Coopy Hook Handler – jest to uchwyt systemowy zaimplementowany jako biblioteka DLL, która udostępnia jeden interfejs ICopyHook, który posiada jedną metodę CopyCallback. Kiedy folder ma zostać przeniesiony, powłoka wywołuje ową metodę, która zawiera informacje o:

- folderze źródłowym,

- folderze docelowym

- rodzaj wykonywanej operacji,

- atrybuty

Na wyjściu metoda ta zwraca jedną z trzech wartości, które decydują o losie wykonywanej operacji:

- IDYES – operacja jest dozwolona,

- IDNO – operacja jest blokowana, ale powłoka może wykonywać pozostałe, zaakceptowane operacje

- ICANCEL – blokuje operacje oraz odwołuje wszystkie inne operacje przeprowadzane na tym folderze

Standardowo, użytkownik podczas kopiowania, przenoszenia, usuwania czy zmiany nazwy folderu jest ograniczany jedynie przez prawa dostępu do tego folderu. Dzięki tej metodzie jesteśmy w stanie wprowadzić dodatkowe zasady kopiowania uwzględniające dowolnie wybrany przez nas czynnik, np. użytkownik PRACOWNIK1 ma prawo kopiować folder „D:\folder\” jedynie o godzinie 9:30. Dodatkowo samo zarejestrowanie żądania kopiowania daje nam możliwość tworzenia logów tej operacji.

Rozwiązanie to umożliwia implementację wszystkich funkcjonalności mojej aplikacji, jednakże obsługuje jedynie kopiowanie całych folderów, bez możliwości monitorowania pojedynczych plików, przez co staje się całkowicie bezużyteczna.

Schowek systemu Windows jest to wydzielona część pamięci ulotnej oraz zestaw metod do jej obsługi. Schowek przeznaczony jest dla użytkowników do tymczasowego przechowywania informacji w celu ich przeniesienia. Pamięć ta jest ogólnodostępna dla wszystkich użytkowników i można w nim przechowywać tekst, obraz lub pliki. Kopiowanie danych do schowka polega na zaznaczeniu jakiegoś fragmentu tekstu, grafiki lub pliku i wybranie polecenia kopiuj (CTRL +C lub PPM->kopiuj).

Do obsługi schowka, użytkownik ma do wyboru następujące polecenia:

- kopiuj – kopia obiektu, na którym wykonywane jest to polecenie, przenoszona jest do pamięci a sam obiekt pozostaje nienaruszony.

- wytnij – obiekt jest przenoszony do pamięci i usunięty z dotychczasowej lokalizacji.

- wklej – dane umieszczone w pamięci schowka są zapisywane we wskazanej lokalizacji.

W odróżnieniu od polecenia „wklej”, każdorazowe wywołanie polecenia „kopiuj” lub „wytnij” skutkuje nadpisaniem informacji zawartych w pamięci schowka.

Windows .NET posiada klasę Clipboard, która udostępnia metody do zarządzania schowkiem systemowym. Dzięki tej klasie jesteśmy wstanie nie tylko odczytywać i zapisywać, ale także podmieniać czy analizować dane zawarte w schowku, lub całkowicie go wyczyścić.

Podstawową metodą klasy Clipboard, która będzie użyteczna podczas pisania aplikacji, jest ContainsFileDropList(), która informuje nas czy w pamięci schowka znajdują się jakieś pliki, oraz metoda GetFileDropList(), która zwraca w postaci ciągu znakowego, listę plików w niej zawartych. Dzięki tym dwóm metodom możliwym jest wykrycie, że w schowku znajduje się plik, oraz uzyskanie informacji o ścieżce docelowej tego pliku, czym jest podstawowa funkcjonalność mojej aplikacji. Moja aplikacja, w zależności od decyzji administratora, powinna blokować nieautoryzowane kopiowanie, co jest możliwe dzięki metodzie Clear(), która usuwa wszystkie dane ze schowka, lub ma fałszować kopiowane dane. Aby tego dokonać, należy posłużyć się funkcją SetFileDropList(), która czyści pamięć schowka i umieszcza w niej wcześniej ustaloną listę plików.

Rozwiązanie to z pozoru jest sposobem na implementację wszystkich funkcjonalności mojej aplikacji, jednakże po przeprowadzeniu wstępnych testów, okazało się, iż nie wszystkie próby kopiowania plików zostały wyłapane przez aplikację. Po głębszej analizie okazało się, że metod kopiowania jest wiele, a użycie schowka systemowego, użyte w Windows Explorer jest tylko jedną z nich, a co za tym idzie nie ma jednoznacznej metody na wykrycie kopiowania.

Kolejnym moim pomysłem było sprawdzanie zmian na dysku, w poszukiwaniu nowo pojawiających się plików. Tą drogą znalazłem klasę FileSystemWatcher. Klasa ta nasłuchuje system plików w poszukiwaniu zmian w plikach i katalogach. Możliwye do wykrycia są następujące zmiany:

- Created – zachodzi, gdy w podanej ścieżce pojawi się nowy plik lub katalog,

- Deleted – zachodzi, gdy plik lub katalog z podanej ścieżki zostaje usunięty,

- Renamed – zachodzi, gdy plik lub katalog zmieni swoją nazwę,

- Changed – zachodzi, gdy jeden z nasłuchiwanych atrybutów zostanie zmieniony.

Dzięki tej klasie jesteśmy w stanie filtrować zbierane dane, monitorując jedynie wybrane typy plików, sprawdzać zmiany rozmiaru, czas ostatniej modyfikacji czy nazwy pliku.

Wszelkie zmiany wyłapane przez tą klasę przetrzymywane są w buforze stworzonym przez FileSystemWatcher. W przypadku, gdy w krótkim czasie zdarzy się zbyt wiele zmian, bufor może się przepełnić, co skutkowałoby chwilową przerwą w zbieraniu nowych danych. Pojemność bufora jest rozszerzalna, poprzez ustawienie odpowiedniej wartości argumentowi InternalBufferSize. Wielkość bufora może zostać ustawiona w przedziale 4KB – 64KB. Jeśli zostanie ustawiona na wartość większą lub mniejszą niż podane, wartości te zostaną nadpisane poprzez wartości graniczne.

FileSystemWatcher rozpoznaje tylko elementarne działania na plikach i folderach, tj. utworzenie, edycja czy usunięcie, przez co operacja kopiowania rozpoznawana jest jedynie jako utworzenie nowego pliku w folderze, do którego został przekopiowany. Niestety, nawet jeśli folder, z którego kopiujemy plik, jest monitorowany to ze względu, że ani obserwowany folder ani plik źródłowy nie zostały zmienione, nie zostaniemy o tym poinformowani. W tym przypadku pomocną może okazać się suma kontrolna obliczana z pliku. Na początku niezbędnym by było obliczenie sum kontrolnych ze wszystkich plików w interesującym nas obszarze i umieszczenie ich w bazie danych. W przypadku wykrycia nowego pliku o niezerowej pojemności, należałoby obliczyć z niego sumę kontrolną i sprawdzić w bazie danych czy nie występuje już identyczny ciąg. Biorąc pod uwagę średnią ilość plików na dysku, problematyczna może być wydajność takiego rozwiązania.