Modalitati de interconectare a echipamentelor si solutiilor TIC cu aplicatia AMAP (AUTOMATED MONITORING & ANALYSIS PLATFORM)

Abstract: *Proiectul "AUTOMATED MONITORING & ANALYSIS PLATFORM (AMAP)", Cod SMIS 142811, finantat prin Programul Operațional Competitivitate, Axa prioritară 2 - Tehnologia Informației și Comunicațiilor (TIC) pentru o economie digitală competitivă, Prioritatea de investiții 2b - Dezvoltarea produselor și serviciilor TIC, a comerțului electronic și a cererii de TIC, Obiectiv Specific OS 2.2 - Creșterea contribuției sectorului TIC pentru competitivitatea economică, Acțiunea 2.2.1 - Sprijinirea creșterii valorii adăugate generate de sectorul TIC și a inovării în domeniu prin dezvoltarea de clustere, Apelul de proiecte nr. 3 a vizat dezvoltarea unei aplicații software inovative pentru monitorizarea automată și analiza integrală a tuturor informațiilor primite din partea echipamentelor TIC cu posibilitatea de analiza și procesare a parametrilor, monitorizarea log-urilor și corelarea evenimentelor, care să răspundă diferitelor cerințe ale piețelor potențiale identificate – Automated Monitoring & Analysis Platform (AMAP). Pentru atingerea obiectivelor asumate de proiectul AMAP, în prezenta cercetare sunt analizate principalele modalități de interconectarea echipamentelor și soluțiilor IT bazate pe platforme de gestionare a API-urilor, funcții „hook"(cârlig) respectiv elemente de tip „wrapper" (înveliș). S-a acordat o atenție deosebită dezvoltării de interfețe flexibile și scalabile API, care să faciliteze integrarea cu diverse echipamente și soluții TIC existente pe piață. În continuare s-au analizat modalități de stocare a datelor și a fost realizată o cercetare amplă asupra metodelor de analiză a datelor, dezvoltându-se modele și algoritmi avansați pentru a extrage informații relevante și pentru a genera rapoarte personalizate, adaptate nevoilor utilizatorilor și piețelor țintă ale aplicației. În ultima etapă au fost prezentate principiile care stau la baza elaborării de algoritmi avansați pentru analizarea datelor colectate, identificarea anomaliilor și generarea de alerte personalizate, furnizând astfel utilizatorilor informațiile necesare pentru a lua decizii fundamentate și a reduce riscurile asociate activității lor.*

În perspectiva modernă, digitalizarea înseamnă transformarea interacțiunilor, comunicărilor, relațiilor, funcțiilor de business și a modelelor de afaceri în (mai multe) procese digitale, care adesea se reduc la o combinație de digital și fizic (cum ar fi serviciul pentru clienți - omnichannel, marketingul integrat sau industrializarea producției și operațiilor manuale, serviciile electronice și așa mai departe). Digitalizarea utilizează informații digitizate (sau obținute direct în format digital).

Versiunile analogice / fizice, cum ar fi documente de hârtie, imagini, fotografii, sunete etc., trebuie convertite în format digital prin digitizare. Odată ce informațiile analogice au fost digitizate, ele pot fi integrate și apoi folosite în diverse aplicații software (cu premise bune pentru automatizări ulterioare).

O caracteristică esențială a Monitorizării TIC este, pe lângă măsurarea practică a tuturor indicatorilor și obiectivelor necesare atingerii țintelor asumate, facilitarea interacțiunii active a cetățenilor, mediului de afaceri, mediului academic și alți reprezentanți ai societății civile în definirea, prioritizarea și urmărirea rezultatelor. Astfel, prin prezentul proiect se va avea în vedere dezvoltarea unei platforme software care, pe baza Principiilor Directoare și Criteriilor pentru Serviciile Publice, va plasa interesul acestora în centrul inițiativelor pe tot parcursul ciclului de viață, de la definire până la monitorizarea efectelor.

Există o varietate de soluții pentru interconectarea echipamentelor și soluțiilor IT. Iată câteva dintre cele mai populare și utilizate:

- Middleware: Middleware-ul este un software intermediar care facilitează comunicarea între diferite aplicații, sisteme sau dispozitive. Aceasta poate fi utilizată pentru a interconecta și a integra echipamente și soluții IT diverse. Exemple de middleware includ Apache Kafka, RabbitMQ și Apache ActiveMQ.
- Platforme de integrare (Integration Platforms): Aceste platforme oferă facilități de integrare și interconectare a diferitelor aplicații și sisteme IT prin intermediul interfețelor și protocoalelor standardizate. Exemple de platforme de integrare includ MuleSoft Anypoint Platform, Dell Boomi, IBM Integration Bus și Microsoft Azure Logic Apps.
- Sisteme de management al rețelei (Network Management Systems): Aceste sisteme sunt utilizate pentru monitorizarea, administrarea și gestionarea echipamentelor și rețelelor IT. Ele pot include funcționalități de interconectare și integrare a diferitelor dispozitive și soluții IT. Exemple de sisteme de management al rețelei includ Cisco Prime Infrastructure, SolarWinds Network Performance Monitor și Nagios.
- Sisteme de automatizare a proceselor (Business Process Automation Systems): Aceste sisteme permit automatizarea și optimizarea fluxurilor de lucru și a proceselor de afaceri. Ele pot include funcționalități de interconectare și integrare a diferitelor aplicații și sisteme IT. Exemple de sisteme de automatizare a proceselor includ UiPath, Automation Anywhere și Pega Platform.
- Platforme de gestionare a API-urilor (API Management Platforms): Aceste platforme facilitează gestionarea și securizarea interfețelor de programare a aplicațiilor (API-uri) utilizate pentru interconectarea și comunicarea între diferite aplicații și servicii IT. Exemple de platforme de gestionare a API-urilor includ Apigee (Google Cloud), MuleSoft Anypoint Platform și IBM API Connect.
- Funcții hook- reprezintă un mecanism care permite unui programator să intervină în execuția unei funcții sau a unui eveniment într-o aplicație. Prin utilizarea hook-urilor, un programator poate înlocui, modifica sau adăuga cod suplimentar înainte, în timpul sau după execuția unei funcții. Aceasta oferă posibilitatea de a controla și personaliza comportamentul aplicației. Hook-urile pot fi utilizate într-o varietate de contexte, cum ar fi hook-uri de sistem pentru interceptarea evenimentelor de la nivel de sistem, hook-uri de aplicație pentru modificarea comportamentului aplicațiilor sau hook-uri de funcții pentru a modifica funcționalitatea unei funcții specifice.
- Wrapperele: Un wrapper este un concept utilizat pentru a înconjura o funcție sau o clasă existentă cu un alt cod care oferă o funcționalitate suplimentară sau modificată. Wrapperele sunt utilizate adesea pentru a extinde funcționalitatea unei biblioteci sau a unui framework fără a modifica codul sursă original. Prin încapsularea funcțiilor sau claselor existente într-un wrapper, dezvoltatorii pot adăuga sau modifica comportamentul acestora într-un mod controlat și izolat. Acesta oferă flexibilitate și modularitate în dezvoltarea software.

În urma analizei efectuate pentru proiectul AMAP, deoarece nivelul de interconectare trebuie să fie maxim, fiind vizate platformei cu o arie largă de specificații, soluțiile de interconectare bazate cu caracter nișat: middleware, platforme de interconectare, sisteme de optimizare a procesor, etc. au fost eliminate. S-au păstrat doar soluțiile cu caracter general care permit cel mai larg spectru de interconectare.

Windows API

Folosind API-ul furnizat de Windows pot fi dezvoltate aplicații care rulează cu succes pe toate versiunile sistemelor de operare Windows, profitând în același timp de caracteristicile și capabilitățile unice pentru fiecare versiune (Microsoft, 2022i). Windows API a fost numit anterior API-ul Win32. Numele Windows API reflectă mai precis rădăcinile sale în Windows pe 16 biți și suportul său pentru Windows pe 64 de biți.

API-ul Windows, în mod informal „WinAPI”, este setul de bază de interfețe de programare a aplicațiilor (API-uri) Microsoft în sistemele de operare Microsoft Windows. Numele Windows API se referă în mod colectiv la mai multe implementări de platforme diferite, care sunt adesea menționate prin propriile nume (spre

exemplu, Win32 API). Aproape toate programele Windows interacționează cu API-ul Windows. Pe linia de sisteme de operare Windows NT, un număr mic (cum ar fi programele începute la începutul procesului de pornire Windows) utilizează API-ul nativ.

API-ul Windows (Win32) se concentrează în principal pe limbajul de programare C, deoarece funcțiile sale expuse și structurile de date sunt descrise în acel limbaj în versiunile recente ale documentației sale. Cu toate acestea, API-ul poate fi utilizat de orice compilator sau asamblator de limbaj de programare capabil să gestioneze structurile de date de nivel scăzut (bine definite) împreună cu convențiile de apelare prescrise pentru apeluri și apeluri inverse. În mod similar, implementarea internă a funcției API a fost dezvoltată în mai multe limbi.

În ciuda faptului că C nu este un limbaj de programare orientat pe obiecte, API-ul Windows și Windows sunt descrise ca fiind orientate pe obiecte. Au existat, de asemenea, multe clase de tip „wrapper" și extensii (de la Microsoft și altele) pentru limbaje orientate pe obiecte care fac această structură orientată pe obiecte mai explicită.

Versiuni Windows API

Aproape fiecare versiune nouă de Microsoft Windows a introdus propriile adăugiri și modificări la API-ul Windows. Cu toate acestea, numele API-ului a rămas consecvent între diferitele versiuni de Windows, iar modificările de nume au fost limitate la modificări majore ale arhitecturii și ale platformei pentru Windows. În cele din urmă, Microsoft a schimbat numele familiei actuale de API Win32 în Windows API și a transformat-o într-un termen general pentru versiunile API anterioare și viitoare.

Win16 este API-ul pentru primele versiuni pe 16 biți ale Microsoft Windows. Acestea au fost denumite inițial pur și simplu API-ul Windows, dar ulterior au fost redenumite în Win16, într- un efort de a le distinge de versiunea mai nouă, pe 32 de biți, a API-ului Windows. Funcțiile Win16 API rezidă în principal în fișierele de bază ale sistemului de operare: „kernel.exe" (sau „krnl286.exe", respectiv „krnl386.exe"), „user.exe" și „gdi.exe". În ciuda extensiei de fișier, acestea sunt de fapt biblioteci cu link-uri dinamice.

Win32 este interfața de programare a aplicațiilor pe 32 de biți pentru versiunile de Windows pe 32 de biți (NT, 95 și versiuni ulterioare). API-ul constă din funcții implementate, ca și în cazul Win16, în DLL-urile de sistem. DLL-urile de bază ale Win32 sunt „kernel32.dll", „user32.dll" și „gdi32.dll". Win32 a fost introdus cu Windows NT. Versiunea de Win32 livrată cu Windows 95 a fost inițial denumită Win32c, „c" însemnând compatibilitate. Acest termen a fost ulterior abandonat de Microsoft în favoarea Win32. Win32s este o extensie pentru familia Windows 3.1x a Microsoft Windows care a implementat un subset al API-ului Win32 pentru aceste sisteme („S" înseamnă „subset").

Win64 este varianta API-ului implementată pe platformele pe 64 de biți ale arhitecturii Windows (din 2021, x86-64 și AArch64). Ambele versiuni pe 32 de biți și 64 de biți ale unei aplicații poate fi încă compilat dintr-o bază de cod, deși unele API-uri mai vechi au fost depreciate, iar unele dintre API-urile care erau deja depreciate în Win32 au fost eliminate. Toți pointerii de memorie sunt implicit pe 64 de biți (modelul LLP64), astfel încât codul sursă trebuie verificat pentru compatibilitate cu aritmetica pointerului pe 64 de biți și rescris după cum este necesar.

Funcții Windows API

Funcțiile oferite de API-ul Windows pot fi grupate în opt categorii :

1. ***Servicii de bază***

Aceste funcții oferă acces la resursele de bază e pentru un sistem Windows. Sunt incluse lucruri precum sistemele de fișiere, dispozitivele, procesele și gestionarea erorilor. Aceste funcții se află în fișierele kernel.exe, krnl286.exe sau krnl386.exe pe Windows pe 16 biți și kernel32.dll și KernelBase.dll pe Windows pe 32 și 64 de biți. Aceste fișiere se află în directorul „\Windows\System32" pe toate versiunile de Windows.

2. *Servicii avansate*

Aceste servicii oferă acces la funcții dincolo de nucleu. Sunt incluse lucruri precum registrul Windows, închiderea/repornirea sistemului (sau anularea), pornirea/oprirea/crearea unui serviciu Windows, gestionarea conturilor de utilizator. Aceste funcții se află în „advapi32.dll" și „advapires32.dll" pe Windows pe 32 de biți.

3. *Interfața dispozitivului grafic GUI*

Interfața dispozitivului grafic permite afișarea conținutului grafic pe monitoare, imprimante și alte dispozitive de ieșire. Se află în „gdi.exe" pe Windows pe 16 biți și „gdi32.dll" pe Windows pe 32 de biți în modul utilizator. Suportul pentru modul Kernel este oferit de „win32k.sys" care comunică direct cu driverul grafic.

4. *Interfața cu utilizatorul*

Aceasta oferă funcțiile pentru a crea și gestiona ferestrele de pe ecran și majoritatea controalelor de bază, cum ar fi butoanele și barele de defilare, primirea intrării mouse-ului și a tastaturii și alte funcții asociate cu interfața grafică cu utilizatorul (GUI) din Windows. Această unitate funcțională se află în „user.exe" pe Windows pe 16 biți și „user32.dll" pe Windows pe 32 de biți. Începând cu versiunile Windows XP, controalele de bază se află în „comctl32.dll", împreună cu controalele comune (Common Control Library).

5. *Common Dialog Box Library*

Această funcție oferă aplicațiilor casetele de dialog standard pentru deschiderea și salvarea fișierelor, alegerea culorii și fontului. Biblioteca se află într-un fișier numit „commdlg.dll" pe Windows pe 16 biți și „comdlg32.dll" pe Windows pe 32 de biți.

6. *Common Control Library*

Această funcționalitate oferă aplicațiilor acces la unele controale avansate furnizate de sistemul de operare. Acestea includ lucruri precum bare de stare, bare de progres, bare de instrumente și file. Biblioteca se află într-un fișier de bibliotecă cu legături dinamice numit „commctrl.dll" pe Windows pe 16 biți și „comctl32.dll" pe Windows pe 32 de biți.

7. *Windows Shell*

Componenta API-ului Windows permite aplicațiilor să acceseze funcțiile furnizate de shell-ul sistemului de operare și să o schimbe sau/și îmbunătățească. Componenta se află în „shell.dll" pe Windows pe 16 biți și „shell32.dll" pe Windows pe 32 de biți. Funcțiile „Shell Lightweight Utility" sunt în „shlwapi.dll".

8. *Network Services*

Aceste servicii oferă acces la diferitele abilități de rețea ale sistemului de operare. Subcomponentele sale includ NetBIOS, Winsock, NetDDE, apel de procedură la distanță (RPC) și multe altele. Această componentă se află în „netapi32.dll" pe Windows pe 32 de biți.

Interacțiunea cu programe și biblioteci de tip „wrapper"

API-ul Windows este conceput în principal pentru interacțiunea dintre sistemul de operare și aplicații. Pentru comunicarea între diferite aplicații Windows, Microsoft a dezvoltat o serie de tehnologii alături de principalul API Windows. Acest lucru a început cu Dynamic Data Exchange (DDE), care a fost înlocuit de Object Linking and Embedding (OLE) și mai târziu de Component Object Model (COM), Automation Objects, controale ActiveX și .NET Framework.

Varietatea termenilor este practic rezultatul grupării mecanismelor software care se referă la un anumit aspect al dezvoltării software. Automatizarea se referă în mod specific la exportul funcției unei aplicații sau componente (ca o interfață de programare a aplicațiilor), astfel încât să poată fi controlată de alte aplicații în loc de utilizatori umani, .NET este o metodologie și o tehnologie generală autonomă pentru dezvolta aplicații desktop și web scrise într-o varietate de limbaje compilate just-in-time (JIT). „Windows.pas" este o unitate Pascal/Delphi care conține declarațiile API specifice Windows. Este echivalentul Pascal cu „windows.h" folosit în C.

Diverse „wrappere” au fost dezvoltate de Microsoft care au preluat unele dintre funcțiile de nivel mai scăzut ale API-ului Windows și au permis aplicațiilor să interacționeze cu API-ul într-un mod mai abstract. Aceste biblioteci au ca scop principal apelarea unei subrutine sau un apel de sistem cu puține sau deloc calcule suplimentare. Astfel, bibliotecile de tip „wrapper” permit optimizarea interacțiunii cu Windows API, precum și a interacțiunii dintre Windows API și alte programe.

Microsoft Foundation Class (MFC) a inclus funcționalitatea Windows API în clasele C++ și, astfel, permite o modalitate mai orientată pe obiect de a interacționa cu API-ul. Biblioteca Active Template (ATL) este un wrapper orientat spre șablon pentru COM. Biblioteca de șabloane Windows (WTL) a fost dezvoltată ca o extensie pentru ATL și intenționată ca o alternativă mai mică la MFC.

Aplicabilitatea funcțiilor „hook”

O funcție tip „hook” („cârlig”) reprezintă o metodă din mecanismul de gestionare a mesajelor de sistem, în care o aplicație poate instala o subrutină pentru a monitoriza traficul de mesaje în sistem și a procesa anumite tipuri de mesaje înainte ca acestea să ajungă la procedura ferestrei țintă (Microsoft, 2022c).

De asemenea, funcțiile de tip „hook” oferă unei aplicații capabilitățile de a intercepta evenimente, cum ar fi mesaje, acțiuni ale mouse-ului și apăsări de taste. O funcție care interceptează un anumit tip de eveniment este cunoscută ca procedură de tip „hook”. O procedură de tip „hook” poate acționa asupra fiecărui eveniment pe care îl primește și apoi poate modifica sau elimina evenimentul.

Următoarele sunt exemple de utilizare pentru funcțiile tip „hook”:

Monitorizarea mesajelor în scopuri de depanare.

Colectarea datelor de elementele hardware ale sistemului

Colectarea log-urilor de la servicii și aplicații

Oferirea de suport pentru înregistrarea și redarea macro comenzilor.

Oferirea de suport pentru o tastă de ajutor (F1).

Simularea introducerii mouse-ului și a tastaturii.

Implementarea unei aplicații de instruire pe computer.

1.1. Lanțuri de funcții tip „hook”

Un sistem de operare Windows acceptă multe tipuri diferite de funcții tip „hook”. Fiecare tip oferă acces la un aspect diferit al mecanismului său de gestionare a mesajelor. Spre exemplu, o aplicație poate folosi „WH_MOUSE” pentru a monitoriza traficul de mesaje dat de mouse.

Sistemul menține un lanț separat pentru fiecare funcție de tip „hook”. Un lanț reprezintă o listă de indicatori către funcții speciale, definite de aplicație, numite proceduri de conectare. Când apare un mesaj care este asociat cu un anumit tip de funcție „hook”, sistemul transmite mesajul fiecărei proceduri la care se face referire în lanț, unul după altul. Acțiunea pe care o poate lua o procedură depinde de tipul de funcție „hook” utilizată. Unele proceduri pot monitoriza doar mesajele; altele pot modifica mesajele sau pot opri progresul lor prin lanț, împiedicând realizarea următoarei proceduri.

1.2. Proceduri de funcții tip „hook”

Pentru utilizarea unei funcții de tip „hook”, se poate folosi o procedură bazată pe funcția

„SetWindowsHookEx”. Această funcție poate fi utilizată pentru a instala funcția „hook” în lanțul asociat. O procedură de tip „hook” trebuie să aibă sintaxa prezentată în figura de mai jos.

```
LRESULT CALLBACK HookProc(
  int nCode,
  WPARAM wParam,
  LPARAM lParam
)
{
   // process event
   ...

   return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

Sintaxa unei proceduri de tip "Hook"

În Figura de mai sus, „HookProc” este un substituit pentru un nume definit de aplicație. Parametrul „nCode” este un cod de tip „hook” pe care procedura îl folosește pentru a determina acțiunea de efectuat. Valoarea codului depinde de tipul funcției „hook”; fiecare tip are propriul său set caracteristic de coduri. Valorile parametrilor „wParam” și „lParam” depind de codul de tip„hook”, dar de obicei conțin informații despre un mesaj care a fost trimis sau postat.

Funcția „SetWindowsHookEx” instalează întotdeauna o procedură la începutul unui lanț de funcții tip „hook”. Când are loc un eveniment care este monitorizat de un anumit tip de funcție „hook”, sistemul apelează procedura la începutul lanțului. Fiecare procedură din lanț determină dacă se trece evenimentul la următoarea procedură. O procedură de conectare transmite un eveniment la următoarea procedură prin apelarea funcției „CallNextHookEx”. Procedurile pentru unele tipuri de funcții „hook” pot monitoriza doar mesajele. Sistemul transmite mesaje fiecărei proceduri, indiferent dacă o anumită procedură apelează „CallNextHookEx”.

O funcție „hook” globală monitorizează mesajele pentru toate firele de execuție din același desktop ca firul de apelare. O funcție „hook” specifică unui fir monitorizează mesajele doar pentru un fir individual. O procedură globală poate fi apelată în contextul oricărei aplicații din același desktop cu firul de apelare, deci procedura trebuie să fie într-un modul separat DLL. O procedură specifică firului este apelată numai în contextul firului asociat. Dacă o aplicație instalează o procedură de tip „hook” pentru unul dintre propriile fire de execuție, procedura de tip „hook” poate fi în același modul cu restul codului aplicației sau într-un DLL. Dacă aplicația instalează o procedură pentru un fir al unei aplicații diferite, procedura trebuie să fie într-un DLL.

1.3. Tipuri de funcții tip „hook”

WH_CALLWNDPROC și WH_CALLWNDPROCRET

Funcțiile „WH_CALLWNDPROC” și „WH_CALLWNDPROCRET” permit monitorizarea mesajelor trimise la procedurile unei ferestre. Sistemul apelează o procedură de tip „hook”„WH_CALLWNDPROC” înainte de a trece mesajul către procedura ferestrei de primire și apelează procedura „WH_CALLWNDPROCRET” după procesarea mesajului.

Funcția „WH_CALLWNDPROCRET” transmite către procedura de funcție un indicator către o structură „CWPRETSTRUCT”. Structura conține valoarea returnată din procedura ferestrei care a procesat mesajul, precum și parametrii mesajului asociați mesajului.

WH_CBT

Sistemul apelează o procedură „WH_CBT” înainte de a activa, crea, distruge, minimiza, maximiza, muta sau dimensiona o fereastră. Înainte de a finaliza o comandă de sistem, înainte de a elimina un eveniment de mouse sau tastatură din coada de mesaje de sistem, înainte de a seta focalizarea de intrare sau înainte de sincronizarea cu coada de mesaje a sistemului. Valoarea pe care o returnează procedura determină dacă sistemul permite sau împiedică una dintre aceste operațiuni.

WH_DEBUG

Sistemul apelează o procedură „WH_DEBUG” înainte de a apela procedurile asociate cu orice altă funcție de tip „hook” din sistem. Acest tip de funcție „hook” se poate utiliza pentru a determina dacă se permite sistemului să apeleze procedurile asociate cu alte tipuri de funcții „hook”.

WH_FOREGROUNDIDLE

Funcția „WH_FOREGROUNDIDLE” permite efectuarea unor sarcini cu prioritate scăzută în perioadele în care firul său din prim-plan este inactiv. Sistemul apelează o procedură „WH_FOREGROUNDIDLE” atunci când firul din prim-plan al aplicației este pe cale să devină inactiv.

WH_GETMESSAGE

Funcția „WH_GETMESSAGE” permite unei aplicații să monitorizeze mesajele pe cale să fie returnate de funcția „GetMessage” sau „PeekMessage”. Funcția „WH_GETMESSAGE” poate fi utilizată pentru a monitoriza datele introduse (engl. „input”) cu ajutorul mouse-ului și a tastaturii și alte mesaje postate în coada de mesaje.

WH_KEYBOARD_LL

Funcția „WH_KEYBOARD_LL” permite monitorizarea evenimentelor de introducere a datelor de la tastatură pe cale să fie postate într-o coadă de introducere a firelor.

WH_KEYBOARD

Funcția „WH_KEYBOARD” permite unei aplicații să monitorizeze traficul de mesaje pentru mesajele „WM_KEYDOWN” și „WM_KEYUP” pe cale să fie returnate de funcția „GetMessage” sau „PeekMessage”. Funcția „WH_KEYBOARD” poate fi utilizată pentru a monitoriza datele introduse de la tastatură, postate într-o coadă de mesaje.

WH_MOUSE_LL

Funcția „WH_MOUSE_LL” permite monitorizarea evenimentelor de introducere a datelor de la mouse care urmează să fie postate într-o coadă de mesaje.

WH_MOUSE

Funcția „WH_MOUSE” permite monitorizarea mesajelor mouse-ului pe cale să fie returnate de funcția „GetMessage” sau „PeekMessage”. Funcția „WH_MOUSE” poate fi utilizată pentru a monitoriza datele de intrare ale mouse-ului postate într-o coadă de mesaje.

WH_MSGFILTER and WH_SYSMSGFILTER

Funcțiile „WH_MSGFILTER” și „WH_SYSMSGFILTER” permit monitorizarea mesajelor care urmează să fie procesate de un meniu, bară de prezentare, casetă de mesaje sau casetă de dialog și detectarea când o altă fereastră este pe cale să fie activată ca urmare a apăsării combinațiilor de taste „ALT+TAB” sau „ALT+ESC” de către utilizator. Funcția „WH_MSGFILTER” poate monitoriza numai mesajele transmise unui meniu, bară de prezentare, casetă de mesaje sau casetă de dialog create de aplicația care a instalat procedura.

Funcția „WH_SYSMSGFILTER” monitorizează astfel de mesaje pentru toate aplicațiile.

Funcțiile „WH_MSGFILTER” și „WH_SYSMSGFILTER” permit filtrarea mesajelor în timpul buclelor modale. Un exemplu de buclă modală este reprezentat de un mesaj tip „pop-up” care apare în timpul utilizării unei aplicații. Filtrarea mesajelor în timpul buclelor modale este echivalentă cu filtrarea efectuată în bucla principală a mesajelor. Un exemplu de buclă principală a mesajelor este reprezentat de comunicarea dintre aplicații și sistemul de operare. Spre exemplu, o aplicație examinează adesea un mesaj nou în bucla principală între momentul în care preia mesajul din coadă și momentul în care trimite mesajul, efectuând o procesare specială, după caz. Cu toate acestea, în timpul unei bucle modale, sistemul preia și trimite mesaje fără a permite unei aplicații șansa de a filtra mesajele în bucla principală de mesaje. Dacă o aplicație instalează o procedură precum cea de tip „WH_MSGFILTER” sau „WH_SYSMSGFILTER”, sistemul apelează procedura în timpul buclei modale.

O aplicație poate apela direct „WH_MSGFILTER” apelând funcția „CallMsgFilter”. Folosind această funcție, aplicația poate folosi același cod pentru a filtra mesajele în timpul buclelor modale așa cum îl folosește în bucla principală a mesajelor. Pentru a face acest lucru, operațiunile de filtrare trebuie încapsulate într-o procedură „WH_MSGFILTER”, iar apoi trebuie apelată funcția „CallMsgFilter” între apelurile la funcțiile „GetMessage” și „DispatchMessage”. Sintaxa utilizării acestui tip de cârlig poate fi urmărită în următoarea figură

```
while (GetMessage(&msg, (HWND) NULL, 0, 0))
{
    if (!CallMsgFilter(&qmsg, 0))
        DispatchMessage(&qmsg);
}
```

Sintaxa utilizării unui cârlig pentru filtrarea de mesaje

Ultimul argument al funcției „CallMsgFilter” este transmis direct procedurii. Aici poate fi introdusă orice valoare. Procedura, prin definirea unei constante precum „MSGF_MAINLOOP”, poate folosi această valoare pentru a determina de unde a fost apelată.

WH_SHELL

O aplicație de tip „shell” poate folosi „WH_SHELL” pentru a primi notificări importante. Sistemul apelează o procedură „WH_SHELL” atunci când aplicația este pe cale să fie activată și când o fereastră de nivel superior este creată sau distrusă.

Aplicațiile „shell” personalizate nu primesc mesaje „WH_SHELL”. Prin urmare, orice aplicație care se înregistrează drept „shell”, implicit, trebuie să apeleze funcția „SystemParametersInfo” înainte ca aceasta (sau orice altă aplicație) să poată primi mesaje „WH_SHELL”. Această funcție trebuie apelată prin utilizarea funcției „SPI_SETMINIMIZEDMETRICS” și o structură „MINIMIZEDMETRICS”. Variabila „iArrange” a acestei structuri trebuie setată la „ARW_HIDE”.

Funcții specifice de tip „hook”

Funcția „CallNextHookEx” (Microsoft, 2021a) transmite informațiile la următoarea procedură din lanțul curent. O astfel procedură poate apela această funcție fie înainte, fie după procesarea informațiilor.

Funcția „CallMsgFilter” (Microsoft, 2022a) transmite mesajul specificat și codul procedurilor asociate „WH_SYSMSGFILTER” și „WH_MSGFILTER”.

Funcția „CallWndProc” (Microsoft, 2018a) este o funcție de apel invers (engl. „callback”), utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție înainte de a apela procedura ferestrei pentru a procesa un mesaj trimis firului de execuție.

Funcția „CallWndRetProc” (Microsoft, 2021d) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție după ce este apelată funcția „SendMessage”. Procedura poate examina mesajul; însă nu îl poate modifica.

Funcția „CBT-Proc” (Microsoft, 2018b) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție înainte de a activa, crea, distruge, minimiza, maximiza, muta sau dimensiona o fereastră, înainte de a finaliza o comandă de sistem, înainte de a elimina un eveniment de mouse sau tastatură din coada de mesaje de sistem, înainte de a seta focalizarea tastaturii, sau înainte de sincronizarea cu coada de mesaje a sistemului. O aplicație de instruire pe calculator utilizează această procedură pentru a primi notificări utile de la sistem.

Funcția „Debug-Proc” (Microsoft, 2018c) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție înainte de a apela procedurile asociate cu orice tip de

funcție „hook”. Sistemul transmite informații despre „hook”-ul care urmează să fie apelat de procedura „Debug-Proc”, care examinează informațiile și determină dacă permite apelarea funcției „hook”.

Funcția „ForegroundIdleProc” (Microsoft, 2018d) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție ori de câte ori firul din prim- plan este pe cale să devină inactiv.

Funcția „GetMsgProc” (Microsoft, 2021c) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție ori de câte ori funcția „GetMessage” sau „PeekMessage” a preluat un mesaj dintr-o coadă de mesaje a aplicației. Înainte de a returna mesajul preluat, sistemul transmite mesajul procedurii de conectare.

Funcția „KeyboardProc” (Microsoft, 2022d) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție ori de câte ori o aplicație apelează funcția „GetMessage” sau „PeekMessage” și există un mesaj de la tastatură („WM_KEYUP” sau „WM_KEYDOWN”) de procesat.

Funcția „LowLevelKeyboardProc” (Microsoft, 2020a) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție de fiecare dată când un nou eveniment de introducere de la tastatură este pe cale să fie postat într-o coadă de introducere a firelor de execuție.

Funcția „LowLevelMouseProc” (Microsoft, 2020b) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție de fiecare dată când un nou eveniment de introducere a mouse-ului este pe cale să fie postat într-o coadă de introducere a firelor de execuție.

Funcția „MessageProc” (Microsoft, 2021e) este o funcție de apel utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție după ce apare un eveniment de intrare într-o casetă de dialog, casetă de mesaj, meniu sau bară de accesare, dar înainte ca mesajul generat de evenimentul de intrare să fie procesat. Procedura poate monitoriza mesajele pentru o casetă de dialog, o casetă de mesaje, un meniu sau o bară de accesare creată de o anumită aplicație sau de toate aplicațiile.

Funcția „MouseProc” (Microsoft, 2020c) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție ori de câte ori o aplicație apelează funcția „GetMessage” sau „PeekMessage” și există un mesaj de mouse care trebuie procesat.

Funcția „ShellProc” (Microsoft, 2022h) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Funcția primește notificări ale evenimentelor de tip „shell” de la sistem.

Funcția „SysMsgProc” (Microsoft, 2021f) este o funcție de apel invers utilizată cu funcția „SetWindowsHookEx”. Sistemul apelează această funcție după ce apare un eveniment de intrare într-o casetă de dialog, casetă de mesaj, meniu sau bară de accesare, dar înainte ca mesajul generat de evenimentul de intrare să fie procesat. Funcția poate monitoriza mesajele pentru orice casetă de dialog, casetă de mesaje, meniu sau bară de accesare din sistem.

Funcția „UnhookWindowsHookEx” (Microsoft, 2021g) este o funcție care elimină o procedură de tip „hook” instalată într-un lanț de cârlig prin funcția „SetWindowsHookEx”.

Utilizarea metodelor de tip „hook”

Funcția „SetWindowsHookEx” (Microsoft, 2022g) instalează o procedură definită de aplicație într-un lanț de funcții tip „hook”. O astfel de procedură este instalată pentru a monitoriza sistemul pentru anumite tipuri de evenimente. Aceste evenimente sunt asociate fie cu un anumit fir de execuție, fie cu toate firele de execuție din același desktop folosit de firul de execuție apelant. Parametrii acestei funcții sunt următorii:

- „nCode” de tip „int”: Codul transmis procedurii curente. Următoarea procedură de tip „hook” utilizează acest cod pentru a determina modul de procesare a informațiilor.

- „wParam” de tip „WPARAM”: Valoarea „wParam” transmisă procedurii de tip „hook” curente. Semnificația acestui parametru depinde de tipul funcție „hook” asociată lanțului de funcții curent.
- „lParam” de tip „LPARAM”: Valoarea „lParam” transmisă procedurii de tip „hook” curente. Semnificația acestui parametru depinde de tipul funcție „hook” asociată lanțului de funcții curent.

Această metodă returnează o valoare cu tipul „LRESULT”. Această valoare este returnată de următoarea procedură din lanț. Procedura actuală trebuie, de asemenea, să returneze această valoare. Semnificația valorii returnate depinde de tipul de funcție „hook”.

Funcția „CallNextHookEx” (Microsoft, 2021b) transmite informațiile la următoarea procedură din lanțul de funcții „hook” curent. O procedură poate apela această funcție fie înainte, fie după procesarea informațiilor. Parametrii acestei funcții sunt următorii:

- „hhk” de tip „HHOOK”: acest parametru este ignorat;
- „nCode” de tip „int”: Codul transmis procedurii curente. Următoarea procedură de tip
- „hook” utilizează acest cod pentru a determina modul de procesare a informațiilor.
- „wParam” de tip „WPARAM”: Valoarea „wParam” transmisă procedurii de tip „hook” curente. Semnificația acestui parametru depinde de tipul funcție „hook” asociată lanțului de funcții curent.
- „lParam” de tip „LPARAM”: Valoarea „lParam” transmisă procedurii de tip „hook” curente. Semnificația acestui parametru depinde de tipul funcție „hook” asociată lanțului de funcții curent.

Funcția „UnhookWindowsHookEx” (Microsoft, 2021h) elimină o procedură instalată într-un lanț prin funcția „SetWindowsHookEx”. Această funcție are un singur parametru, respectiv

- „hhk” de tip „HHOOk”, care reprezintă un mâner (engl. „handle”) către funcția care trebuie eliminată. Acest parametru reprezintă un mâner pentru rezultatul funcției „hook” anterioare, prin apelul către „SetWindowsHookEx”. Valoarea returnată de această metodă este 0 dacă funcția eșuează și diferită de 0, dacă aceasta este executată cu succes.

Utilizarea procedurilor de tip „hook”

O procedură de tip „hook” poate fi instalată apelând funcția „SetWindowsHookEx” și specificând tipul de funcție „hook” care apelează procedura, dacă procedura ar trebui să fie asociată cu toate firele de execuție din același desktop cu firul de execuție apelant sau cu un anumit fir de execuție și un indicator către procedură ca punct de intrare (Microsoft, 2021i).

O procedură globală de tip „hook” trebuie plasată într-un DLL separat de aplicația care instalează procedura de tip „hook”. Aplicația de instalare trebuie să aibă mânerul (engl.

„handle”) pentru modulul DLL înainte de a putea instala procedura. Pentru a prelua un astfel de mâner pentru modulul DLL, se apelează funcția „LoadLibrary” cu numele DLL-ului. După ce este obținut mânerul, se poate apela funcția „GetProcAddress” pentru a prelua un indicator către procedura de tip „hook”. În cele din urmă, se utilizează „SetWindowsHookEx” pentru a instala adresa procedurii în lanțul corespunzător. „SetWindowsHookEx” transmite mânerul modulului, un indicator către punctul de intrare al procedurii și 0 pentru identificatorul firului de execuție, indicând că procedura ar trebui să fie asociată cu toate firele de execuție din același desktop cu firul de execuție apelant. Această secvență este prezentată în exemplul următor.

Exemplu procedură de tip „hook”.

```
HOOKPROC hkprcSysMsg;
static HINSTANCE hinstDLL;
static HHOOK hhookSysMsg;

hinstDLL = LoadLibrary(TEXT("c:\\myapp\\sysmsg.dll"));
hkprcSysMsg = (HOOKPROC)GetProcAddress(hinstDLL, "SysMessageProc");

hhookSysMsg = SetWindowsHookEx(
                    WH_SYSMSGFILTER,
                    hkprcSysMsg,
                    hinstDLL,
                    0);
```

Eliminarea unei proceduri de tip „hook” specifică firului de execuție se poate realiza apelând funcția „UnhookWindowsHookEx”, specificând mânerul procedurii vizate pentru eliminare. O procedură trebuie eliminată imediat ce aplicația nu mai are nevoie de ea. Eliminarea unei proceduri cu „UnhookWindowsHookEx”, nu elimină însă DLL-ul care conține procedura de tip „hook”. Acest lucru se datorează faptului că procedurile globale sunt apelate în contextul procesului fiecărei aplicații de pe desktop, provocând un apel implicit la funcția „LoadLibrary” pentru toate aceste procese. Sistemul elimină în cele din urmă DLL-ul după ce toate procesele legate în mod explicit de DLL fie s-au terminat, fie au numit „FreeLibrary” și toate procesele care au numit procedura „hook„ au reluat procesarea în afara DLL-ului.

O metodă alternativă pentru instalarea și eliminarea unei proceduri de conectare globală este de a furniza o funcție de instalare în DLL, împreună cu procedura de conectare. Cu această metodă, aplicația de instalare nu are nevoie de mânerul pentru modulul DLL. Prin conectarea la DLL, aplicația obține acces la funcția de instalare. Funcția de instalare poate furniza mânerul modulului DLL și alte detalii în apelul către „SetWindowsHookEx”. DLL-ul poate conține, de asemenea, o funcție care elimină procedura globală; aplicația poate apela această funcție de eliminare la terminare.

Cercetarea s-a derulat în cadrul proiectului “AUTOMATED MONITORING & ANALYSIS PLATFORM (AMAP)”, Cod SMIS 142811, finantat prin Programul Operațional Competitivitate, Axa prioritară 2 - Tehnologia Informației și Comunicațiilor (TIC) pentru o economie digitală competitivă, Prioritatea de investiții 2b - Dezvoltarea produselor și serviciilor TIC, a comerțului electronic și a cererii de TIC, Obiectiv Specific OS 2.2 - Creșterea contribuției sectorului TIC pentru competitivitatea economică, Acțiunea 2.2.1 - Sprijinirea creșterii valorii adăugate generate de sectorul TIC și a inovării în domeniu prin dezvoltarea de clustere, Apelul de proiecte nr. 3.

„Conţinutul acestui material nu reprezintă în mod obligatoriu poziţia oficială a Uniunii Europene sau a Guvernului României”.