Když proces hlídá proces

Vložil/a RubberDuck [1], 20 Únor, 2013 - 01:29

- Programming [2]

... Neznámá předtucha mě nutí otevřít Task Manager. Spěšně ho prohlížím. V záplavě nejrůznějších procesů upoutá mou pozornost jeden s názvem kernel64.exe. Co to je za nesmysl? Proces vypnu a jdu si nalít trochu čaje. Když se znovu podívám do Task Managera, proces kernel64.exe je zpět. WTF?...

Cílem následujícího článku je ukázat, jak jednoduše lze hlídat běh procesu a v případě potřeby ho znovu nastartovat. Před pár dny jsem si při sledování filmu Robin Hood vzpomněl, jak jsem před mnoha lety četl článek o vtipném pokusu jedné společnosti, která konkurenci (XEROX nebo IBM?) do tiskárny nasadila dvě binárky, které se ve formě procesů neustále hlídaly a pokud byl jeden z procesů zabit, druhý ho okamžitě znovu spustil. Tento vtip nadělal administrátorům hodně vrásek na čele. Paradoxně dal také vzniknout jedné z technik, s níž jsme se mohli, můžeme a pravděpodobně i budeme moci setkávat u malware. Z dnešního pohledu už se nejedná o nijak závažný problém. Ale před třemi čtyřmi desítkami let se odstranění tohoto žertíku opravdu rovnalo neřešitelnému úkolu. Možná se teď leckdo z vás ptá, proč jsem si na tohle vzpomněl zrovna u filmu Robin Hood. Odpověď je jednoduchá: Ty dva procesy se totiž jmenovaly Robin a Big John :)

Oč se tedy jedná? Princip je velice jednoduchý. Máme dva procesy, které se vzájemně hlídají. Pokud jeden z nich zjistí, že ten druhý "umřel", okamžitě ho znovu nastartuje. Způsobů, jak tento úkol realizovat je mnoho. Pokusím se obecně popsat tři nejzákladnější a navrch přidám popis čtvrtého, trošku komplikovanějšího řešení.

První a úplně nejjednodušší řešení napadlo snad každého. Každý program bude procházet cyklicky procesy a hledat v nich název svého souputníka. Pokud ho najde, pokračuje dál. Pokud ho nenajde, spustí ho znovu a pokračuje dál. Mezi obecně nejpoužívanější Windows API pro tento účel je sada funkcí z knihovny Tool Help. Konkrétně CreateToolhelp32Snapshot, Process32First a Process32Next. Windows API funkce CreateToolhelp32Snapshot s prvním argumentem TH32CS_SNAPPROCESS, jenž říká, že budeme pracovat s procesy, vytvoří handle. Tento handle se dále předá do API funkce Process32First jako první argument. Jako druhý je pak pointer na strukturu PROCESSENTRY32. Tuto strukturu API funkce Process32Fist naplní informacemi o prvním běžícím procesu. Pro nás je nejdůležitější pole szExeFile udávající název souboru, z kterého byl daný proces vytvořen. Pro vylistování dalších procesů použijeme API funkci Process32Next. Tato API funkce přebírá stejné argumenty jako předchozí. Jednoduchý kód pro ověření, zda daný proces v paměti existuje, může vypadat následovně:

```
#include <windows.h>
#include <tlhelp32.h>
#include <stdio.h>

#define MY_COMRADE      "processxx.exe"

int main(){
  PROCESSENTRY32 pe;
  STARTUPINFO si;
  PROCESS_INFORMATION pi;
  HANDLE hSnap = NULL;
  BOOL bIsDown = true;

  ZeroMemory(&pe, sizeof(pe));
```

```
  ZeroMemory(&si, sizeof(si));
  ZeroMemory(&pi, sizeof(pi));
  si.cb = sizeof(si);
  pe.dwSize = sizeof(PROCESSENTRY32);

  hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

  // is handle valid?
  if(hSnap != INVALID_HANDLE_VALUE){
    // get data of first process
        if(Process32First(hSnap, &pe) != false){
      do{
        // is this process my comrade?
        if(strcmp(pe.szExeFile, MY_COMRADE) == 0){
          // YES! This is my comrade! ;)
          bIsDown = false;
          printf("%s\n", pe.szExeFile);
        }
      }while(Process32Next(hSnap, &pe));

      // was found my comrade?
      if(bIsDown == true){
        // NO! my comrade is dead! :(
        CreateProcessA(MY_COMRADE, NULL,
                       NULL, NULL,
                       FALSE, NORMAL_PRIORITY_CLASS,
                       NULL, NULL, &si, &pi);
                Sleep(100);
      }else{
        bIsDown = true;
      }
    }
  }

  CloseHandle(hSnap);

  return 0;
}
```

Jak je vidět, kód je velice jednoduchý a prakticky nepotřebuje dalšího vysvětlování. Druhou možností je využití takzvaného mutexu. Mutex je synchronizační objekt, který může, ale nemusí mít jméno/pojmenování. Využívá signalizaci, aby dal procesům nebo vláknům, která se mají synchronizovat, echo, jak celý proces synchronizace probíhá (ještě se stále synchronizuje nebo už můžeme pokračovat?). Z těch několika málo API funkcí sloužících pro práci s mutexy si vystačíme s jednou, maximálně se dvěma (v závislosti na chutích jedinců ;)). První je CreateMutex a případná druhá OpenMutex. API funkce CreateMutex přebírá tři argumenty. První nastavuje bezpečnostní atributy pro práci s mutexem, druhým si může tvůrce mutexu zajistit počáteční vlastnictví mutexu a konečně třetí argument tvoří název mutexu. Důležitou vlastnostní mutexu je pro nás fakt, že jméno mutexu je v rámci systému unikátní, a proto existuje vždy pouze v jediném exempláři. Pokusíme-li se vytvořit mutex se jménem, které již existuje, systém nastaví funkci GetLastError na hodnotu ERROR_ALREADY_EXISTS. Díky tomu jsme schopni zjistit, zda náš spřízněný proces stále běží. Vyjádřeno programovacím jazykem:

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

#define BROTHERMUTEXNAME  "Kain_XYZ" /*Abel_XYZ*/
#define MYMUTEXNAME    "Abel_XYZ" /*Kain_XYZ*/
```

```
#define PROCESSNAME    "Kain.exe" /*Abel.exe*/

int main(){
  HANDLE hMutex = NULL;
  STARTUPINFO si;
  PROCESS_INFORMATION pi;

  CreateMutexA(NULL, FALSE, MYMUTEXNAME);

  while(1){
    ZeroMemory(&si, sizeof(si));
    ZeroMemory(&pi, sizeof(pi));
    si.cb = sizeof(si);

    hMutex = CreateMutexA(NULL, FALSE, BROTHERMUTEXNAME);

    if(GetLastError() != ERROR_ALREADY_EXISTS){
      if(CreateProcessA(PROCESSNAME, NULL,
                        NULL, NULL,
                        FALSE, NORMAL_PRIORITY_CLASS,
                        NULL, NULL, &si, &pi)){
            Sleep(100);
      }
    }

    if(hMutex != NULL){
      CloseHandle(hMutex);
      hMutex = NULL;
    }
  }

  return 0;
}
```

Opět se jedná o velice jednoduchý kód. Stejným způsobem můžeme využít i semafor. Semafor je 'inteligentnější' verze mutexu. 'Inteligentnější' je o schopnost počítat maximální počet procesů nebo vláken přistupujících k mutexu. Ostatní se holt musí hezky seřadit do řady a tiše čekat, až na ně přijde řada. Pro práci se semafory se nejčastěji setkáte s API funkcemi CreateSemaphore pro vytvoření semaforu, OpenSemaphore pro otevření existujícího semaforu a ReleaseSemaphore pro navýšení počítadla semaforu.

Tři nejjednodušší řešení máme za sebou. Na začátku jsem slíbil ještě jedno trochu složitější řešení. Ve skutečnosti se opět jedná o velice jednoduché řešení ;) Myšlenka je jednoduchá: Aplikace si vytvoří klienta a server. Server slouží přípojný bod pro druhý proces, aby byl schopný zjistit existenci prvního procesu. Klientem se pokusí připojit na server spolubojovníka. Pokud se mu to povede, čeká dokud nedostane zprávu o chybě. Pak se znovu pokusí připojit. Pokud se mu to nepodaří, spustí kamarádský proces a pokračuje s připojováním. Vytvoření tohoto kódu již nechám jako cvičení na každém ze čtenářů :)

URL článku: https://www.security-portal.cz/clanky/kdy%C5%BE-proces-hl%C3%ADd%C3%A1-proces

Odkazy:
[1] https://www.security-portal.cz/users/rubberduck
[2] https://www.security-portal.cz/category/tagy/programming