4. KOMUNIKACJA MIĘDZY PROCESAMI

Obiekty**J**ądra mogą być współużytkowane przez wątki działające w **oddzielnych** procesach.

1. **ObiektMapowaniePliku** pozwala korzystać z tych samych danych dwóm procesom.
2. **Muteksy**, **semafory** pozwalają wątkom różnych procesów synchronizować działanie.
3. **Gniazda pocztowe**, **łącza z nazwą** pozwalają przesyłać dane między procesami działającymi w różnych komputerach.

Uchwyty **O**biektów**J**ądra **nie są wartościami ogólnosystemowymi**, są związane z określonymi procesami, co wyklucza ich **proste** współużytkowanie.

Obiekty**J**ądra są chronione i inny proces musi uzyskać **zgodę** na dostęp do takiego Obiektu.

Metody dzielenia Obiektów z innymi procesami:

-dziedziczenie uchwytów,
-nadawanie Obiektom nazw,
-tworzenia duplikatów.

4.4.1. Dziedziczenie uchwytów Obiektów

Tylko wtedy, gdy jeden proces jest procesem potomnym drugiego.

Proces nadrzędny, który ma dostęp do uchwytów ObiektówJądra, może utworzyć proces **pod**rzędny i umożliwić mu dostęp do swoich Obiektów.

➔ Proces **nadrzędny** informuje system, że **uchwyty** tworzonych **O**biektów**J**ądra mają być dziedziczone (tylko ***uchwyt*** a nie sam Obiekt).

W tym celu inicjalizuje strukturę SECURITY_ATTRIBUTES i przekazuje jej adres do konkretnej funkcji ***CreateNazwa***(...).

```
typedef struct _SECURITY_ATTRIBUTES { // sa
    DWORD     nLength;
    LPVOID    lpSecurityDescriptor;
    BOOL      bInheritHandle;
} SECURITY_ATTRIBUTES;
```

Poniższy **kod** należy zamieścić w procesie macierzystym przed wywołaniem **CreateFileMapping**.

```
SECURITY_ATTRIBUTES sa;
sa.nLength = sizeof(sa);
sa.lpSecurityDescriptor = NULL;
sa.bInheritHandle = TRUE;
        hNazwaObiektu = CreateNazwaObiektu(…, &sa , ...);
```

➔ Następnie proces nadrzędny tworzy proces **potomny** funkcją ***CreateProcess*** :

BOOL **CreateProcess**(❶,❷,❸,❹, ***bInheritHandles***, ❻,❼,❽,❾,❿);

bInheritHandle = **TRUE** oznacza, że tworzony proces potomny odziedziczy ***jakieś*** uchwyty rodzica.

➲ Wówczas SO tworząc proces potomny wstrzymuje jego natychmiastowe wykonanie.

Proces potomny (nowy proces) otrzymuje pustą Tablicę Uchwytów.

SO przegląda Tablicę Uchwytów procesu **nadrzędnego** i po znalezieniu pozycji z ważnymi uchwytami ***dziedzicznymi*** kopiuje je w niezmienionej postaci do Tablicy Uchwytów procesu potomnego, w to samo miejsce, które zajmowały w Tablicy procesu **nadrzędnego**.

Wartości uchwytów identyfikujących **dany** **O**biekt**J**ądra jest identyczna w **obu** procesach.

System zwiększa licznik użyć tych Obiektów, gdyż aktualnie każdy z nich używany jest przez dwa procesy.

Zniszczenie takiego Obiektu wymaga, aby oba procesy wywołały dla niego funkcję ***CloseHandle***.

Proces nadrzędny może zamknąć swój uchwyt Obiektu zaraz po powrocie z funkcji ***CreateProcess*** i nie wpłynie to na możliwości operowania tym Obiektem w procesie potomnym.

Dziedziczenie uchwytów Obiektów następuje tylko w chwili tworzenia procesu potomnego.

Jeżeli później proces nadrzędny utworzy nowy ObiektJądra z dziedzicznym uchwytem, żaden z już działających jego procesów potomnych nie odziedziczy tego uchwytu.

☛ **Proces potomny dziedziczący uchwyt nie jest w stanie wykryć tego faktu.**

Dziedziczenie uchwytu **O**biektu**J**ądra jest praktycznie przydatne, gdy proces potomny będzie poinformowany o tej możliwości (np.: dokumentacja).

W Tablicy procesu potomnego znajduje się wartość uchwytu identyfikująca Obiekt powstały w procesie rodzicielskim, lecz dzieciak nie wie, jaka to wartość.

☛ **Dziedziczoną wartość uchwytu należy świadomie przesłać do procesu potomnego**.

Proces **Map1a**Dziedziczenie sortuje zbiór liczb typy double.

Proces potomny **Map1b**Dziedziczenie generuje dane dla programu **Map1a**Dziedziczenie.

Wykorzystywana jest technika **mapowania pliku** na PAO do przekazywania danych między procesami.

Proces **Map1a**Dziedziczenie ustawia pola struktury **SECURITY_ATTRIBUTES sa**

Obiekt**M**apowany**P**lik tworzony jest przez **rodzica - Map1a**Dziedziczenie.

Wartość **uchwytu** **OMP** przekazywana jest to procesu potomnego - **Map1b**Dziedziczenie, poprzez argumentu linii polecenia (**cmd1**).

```
                                  //wspoldzielenie pliku danych przez DZIEDZICZENIE UCHWYTU
                                  // sortuje dane wygenerowane przez proces potomny Map1bDziedziczenie
#include<windows.h>
void BubbleSort(double *, int);
void DispV(int, int, double *, char *);
int main(int argc, char *argv[ ])                    // Map1aDziedziczenie     - macierzysty
{
HANDLE hFile = NULL;                                 // uchwyt do pliku
HANDLE hMapFile = NULL;                              // uchwyt do obiektu reprezentujacego plik zmapowany
BYTE *pMapFile;                                      // wskaznik na obszar RAM zmapowanego pliku
int nData = 40000;   double pocz = 3.3,   kon = 99.8;          // parametry generatora
char cmd1[128],   nameF[33] = "Map1a_Dziedz.bin";
DWORD sizeF = nData*sizeof(double);                  // rozmiar pliku dyskowego w bajtach
double *A = new double[nData];
hFile = CreateFile(nameF, GENERIC_READ|GENERIC_WRITE,0,0, CREATE_ALWAYS,0,0);
    if (hFile == INVALID_HANDLE_VALUE) {   printf("CreateFile error: %d.\n", GetLastError()); getchar(); return(1); }
  SECURITY_ATTRIBUTES   sa;
    sa.nLength = sizeof(sa);      sa.lpSecurityDescriptor = NULL;
    sa.bInheritHandle = TRUE;                        // zmiana uchwytu na dziedziczny.
hMapFile = CreateFileMapping (hFile, &sa, PAGE_READWRITE, 0, sizeF,  NULL);
            if(hMapFile==NULL) { printf("CreateFileMapping error: %d\n", GetLastError()); getchar(); return 1; }
pMapFile = (BYTE *)MapViewOfFile(hMapFile, FILE_MAP_WRITE, 0,  0,  0);
       if(pMapFile==NULL) { printf("in Map1aDz MapVievOfFile error: %d\n", GetLastError()); getchar(); return 1; }
STARTUPINFO si = {0};  si.cb = sizeof(STARTUPINFO);  PROCESS_INFORMATION  pi;
    sprintf(cmd1, "Map1bDziedziczenie %d %g %g %p", nData, pocz, kon, hMapFile);
BOOL OK = CreateProcess(0, cmd1, 0,0,
                                  TRUE,      // proces potomny odziedziczy uchwyt rodzica
                                  0,0,0, &si, &pi );
                        if(!OK) { printf("CreateProcess error: %d\n", GetLastError());  getchar(); return 1; }
    Sleep(2000);      // czekanie na zakończenie procesu potomnego
memcpy(A, pMapFile, sizeF);    // odczyt danych z pliku w PAO i przesłanie do tablicy A
      BubbleSort(A, nData);    // sortowanie tablicy A
memcpy(pMapFile, A, sizeF);    // zapis do pliku mapowanego w PAO zawartości tablicy A
delete [ ] A;
UnmapViewOfFile (pMapFile);
CloseHandle(hMapFile);    CloseHandle(hFile);

puts("\n------ odczyt kontrolny z pliku dyskowego w Map1aDziedziczenie:");
    double *B = new double[nData];
    FILE *pF1 = fopen(nameF, "rb");
      fread(B,  sizeof(double),  nData,  pF1); fclose(pF1);
      DispV(0, 16, B,  "" );  delete [ ] B;
puts("Koniec Map1aDziedziczenie");
 return 0;
 }
```

Zadanie 4.1
Odczyt kontrolny zawartości pliku funkcjami języka C++, **zastąpić funkcjami Systemu Operacyjnego**


```
                  // Proces potomny dla Map1aDziedziczenie,   generuje dane losowe.
                  // Parametry do generowania pobiera od procesu macierzystego
#include<windows.h>
#include<cmath>
#include<cstdio>
#include<cstdlib>
using namespace std;
void DispV(int, int, double *,, char *);
double Generuj(float a, float b)
    { double w = (a + (b - a)*(double)rand()/RAND_MAX);
       // for (long k=0; k < 1000; k++) log(pow((pow(sin(k)+1.1, 3.3)), 2.2));  // spowalniacz
    return floor(w*100+0.5)/100; }
int main(int argc, char *argv[ ])                    // Map1bDziedziczenie  - potomny
{
HANDLE hMapFile;
int nData;
double oda, dob;
if (argc <= 1 ) { puts("Brak Danych"); return 10; }
     else { nData = atoi(argv[1]);   oda = atof(argv[2]);   dob = atof(argv[3] );
     sscanf(argv[4], "%p", &hMapFile); }      // pobranie wartości uchwytu od rodzica
puts(".....poczatek procesu Map1bDziedziczenie.....");
double *A = new double[nData];
DWORD sizeF = nData*sizeof(double);                  // rozmiar pliku dyskowego w bajtach
   for (int i = 0; i < nData; i++)  A[i] = Generuj(oda, dob);
                  DispV(0, 16, A, "  Fragment danych w Map1bDziedziczenie (losowe):" );
BYTE *pMapFile = (BYTE *)MapViewOfFile(hMapFile, FILE_MAP_WRITE, 0,0, 0);
       if(pMapFile==NULL) { printf("in Map1bDz MapVievOfFile error: %d\n", GetLastError());  getchar(); return 1; }
       puts("  -->zapis Danych do pliku mapowanego");
       memcpy(pMapFile, A, sizeF);  // zapis danych z tablicy A do pliku mapowanego
// UnmapViewOfFile(pMapFile);
// CloseHandle(hMapFile);
delete [ ] A;
puts(".....koniec procesu Map1bDziedziczenie.");
// getchar();
return 0;
}
```

```
.....poczatek procesu Map1bDziedziczenie.....
 Fragment danych w Map1bDziedziczenie (losowe):
 3.42  57.69  21.95  81.34  59.75  49.61  37.10  89.76 82.70  75.35  20.10  86.19
71.86  52.86  32.64  4.75
 -->zapis Danych do pliku mapowanego
.....koniec procesu Map1bDziedziczenie.....
---- odczyt kontrolny z pliku dyskowego w Map1aDziedziczenie:
 3.36  3.38  3.38  3.42  3.46  3.47  3.51  3.66  3.71  3.72  3.74  3.75  3.77  3.80
3.86  3.87
Koniec Map1aDziedziczenie
```

4.2. Nadawanie nazw Obiektom

Niektórym Obiektom można nadawać nazwy (nie wszystkim), co pozwala organizować współużytkowania ObiektówJądra przez różne procesy.

Proces współużytkujący nie musi być potomkiem innego procesu.

Może zostać utworzony przez dowolną aplikację.

HANDLE CreateFileMapping(🡅, PSECURITY_ATTRIBUTES psa, 🡅, 🡅, PCTSTR ***pszNAME***);

HANDLE CreateMutex(PSECURITY_ATTRIBUTES psa, 🡅, PCTSTR ***pszNAME***);

Jeżeli ***pszNAME*** = **NULL**, to taki Obiekt może być współużytkowany przez różne procesy, po zastosowaniu mechanizmu ***dziedziczenia*** lub funkcji ***DuplicateHandle***.

➔Aby współużytkować Obiekt za pomocą **nazwy** należy ustawić w funkcji **Create**NazwaObiektu(...) parametr ***pszName*** na "łańcuch_Tekstowy".

Proces **A** został właśnie uruchomiony i wywołuje funkcję:

HANDLE hNProcess**A** = **Create*Nazwa***(🡅,...,🡅, "**MojaNazwa**");

Później **zupełnie inny** proces uruchamia Proces **B**.

Proces **B** rozpoczyna działanie, wykonując kod:

HANDLE hNProcess**B** = **Create*Nazwa*** 🡅,...,🡅,"**MojaNazwa**");

W momencie wywołania przez proces **B** funkcji **CreateNazwa** system sprawdza, czy nie istnieje już **O**biekt**J**ądra o nazwie "**MojaNazwa**"; Po znalezieniu takiego **O**biektu, system sprawdza jego **typ**.

Ponieważ jest to **O**biekt**Nazwa**, a proces **B** również próbuje utworzyć **O**biekt**Nazwa**, system sprawdza z kolei zabezpieczenia ***istniejącego*** **O**biektu**Nazwa**.

Jeśli proces **B** ma pełne prawa dostępu do tego Obiektu, system umieszcza w jego **T**ablicy **U**chwytów wskaźnik do istniejącego obiektu "**MojaNazwa**".

Jeżeli typy obu Obiektów nie są identyczne lub proces **B** nie praw dostępu, wywołanie funkcji *Create***Nazwa** kończy się niepowodzeniem.

Proces **B** uzyskuje wartość uchwytu identyfikujący **O**biekt**Nazwa**, który już istnieje w jądrze.

Powstanie w Tablicy Uchwytów procesu **B** nowego odwołania do **O**biektu**Nazwa** powoduje automatycznie zwiększenie wartości jego licznika użyć.

Aby zniszczyć **O**biekt**Nazwa** należy zamknąć jego uchwyt w procesie **A** i **B**.

Problem: otwierając **O**biekt**J**ądra o nazwie "**MojaNazwa**", nie ma żadnej gwarancji, że Obiekt o takiej nazwie już **nie istnieje**;

☛ ponadto wszystkie te **O**biekty mają wspólną przestrzeń nazw.

Z tego powodu poniższe wywołanie funkcji **CreateFileMapping** zwróci **error**:

```
HANLE hMutex = Create Mutex (NULL, FALSE, "MNazwa");
HANDLE hMap = CreateFileMapping(🡅, PSECURITY_ATTRIBUTES psa, 🡅, 🡅, "MNazwa");
DWORD dwErrorCode = GetLastError();     // dwErrorCode == ERROR_INVALID_HANDLE (6)
```


UWAGA: Proces **B** wywołując funkcję ***Create*Nazwa**, przekazuje przez jej parametry informacje istotne dla bezpieczeństwa Obiektu**.**

Jeśli Obiekt o podanej nazwie istnieje, parametry te są ignorowane!

Aplikacja może sprawdzić, czy utworzono nowy Obiektu, czy otwarto już istniejący.
Wystarczy zaraz po wywołaniu funkcji *Create***Nazwa** sprawdzić *GetLastError.*

```
HANDLE hMutex = CreateNazwa(🡅,...,🡅,"MojaNazwa");
    if (GetLastError() == ERROR_ALREADY_EXISTS)
      {
      // Otwarcie uchwytu już istniejącego obiektu.}
```

Proces **A**: **hFile** = **CreateFile**(nameF, GENERIC_WRITE, 0,0, OPEN_ALWAYS, 0,0);
hMapFile = **CreateFileMapping**(**hFile**, 🡅,...,🡅, "**Alfa**");

Proces **B**: hMapFile = **CreateFileMapping**((HANDLE)0xFFFFFFFF, 🡅,...,🡅, "**Alfa**");

Zamiast wywoływać **Create*****, proces może wywołać jedną z funkcji **Open*****:

HANDLE **Open**FileMapping(🡅,...,🡅, LPCSTR ***pszName***);
HANDLE **Open**Mutex(🡅,...,🡅,LPCSTR ***pszName***);
HANDLE **Open**Event(🡅,...,🡅, LPCSTR ***pszName***);
HANDLE **Open**Semaphore(🡅,...,🡅, LPCSTR ***pszName***);
HANDLE **Open**WaitableTimer(🡅,...,🡅, LPCSTR ***pszName***);

Ostatni parametr ***pszName*** (wskazuje nazwę ObiektuJądra) **nie może** przyjąć wartości NULL.

Należy ustawić ***pszName*** = "łańcuch_Tekstowy".

Funkcje **Open** sprawdzają, czy podana **nazwa występuje** już w przestrzeni nazw ObiektówJądra.

Jeśli **nie** to funkcje zwracają NULL, a *GetLastError* == ERROR_FILE_NOT_FOUND (2).

Jeśli **występuje**, i ma taki sam typ, jak podany w funkcji *Open****, system sprawdza, czy żądany dostęp (parametr *dwDesiredAccess)* jest dozwolony.

Jeśli tak, następuje aktualizacja Tablicy Uchwytów procesu wywołującego oraz zwiększenie licznika użyć Obiektu.

Jeśli parametr ***bInheritHandle*** przyjmie wartość TRUE, zwrócony przez funkcję uchwyt będzie dziedziczny.

Jeżeli podany Obiekt w funkcjach **Create*Nazwa*** i **Open*Nazwa*** nie istnieje to:

funkcja ***CreateNazwa*** utworzy go,

funkcja ***OpenNazwa*** zakończy się niepowodzeniem.

Na dysku znajduje się plik tekstowy: ***nowy.txt***

Program **Map1a**Nazwa mapuje plik dyskowy nowy.txt na PAO i wyświetla jego zawartość.

Następnie tworzy proces potomny ***Map1b****Nazwa,* który modyfikuje tekst w pliku, **nie** odwołując się do bezpośrednio do pliku dyskowego.

Na koniec program **Map1a**Nazwa wyświetla kontrolnie (funkcjami języka C/C++) zawartość zmodyfikowanego pliku tekstowego bezpośrednio z dysku.

!!! W programach docelowych NIE stosować funkcji plikowych języka C/C++ !!!

```
#include<windows.h>
#include<cstdio>
#include<cstdlib>
using namespace std;
int main()                                  // Map1aNazwa  -  mapowanie przez nazwę
{
HANDLE  hFile = NULL,
HANDLE   hMapFile = NULL;
char *pMapFile;                             // wskaźnik na początek obszaru PAO zmapowanego pliku
char nameF[30] = "nowy.txt";
DWORD  sizeF = 30;                  // założony rozmiar pliku dyskowego, który będzie mapowany
                                            // wyłącznie roboczy zapis danych We dla Map1aNazwa
                                              FILE *pF = fopen(nameF, "w");
                                              fputs("abcdefghijklmnoprstuvwz", pF);   fclose(pF);

hFile = CreateFile(nameF, GENERIC_READ|GENERIC_WRITE, 0,0, OPEN_ALWAYS,0,0);
     if (hFile == INVALID_HANDLE_VALUE) { printf("CreateFile error: %d.\n", GetLastError());  getchar(); return(1); }

hMapFile = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0,   sizeF,  "Alfa1" );
     if(hMapFile==NULL) { printf("CreateFileMapping error: %d\n", GetLastError()); getchar();  return 1; }

pMapFile = (char *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0,  0,  0);
    printf("\nOdczytane Dane ze zmapowanego pliku:\n");   puts(pMapFile);
    STARTUPINFO si = {0};  si.cb = sizeof(STARTUPINFO); PROCESS_INFORMATION  pi;
    CreateProcess(0,  "Map1bNazwa   Open",  0,0,0,0,0,0,  &si, &pi);  //❶
    Sleep(1000);                    // Czy konieczne ?  -  usunąć , zmienić parametr
UnmapViewOfFile(pMapFile);
CloseHandle(hFile);
CloseHandle(hMapFile);

puts("—-odczyt kontrolny z pliku dyskowego Map1aNazwa:        // realizacja funkcjami języka C++
char zn;    FILE *pF1 = fopen(nameF, "r");
while ((zn=getc(pF1)) !=EOF)  putc(zn, stdout);  fclose(pF1);
//getchar();
return 0;
}
```

Zadanie 4.2
Odczyt kontrolny zawartości pliku funkcjami języka C++, **zastąpić funkcjami Systemu Operacyjnego**


```
#include<windows.h>
#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
int main(int argc, char *argv[ ])                    // Map1bNazwa  - potomny
{
DWORD  sizeF = 30;
HANDLE  hMapFile;
HANDLE  hFile = (HANDLE)0xFFFFFFFF;                          // adres pliku wymiany

if (strcmp(argv[1], "Open") == 0)  {
     puts( ".....proces Map1bNazwa z Open ........");
     hMapFile = OpenFileMapping(FILE_MAP_WRITE,  FALSE,  "Alfa1");
     }
else if (strcmp(argv[1], "Create") == 0) {
     puts("......proces Map1bNazwa z Create ........");
    hMapFile=CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, sizeF,  "Alfa1");
   }
     if(hMapFile==NULL) { printf("***FileMapping error: %d\n", GetLastError());  getchar(); return 1; }

char *pMapFile = (char *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0,  0,  0);
      puts("     ***modyfikacja pliku bezposrednio w PAO");
      memcpy(pMapFile+1*6,  "12%456",  6);

UnmapViewOfFile (pMapFile);
CloseHandle(hMapFile);
puts(".....koniec procesu Map1bNazwa.");
return 0;
}
```

Odczytane Dane ze zmapowanego pliku:
abcdefghijklmnoprstuvwz
.....proces Map1bNazwa z Open .....
***modyfikacja pliku bezposrednio w PAO
.....koniec procesu Map1bNazwa.....
---- kontrolny odczyt z pliku dyskowego Map1aNazwa:
abcdef12%456mnoprstuvwz

Zadanie 4.3

Program **Map1b**Nazwa uruchomiony został z wykorzystaniem **Open**FileMapping().

Uruchomić program z wykorzystaniem **Create**FileMapping(…, **"Test"**).

Zadanie 4.4

Programy **`Map1aDziedziczenie`** i **`Map1bDziedziczenie`** nie działają optymalnie.

Zmodyfikować te programy tak, aby działały optymalnie.

HANDLE **OpenFileMapping(**

DWORD **dwDesiredAccess**, // access mode

BOOL **bInheritHandle**, // inherit flag

LPCTSTR **lpName** // pointer to name of file-mapping object

);

If the function succeeds, the return value is an open handle to the specified file-mapping object.

If the function fails, the return value is NULL.

To get extended error information, call GetLastError.

dwDesiredAccess:

FILE_MAP_WRITE Read-write access.

FILE_MAP_ALL_ACCESS same as FILE_MAP_WRITE

The target file-mapping object must have been created with PAGE_READWRITE protection.

FILE_MAP_READ Read-only access.

The target file-mapping object must have been created with PAGE_READWRITE or PAGE_READ protection.

FILE_MAP_COPY Copy-on-write access.

The target file-mapping object must have been created with PAGE_WRITECOPY protection.

bInheritHandle:

Specifies whether the returned handle is to be inherited by a new process during process creation.

A value of TRUE indicates that the new process inherits the handle.

Aby sprawdzić czy nadana nazwa Obiektu jest unikatowa. wystarczy w *main/WinMain* wywołać funkcję ***Create*Nazwa** (typ obiektu nie jest istotny), a po powrocie z niej sprawdzić *GetLastError.*

Błąd ERROR_ALREADY_EXISTS oznacza, że w systemie działa już druga instancja tej samej aplikacji.

Poniżej przykład kodu, który ilustruje tę metodę.

```
int WINAPI WinMain(HINSTANCE hExe, ⬆,...,⬆)
{
   HANDLE h = CreateMutex(NULL, FALSE, "Dowolna_Nazwa");
     if (GetLastError () == ERROR_ALREADY_EXISTS) {
                cout << W systemie działa już instancja tej aplikacji;
                return(0);
                }
    cout << To jest pierwsza instancja tej aplikacji w systemie;
    CloseHandle(h);
}
```


ANEKS 4.1. TWORZENIE DUPLIKATÓW UCHWYTÓW OBIEKTÓW

Metoda korzysta z funkcji **DuplicateHandle**, która:

-pobiera pozycję z TablicyUchwytów **jednego** procesu

-tworzy jej kopię w TablicyUchwytów **drugiego** procesu.

Funkcja związana jest z trzema procesami uruchomionymi w systemie.

Proces **A** może być procesem źródłowym, który aktualnie ma dostęp do pewnego ObiektuJądra,

Proces **B** jest procesem docelowym, który ma dopiero uzyskać dostęp do tego Obiektu,

Proces **C** jest procesem-katalizatorem, który wywołuje funkcję *DuplicateHandle.*

Jednak rzadko, kiedy wykorzystuje się w niej **aż trzy** różne procesy.

BOOL **DuplicateHandle(**

❶	HANDLE	*hSourceProcessHandle*,	// handle to process with handle to duplicate
❷	HANDLE	*hSourceHandle*,	// handle to duplicate
❸	HANDLE	*hTargetProcessHandle*,	// handle to process to duplicate to
❹	LPHANDLE	*lpTargetHandle*,	// pointer to duplicate handle
❺	DWORD	*dwDesiredAccess*,	// access for duplicate handle
❻	BOOL	*bInheritHandle*,	// handle inheritance flag
❼	DWORD	*dwOptions* **)**;	// optional actions

1-szy i 3-ci: uchwyty ObiektówJądra względem procesu wywołującego funkcję *DuplicateHandle.*

Muszą to być ObiektyProcesy (powstają, gdy uruchamiany jest nowy proces).

2-gi: uchwyt ObiektuJądra dowolnego typu, jego wartość jest podawana na podstawie procesu identyfikowanego przez uchwyt *hSourceProcessHandle.*

4-ty: adres zmiennej, do której trafia indeks pozycji z kopią informacji o uchwycie źródłowym. Wartość ta odnosi się do procesu identyfikowanego przez *hTargetProcessHandle.*

3-ostatnie: określają wartość maski dostępu i flagi dziedziczenia, które mają być użyte w docelowej pozycji kopiowanego uchwytu ObiektuJądra.

dwOptions: może mieć wartość 0 (zero) lub dowolną kombinację dwóch poniższych flag:

DUPLICATE_SAME_ACCESS -uchwyt docelowy ma mieć tę samą maskę dostępu, co uchwyt procesu źródłowego.

W wyniku użycia tej flagi funkcja *DuplicateHandle* ignoruje parametr *dwDesiredAccess.*

DUPLICATE_CLOSE_SOURCE -zamyka uchwyt w procesie źródłowym.

Zastosowania tej flagi nie zmienia licznika użyć ObiektuJądra

➔ Proces docelowy nie jest powiadamiany, że udostępniono w nim nowy ObiektJądra.

Należy „ręcznie” przekazać do procesu wartość uchwytu do udostępnionego Obiektu.

Można użyć mechanizmów komunikacji międzyprocesorowej IPC(Interprocess Communication).

❒ Przykład użycia funkcji *DuplicateHandle*

Proces ma prawo **C**zytania i **P**isania w **O**biekcie**M**apowany**Plik**.

W pewnym momencie zostaje wywołana funkcja, która ma odczytać coś z tego Obiektu.

Warto za pomocą ***DuplicateHandle*** utworzyć nowy uchwyt do OMPlik, nadać mu wyłącznie prawo czytania i dopiero wtedy przekazać go do funkcji ***Czytającej***.

```
int WINAPI WinMain(♥, ♥, ♥, ♥)
{
// utworzenie ObiektuMapowaniePliku z prawem Czytaj/Pisz.
 HANDLE hFileMapRW = CreateFileMapping((HANDLE)0xFFFFFFFF,  // plik będzie współdzielony
                                       NULL   PAGE_READWRITE, 0, 123454, NULL);
HANDLE  hFileMapRO;   // utworzenie dodatkowego uchwytu OMPliku tylko z prawem Czytania.
DuplicateHandle(GetCurrentProcess(),  hFileMapRW,    // dotyczy procesu tworzącego Obiekt
                GetCurrentProcess(), &hFileMapRO,  // dotyczy procesu gdzie będzie przekazany
                FILE_MAP_READ, FALSE, 0
                );
//wywołanie funkcji z prawem czytania OMPliku
My_Read_from_FileMapping(hFileMapRO);
CloIteHandle(hFileMapRO);          // zamknięcie uchwytu obiektu z samym prawem czytania
                  // nadal można czytać i pisać w OMPliku za pomocą hFileMapRW
CloseHandle(hFileMapRW);          // zamykanie  OMPliku  po zakończeniu pracy
}
      // Wywołanie funkcji GetCurrentProcess zwraca pseudouchwyt, identyfikujący bieżący proces.
```


NOTATKI