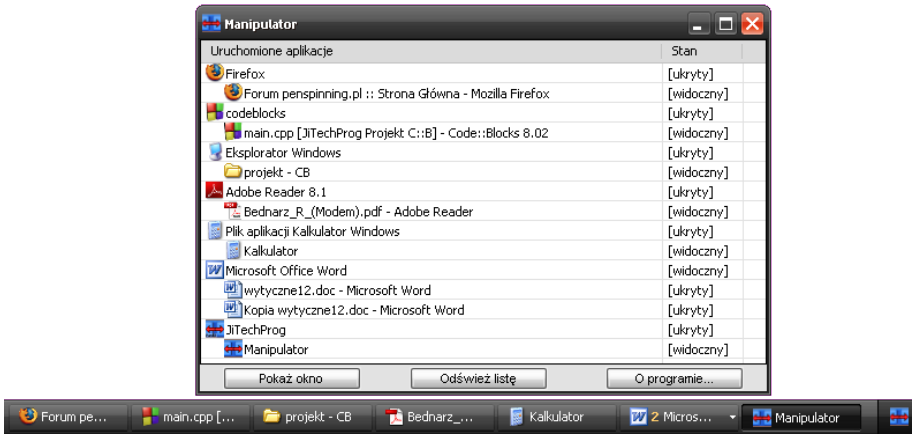
Pracownia Inżynierii Oprogramowania M-74 Instytut Informatyki Stosowanej Wydział Mechaniczny, Politechnika Krakowska	Przedmiot: **Języki i Techniki Programowania**
Osoba odpowiedzialna za przedmiot: dr inż. *Jacek Pietraszek*	Temat: **Manipulowanie przyciskami paska zadań przy użyciu interfejsu WinAPI**
Osoby odpowiedzialne za ćwiczenie: mgr inż. *Przemysław Osocha*	
DOKUMENTACJA	
Wersja: 1.0	Data wprowadzenia: 26.01.2009

Grupa:	Imię i Nazwisko:	Rok akademicki:	Semestr:
1131	Konrad Gadzina	2008/09	zimowy

0. Wstęp

Program Manipulator jest uproszczoną wersją programu PaseX tego samego autora, dostępnego w obecnej wersji pod adresem http://chomikuj.pl/Eisenheim/progsy/PaseX.rar, którego celem było umożliwienie modyfikowania kolejności i widoczności przycisków na pasku zadań systemu Windows XP (z dodatkiem SP2 - na innych systemach nie był testowany).

Manipulator został napisany w języku C++, w środowisku Code::Blocks 8.02. Problem został rozwiązany przy użyciu interfejsu Win32API, który również posłużył do stworzenia interfejsu graficznego programu.



Rys. 1. Okno programu i pasek zadań

1. Informacje o autorze

Konrad Gadzina <fenix.b3@gmail.com>
Grupa 1131, Informatyka, rok akad. 2008/2009
Pracownia Inżynierii Oprogramowania M-74
Instytut Informatyki Stosowanej
Wydział Mechaniczny, Politechnika Krakowska
Języki i Techniki Programowania

2. Krótki opis problemu i sposobu rozwiązania go przez program

W systemie Windows XP każde okno otwierane przez użytkownika dodaje do paska zadań przycisk – chyba, że programista pozbawił okna swojego programu tej możliwości. Przycisk ten jest z powiązany ściśle z konkretnym oknem i pozwala na wygodne zamykanie, minimalizowanie, czy też przywracanie go bez konieczności szukania odpowiednich przycisków na jego pasku tytułowym.

Osoba długo pracująca na konkretnym układzie okien przyzwyczajona jest już do niego i każda zmiana spowodowana zamknięciem któregoś z okien może mieć negatywny wpływ na szybkość jej pracy. Ten problem rozwiązuje Manipulator – pozwala on zmieniać kolejność przycisków, ukrywać niektóre z nich i pokazywać te już ukryte.

Każde uruchomione okno danego programu należy do pewnej grupy, dzięki czemu Eksplorator Windows może grupować na pasku zadań przyciski dotyczące okien jednego programu, jeżeli we właściwościach paska zadań jest zaznaczona odpowiednia opcja. Wtedy tworzony jest dla tych okien jeden przycisk grupowy, który jest ukryty, póki na pasku zadań jest wystarczająco dużo miejsca, by przycisk każdego okna był widoczny. Jeżeli zaś opcja ta nie jest zaznaczona, każde okno tworzy własny przycisk grupowy, który jest stale niewidoczny, bo przyciski powiązane z jednym programem nie są ze sobą kojarzone, więc nie mogą być one zastąpione jednym, wspólnym. Aby zmienić stan zaznaczenia tej opcji należy kliknąć prawym przyciskiem myszki na pasku zadań i z menu kontekstowego wybrać pozycję *Właściwości*, w zakładce *Pasek zadań* odznaczyć lub zaznaczyć opcję *Grupuj podobne przyciski paska zadań.*

Manipulator wczytuje wszystkie przyciski z paska zadań do listy w oknie programu, która przedstawia ikonę danego przycisku, jego nazwę i stan.

3. Opis interfejsu programu i sposób obsługi

Interfejs graficzny został napisany przy użyciu Win32API bez dodatkowych bibliotek. Okno programu minimalizuje się do zasobnika systemowego – przywrócić je można poprzez kliknięcie lewym przyciskiem myszki na ikonę dodaną tam przez program. Głównym elementem okna Manipulatora jest kontrolka ListView, do której ładowane są informacje dotyczące przycisków paska zadań. Oprócz niej znajdują się tam trzy przyciski:

1. *Pokaż/Ukryj okno* – zmienia widoczność danego przycisku
2. *Odśwież listę* – wywołuje funkcję *refresh()*, która ładuje ponownie dane przycisków z paska zadań
3. *O programie…* - otwiera okno z informacjami dotyczącymi programu

Jest możliwość obsługi programu bez konieczności klikania w przyciski – akcję pokazywania/ukrywania można zrealizować klikając dwukrotnie lewym przyciskiem myszy pozycję na liście, odpowiadającą za interesujący nas przycisk paska zadań. Poza tym zdefiniowane zostały skróty klawiszowe:

- *Spacja* – pokazuje/ukrywa zaznaczony przycisk
- *F5* – odświeża listę
- *F1* – otwiera okno z informacjami o programie
- *Esc* – minimalizuje okno do zasobnika systemowego

Najważniejszą funkcjonalnością programu jest oczywiście zmienianie kolejności przycisków na pasku zadań. Aby tego dokonać należy nacisnąć lewy przycisk myszy na elemencie listy, odpowiadającym interesującemu nas przyciskowi i trzymając wciąż wciśnięty

lewy przycisk przeciągnąć go na miejsce na liście, w którym chcemy go umieścić. Dla zachowania jak największej przejrzystości i przystępności kodu Manipulator pozwala przenosić tylko całe grupy przycisków – tylko przenoszenie przycisku grupowego daje żądany efekt, próby zmiany miejsca przycisku konkretnego okna są ignorowane. Pełną funkcjonalność oferuje program PaseX, który jest dostępny pod adresem podanym na końcu dokumentacji.

4. Opis funkcjonalności programu

Przy starcie programu uruchamiana jest funkcja *refresh()*, która przy pierwszym wywołaniu ma za zadanie najpierw znaleźć uchwyt do okna paska zadań i otworzyć proces Eksploratora Windows, by można było pobierać dane obszaru pamięci, z którego korzysta i wykonywać operacje na nim – jest to konieczne, by móc operować na przyciskach paska zadań.

Listing 1.

```
if (!hPasek) //jeżeli nie znany jest jeszcze uchwyt paska zadań
{
    hPasek = FindWindowEx(NULL, NULL, "Shell_TrayWnd", NULL);
    EnumChildWindows(hPasek, EnumChildProc, (LPARAM)txt);

    col.pszText = txt;
    col.cchTextMax = strlen(txt);

    GetWindowThreadProcessId(hPasek, &pid); //pobieranie ID procesu explorera
    proces = OpenProcess(PROCESS_VM_OPERATION | PROCESS_VM_READ, false, pid); //otwieranie
procesu explorera
}
```

Jak widać na listingu 1., który jest fragmentem funkcji *refresh*, jeżeli uchwyt paska zadań jest jeszcze nie znany, czyli funkcja ta jest wywoływana po raz pierwszy, to przy pomocy funkcji *FindWindowEx* i *EnumChildWindows* znajdowany jest najpierw uchwyt całego paska zadań (czyli całego okna, zawierającego przycisk Start, pasek szybkiego uruchamiania, pasek z przyciskami okien i zasobnik systemowy), którego klasa to *Shell_TrayWnd.* Następnym krokiem jest przeszukanie przy pomocy funkcji *EnumChildProc* (do której wskaźnik jest przekazywany funkcji *EnumChildWindows*) okien pochodnych paska zadań w celu znalezienia okna klasy *ToolbarWindow32*, którego dzieckiem jest okno klasy *MSTaskSwWClass*. To ostatnie jest tym, co nas interesuje – jest to okno z przyciskami powiązanymi z oknami innych programów. Parametrem funkcji *EnumChildWindows* jest również wskaźnik do tablicy znaków *txt*, do której będzie zapisana nazwa okna z przyciskami.

Listing 2.

```
BOOL CALLBACK EnumChildProc(HWND hwnd, LPARAM lParam)
{
    const int MAX = 256;
    char klasa[MAX];
    GetClassName(hwnd, klasa, MAX);
    if (!strcmp(klasa, "ToolbarWindow32"))
    {
        HWND hParent = GetParent(hwnd);
        GetClassName(hParent, klasa, MAX);
        if (!strcmp(klasa, "MSTaskSwWClass"))
        {
            GetWindowText(hwnd, (char*)lParam, MAX);
            hPasek = hwnd;
            return false;
        }
    }
```

```
    return true;
}
```

Posiadając już uchwyt paska zadań i jego procesu można zacząć wykonywać na nim operacje. Należy jednak jeszcze zaalokować pamięć, gdzie będą zapisywane dane dotyczące obecnie pobieranego przycisku i skąd będą kopiowane do zmiennej, która pozwoli operować informacjami na jego temat. Jako, że okno paska zadań jest toolbarem, informacje na temat jego konkretnego elementu można zapisać do zmiennej typu *TBBUTTON*.

Listing 3.

```
ptbb = (TBBUTTON*)VirtualAllocEx(proces, NULL, sizeof(TBBUTTON), MEM_COMMIT, PAGE_READWRITE);
ptxt = (char*)VirtualAllocEx(proces, NULL, 256, MEM_COMMIT, PAGE_READWRITE);

ile = SendMessage(hPasek, TB_BUTTONCOUNT, 0, 0);
for (int i = 0; i < ile; i++)
{
    SendMessage(hPasek, TB_GETBUTTON, (WPARAM)i, (LPARAM)ptbb);
    ReadProcessMemory(proces, (void*)ptbb, (void*)&tbb, sizeof(TBBUTTON), NULL);

    SendMessage(hPasek, TB_GETBUTTONTEXT, (WPARAM)tbb.idCommand, (LPARAM)ptxt);
    ReadProcessMemory(proces, (void*)ptxt, (void*)txt, 256, NULL);

    if (tbb.fsStyle & BTNS_WHOLEDROPDOWN)   //jeżeli przycisk jest przyciskiem grupowym
        item.iIndent = 0;
    else
        item.iIndent = 1;

    ListView_InsertItem(hList,&item);
}

VirtualFreeEx(proces, ptbb, 0, MEM_RELEASE);
VirtualFreeEx(proces, ptxt, 0, MEM_RELEASE);
```

Pierwsze dwie linie listingu 3. przedstawiają wyżej omówione „przygotowanie” do operacji na przyciskach. By móc kontynuować należy sprawdzić, ile przycisków jest obecnie na pasku zadań – po wysłaniu do paska komunikatu *TB_BUTTONCOUNT* i zapisaniu rezultatu do zmiennej można w pętli pobierać dane związane z przyciskami. Przed wstawieniem elementów do listy należy sprawdzić, czy dany przycisk jest grupowy, czy nie – jeśli tak, to wcięcie ustawiamy na 0, w przeciwnym wypadku ustawiamy je na 1. Daje to przejrzystość samej listy i pozwala później łatwo rozróżniać, z jakim przyciskiem mamy do czynienia. Po zakończeniu pobierania informacji należy zwolnić zarezerwowaną wcześniej pamięć w procesie paska zadań.

Ukrywanie/pokazywanie przycisków na pasku zadań następuje poprzez sprawdzenie widoczności wybranego elementu za pomocą komunikatu *TB_ISBUTTONHIDDEN* i następnie ustawienie jej na przeciwną z użyciem komunikatu *TB_HIDEBUTTON*.

Listing 4.

```
ptbb = (TBBUTTON*)VirtualAllocEx(proces, NULL, sizeof(TBBUTTON), MEM_COMMIT, PAGE_READWRITE);
poz = ListView_GetNextItem(hList, (WPARAM)-1, LVNI_SELECTED);

SendMessage(hPasek, TB_GETBUTTON, (WPARAM)poz, (LPARAM)ptbb);
ReadProcessMemory(proces, (void*)ptbb, (void*)&tbb, sizeof(TBBUTTON), NULL);

hidden = SendMessage(hPasek, TB_ISBUTTONHIDDEN, (WPARAM)tbb.idCommand, 0);
SendMessage(hPasek, TB_HIDEBUTTON, (WPARAM)tbb.idCommand, (LPARAM)!hidden);

VirtualFreeEx(proces, ptbb, 0, MEM_RELEASE);
```

Oczywiście, jak widać na listingu 4., najpierw trzeba ponownie zaalokować pamięć, by można było pobierać informacje dotyczące przycisku. Należy również pobrać indeks

zaznaczonego elementu za pomocą funkcji *ListView_GetNextItem*. Po pobraniu danych i zakończeniu operacji należy zwolnić pamięć, którą zaalokowaliśmy wcześniej.

Teraz czas przejść do najistotniejszej rzeczy – zmieniania kolejności przycisków. Jak wspomniane zostało na początku, przenoszenie elementów listy jest wykonywane metodą drag & drop (z ang. *przeciągnij i upuść*). „Złapanie" elementu jest przekazywane do aplikacji przez komunikat *WM_NOTIFY*, który odnosi się do kontrolki ListView i którego element *code* parametru *lParam* jest równy *LVN_BEGINDRAG*. W obsłudze tego komunikatu tworzymy „duszka", znaczy półprzezroczysty obrazek, który pokazuje elementy, które przenosimy. Użytkownik wie wtedy, co aktualnie jest przeciągane. W pętli dodawane są obrazki dotyczące elementów danej grupy, by wiadomo było, że cała grupa jest przenoszona. Po stworzeniu „duszka" pobieramy współrzędne kursora i przeliczamy je z układu zależnego od współrzędnych okna do układu absolutnego – współrzędnych ekranu.

Listing 5.

```
case WM_NOTIFY:
else if (((LPNMHDR)lParam)->hwndFrom == hList && ((LPNMHDR)lParam)->code == LVN_BEGINDRAG)
{
    //wiadomość wysyłana do okna głównego, gdy rozpoczynamy drag & drop na liście
    //czyli łapiemy element lewym przyciskiem myszy
    HIMAGELIST hOneImageList, hTempImageList;
    IMAGEINFO imginf;
    int x = 0, wys;
    pkt.x = 1;
    pkt.y = 1;

    poz = ListView_GetNextItem(hList, (WPARAM)-1, LVNI_SELECTED);  //pobieranie indexu
zaznaczonego elementu
    item.iItem = poz;
    item.mask = LVIF_IMAGE | LVIF_INDENT | LVIF_PARAM; //ustawienie maski elementu listy do
pobrania
    ListView_GetItem(hList, &item);    //pobieranie danego elementu do zmiennej item

    if (item.iIndent) //element z wcięciem nas nie interesuje
        break;

    hDragImgList = ListView_CreateDragImage(hList, poz, &pkt); //tworzenie "duszka" do d&d
    ImageList_GetImageInfo(hDragImgList, 0, &imginf);
    wys = imginf.rcImage.bottom;

    while(true)    //dodawanie elementów danej grupy do "duszka" w pętli
    {
        if (++item.iItem >= ile)
            break;
        item.mask = LVIF_IMAGE | LVIF_INDENT | LVIF_PARAM; //ustawianie maski pobierania
danych elementu
        ListView_GetItem(hList, &item);
        if(item.iIndent == 0) //jeżeli napotkano kolejny przycisk grupowy trzeba przerwać
            break;

        hOneImageList = ListView_CreateDragImage(hList, item.iItem, &pkt);
        hTempImageList = ImageList_Merge(hDragImgList, 0, hOneImageList, 0, 0, wys);
        ImageList_Destroy(hDragImgList);
        ImageList_Destroy(hOneImageList);
        hDragImgList = hTempImageList;
        ImageList_GetImageInfo(hDragImgList, 0, &imginf);
        wys = imginf.rcImage.bottom;
    }

    item.iItem = poz;
    item.mask = LVIF_IMAGE | LVIF_INDENT | LVIF_PARAM;
    ListView_GetItem(hList, &item);

    ImageList_BeginDrag(hDragImgList, 0, x, 0);
    pkt = ((NM_LISTVIEW*) ((LPNMHDR)lParam))->ptAction;
    ClientToScreen(hList, &pkt);
    ImageList_DragEnter(GetDesktopWindow(), pkt.x, pkt.y);
    bDragging = true; //by wiadomo było, że d&d jest w trakcie
```

```
    SetCapture(hWnd);
}
break;
```

Jak widać na listingu 5., funkcją *ImageList_BeginDrag* sygnalizujemy programowi, że rozpoczynamy przeciąganie „duszka”. Po pobraniu współrzędnych kursora i przeliczeniu ich funkcją *ClientToScreen* należy użyć funkcji *ImageList_DragEnter* z uchwytem okna pulpitu, by operacja przeciągania odnosiła się do niego i by program wiedział, że ma zacząć rysować przenoszony obrazek.

Po obsłużeniu komunikatu omówionego wyżej trzeba zająć się komunikatem zmiany położenia kursora – *WM_MOUSEMOVE*. Oczywiście nie trzeba się nim zajmować, jeżeli przeciąganie nie zostało rozpoczęte – w tym momencie wykorzystujemy zmienną *bDragging*.

Listing 6.

```
case WM_MOUSEMOVE:  //zdarzenie ruchu kursora myszy
if (!bDragging) //jeżeli nie obsługujemy akurat d&d na liście to nie trzeba nic robić
    break;

pkt.x = LOWORD(lParam);
pkt.y = HIWORD(lParam);
ClientToScreen(hWnd, &pkt);
ImageList_DragMove(pkt.x, pkt.y);
break;
```

Jeżeli zmienna ta ma wartość *true*, to pobieramy współrzędne kursora i przeliczamy je ponownie funkcją *ClientToScreen*. Po tym możemy ruszyć przeciągany obrazek za pomocą funkcji *ImageList_DragMove* i przerysować go w danym miejscu.

Ostatnim etapem przeciągania elementu listy na inne miejsce jest obsłużenie komunikatu puszczania lewego przycisku myszy – *WM_LBUTTONUP*. W tym komunikacie kończymy operację przeciągania, po czym funkcją *ListView_HitTest* sprawdzamy, w jakim miejscu został upuszczony przeciągany element.

Listing 7.

```
case WM_LBUTTONUP:
if (!bDragging) //jeżeli nie obsługujemy akurat d&d na liście to nie trzeba nic robić
    break;

LVHITTESTINFO lvhti;
char buf[256], sub[12];

/*---puszczamy przycisk, więc przeciąganie się kończy---*/
bDragging = false;
ImageList_DragLeave(hList);
ImageList_EndDrag();
ReleaseCapture();


/*---sprawdzanie, czy przeciągany element został upuszczony na inny element listy---*/
lvhti.pt.x = LOWORD(lParam);
lvhti.pt.y = HIWORD(lParam);
ClientToScreen(hWnd, &lvhti.pt);
ScreenToClient(hList, &lvhti.pt);
ListView_HitTest(hList, &lvhti);

    ListView_GetItemText(hList, poz, 0, buf, 256); //pobieranie tekstu danego elementu listy
    while (true)
    {
        int tmp = poz;

        item.iItem = poz;
        item.iSubItem = 0;
        item.cchTextMax = 256;
        item.pszText = buf;
        item.stateMask = ~LVIS_SELECTED;
```

```
        item.mask = LVIF_STATE | LVIF_IMAGE | LVIF_INDENT | LVIF_TEXT | LVIF_PARAM;
        ListView_GetItem(hList, &item);
        ListView_GetItemText(hList, poz, 1, sub, 12);

        SendMessage(hPasek, TB_MOVEBUTTON, (WPARAM)poz, (LPARAM)lvhti.iItem);
        if (lvhti.iItem > poz && lvhti.iItem < ile)
            lvhti.iItem++;
        item.iItem = lvhti.iItem;

        ListView_InsertItem(hList, &item);
        ListView_SetItemText(hList, item.iItem, 1, sub);
        if (lvhti.iItem < poz)
            poz++;
        ListView_DeleteItem(hList, poz);
        if (lvhti.iItem > tmp)
            lvhti.iItem--;
        if (lvhti.iItem < tmp)
            lvhti.iItem++;
        ListView_GetItemText(hList, poz, 0, buf, 256);
        item.iItem = poz;
        item.iSubItem = 0;
        item.cchTextMax = 256;
        item.pszText = buf;
        ListView_GetItem(hList, &item);
        if (!item.iIndent || poz > ile-1)
            break;
    }
}
break;
```

Najważniejszym i właściwie jedynym koniecznym do wykonania pożądanej operacji, czyli zmiany kolejności przycisków na pasku zadań jest wysłanie do niego komunikatu *TB_MOVEBUTTON* ze starą i pożądaną pozycją danego przycisku jako parametrami. Pozostałe linie konieczne są do przesuwania elementów listy. Łatwiejszym rozwiązaniem jest oczywiście wywołanie funkcji *refresh* po przesunięciu przycisku na pasku, jednak byłoby to wykonywanie ponownie zbędnych operacji pobierania wszystkich przycisków.

5. Dodatkowe informacje

Manipulator może zostać uruchomiony z parametrem *–hide*. Wtedy okno programu automatycznie zminimalizuje się do zasobnika systemowego – jest to bardzo wygodne, jeżeli chcemy uruchamiać program ze startem systemu. By tak zrobić wystarczy kliknąć prawym przyciskiem na ikonie programu, z menu kontekstowego wybrać *Wyślij do -> Pulpit (utwórz skrót)*, następnie we właściwościach utworzonego skrótu w polu *Element docelowy* należy dopisać *–hide* do ścieżki programu, która już tam jest wpisana.

6. Literatura

[1] Microsoft Corporation, *Microsoft Developer Network,* http://msdn.microsoft.com/en-us/library/default.aspx,