6 lutego 2023

Oprogramowanie Systemowe

Sprawozdanie z projektu „Winfsp”

Autor: Tomasz Piwowski

1. Czym jest WinFsp

Windows File System Proxy (WinFSP) to aplikacja działająca jako usługa w systemie Windows. Umożliwia stworzenie systemu plików z uprawnieniami użytkownika (user-mode).

Pracowanie w trybie usługi oznacza, że ilekroć system windows będzie chciał się odwołać do plików z sieciowego dysku (przydziału), będzie wywoływał naszą aplikację, która następnie zmapuje standardowe funkcje systemu plików NTFS jak otwieranie, tworzenie, zamykanie pliku itd. na funkcje zdefiniowane przez nas.

2. Uruchamianie

Po napisaniu kodu aplikacji wg schematu przedstawionego w samplu WinFsp, należy go skompilować i wyeksportować jako plik wykonywalny. Po tym etapie mamy 2 możliwości:

- Wywołać nasz system jako usługę – wówczas należy uzupełnić rejestr Windowsa, w katalogu należącym do WinFsp i podkatalogu o nazwie naszego systemu plików, o informacje o ścieżce do pliku wykonywalnego, argumentach wywołania, id grupa kontrolnej, do której należy proces oraz ciąg oznaczający uprawnienia naszego systemu plików (komenda „regedit” w polu wyszukiwania aplikacji pod przyciskiem „Windows”, by dostać się do rejestru w trybie graficznym).
  Następnie można już używać nazwy naszego systemu plików wraz z komendą „net use”. Natomiast do ustawienienia

odpowiednich pól w rejestrze można użyć pliku „fsreg.bat”, który znajduje się w folderze „bin” w katalogu „Winfsp” (tam gdzie zainstalowana jest aplikacja).
- Bezpośrednio uruchomić plik wykonywalny z odpowiednimi parametrami w jednym oknie konsoli, a następnie dostać się na wybrany dysk sieciowy w drugim oknie.

Uruchomiony dysk sieciowy z naszym systemem plików można również obsługiwać z poziomu Windows Explorera (nie tylko terminal).

3. **Największe napotkane trudności**

Na co warto zwrócić uwagę przed zabraniem się za analizę niskopoziomowego oprogramowania (najcz. napisanego w języku C) oraz wykorzystania związanych z nim bibliotek umożliwiających na stworzenie czegoś własnego.

- Wybrakowana dokumentacja — dostępna online dokumentacja nie musi zawierać wszystkich dostępnych funkcji, nie musi nawet dokładnie omawiać tych funkcji, które już się w niej pojawiły. Niektóre elementy są po prostu uznawane za znane dla czytelnika z przysłowiowego kosmosu.
  Rozwiązanie: Gdy mamy do czynienia z aplikacją otwarto źródłową najlepiej bezpośrednio otworzyć jej kod w drugim oknie i przelecieć po nazwach wszystkich funkcji. Sprawdzić dokładne działanie niektórych z nich. Czasem komentarze napisane w samym kodzie są dużo lepsze niż te udostępnione w dokumentacji online. No i nie musimy się martwić, że dana funkcja działa inaczej niż opisana.
- Niewiele łatwo dostępnych, dobrze opisanych bibliotek do ogólnych celów (język C) — w przeciwieństwie do języków

wyższego poziomu jak Java lub Python, C nie posiada wielu rozbudowanych bibliotek lub są one trudne do znalezienia i wykorzystania we własnym projekcie.
Rozwiązanie: Należy się przygotować, że większość funkcji będzie trzeba napisać samemu od zera lub skorzystać ze wcześniej przygotowanych bibliotek swojego autorstwa. Nie ma dróg na skróty.

- Utrudnione debugowanie — Aplikacja w trybie usługi nie ma swojego okna poleceń, tzn. nie można niczego wydrukować dla testu ani ustawiać breakpointów w debbugerze. Taka budowa znacznie utrudnia pracę nad programem.
  Rozwiązanie: Skorzystanie z generowanych dla systemu Windows wydarzeń, które można później podejrzeć w formie graficznej za pomocą aplikacji „Podgląd Zdarzeń" systemu Windows lub za pomocą narzędzia „debugview" do przeglądania ich w konsoli.
- Nieczytelny kod — aplikacje pisane w C, w szczególności dla systemu Windows, mają to do siebie, że posiadają wiele rozwiązań wynikających z wcześniejszych ułomności kompilatorów i nie tylko, co nie wpływa na czytelność ich kodów źródłowych. Jednymi z przykładów mogą być:
  - Niekonsekwencja względem nowoczesnych sposobów nazewnictwa zmiennych i funkcji — popularną praktyką jest by funkcje i zmienne zaczynać z małej litery, natomiast stałe pisać wielkimi literami. Nie możemy być pewni, że konwencja ta zostanie zachowana w tego typu programach.
  - Niekonsekwencja w nazewnictwie zmiennych i funkcji — niekiedy w samym programie funkcje mogą być zapisane czasem z małej litery (np. funkcje zadeklarowane za pomocą #define) a inne już nie. Zależy to nie tylko od widzimisię projektanta kodu, ale również od użycia funkcji

z bibliotek o innej konwencji nazewnictwa. Podobnie z nazwami zmiennych.

- Funkcje w define — często tworzenie funkcji za pomocą dyrektywy define może utrudniać czytanie kodu i wymagać wertowania po nim kilkukrotnie celem zrozumienia, co tak naprawdę się dzieje.
- Wykorzystanie goto — wykorzystanie instrukcji uznawanej za relikt przeszłości, która utrudnia czytanie kodu, koniecznością wertowania po nim góra dół za każdym razem gdy wystąpi oraz upewniania się, że działa tak jak powinna bez żadnych dodatkowych zależności.
- Przepisywanie typów, typy jako elementy dyrektywy define — aplikacje pisane dla Windows często stronią od używania standardowo zdefiniowanych typów, jak int, float, int* itd. zamiast tego wolą korzystać z typów, które zapisane wielkimi literami bardziej szczegółowo określają znaczenie danej zmiennej. Często jednak zbytnia szczegółowość bardziej przeszkadza niż pomaga, gdy nie znamy wszystkich rozwinięć skrótów i zasad działania każdego z typów (i tak jest wymagane zajrzenie do dokumentacji lub kodu źródłowego, by przekonać się co dany typ może przedstawiać). Lądujemy zatem ze stertą bliżej nieokreślonych danych, które do tego czasem stoją tuż obok typów prostych zapisanych małymi literami. Niekiedy również typy są przepisywane za pomocą #define bez konkretnego powodu, bez zmiany zasady działania lub dużej zmiany w znaczeniu — powoduje to m.in. dodatkowy krok potrzebny do rozszyfrowania prawdziwego znaczenia struktury, pod którą się kryje dana nazwa.
- Kod wymagający tłumaczenia — kod zapisany w taki sposób, że nie da się go przeczytać jak opis wykonania danej funkcji, a zamiast tego potrzebne są komentarze

i/lub dokumentacja objaśniająca co się w nim dzieje. Nie jest jednak powiedziana, że owa dokumentacja/komentarze zawsze się pojawiają i to w dobrej jakości.

- Brak komentarzy w plikach nagłówkowych/ dołączonej dokumentacji — powoduje to konieczność sprawdzenia działania funkcji bezpośrednio w kodzie źródłowym lub w dokumentacji online (jeśli występuje). Konsekwencją jest brak podpowiedzi na poziomie pisania kodu w IDE.
- Brak nowoczesnych udogodnień, poprawiających czytelność — mowa o przestrzeniach nazw, dzięki którym długie nazwy wielu funkcji, które muszą uwzględniać swoje pochodzenie z danej biblioteki, mogłyby zyskać wiele na czytelności. Jako że funkcjonalność wydaje się możliwe do zaimplementowania w sposób obciążający jedynie kompilator, nie wiem dlaczego nie jest obecna w dzisiejszych programach pisanych w C.

4. Ogólny zarys szkieletu programu

Najważniejszym elementem programu określającego działanie naszego systemu pliku jest struktura o typie FSP_FILE_SYSTEM_INTERFACE. Posiada ona pola na wskaźniki wszystkich funkcji, jakie zazwyczaj obsługuje system plików. Gdy pole przyjmuje wartość zera oznacza to, że dana funkcja nie jest obsługiwana przez nasz system plików. Jednakże istnieją pola obowiązkowe — są to:

- File open
- File close
- Get Security by Name (rekomendowane) – służy do pobrania atrybutów pliku i informacji dotyczących praw dostępu.

Jest ona parametrem wywołania funkcji FspFileSystemCreate — pierwszej z 3, które muszą być wywołane przy inicjalizacji naszego systemu plików. Są to (wykonywane w podanej kolejności):

1. FspFileSystemCreate — tworzy strukturę zawierająca informacje o stworzonym systemie plików
2. FspFileSystemSetMountPoint — ustawia punkt montowania dysku (katalog lub dysk wirtualny)
3. FspFileSystemStartDispatcher — rozpoczyna pracę naszego systemu plików jako usługi systemu Windows (inne aplikacja będąc w katalogu/dysku (mountpoint) i wywołując operacje systemu plików zaczną od tego momentu odwoływać się do funkcji zdefiniowanych przez nas.

Natomiast dla ułatwienia zarządzania zasobami i pilnowania ich zwolnienia, gdy program przestanie działać powstała jeszcze funkcja FspServiceRun, która jako jedyna uruchamiana z maina (wmaina) przyjmuje parametry m.in. takie jak:

- Wskaźnik do funkcji inicjalizującej system plików (wywołuje ww. 3 kluczowe metody)
- Wskaźnik do funkcji kończącej działanie systemu plików (zatrzymuje działanie dispatcher poprzez wywołania FspFileSystemStopDispatcher i zwalnia wszystkie zużyte zasoby)

5. Szczegółowe omówienie funkcji

Nazwy poniższych funkcji nie są przypisane z góry, tzn. mogą przyjąć dowolną postać, jednak dla zobrazowania ich funkcjonalności oraz dla zgodności z przykładowym kodem systemu plików napisanym przez twórców WinFsp, będą prezentowane w takiej a nie innej formie:

- SvcStart (skr. Service Start) — uruchamiana na starcie usługi. Przyjmuje parametry wywołania funkcji głównej oraz wskaźnik

do structury nadrzędnej dla całej usługi (FSP_SERVICE). Jej głównym zadaniem jest odczytanie paramtetrów wejściowych i zainicjalizowanie systemu plików (patrz punkt wyżej).

- SvcStop (skr. Service Stop) — analogicznie do SvcStart, tylko tym razem wykonywana, gdy usługa ma zostać wyłączona (np. po odmontowaniu dysku sieciowego z naszym systemem plików). Jedynym paremetrem jest wskaźnik na FSP_SERVICE (główne cele wywołania zostały opisane punkt wyżej).

Funkcje mapujące standardowe operacje na plikach: (W większości zwracają status czy operacja się udała, a struktury wyjściowe są przekazywane za pomocą wskaźników jako parametrów funkcji)

- GetVolumeInfo — udostępnia informacje o systemie plików dla systemu Windows, takie jak: całkowite zajęte miejsce, wolna przestrzeń oraz nazwa.
  Parametry wyjściowe:
  - FSP_FSCTL_VOLUME_INFO* VolumeInfo
- SetVolumeLabel – ustawia nową nazwę dla wirtualnego dysku z podpiętym naszym systemem plików.
  Parametry wyjściowe:
  - FSP_FSCTL_VOLUME_INFO* VolumeInfo
- GetSecurityByName — udostępnia listę atrybutów pliku oraz informacje o prawach dostępu, zanim zostanie otworzony.
  Parametry wyjściowe:
  - PUINT32 PFileAttributes,
  - PSECURITY_DESCRIPTOR SecurityDescriptor
  - SIZE_T* PSecurityDescriptorSize
- Create — tworzy nowy plik/katalog.
  Parametry wyjściowe:
  - PVOID* PFileContext
  - FSP_FSCTL_FILE_INFO* FileInfo

  Open — otwiera istniejący plik. Warto zaznaczyć, że ta funkcja jest wywoływana wielokrotnie przez system Windows i

zdecydowanie więcej niż raz przy każdym poleceniu otworzenia nowego pliku.
Parametry wyjściowe:
  - PVOID* PFileContext
  - FSP_FSCTL_FILE_INFO* FileInfo
- Overwrite —funkcja wywoływana przez system Windows, gdy nastąpi operacja nadpisania (tworzony jest nowy plik o tej samej ścieżce dostępu). Atrybuty pliku powinny być zastąpione nowymi lub zespolone ze starymi zależnie od wartości parametru ReplaceFileAttributes. Natomiast rozmiar pliku zawsze powinien być zmniejszony do zera.
  Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- Cleanup — operacja wywoływana tuż przed zamknięciem pliku (przy okazji wywoływania CloseHandle na uchwycie do pliku). Niezbędna, gdy implementowana jest również możliwość usuwania pliku, gdyż po tej operacji uchwyt do tego pliku staje się nieważny.
- Close — wywoływane po każdej operacji „open”. Ostatnia operacja dla każdego otworzonego pliku. Warto w niej chociażby opróżnić directory buffer tworzony przez WinFsp (funkcja FspFileSystemDeleteDirectoryBuffer).
- Read — wywoływana przy próbie odczytania pliku.
  Parametry wyjściowe:
  - PVOID Buffer
- Write — wywoływana przy próbie zapisu do pliku.
  Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- Flush — wywoływana w momencie, gdy system Windows przepisze już wszystkie dane z tymczasowych buforów do struktur systemu plików (za pomocą funkcji Write). Jeśli system plików obsługuje dodatkowe cachowanie (poza tym, które

oferuje z góry WinFsp) to jest to odpowiednie miejsce, by je obsłużyć (przepisać na stałe miejsca w pamięci).
Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- GetFileInfo — udostępnia informacje o atrybutach plików (dostępne np. poprzez kliknięcie prawym przyciskiem myszy na plik w eksploratorze i wybranie opcji „properties(właściwości)”).
Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- SetBasicInfo — służy do ustawienia kilku podstawowych atrybutów pliku (LastAccessTime, LastWriteTime, ChangeTime, Creation Time).
Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- SetFileSize — służy do ustawienia ilości zaalokowanej pamięci dla pliku. Możliwe jest również zaimplementowanie części dla ustawionej flagi SetAllocationSize, która pozwala na zaktualizowanie informacji o faktycznej ilości zajętych przez plik bajtów. Jednak należy pamiętać, że zawsze AllocationSize >= FileSize.
Parametry wyjściowe:
  - FSP_FSCTL_FILE_INFO* FileInfo
- Rename — operacja wywoływana podczas próby zmiany nazwy pliku lub jego przesunięcie (zmiana pełnej ścieżki do pliku).
- GetSecurity — służy do pobrania listy uprawnionych podmiotów do korzystania z pliku oraz dokładnego rodzaju tych uprawnień (wywoływana, gdy system plików obsługuje ACL). Uboższa wersja GetSecurityByName.
Parametry wyjściowe:
  - PSECURITY_DESCRIPTOR SecurityDescriptor
  - SIZE_T* PSecurityDescriptorSize
- SetSecurity — analogicznie do GetSecurity (potrzebne tylko wtedy, gdy system plików obsługuje ACL).

- ReadDirectory — służy do wylistowania zawartości katalogu celem obsłużenia tej funkcjonalności w Windows Explorerze (przy otwieraniu katalogu). Ponieważ funkcja uwzględnia pokazanie plików wg kryterium wyszukiwania po nazwie lub ładowanie etapami, stąd parametry wywołania: Pattern i Marker. Pattern to kryterium wyszukiwania, a Marker sygnalizuje że jedynie pliki o nazwach o wartości większej niż Marker mają być wylistowane. Celem utworzenia takiego buforu trzeba przywłaszczyć wcześniej odpowiedni semafor, służą do tego funkcje:
    - FspFileSystemAquireDirectoryBuffer — przywłaszcza semafor dla danego katalogu.
    - FspFileSystemReleaseDirectoryBuffer — zwalnia semafor. Musi być konieczne wywołana pod koniec funkcji ReadDirectory.

  Kluczowe do odpowiedniego wypełnienia wspomnianego buforu są jeszcze 2 funkcje:

    - FspFileSystemFillDirectoryBuffer — uzupełnia tymczasowy bufor zablokowany przez ww. semafor o pojedynczą pozycję.
    - FspFileSystemReadDirectoryBuffer — przepisuje informacje z tymczasowego bufora do odpowiedniego paramteru wyjściowego funkcji ReadDirectory.

  Parametry wyjściowe:
    - PVOID Buffer
    - PULONG PBytesTransferred
- SetDelete — sprawdza czy usunięcie pliku/katalogu jest dozwolone i sygnalizuje to odpowiednim statusem wyjściowym. SetDelete nie powinno ustawiać flagi pliku sugerującej jej gotowość do usunięcia (robią to wewnętrzne funkcje WinFsp). Warto jeszcze zauważyć, że funkcja jest wywoływana jedynie, gdy plik zostanie usunięty poprzez ustawienie odpowiedniej

flagi w jego atrybutach (wywołanie SetInformationByHandle dla struktury FileDispositionHandle; wykorzystywane przez takie funkcje jak DeleteFileW i RemoveDirectoryW). Usunięcie pliku w inny sposób nie spowoduje wywołania SetDelete (poprzez CreateFileW z flagą FILE_FLAG_DELETE_ON_CLOSE lub poprzez ww. funkcję Rename).