Instrukcja laboratoryjna nr 4
Synchronizacja zdarzeniami i semaforami

oprac. Robert Tomaszewski

Ogólna zasada pracy ze zdarzeniami przypomina nieco sekcje krytyczne (obiekt może być w jednym z dwóch stanów: zasygnalizowanym ("zielone światło") i niezasygnalizowanym ("czerwone światło"):

1. Deklaracja obiektu zdarzenia (typ HANDLE)
2. Utworzenie obiektu zdarzenia - CreateEvent (obiekt można utworzyć jako wstępnie zasygnalizowany lub niezasygnalizowany)
3. Zasygnalizowanie obiektu - SetEvent ("zapalenie zielonego światła")
4. Odsygnalizowanie obiektu - ResetEvent ("zapalenie czerwonego światła")
5. Czekanie na zasygnalizowanie obiektu zdarzenia - WaitForSingleObject (podobne do wejścia do sekcji krytycznej)
6. Czekanie na zasygnalizowanie jednego lub całej grupy obiektów zdarzeń - WaitForMultipleObjects (funkcja bardziej zaawansowana od poprzedniczki) - poniżej przykład użycia zaczerpnięty z dokumentacji WinAPI Microsoftu:

```
HANDLE hEvents[2];
DWORD i, dwEvent;

/* Tworzymy dwa obiekty zdarzeń */
for (i = 0; i < 2; i++)
{
    hEvents[i] = CreateEvent(
        NULL,   /* bez atrybutów bezpieczeństwa */
        FALSE,  /* obiekt sam się zresetuje po udanym wykonaniu funkcji
                   WaitForSingleObject/WaitForMultipleObjects co czyni funkcję ResetEvent
                   zbędną w tym programie */
        FALSE,  /* stan początkowy - niezasygnalizowany */
        NULL);  /* nie będziemy nazywać obiektu */
    if (hEvents[i] == NULL) {
        printf("CreateEvent error: %d\n", GetLastError() );
        ExitProcess(0);
    }
}

/* Wątek główny ("stwórca") czeka, aż pozostałe wątki zasygnalizują obiekty */
dwEvent = WaitForMultipleObjects(
    2,          /* liczba obiektów zdarzeń w tablicy */
    hEvents,    /* tablica owych obiektów */
    FALSE,      /* czekaj, aż przynajmniej jeden obiekt będzie zasygnalizowany */
    INFINITE);  /* czekaj nieskończenie długo */

/* A teraz sprawdzimy który obiekt zdarzenia został zasygnalizowany */
switch (dwEvent) {
    /* hEvent[0] został zasygnalizowany */
    case WAIT_OBJECT_0 + 0:
        /* Wykonaj instrukcje synchronizowane tym zdarzeniem */
        break;
    /* hEvent[1] został zasygnalizowany */
    case WAIT_OBJECT_0 + 1:
        /* Wykonaj instrukcje synchronizowane tym zdarzeniem */
        break;
    /* Zwrócona wartość (dwEvent) jest błędna */
    default:
        printf("Wait error: %d\n", GetLastError());
        ExitProcess(0);
}
```

Semafory

Obiekty służące do synchronizacji operacji w programach wielowątkowych przypominające obiekty zdarzeń (events). Podobnie jak zdarzenia semafory mogą być w jednym z dwóch stanów: **zasygnalizowanym** ("zielone" światło) i **niezasygnalizowanym** ("czerwone" światło), ale w przeciwieństwie do obiektów zdarzeń **semafory posiadają licznik** przechowujący wartość od zera do pewnej wartości maksymalnej (liczba

całkowita dodatnia). Semafor jest w stanie zasygnalizowanym dopóty, dopóki licznik jego licznik jest większy od zera. Każdorazowe udane wywołanie w programie/wątku funkcji oczekiwania na obiekt (np. WaitForSingleObject) powoduje dekrementację licznika semafora. Wywołanie w programie/wątku funkcji oczekiwania w momencie, kiedy licznik osiągnie zero spowoduje zatrzymanie programu/wątku na takim wywołaniu. Zwiększenie licznika semaforu (czyli wznowienie zawieszonego wątku) odbywa się poprzez wywołanie odpowiedniej funkcji w programie/wątku, który nie został zawieszony (tzn. działa w momencie zawieszenia innego wątku/watków). Semafor działa jak zawór limitujący jednoczesny dostęp do pewnego zasobu dla podzbioru wątków z większego ich zbioru (np. ze zbioru 10 wątków tylko 5 ma prawo działać współbieżnie, reszta musi czekać). Zdarzenia (events) są formą semafora binarnego (wartość 0 lub 1).

Operacje na semaforach:

1. Deklaracja obiektu semafora (typ HANDLE)
2. Utworzenie obiektu semafora - CreateSemaphore (należy określić wstępną i maksymalną wartość licznika, żadna z nich nie może być mniejsza od zera a dodatkowo wartość wstępna nie może przekroczyć maksymalnej)
3. Czekanie na zasygnalizowanie obiektu zdarzenia - WaitForSingleObject (jeżeli licznik semaforu jest większy od zera to nastąpi przepuszczenie przez tą funkcję oraz dekrementacja licznika, w przeciwnym wypadku funkcja zawiesi wątek)
4. Zwolnienie semaforu (inkrementacja licznika) - ReleaseSemaphore
5. Usunięcie zasobów związanych z semaforem – CloseHandle

Program przykładowy ex4.c (przerobione ex1.c i ex2.c)

```
/* Program demonstrujący użycie obiektu semafora do synchronizacji wątków */

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

/* Definicja typu strukturalnego i typu wskaźnikowego na strukturę */
typedef struct
{
   char napis[10];
   int liczba;
} TPARAM, *PTPARAM;

/* Deklaracja obiektu semafora - można stworzyć kilka obiektów:
   sem1 - do synchronizacji wątków roboczych z wątkiem głównym
   sem2 - do synchronizacji wątków roboczych pomiędzy sobą
*/
HANDLE sem1 = NULL, sem2 = NULL;

/* Funkcja wątku - kilka wątków może wykonywać tą samą lub różne funkcje */
DWORD WINAPI proc(LPVOID arg)
{
 int i,licznik1=0; /* prywatne dane wątku */
 PTPARAM parametr; /* zmienna na parametr przekazany do wątku - u nas struktura */

 /* Wejście do sekcji "zasemaforowanej" */
 WaitForSingleObject(sem1,INFINITE);
 for (i=0;i<10;i++)
 {
  ++licznik1;
  printf("WATEK-licznik watku: %d\n",licznik1);
 }
 parametr = (PTPARAM) arg; /* przypisanie argumentu do zmiennej prywatnej */
 /* zabezpieczenie semaforem niepodzielności operacji na polach struktury */
 WaitForSingleObject(sem2,INFINITE);
 printf("Parametr watku:%s, %u\n",parametr->napis,parametr->liczba++);
 /* Opuszczenie obiektów semaforów - zwiększenie ich liczników o 1 */
 ReleaseSemaphore(sem2,1,NULL);
 ReleaseSemaphore(sem1,1,NULL);
}

int main(void)
{
 TPARAM zm; /* tą zmienną przekażemy do funkcji wątków */
```

```
 DWORD id1, id2;
 int licznik=0;

 /* Wypełnienie pól struktury */
 printf("Wpisz jakis tekst (do 10 znakow):");
 scanf("%s",zm.napis);
 printf("A teraz liczbe calkowita dodatnia:");
 scanf("%u",&zm.liczba);
 /* Utworzenie obiektu semafora do synchronizacji 2ch wątków z wątkiem głównym z
    licznikiem maks. i początkowym równym 2 */
 sem1 = CreateSemaphore(NULL,2,2,NULL);
 /* Utworzenie obiektu semafora do synchronizacji 2ch wątków między sobą z licznikiem
    maks. i początkowym równym 1 */
 sem2 = CreateSemaphore(NULL,1,1,NULL);
 /* Utworzenie wątków - bez zapamiętania uchwytów */
 if (CreateThread(NULL,0,proc,&zm,0,&id1) != NULL)
  printf("Watek utworzony!\n");
 if (CreateThread(NULL,0,proc,&zm,0,&id2) != NULL)
  printf("Watek utworzony!\n");
 /* Program/wątek główny też coś robi */
 for (licznik=0;licznik<=5;licznik++)
  printf("PROGRAM: Licznik globalny programu: %d\n",licznik);
 printf("Program czeka na wejscie do sekcji semafora...\n");
 WaitForSingleObject(sem1,INFINITE);
 printf("Parametr watku (zmieniony przez watek):%s, %u\n",zm.napis,zm.liczba);
 CloseHandle(sem1);
 CloseHandle(sem2);
 system("PAUSE");
 return 0;
}
```

ZADANIE: Zapoznaj się z przykładami w tej instrukcji demonstrującym sposób przekazywania parametrów do funkcji wątków oraz zasady użycia zdarzeń i semaforów do synchronizacji. Napisz program, który pobiera od użytkownika 20 liczb (zapisując je w zmiennej tablicowej), tworzy dwa wątki i przesyła do nich (jako parametry ich funkcji) po połowie tablicy. Wątki sortują otrzymane połówki tablicy, informują program/wątek główny o zakończeniu swojej pracy, zaś ten wyświetla posortowane połówki, następnie scala je w jedną posortowaną całość i ją wyświetla.
Napisz dwie wersje programu: w pierwszej do synchronizacji wątków roboczych z wątkiem głównym użyj obiektów zdarzeń, w drugiej - semaforów.

PRZYDATNE ŹRÓDŁA:
Opis wątków i procesów w Windows (interesują nas tylko wątki - threads):
http://msdn2.microsoft.com/en-us/library/ms684841.aspx
Pisanie programów wielowątkowych w Windows:
http://msdn2.microsoft.com/en-us/library/y6h8hye8(VS.71).aspx
Synchronizacja wielobieżności w programach Windows (interesują nas sekcje krytyczne, semafory i zdarzenia – critical section, semaphores, events)
http://msdn2.microsoft.com/en-us/library/ms686353.aspx
Opisy biblioteczne funkcji i struktur używanych w programowaniu wielowątkowym:
http://msdn2.microsoft.com/en-us/library/aa908719.aspx