PROCESY WSPÓŁBIEŻNE

PROCESOR

Procesor jest elementem (jednostką) wykonującym rozkazy.

Procesor może być:

- sprzętowy - wykonuje ciąg operacji wybranych z listy rozkazów procesora. Pobiera rozkazy z pamięci operacyjnej i kolejno je wykonuje.
- sprzętowo-programowy - jest wynikiem połączenia procesora sprzętowego z oprogramowaniem. Rozkazy interpretowane są przez oprogramowanie.

PROGRAM

Program jest to statyczny zestaw instrukcji wykonywany przez jednostkę centralną (procesor) realizujący określone zadanie.

Zestawy instrukcji mogą być w formie:

- źródłowej (tekstowej) – zrozumiałej dla programisty
- binarnej (wynikowej) zrozumiałej dla maszyny.

PROCES

Proces (zadanie) jest to dynamiczny ciąg działań wykonywanych za pośrednictwem programu lub sprzętu.

- Proces można nazwać „wykonywanym programem”.
- W danej chwili na jednym procesorze tylko jeden proces jest aktywny.
- W skład procesu wchodzą: kod programu, licznik rozkazów, sekcja danych oraz stos.
- Procesowi mogą być przydzielone: procesor, pamięć, dostęp do urządzeń we/wy oraz pliki.

PROGRAM A PROCES

program

procesor

proces

PROCES

- Jeżeli program uruchamiany jest przez np. 3 użytkowników to powstaną 3 procesy.
- Jeden proces może korzystać z wielu programów.
- W skutek uruchomienia programu może powstać wiele procesów

STANY PROCESU

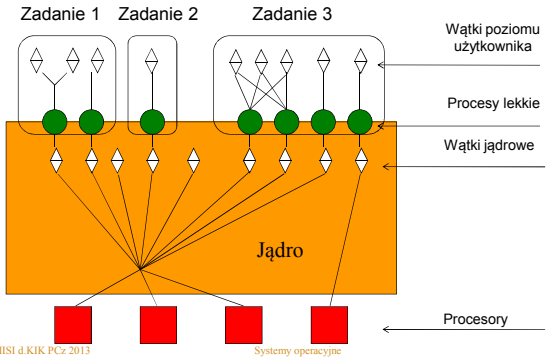
- Nowy
  – proces został utworzony.
- Wykonywany (aktywny, bieżący)
  – instrukcje procesu są aktualnie wykonywane
- Wykonywalny (gotowy)
  – proces czeka na przydział procesora
- Oczekujący (niewykonywalny)
  – proces czeka na wystąpienie jakiegoś zdarzenia.
- Zakończony
  – proces zakończył działanie.

WĄTEK

Wątek to proces lekki – wątki jednego zadania współdzielą kod i dane.

- Podstawowa cecha różniąca wątek od procesu: każdy proces posiada własną przestrzeń adresową, natomiast wątki posiadają wspólną sekcję danych oraz kod, ale różne stosy i liczniki rozkazów.
- Wątek jest podstawową jednostką wykorzystania procesora.
- Proces tradycyjny (nazywany ciężkim) jest równoważny zadaniu z jednym wątkiem.

WĄTKI W SYSTEMIE SOLARIS



WĄTKI I PROCESY W SYSTEMIE SOLARIS - BUDOWA

- Wątek jądrowy złożony jest z danych i stosu. Przełączanie wątków jądrowych jest stosunkowo szybkie.
- Proces lekki (LWP) zawiera blok kontrolny procesu z danymi rejestrowymi, informacjami rozliczeniowymi i informacjami dotyczącymi pamięci. Przełączanie procesów jest dość wolne.
- Wątek poziomu użytkownika wymaga tylko stosu i licznika rozkazów nie są mu potrzebne zasoby jądra, więc ich przełączanie jest szybkie. Bez względu na ilość wątków poziomu użytkownika dla jądra widoczne będą tylko procesy lekkie zadania.

WĄTKI I PROCESY W SYSTEMIE SOLARIS - PODSUMOWANIE

- Dowolne zadanie może mieć wiele wątków poziomu użytkownika.
- Wątki te mogą być planowane i przełączane bez interwencji jądra.
- Zablokowanie jednego z wątków użytkownika i podjęcie działania przez inny wątek nie wymaga przełączania kontekstu,
- Każdy proces LWP jest przyłączony do jednego wątku jądrowego,
- Wszystkie wątki poziomu użytkownika są od jądra niezależne.
- W zadaniu może być wiele procesów LWP, lecz są one używane tylko do komunikacji z jądrem.
- Gdy jeden proces lekki w zadaniu zostanie zablokowany, inne mogą kontynuować działanie w ramach zadania

WSPÓŁBIEŻNOŚĆ

Współbieżność polega na wykonywaniu wielu procesów jednocześnie.

- Należy zadbać o to by jedno nie zadanie miało negatywnego wpływu na inne zadania.

RODZAJE WSPÓŁBIEŻNOŚCI

- Współbieżność pełna.
- Współbieżność pozorna.
- Współbieżność koleżeńska
- Współbieżność z wywłaszczeniem

WSPÓŁBIEŻNOŚĆ PEŁNA

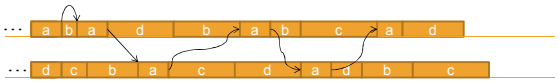
- Każdy proces jest wykonywany do końca przez jeden procesor
- Liczba procesów jest mniejsza bądź równa liczbie procesorów.

WSPÓŁBIEŻNOŚĆ POZORNA

- Występuje wówczas, gdy liczba procesów przewyższa liczbę procesorów, czyli liczba czynności jest większa od liczby wykonawców.
- System będzie dokonywał przełączeń pomiędzy procesami
- Konieczne jest zapamiętanie stanu procesu przy przełączaniu.

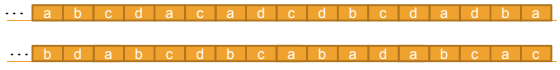
WSPÓŁBIEŻNOŚĆ KOLEŻEŃSKA

- Dany proces sam zwraca sterowanie do systemu operacyjnego, aby mógł on wykonać zadanie kolejne.
- Proces może zatem sam zarządzać czasem procesora.



WSPÓŁBIEŻNOŚĆ Z WYWŁASZCZENIEM

- Procesy same nie zwracają sterowania.
- System wywłaszcza proces z procesora.
- System operacyjny przekazuje sterowanie kolejnemu procesowi bez względu na proces poprzedni.



KOMUNIKACJA MIĘDZY PROCESAMI

Procesy:

- współdziałają w celu wykonania zadania zleconego przez użytkownika
- współzawodniczą o zasoby systemu

KATEGORIE RODZAJÓW KOMUNIKACJI:

- wzajemne wyłączanie
- synchronizacja
- zakleszczenie

WZAJEMNE WYŁĄCZNIE

Polega na zapewnieniu, aby tylko jeden proces naraz mógł korzystać z zasobów niepodzielnych.

ZASOBY DZIELI SIĘ NA:

- zasoby podzielne
- zasoby niepodzielne

ZASOBY PODZIELNE

- kilka procesów może korzystać z nich współbieżnie
- zabranie zasobu przez jakiś proces inny odbywa się bez żadnej szkody.

- Do zasobów podzielnych należą:
  - jednostki centralne
  - pliki przeznaczone wyłącznie do odczytu
  - pamięć wyłącznie do odczytu (kod lub stałe)

ZASOBY NIEPODZIELNE

- mogą być wykorzystywane tylko przez jeden proces
- Do zasobów niepodzielnych należą:
  - większość urządzeń zewnętrznych
  - pliki otwarte do zapisu
  - pamięć przeznaczona do zapisu

NIEPODZIELNOŚĆ ZASOBÓW WYNIKA Z

- natury fizycznej danego zasobu, która nie pozwala na jego współdzielenie. Typowym przykładem jest drukarka.
- tego, że czynności jednego procesu mogą zakłócać wykonywanie innego procesu.

PRZYKŁAD

proces A	zmienna x=5	proces B
czytaj x (5) ←	5	
	5 →	czytaj x (5)
dodaj 1 (6)	5	
	5	odejmij 1 (4)
zapisz x (6) →	6	
	4 ←	zapisz x (4)

Po dodaniu i odjęciu 1 zmienna powinna mieć wartość 5.
Niestety nie można przewidzieć wyniku!!!

SYNCHRONIZACJA

Procesy z założenia działają asynchronicznie.

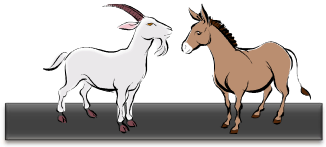
- Aby uzyskać zadowalającą współpracę, wyznacza się pewne punkty, w których procesy muszą synchronizować swoje działanie. Poza te punkty jeden proces nie może przejść do czasu, aż drugi proces mu nie zezwoli.

ZAKLESZCZENIE (BLOKADA)

Jeżeli kilka procesów współzawodniczy o zasoby niepodzielne, to może się zdarzyć, że każdy z nich będzie chciał skorzystać z zasobów, których używają inne procesy. Co prowadzi do tego, że żaden nie będzie mógł dalej działać.

Taka sytuacja nazywa się zakleszczeniem.

PRZYKŁADY ZAKLESZCZENIA



SEMAFORY

SEMAFOR – jest to nieujemna liczba całkowita, na której – z wyjątkiem nadawania wartości początkowych – mogą działać jedynie niepodzielne operacje czekaj i sygnalizuj.

- czekaj(S): while S <= 0 do nic;
  S:= S – 1;
- sygnalizuj(S): S:= S + 1;

SEMAFORY

Zmiany wartości semafora muszą być wykonywane za pomocą operacji czekaj i sygnalizuj w sposób **niepodzielny**.

Oznacza to, że gdy jeden proces modyfikuje wartość semafora, to inny proces nie może jednocześnie wartości tej zmieniać.

WZAJEMNE WYŁĄCZANIE

Gdy proces wykonuje operacje na zasobach niepodzielnych, żaden inny proces nie może wykonywać operacji na tych zasobach.

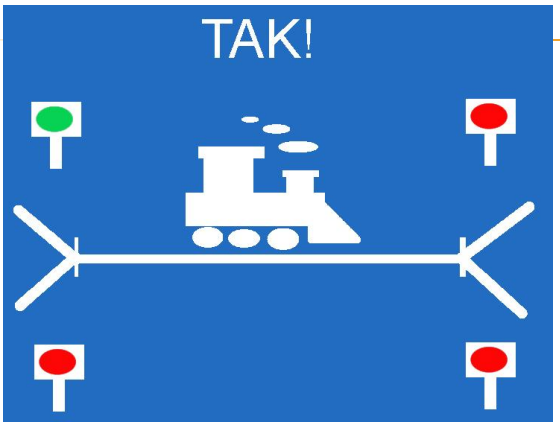
Fragment kodu, w którym proces odwołuje się do zasobów niepodzielnych, nazwany jest sekcją krytyczną.

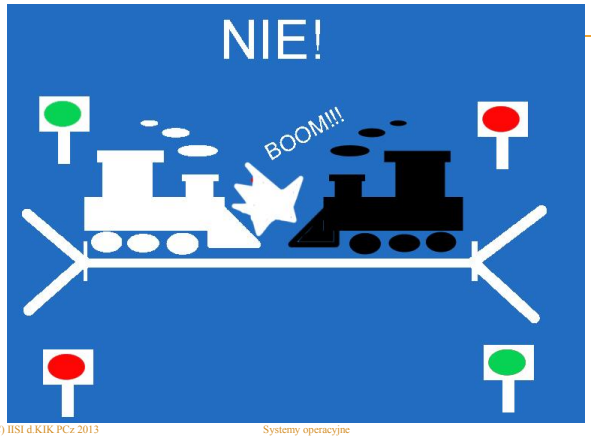
WZAJEMNE WYŁĄCZANIE

Ograniczenie dostępu do zasobów niepodzielnych przez niewspółbieżne wykonanie sekcji krytycznej jest realizowane następująco:

czekaj(s)
  sekcja krytyczna
sygnalizuj(s)

przy założeniu, że wartość początkowa s=1





WZAJEMNE WYŁĄCZANIE

Jeżeli do zasobu niepodzielnego może odwoływać się kilka procesów jednocześnie (np. pula drukarek), to wartość początkową semafora ustawia się na żądaną.

SYNCHRONIZACJA

proces A	proces B
czekaj(s)	
	sygnalizuj(s)

Proces A czeka na proces B.
Proces A jest synchronizowany z procesem B.
Wartość początkowa semafora s=0.

SYNCHRONIZACJA

proces A	proces B
czekaj(s1)	
	sygnalizuj(s1)
	czekaj(s2)
sygnalizuj(s2)	

Procesy synchronizują się wzajemnie.
Wartość początkowa semaforów s1 i s2 = 0.

PRZYKŁAD

Problem producenta i konsumenta

Proces producenta wytwarza jakieś informacje, umieszcza je w buforze N.

Proces konsumenta pobiera dane z bufora i je „konsumuje”.

PROCES PRODUCENTA

repeat nieskończenie
begin
  wytwórz element;
  czekaj(miejsce dostępne);
  czekaj(manipulowanie buforem);
  umieść element w buforze;
  sygnalizuj(manipulowanie buforem);
  sygnalizuj(element dostępny);
end;

PROCES KONSUMENTA

repeat nieskończenie
begin
  czekaj(element dostępny);
  czekaj(manipulowanie buforem);
  pobierz element z bufora;
  sygnalizuj(manipulowanie buforem);
  sygnalizuj(miejsce dostępne);
  zużyj element;
end;

PROBLEM PRODUCENTA I KONSUMENTA

repeat nieskończenie
begin
  wytwórz element;
  czekaj(miejsce dostępne);
  czekaj(manipulowanie buforem);
  umieść element w buforze;
  sygnalizuj(manipulowanie buforem);
  sygnalizuj(element dostępny);
end;

repeat nieskończenie
begin
  czekaj(element dostępny);
  czekaj(manipulowanie buforem);
  pobierz element z bufora;
  sygnalizuj(manipulowanie buforem);
  sygnalizuj(miejsce dostępne);
  zużyj element;
end;

Wartości początkowe semaforów:

miejsce dostępne=rozmiar bufora
element dostępny=0
manipulowanie buforem=1

PROBLEM PISARZY I CZYTELNIKÓW

Dowolna liczba pisarzy i czytelników.

Gdy pisarz pisze, to inny pisarz nie może pisać ani żaden czytelnik nie może czytać.

Gdy czytelnik czyta, to inni czytelnicy mogą czytać, ale pisarze nie mogą pisać.

PROCES PISARZA

czekaj (s)
pisarz pisze
sygnalizuj (s)

Wartość początkowa semafora s=1.

PROCES CZYTELNIKA

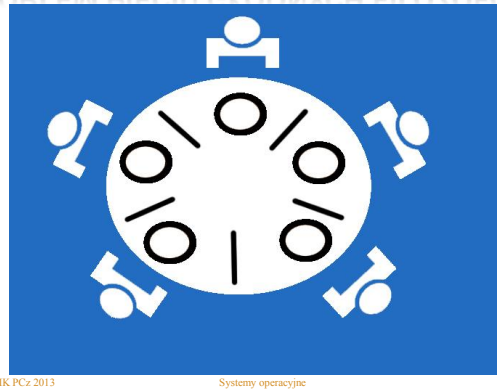
czekaj (M)
licznik:=licznik+1
if licznik = 1 **then** *czekaj (s)*
sygnalizuj (M) (zwalniany jest semafor M)
Odczyt
czekaj (M)
licznik:=licznik-1
if licznik=0 **then** *sygnalizuj (s)*
sygnalizuj (M)

Wartość początkowa semaforów s=1, M=1.

PROBLEM PIĘCIU GŁODNYCH FILOZOFÓW

- Filozofowie myślą i jedzą na zmianę.
- Każdy ma talerz i po obu stronach ma pałeczkę (jest tylko 5 pałeczek i każdy z filozofów może jeść tylko dwoma pałeczkami).
- Każdy z procesów (filozof) by działał musi wziąć 2 zasoby (pałeczki)
- Filozof żeby myśleć musi jeść.
- Co się stanie, gdy wszyscy filozofowie na raz zechcą jeść?

PROBLEM PIĘCIU GŁODNYCH FILOZOFÓW



PROBLEM PIĘCIU GŁODNYCH FILOZOFÓW

repeat
czekaj (S[i]) (bierze pałeczkę)
czekaj (S[i+1]mod 5) (2 pałeczki)
jedzenie
sygnalizuj (S[i])
sygnalizuj(S[i+1] mod 5)
until false

Jeżeli wszyscy filozofowie w tej samej chwili będą chcieli jeść (wszystkie procesy chcą skorzystać jednocześnie ze swoich zasobów), to powstanie ZAKLESZCZENIE.

PROBLEM PIĘCIU GŁODNYCH FILOZOFÓW

repeat
czekaj (j)
czekaj (S[i])
czekaj(S[i+1])
jedzenie
sygnalizuj(S[i])
sygnalizuj(S[i+1])
sygnalizuj (j)
until false

Można ograniczyć filozofom dostęp do jedzenia wprowadzając maksymalną liczbę próbujących jeść filozofów do 4.

Nawet jeśli wartość semafora ustawi się na 4 nie będzie zjawiska zakleszczenia ponieważ procesy pobierają dopuszczalną ilość zasobów i oddają je.

ZAKLESZCZENIE

Niewłaściwe odwołanie się do semaforów s1 i s2 (o wartościach 1) prowadzi do zakleszczenia.

proces A	proces B
.	.
.	.
czekaj(s1)	*czekaj(s2)*
.	.
.	.
czekaj(s2)	*czekaj(s1)*
.	.

KONSTRUKCJE SYNCHRONIZACYJNE

- monitor
- region krytyczny

MONITOR

Monitor składa się z:

- - danych
- - procedur dostępu
- - programu inicjującego

MONITOR

type m = monitor
  deklaracja zmiennych
  procedura p1
  procedura p2
  .....
  procedura pN
end;

begin
  inicjalizacje
end.

- Dostęp do zmiennych jest ograniczony.
- Można się do nich odwoływać wyłącznie przez procedury .
- W procedurach kompilator automatycznie umieszcza odwołania do niejawnie zadeklarowanych semaforów.
- Deklaracja monitora przypomina deklarację obiektu.

REGION KRYTYCZNY

Zadeklarowana zmienna *v* typu *T,* będzie używana wspólnie przez wiele procesów:

var *v*: **shared** *T*;

Zmienna *v* będzie dostępna tylko w obrębie instrukcji **region** o następującej postaci:

region *v* **when** *B* **do** *S*;

SEMAFORY W SYSTEMIE WINDOWS

- Create Semaphore(pSA, wo, w max, pN) : HANDLE
- Tworzy semafor o określonej nazwie. Zwraca uchwyt do niego.
- pSA – wskaźnik do struktury
- wo – wartość początkowa
- w max – wart. max semafora (ograniczenie od góry)
- pN – wskaźnik do nazwy semafora

SEMAFORY W SYSTEMIE WINDOWS

- Open Semaphore(at, if, pN) : HANDLE
- Zwraca uchwyt do semafora
- at – flaga dostępu do semafora
- if – mówi czy może być dziedziczony
- pN – wskaźnik do nazwy semafora

SEMAFORY W SYSTEMIE WINDOWS

- Release Semaphore(HANDLE, Const, pV) : BOOL
- Operacja sygnalizuj, zwiększa wartość semafora, którego wartość zostanie podana jako parametr ; musimy określić krok o ile będzie zmieniany (liczba całkowita). Podajemy liczbę o jaką możemy zwiększyć, niekoniecznie o 1.
- pV – wskaźnik do miejsca w pamięci, gdzie została umieszczona poprzednia wartość semafora (przed zwiększeniem)

SEMAFORY W SYSTEMIE WINDOWS

- Wait For Single Object(HANDLE, tms) : DWORD
- Czekaj na pojedynczy semafor (obiekt). Podajemy jako parametr uchwyt do semafora, a drugim parametrem jest czas oczekiwania. Jeśli czas zostanie przekroczony to operacja zostanie zakończona.
- WAIT_OBJECT_0 – semafor jest w stanie signaled, operacja zakończona powodzeniem
- WAIT_ABANDONED - gdy obiekt przestanie istnieć
- WAIT_TIMEOUT - jeśli czas oczekiwania zostanie przekroczony

SEMAFORY W SYSTEMIE WINDOWS

- Close Handle (HANLDE) : BOOL
- Funkcja usuwa pojedynczy semafor. Parametrem jest uchwyt do semafora. Funkcja ta usuwa też inne obiekty.

PRZEKAZYWANIE KOMUNIKATÓW

W skład narzędzi komunikacji międzyprocesowej wchodzą dwie podstawowe operacje:

- nadaj (komunikat)
- odbierz (komunikat).
- Długość komunikatu może być stała lub zmienna.
- Komunikaty mogą być kierowane do jednego procesu lub do wielu (rozgłoszeniowe).
- Można żądać potwierdzenia otrzymania komunikatu.
- Obie powyższe operacje mogą mieć różne postacie.

PRZEKAZYWANIE KOMUNIKATÓW

operacja blokująca - czeka, aż proces odbierający przyjmie komunikat.

operacja nieblokująca - po prostu umieszcza komunikat w pewnego rodzaju kolejce i pozwala, aby nadawca działał w dalszym ciągu.

PRZEKAZYWANIE KOMUNIKATÓW

Komunikacja bezpośrednia - każdy proces, który chce się komunikować, musi jawnie nazwać odbiorcę lub nadawcę uczestniczącego w tej wymianie informacji.

nadaj(P, komunikat)-nadaj komunikat do procesu *P*

odbierz(Q, komunikat)-odbierz komunikat od procesu *Q*

PRZEKAZYWANIE KOMUNIKATÓW

Komunikację bezpośrednią można przedstawić na przykładzie producenta i konsumenta.

Proces producenta:

```
repeat
   ...
   wytwarzaj jednostkę w nastp
   ...
   nadaj (konsument, nastp);
until false;
```

Proces konsumenta:

```
repeat
   odbierz (producent, nastk);
   ...
   konsumuj jednostkę z nastk
unitl false;
```


PRZEKAZYWANIE KOMUNIKATÓW

Komunikacja pośrednia - komunikaty są nadawane i odbierane za pośrednictwem *skrzynek pocztowych,* nazywanych także *portami*. Możliwość komunikacji między dwoma procesami istnieje tylko wtedy, gdy mają one jakąś wspólną skrzynkę pocztową.

nadaj*(A, komunikat)*-nadaj komunikat do skrzynki *A*

odbierz*(A, komunikat)*-odbierz komunikat ze skrzynki *A*

PRZEKAZYWANIE KOMUNIKATÓW

Buforowanie komunikatów - bufor ma pewną pojemność określającą liczbę komunikatów, które mogą w nim czasowo przebywać.

- Pojemność zerowa: Maksymalna długość kolejki wynosi 0, czyli łącze nie dopuszcza, by czekał w nim jakikolwiek komunikat. W tym przypadku nadawca musi czekać, aż odbiorca odbierze komunikat. Oba procesy muszą być zsynchronizowane.
- Pojemność ograniczona: Kolejka ma skończoną długość n, może w niej zatem pozostawać co najwyżej n komunikatów.
- Pojemność nieograniczona: Kolejka ma potencjalnie nieskończoną długość; może w niej oczekiwać dowolna liczba komunikatów. Nadawca nigdy nie jest opóźniany.

PRZEKAZYWANIE KOMUNIKATÓW

Sytuacje wyjątkowe:

- Zakończenie procesu - może się zdarzyć, że nadawca lub odbiorca zakończy działanie przed zakończeniem przetwarzania komunikatu. Pozostaną wówczas komunikaty, których nikt nigdy nie odbierze, lub jakieś procesy będą czekać na komunikaty, które nigdy nie zostaną wysłane.
- Utrata komunikatów - komunikat nadany przez proces *P* do procesu *Q* może zaginąć w sieci komunikacyjnej z powodu awarii sprzętu lub linii komunikacyjnej.
- Zniekształcenia komunikatów - komunikat może dojść do celu zniekształcony po drodze (np. wskutek zakłóceń w kanale komunikacyjnym).

STRUKTURA MSG W SYSTENIE WINDOWS

```
typedef struct {
   HWND hwnd;
   UINT message;
   WPARAM wParam;
   LPARAM lParam;
   DWORD time;
   POINT pt;
} MSG, *PMSG;
```

Hwnd – uchwyt do okna, którego procedura okna odbiera komunikaty.
Message – identyfikator komunikatu. Aplikacje mogą używać tylko młodszego słowa; starsze jest zarezerwowane dla systemu.
wParam – dodatkowa informacja o komunikacie. Znaczenie zależy od komunikatu.
lParam – dodatkowa informacja o komunikacie. Znaczenie zależy od komunikatu.
Time – czas wysłania komunikatu.
Pt – pozycja kursora na ekranie

KOMUNIKATY W SYSTEMIE WINDOWS

BOOL PostMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);

Funkcja wysyła komunikat (Msg+wParam, lParam) umieszczając go w kolejce komunikatów okna hWnd, nie czekając na jego przetworzenie.

HWND_BROADCAST – komunikat jest wysyłany do wszystkich okien top-level w systemie. Komunikat nie jest przesyłany do okien potomnych.

KOMUNIKATY W SYSTEMIE WINDOWS

VOID PostQuitMessage(int nExitCode);

Funkcja wysyła komunikat WM_QUIT do wątków aplikacji, najczęściej w odpowiedzi na komunikat WM_DESTROY, argumentem jest kod zamknięcia aplikacji.

KOMUNIKATY W SYSTEMIE WINDOWS

BOOL PostThreadMessage(DWORD idThread, UINT Msg, WPARAM wParam, LPARAM lParam);

Funkcja umieszcza komunikat (Msg+wParam, lParam) w kolejce komunikatów danego wątku idThread i natychmiastowo kończy działanie.

KOMUNIKATY W SYSTEMIE WINDOWS

LRESULT SendMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);

Funkcja wysyła komunikat (Msg+wParam, lParam) do okna hWnd lub wielu okien, wywołując funkcje obsługi danego komunikatu, funkcja kończy działanie dopiero po obsłużeniu komunikatu przez docelowe okno.

KOMUNIKATY W SYSTEMIE WINDOWS

BOOL GetMessage(LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);

Funkcja pobiera komunikat z kolejki okna o uchwycie hWnd i umieszcza go w strukturze lpMsg. Dopóki funkcja odbiera komunikaty różne od WM_QUIT, zwraca wartość większą od zera, w przeciwnym razie 0. wMsgFilterMin i wMsgFilterMax oznacają zakres odbieranych komunikatów.