\Windows\sys...	NVIDIA Driver Helper Service, Version 353.54
nvsvsc.exe	13468	Running	SYSTEM	00	180 K	215	3	C:\Windows\Sys...	C:\Windows\sys...	NVIDIA Driver Helper Service, Version 353.54
nvxdsync.exe	14688	Running	SYSTEM	00	2,836 K	276	11	C:\Program File...	"C:\Program Fil...	NVIDIA User Experience Driver Component
OUTLOOK.EXE	16264	Running	mmurgan	01	38,368 K	4,863	56	C:\Program File...	"C:\Program Fil...	Microsoft Outlook
PresentationFontCac...	4220	Running	LOCAL SERV...	00	24 K	241	4	C:\Windows\Mi...	C:\Windows\Mi...	PresentationFontCache.exe
Procmon.exe	2508	Running	mmurgan	00	24 K	270	1	D:\Saved\Sysint...	"D:\Saved\Sysin...	Process Monitor
Procmon64.exe	8516	Running	mmurgan	01	53,252 K	647	11	C:\Users\mmur...	"C:\Users\mmur...	Process Monitor
RAVBg64.exe	10808	Running	SYSTEM	00	364 K	251	4	C:\Program File...	"C:\Program Fil...	HD Audio Background Process
RAVBg64.exe	15796	Running	SYSTEM	00	408 K	250	4	C:\Program File...	"C:\Program Fil...	HD Audio Background Process

O sa vedeti ca numarul de handle-uri creste. Ok, constatam ca avem o problema – cum o investigam? Sa introducem un nou tool Process Explorer

<https://technet.microsoft.com/en-us/sysinternals/processexplorer.aspx>

Porniti-l cu run as administrator. O sa vedeti ca e un fel de task manager, selectati procesul care ne intereseaza, anume HandleLeak. O data selectat apasati Ctrl+H si o sa vedeti ceva de genul:

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
chrome.exe	0.02	43,092 K	60,844 K	12488	Google Chrome	Google Inc.
WavesSvc64.exe	< 0.01	1,288 K	5,708 K	10120	Waves Max/Audio Service A...	Waves Audio Ltd.
DPMClient.exe	< 0.01	1,668 K	5,148 K	4236	DPM Client	Microsoft Corporation
DPMClientUI.exe	< 0.01	28,620 K	31,804 K	9500		Microsoft Corporation
Skype.exe	0.60	107,080 K	101,600 K	5884	Skype	Skype Technologies S.A.
BTTray.exe	4.496	13,144 K	13,144 K	3428	Bluetooth Tray Application	Broadcom Corporation
BluetoothHeadsetHelper.exe	1.544	7,672 K	10,332 K	10332	Bluetooth Headset Helper	Broadcom Corporation
cmd.exe	4.644	4,644 K	6,776 K	8948	Windows host process (Run...	Microsoft Corporation
OUTLOOK.EXE	0.26	288,324 K	259,480 K	16264	Microsoft Outlook	Microsoft Corporation
WINWORD.EXE	0.01	310,716 K	337,540 K	15652	Microsoft Word	Microsoft Corporation
Procmon.exe	4.580	14,884 K	14,884 K	2508	Process Monitor	Sysinternals - www.sysint...
Procmon64.exe	0.72	105,328 K	87,624 K	8516	Process Monitor	Sysinternals - www.sysint...
nvsvsc.exe	< 0.01	2,852 K	8,868 K	15532	NVIDIA Driver Helper Service	http://nvidia.com
mspaint.exe	< 0.01	42,898 K	47,398 K	15960	Paint	Microsoft Corporation
cmd.exe	1.768	1,768 K	3,360 K	8706	Windows Command Processor	Microsoft Corporation
conhost.exe	< 0.01	6,032 K	13,456 K	2024	Console Window Host	Microsoft Corporation
HandleLeak.exe	0.04	6,16 K	2,543 K	7436		
notepad++.exe	< 0.01	68,320 K	17,524 K	2536	Notepad++ : a free (GNU) s...	Don HO don.h@free.fr
process.exe	3.360	3,360 K	9,804 K	15580	Sysinternals Process Explor...	Sysinternals - www.sysint...
process64.exe	1.04	94,392 K	110,284 K	12736	Sysinternals Process Explor...	Sysinternals - www.sysint...
vpnui.exe	0.25	11,888 K	40,252 K	16904	Cisco AnyConnect User Inte...	Cisco Systems, Inc.
slack.exe	0.01	77,660 K	66,008 K	17476	Slack	Slack Technologies

Ctrl+H deschide o fereastră sub process în care afișează toate handleurile deschise cu informațiile care se pot găsi despre ele – deci o să vedem handleurile de fișiere, de registry, de semafoare, threaduri, etc. Mai există și un alt view Ctrl+D care va afișa toate dll-urile încărcate.

Ok, deci vedem că leakurile le avem pe fișierul D:\Logs\HandleLeak\leak.txt (în exemplul rulat de mine, în cazul vostru va fi probabil fișierul într-o altă cale) – o informație foarte utilă, dar nu ar fi și mai frumos să vedem și cine produce în cod acest leak? Va invit să rulați ProcessMonitor cu filtru pe HandleLeak.exe și să observați acolo stiva unde se creează leakul.

Debugarea unei aplicații

În resursa elemente_windows.zip care are parola windows, o să găsiți un executabil și un dll. Rulați executabilul și urmăriți cu process monitor și strings (voi detalia mai jos ce e strings) faceți ca aplicația să funcționeze.

Strings face parte tot din suita sysinternals și poate fi dat jos de pe:

<https://docs.microsoft.com/en-us/sysinternals/downloads/strings>

(sau îl aveți deja downloadat dacă ați dat jos toată suita sysinternals). Strings e un utilitar care va afișează stringurile dintr-un binar, fie dll fie executabil.

Hint(cautați stringurile ascii din .dll).