Akademia Nauk Stosowanych
Wydział Nauk Inżynieryjnych
Kierunek: Informatyka
studia I stopnia, semestr 2

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Systemy operacyjne

WYKŁAD 10

dr inż. Stanisława Plichta

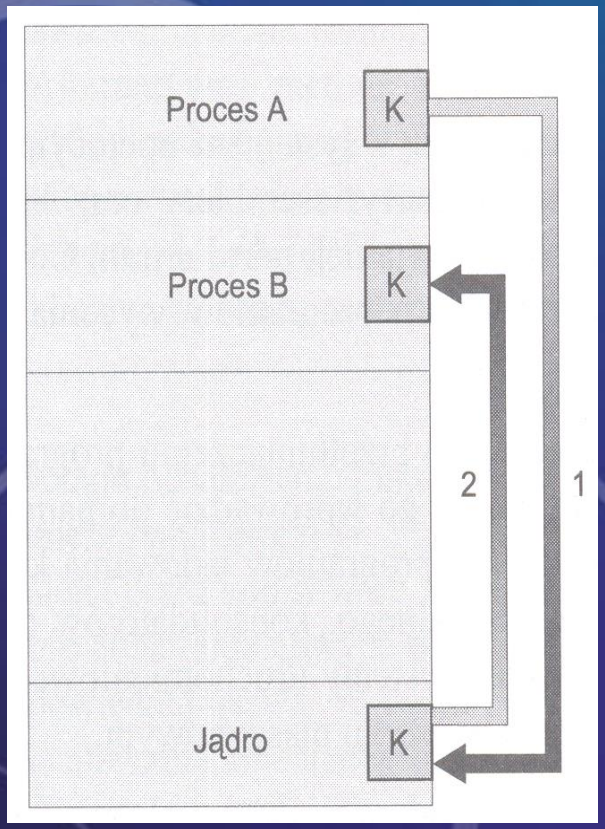
splichta@ans-ns.edu.pl

Komunikacja między procesami system Unix

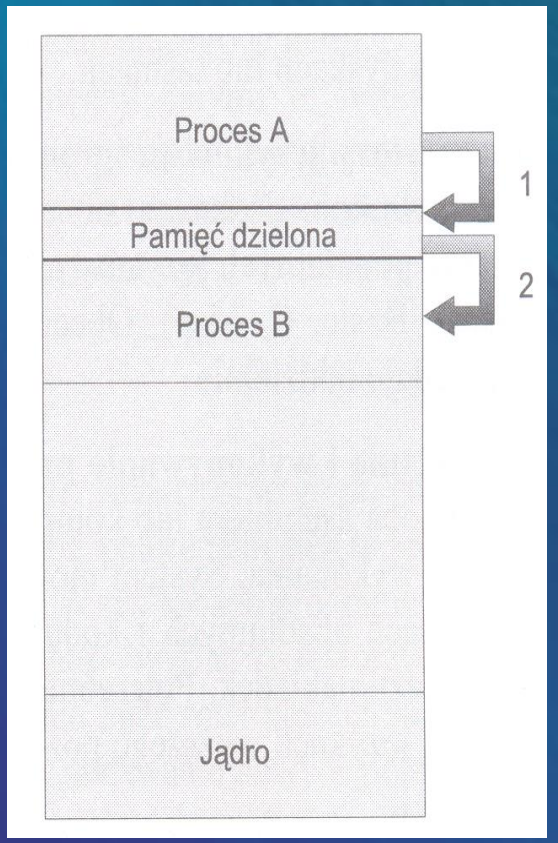
- Pamięć dzielona
- Łącza komunikacyjne
- Łącza nazwane (kolejki FIFO)
- Przekazywanie komunikatów
- Sygnały
- Pliki
- Semafory

autor: dr inż. Stanisława Plichta

Komunikacja między procesami system Unix



przekazywanie komunikatów



pamięć dzielona

autor: dr inż. Stanisława Plichta

Pamięć współdzielona

shmget(key_t key, int size, int shmflg)

- Tworzy segment współdzielonej pamięci o rozmiarze size, identyfikowany przez klucz key lub znajduje segment o takim kluczu, jeśli segment już istnieje.
- Zwraca identyfikator, który służy do odwoływania się do segmentu w pozostałych funkcjach operujących na pamięci współdzielonej.
- shmflg umożliwia przekazanie praw dostępu do kolejki oraz pewnych dodatkowych flag definiujących sposób jej tworzenia (np. IPC_CREAT), połączonych z prawami dostępu operatorem sumy bitowej (np.IPC_CREAT|0660).

Pamięć współdzielona

shmat(int shmid, const void *shmaddr, int shmflg)

Funkcja przydziela segmentowi współdzielonej pamięci, identyfikowanemu przez shmid, adres w przestrzeni adresowej procesu i zwraca ten adres jako wynik

shmdt(const void *shmaddr)

Funkcja powoduje odłączenie segmentu pamięci współdzielonej, umieszczonego pod adresem shmaddr

Pamięć współdzielona

shmctl(int shmid, int cmd, struct shmid_ds *buf)

Funkcja umożliwia wykonywanie operacji kontrolnych na segmencie pamięci współdzielonej, np. usunięcie tego segmentu

- Pamięć współdzielona to najszybszy sposób komunikacji między procesami.
- Komunikacja polega na tym, że ten sam obszar pamięci jest przydzielany dla kilku procesów.

Pamięć współdzielona - przykład

```
void utworzenie ()
  {
    pamiec=shmget(10,256,0777|IPC_CREAT);
    if (pamiec==-1)
      {
        printf("Problemy z utworzeniem
         pamieci dzielonej.\n");
        exit(EXIT_FAILURE);
      }
    else printf("Pamiec dzielona zostala
         utworzona: %d\n",pamiec);
  }
```

Pamięć współdzielona - przykład

```
void przydzielenie()
  {
    adres=shmat(pamiec,0,0);
    if (*adres==-1)
      {
        printf("Problem z przydzieleniem
         adresu.\n");
        exit(EXIT_FAILURE);
      }
    else printf("Przestrzen adresowa zostala
         przyznana : %s\n",adres);
  }
```

Pamięć współdzielona - przykład

```
void odlaczenie()
   {
     odlaczenie1=shmctl(pamiec,IPC_RMID,0);
   if (odlaczenie1==-1)
        {
          printf("Problemy z odlaczeniem
           pamieci dzielonej.\n");
          exit(EXIT_FAILURE);
        }
      else printf("Pamiec dzielona zostala
            odlaczona.\n");
   }
```

Pamięć współdzielona - przykład

```
void wstawienie()
   {
     printf("Wpisz cos do pamieci :");
     scanf("%s",adres);
   }

void pobranie()
   {
     printf("Biore z pamieci : %s",adres);
   }
```

Łącza komunikacyjne nienazwane

- Nie są plikami
- Mają swój i-węzeł
- Nie mają dowiązania
- Dotyczą procesów pokrewnych
- Powolne

- Jeśli proces czytający zbyt wyprzedzi proces piszący, to oczekuje na dalsze dane.
- Jeśli proces piszący zbyt wyprzedzi proces czytający, to zostaje uśpiony.

Łącza komunikacyjne nienazwane

pipe(int filedes[2])

Funkcja tworzy i otwiera potok oraz zwraca tablicę dwóch deskryptorów:

fd[0] umożliwiający czytanie z łącza

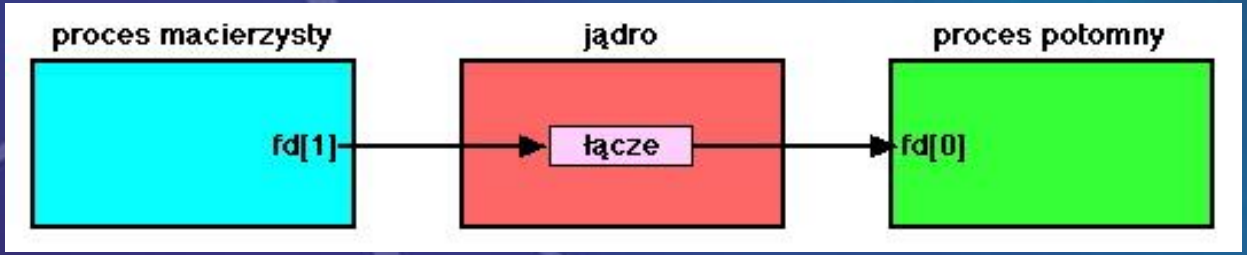
fd[1] umożliwiający pisanie do łącza

- Ponieważ dwa procesy mogą się komunikować przez łącze tylko w jedną stronę, więc żaden z nich nie wykorzysta z obydwu deskryptorów.
- Każdy z procesów powinien zamknąć nieużywany deskryptor łącza za pomocą funkcji:

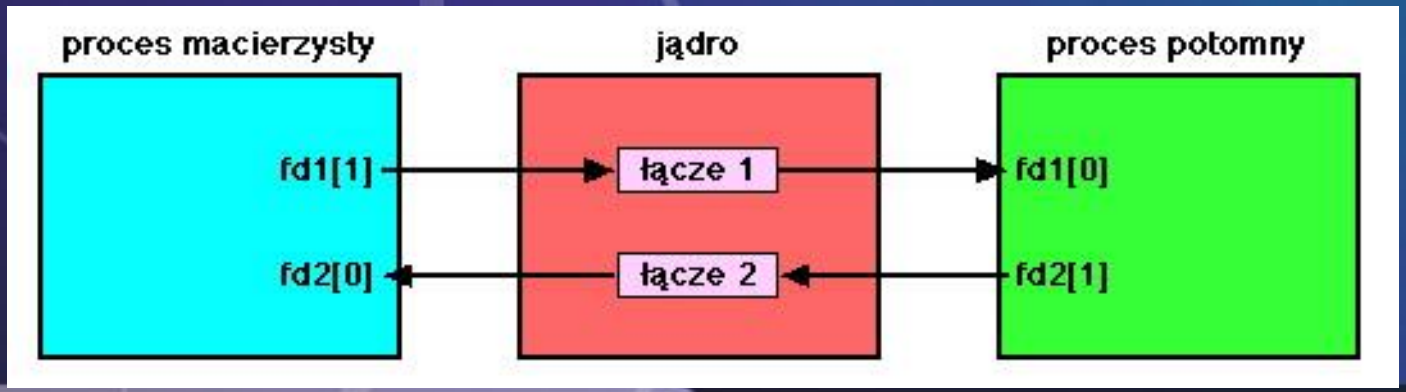
int close(int fd)

Łącza komunikacyjne nienazwane

- Jeden z procesów zamyka łącze do czytania a drugi do pisania.
- Uzyskuje się wtedy jednokierunkowe połączenie między dwoma procesami.



dwukierunkowa komunikacja między procesami



Kolejki FIFO

- Kolejki FIFO są podobne do łączy, zapewniają jednokierunkowy przepływ danych - łączą w sobie cechy pliku i łącza (plik typu p).
- Posiadają swoje nazwy, co umożliwia dostęp procesom niepowiązanym ze sobą.
- Mogą być otwarte przez każdy proces.
- Umożliwiają współpracę wielu procesów piszących i czytających (gwarantują niepodzielność).
- Są powolne.

Kolejki FIFO

mkfifo(const char *pathname, mode_t mode)

- Funkcja tworzy (ale nie otwiera) plik typu kolejka FIFO w katalogu i pod nazwą zawartą w parametrze pathname, z prawami dostępu przekazanymi w parametrze mode.

open(const char *pathname, int flags)

- Funkcja otwiera kolejkę FIFO o nazwie wskazanej przez pathname, podobnie jak otwierany jest plik.
- Funkcja zwraca deskryptor kolejki.
- Kolejka musi zostać otwarta w trybie komplementarnym,
- Kolejka może być otwierana tylko w dwóch trybach:
  - do odczytu (flags = O_RDONLY)
  - do zapisu (flags = O_WRONLY)

Kolejki FIFO

read(int fd, void *buf, size_t count)

- Dane z łącza są odczytywane tak jak z pliku.
- Dane będą odczytywane w kolejności, w której zostały zapisane, a po odczytaniu zostaną usunięte z łącza.

write(int fd, const void *buf, size_t count)

- Dane zapisywane są tak, jak w przypadku zapisu do pliku z wyjątkiem sytuacji, gdy brak jest wystarczającej ilości wolnego miejsca wtedy proces jest blokowany.
- Funkcja zapisze w potoku count bajtów w całości i będą one stanowi ciągły strumień danych, tzn. nie będą się przeplatać z danymi pochodzącymi z innych zapisów (innych wywołań funkcji write).

Kolejki FIFO

close(int fd)

Funkcja działa analogicznie, jak w przypadku zamykania deskryptora zwykłego pliku

- Nazwa FIFO pochodzi od angielskiego określenia first in, first out, czyli pierwszy na wejściu, pierwszy na wyjściu.
- Kolejki FIFO w UNIXIE są podobne do łączy, zapewniają jednokierunkowy przepływ danych.
- W przeciwieństwie do łączy kolejki FIFO są skojarzone z nazwą ścieżki co umożliwia dostęp do nich niespokrewnionym procesom.
-

Przesyłanie komunikatów

- Komunikat to niewielka liczba danych, którą można przesłać do kolejki komunikatów.
- Procesy, które mają uprawnienia mogą pobierać komunikaty z tej kolejki.
- W skład narzędzi komunikacji międzyprocesowej wchodzą dwie operacje:
  - **nadaj** komunikat
  - **odbierz komunikat**

Przesyłanie komunikatów

- Komunikaty mogą mieć stałą lub zmienną długość.
- Jeśli dwa procesy P i Q chcą się ze sobą komunikować wtedy:
  - muszą wzajemnie nadawać i odbierać komunikaty
  - musi między nimi istnieć *łącze komunikacyjne*

Metody implementacji łącza:

- Komunikacja bezpośrednia lub pośrednia.
- Komunikacja symetryczna lub asymetryczna.
- Buforowanie automatyczne lub jawne.

Komunikacja bezpośrednia

Łącze komunikacyjne w tym schemacie ma następujące własności:

- Łącze jest ustawiane automatycznie między parą procesów, które mają się komunikować: do wzajemnego komunikowania się wystarczy, aby procesy znały swoje identyfikatory.
- Dotyczy dokładnie dwu procesów.
- Między każdą parą procesów istnieje dokładnie jedno łącze
- Łącze może być jednokierunkowe, choć zazwyczaj jest dwukierunkowe.

Komunikacja bezpośrednia

Tylko nadawca nazywa odbiorcę, a od odbiorcy nie wymaga się znajomości nazwy nadawcy

Operację nadaj i odbierz będą postaci:

Nadaj(P, komunikat) – nadaj komunikat do procesu P

Odbierz(id, komunikat) – odbierz komunikat od dowolnego procesu .

Komunikacja bezpośrednia

PRODUCENT

```
begin
repeat
...
wytwarzaj jednostkę x
...
nadaj(konsument, x)
end
```

KONSUMENT

```
begin
repeat
...
odbierz(producent, y)
...
konsumuj jednostkę z y
end
```

nadaj(P, komunikat) – nadaj komunikat do procesu P

odbierz(id, komunikat) – odbierz komunikat od dowolnego procesu

Komunikacja pośrednia

nadaj(A, komunikat) – nadaj komunikat do skrzynki A
odbierz(A, komunikat) – odbierz komunikat ze skrzynki A

Łącze komunikacyjne w tym schemacie ma następujące własności:

- Łącze między dwoma procesami jest ustanawiane tylko wtedy, gdy dzielą one jakąś skrzynkę pocztową.
- Łącze może być związane z więcej niż dwoma procesami.
- Każda para komunikujących się procesów może mieć kilka różnych łączy, z których każde odpowiada jakiejś skrzynce pocztowej.
- Łącze może być jednokierunkowe lub dwukierunkowe.

Kolejki komunikatów

msgget(key_t key, int msgflg)

- Funkcja tworzy kolejkę komunikatów, jeśli kolejka o kluczu key jeszcze nie istnieje i zwraca identyfikator tej kolejki.
- Parametr msgflg umożliwia przekazanie praw dostępu do kolejki oraz pewnych dodatkowych flag definiujących sposób jej tworzenia (np. IPC_CREAT), połączonych z prawami dostępu operatorem sumy bitowej (np. PC_CREAT|0660)

Kolejki komunikatów

msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg)

- Funkcja umieszcza w kolejce identyfikowanej przez msgid komunikat.
- Treść komunikatu znajduje się pod adresem msgp w przestrzeni adresowej procesu.
- Komunikat zawiera msgsz bajtów.

Ogólna struktura komunikatu zdefiniowana jest następująco:

```
struct msgbuf {
long mtype; /* typ komunikatu, > 0 */
char mtext[1]; /* treść komunikatu */
};
```

Kolejki komunikatów

msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp, int msgflg)

- Funkcja pobiera z kolejki komunikat który spełnia kryteria określone przez msgtyp (typ komunikatu), pmsgsz (rozmiar bufora w przestrzeni adresowej procesu) i msgflg (flagi specyfikujące zachowanie się funkcji w warunkach nietypowych).
- Wybór komunikatu dokonuje się według następujących zasad: jeśli parametr msgflg ma ustawioną flagę MSG_NOERROR, komunikat przekraczający rozmiar bufora jest ucinany przy odbiorze, w przeciwnym razie odbierane są tylko komunikaty, których treść ma mniejszy rozmiar niż msgsz.

Kolejki komunikatów – system Windows

- Komunikaty w systemie Windows są udostępniane za pomocą tzw. udogodnienia wywoływania procedur lokalnych LPC
- Do nawiązania i utrzymywania połączenia między dwoma procesami używa się obiektu portu
- Każdemu klientowi, odwołującemu się do podsystemu, jest potrzebny kanał komunikacyjny, który jest udostępniany przez obiekt portu i nigdy nie podlega dziedziczeniu

W systemie NT stosuje się dwa typy portów:

- porty łączące
- porty komunikacyjne

Kolejki komunikatów – system Windows

- Klient zaopatruje się w uchwyt do obiektu portu łączącego podsystemu.
- Klient wysyła prośbę o połączenie.
- Serwer tworzy dwa prywatne porty komunikacyjne i przekazuje klientowi uchwyt do jednego z nich.
- Klient i serwer korzystają z odpowiednich uchwytów portowych w celu wysyłania komunikatów lub przywołań oraz nasłuchiwania odpowiedzi.
- Komunikaty nie dłuższe niż 256 bajtów - przekazywanie za pomocą portu określonego przez klienta podczas ustanawiania kanału.
- Komunikaty dłuższe niż 256 bajtów - przekazuje za pomocą obiektu sekcji (pamięci dzielonej).

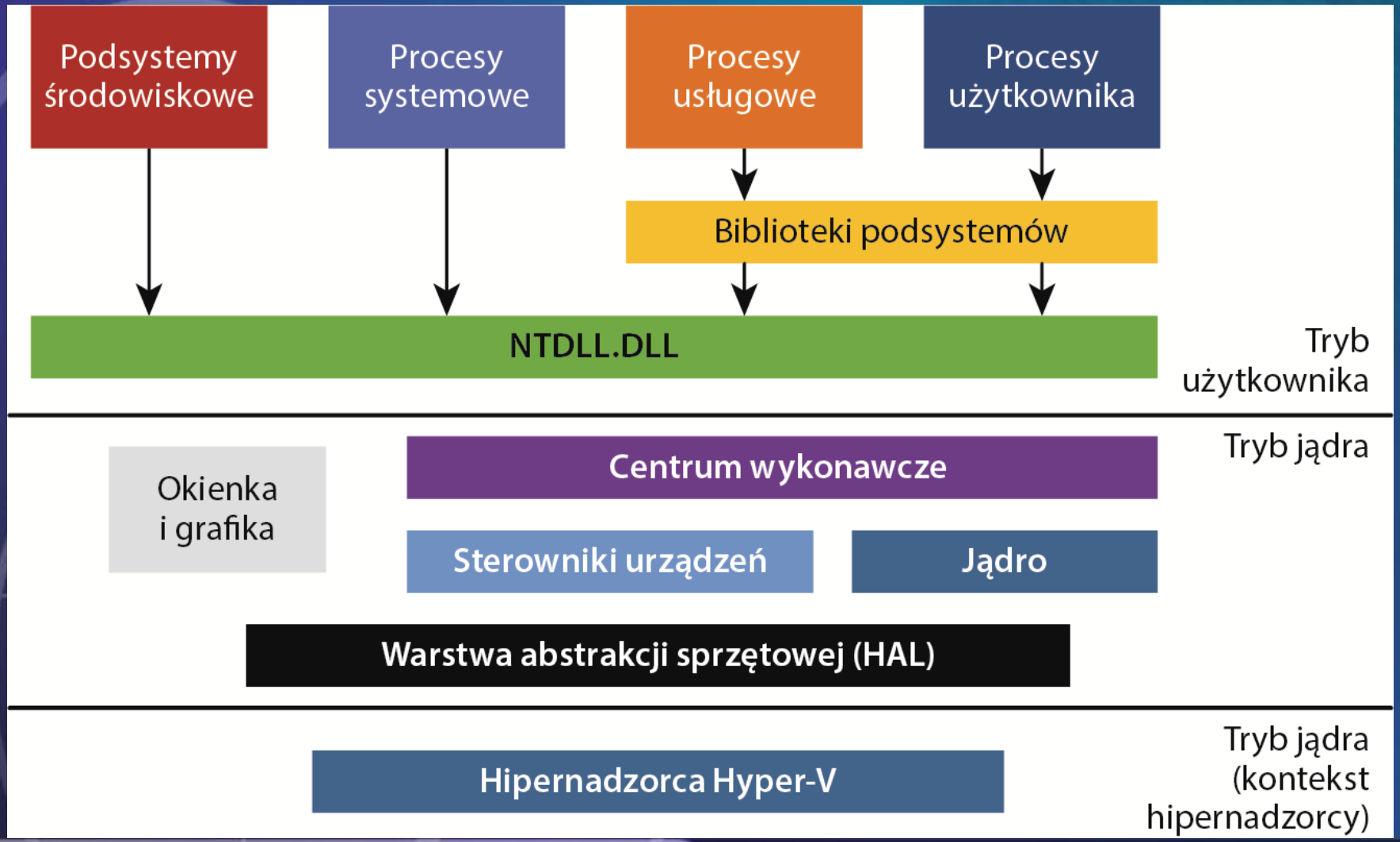
Szybkie wywołania LPC

- Klient wysyła zamówienie połączenia do portu zaznaczając, że będzie używał szybkich wywołań LPC.
- Serwer deleguje do obsługi zamówień:
  - osobny wątek
  - obiekt sekcji wielkości 64 KB
  - obiekt pary zdarzeń (obiekt synchronizacji używany do powiadamiania że wątek klienta przekopiował komunikat do serwer lub na odwrót)
- Jądro daje także wydzielonemu wątkowi priorytet przy planowaniu przydziału procesora.

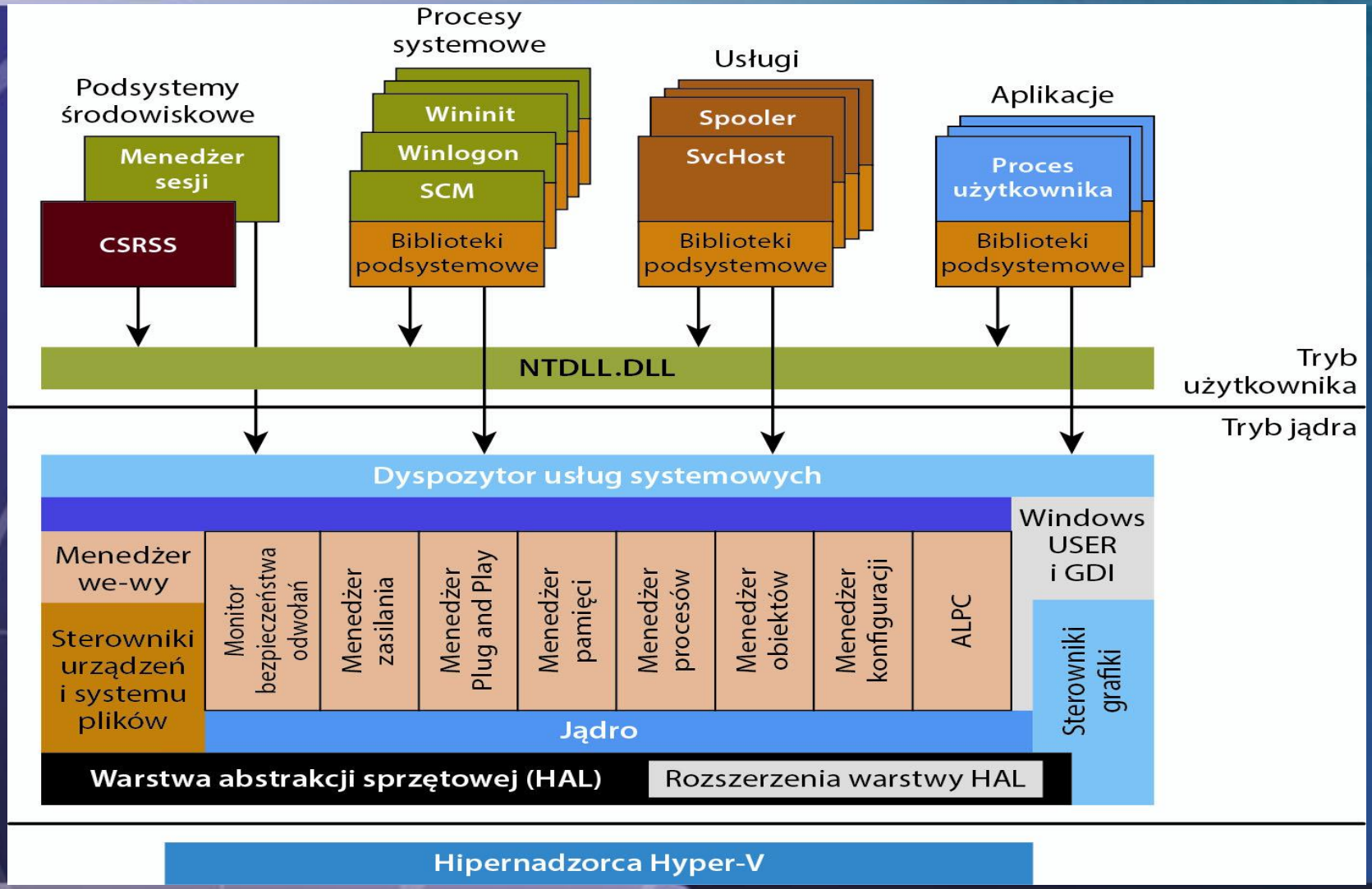
Szybkie wywołania LPC

- Klient wysyła zamówienie połączenia do portu zaznaczając, że będzie używał szybkich wywołań LPC.
- Serwer deleguje do obsługi zamówień:
  - osobny wątek
  - obiekt sekcji wielkości 64 KB
  - obiekt pary zdarzeń (obiekt synchronizacji używany do powiadamiania że wątek klienta przekopiował komunikat do serwer lub na odwrót)
- Jądro daje także wydzielonemu wątkowi priorytet przy planowaniu przydziału procesora.

Architektura systemu Windows



Architektura systemu Windows



Kolejki komunikatów – system Windows

- Klient zaopatruje się w uchwyt do obiektu portu łączącego podsystemu.
- Klient wysyła prośbę o połączenie.
- Serwer tworzy dwa prywatne porty komunikacyjne i przekazuje klientowi uchwyt do jednego z nich.
- Klient i serwer korzystają z odpowiednich uchwytów portowych w celu wysyłania komunikatów lub przywołań oraz nasłuchiwania odpowiedzi.
- Komunikaty nie dłuższe niż 256 bajtów - przekazywanie za pomocą portu określonego przez klienta podczas ustanawiania kanału.
- Komunikaty dłuższe niż 256 bajtów - przekazuje za pomocą obiektu sekcji (pamięci dzielonej).

Zakleszczenia

Zbiór procesów będących w impasie wywołanym przez to, że każdy proces należący do tego zbioru przetrzymuje zasoby potrzebne innym procesom z tego zbioru, a jednocześnie czeka na zasoby przydzielone innym procesom.

- Zakleszczenie jest zjawiskiem niezwykle niepożądanym w systemie i dlatego opracowano rozmaite metody radzenia sobie z nimi.

Model systemu

- Zakłada się, że w systemie jest *m* rodzajów zasobów oznaczanych *Z*(1), *Z*(2), ..., *Z*(*m*). Dla każdego *i*=1, 2, ..., *m*, *E*(*i*) oznacza liczbę egzemplarzy zasobu *Z*(*i*)

Procesy korzystający z zasobu wykonuje następujące działania:

- żąda przydzielenia zasobu
- korzysta z zasobu
- zwalnia zasób

Warunki konieczne

- Warunki konieczne – nie gwarantują zakleszczenia, ale jeśli choć jeden z nich nie jest spełniony, do zakleszczenia nie dojdzie:
  - **Wzajemne wykluczanie**
  - **Przetrzymywanie i oczekiwanie**
  - **Brak wywłaszczania**
  - **Czekanie cykliczne**

Istnieje zbiór czekających procesów {*P*(1), *P*(2), ..., *P*(*n*)}, takich, że:
proces *P*(1) czeka na zasób przydzielony procesowi *P*(2),

proces *P*(2) czeka na zasób przydzielony procesowi *P*(3),

...,

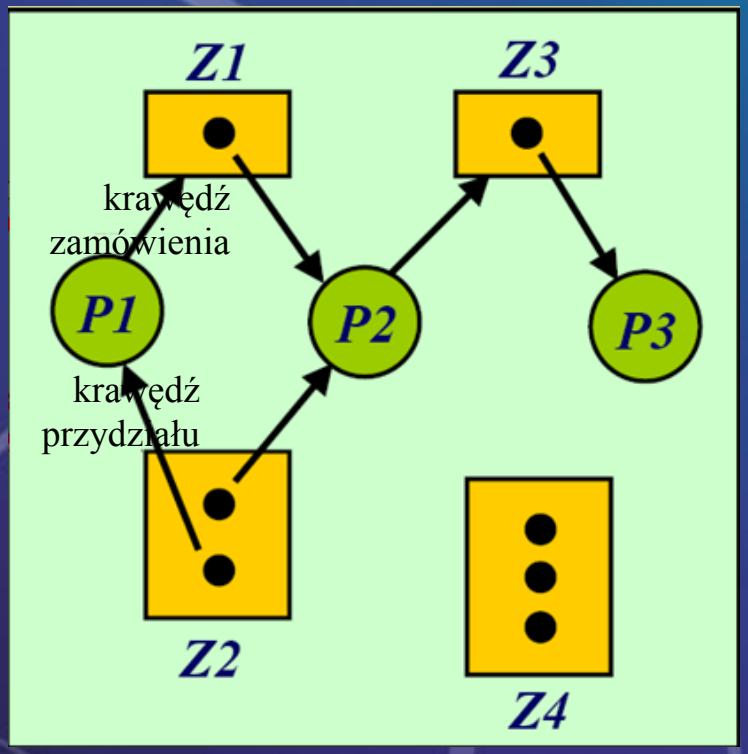
proces *P*(*i*-1) czeka na zasób przydzielony procesowi *P*(*i*),

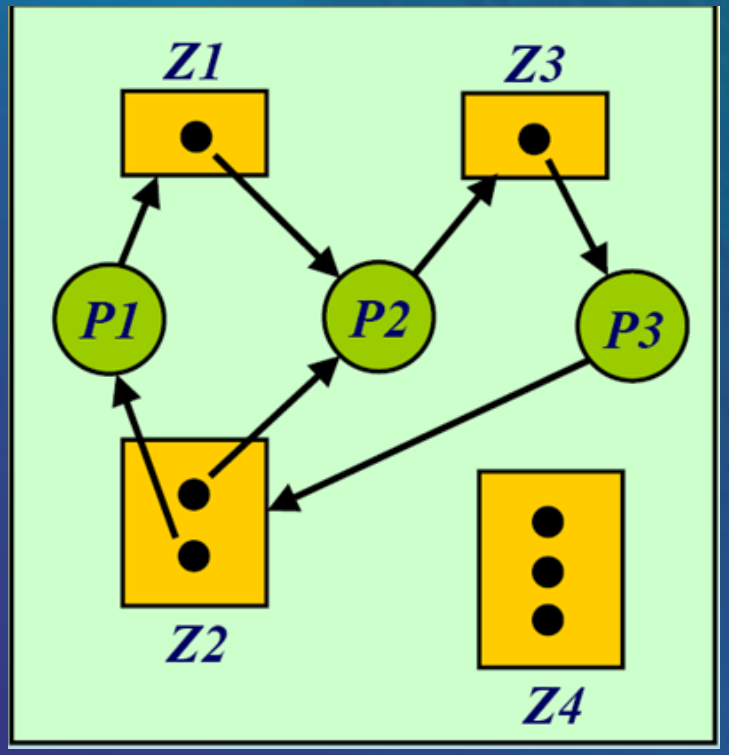
...,

proces *P*(*n*-1) czeka na zasób przydzielony procesowi *P*(*n*),

proces *P*(*n*) czeka na zasób przydzielony procesowi *P*(1).

Grafy przydziału zasobów





graf przydziału bez zakleszczeń

graf przydziału z zakleszczeniem

Jeśli graf nie zawiera cyklu nie ma zakleszczenia

Zapobieganie zakleszczeniom

Wyróżniamy w zasadzie trzy metody radzenia sobie z zakleszczeniami:

- Zapobieganie zakleszczeniom
- Unikanie zakleszczeń
- Wykrywanie zakleszczeń i odtwarzanie

Zapobieganie zakleszczeniom

- Zapobieganie zakleszczeniom polega na zaprzeczeniu co najmniej jednemu z czterech warunków koniecznych zakleszczenia
  - brak wzajemnego wykluczania
  - brak przetrzymywania i oczekiwania
  - wywłaszczanie
  - wykluczenie czekania cyklicznego

Unikanie zakleszczeń

- Wszystkie warunki konieczne muszą być prawdziwe.
- Nie dopuszczamy do zakleszczeń poprzez badanie stanu systemu przed każdym żądaniem przydziału zasobów.
- W algorytmie unikania dynamicznie bada się stan przed każdym żądaniem i sprawdza się, czy jego spełnienie może doprowadzić do czekania cyklicznego.

Stan bezpieczny systemu

Stan przydziału zasobów jest określony przez:

- liczbę dostępnych i przydzielonych zasobów,
- maksymalne zapotrzebowanie procesów.

Stan jest bezpieczny - jeśli istnieje porządek, w którym system operacyjny może przydzielić zasób każdemu procesowi stale unikając blokady

Ciąg procesów $P(1), P(2), ..., P(n)$ nazywamy bezpiecznym, gdy dla i=1,2,...,n max potrzeby procesu P(i) mogą być zaspokojone przy wykorzystaniu dostępnych zasobów oraz zasobów będących w posiadaniu procesów P(1), P(2),...,P(i-1)

Jeśli żaden taki ciąg nie istnieje, stan systemu określa się jako zagrożony

Algorytm bankiera

- Algorytm bankiera służy do sprawdzenia, czy stan jest bezpieczny.
- Polega on na skonstruowaniu ciągu bezpiecznego.

Założenie:
mamy *n* procesów *P(1), P(2), ..., P(n)*
oraz
m zasobów *Z(1), Z(2), ..., Z(m)*

W implementacji algorytmu bankiera występuje kilka struktur danych, struktury te przechowują stan przydziału zasobów.

Algorytm bankiera

***Dostępne*[1..*m*]**	liczba dostępnych egzemplarzy zasobu *Z*(*i*) ***Dostępne[j]=k*** – oznacza, ze jest dostępnych k egzemplarzy zasobu Z(j)
***Max*[1..*n*, 1..m]**	maksymalna liczba egzemplarzy zasobu *Z*(*i*), których może używać proces *P*(*j*) (wg jego deklaracji) ***Max[i,j]=k*** – proces P(i) może zamówić co najwyżej k egzemplarzy zasobu Z(j))
***Przydzielone*[1..*n*, 1..*m*]**	liczba egzemplarzy zasobu *Z*(*i*), przydzielonych do każdego procesu ***Przydzielone[i,j]=k*** – proces P(i) ma przydzielonych k egzemplarzy zasobu Z(j)
***Potrzebne*[1..*n*, 1..*m*]**	przechowuje pozostałe do spełnienia zamówienia każdego z procesów

$$Potrzebne[i,j] = Max[i,j] - Przydzielone[i,j]$$

autor: dr inż. Stanisława Plichta

Algorytm bankiera

Stan systemu jest zadany przez tablice:

- *Potrzebne*
- *Przydzielone*
- *Dostępne*

Algorytm wykorzystuje ponadto tablice:

***Koniec*[1..*n*]**	wektor o wartościach *true*, gdy proces P(j) może być zakończony w dotychczasowej konfiguracji przydziałów
***Robocze*[1..*m*]**	wektor o wartościach odpowiadających liczbie zasobów Z(i) dostępnych po zakończeniu wszystkich procesów, dla których Koniec[i] = *true*

Algorytm bankiera

1. Zainicjuj tablicę *Koniec* wartościami *false*, a tablicę *Robocze* zawartością tablicy *Dostepne*
2. Znajdź takie *j*, że:

   Koniec[*j*] = *false*

   Potrzebne[j] ≤ *Robocze*

   j – nr kolejnego procesu dodawanego na końcu już skonstruowanego ciągu bezpiecznego
3. Jeśli nie ma takiego *j*, idź do kroku 6
4. Jeśli jest takie *j*:

   Robocze = *Robocze* + *Przydzielone*

   Koniec[*j*] := *true*

   po dodaniu procesu P(j) do konstruowanego ciągu bezpiecznego zakłada się, że ten proces kończy się i zwalnia wszystkie swoje zasoby
5. Wróć do kroku 2
6. Jeśli dla każdego *j* = 1, 2, ..., *n*, *Koniec* [*j*] = *true*, to stan jest bezpieczny, w przeciwnym wypadku, stan nie jest bezpieczny

Algorytm bankiera – obsługa żądania

proces P(j) zażądał zasobów Z(i) opisanych za pomocą tablicy Żądanie[j, i]

1. Jeśli Żądanie > Potrzebne, to znaczy, że proces żąda więcej zasobów niż zadeklarowane na początku maksimum → zlecenie jest odrzucane
2. Jeśli Żądanie > Dostępne, to znaczy, że proces żąda więcej zasobów niż jest ich dostępnych → zlecenie jest odrzucane
3. Jeśli żaden z wyżej wymienionych warunków nie jest spełniony, konstruujemy stan, który powstałby po spełnieniu tego żądania i sprawdzamy, czy jest on bezpieczny

Stan bezpieczny systemu

Stan przydziału zasobów jest określony przez:

- liczbę dostępnych i przydzielonych zasobów,
- maksymalne zapotrzebowanie procesów.

Stan jest bezpieczny - jeśli istnieje porządek, w którym system operacyjny może przydzielić zasób każdemu procesowi stale unikając blokady

Ciąg procesów $P(1), P(2), ..., P(n)$ nazywamy bezpiecznym, gdy dla i=1,2,...,n max potrzeby procesu P(i) mogą być zaspokojone przy wykorzystaniu dostępnych zasobów oraz zasobów będących w posiadaniu procesów P(1), P(2),...,P(i-1)

Jeśli żaden taki ciąg nie istnieje, stan systemu określa się jako zagrożony

Algorytm bankiera – obsługa żądania

Nowy stan otrzymamy wykonując następujące zmiany w tablicach:

Dostępne = Dostępne – Żądanie

Przydzielone = Przydzielone + Żądanie

Potrzebne = Potrzebne - Żądanie

- Jeśli taki stan jest bezpieczny, żądanie jest spełniane.
- Jeśli stan nie jest bezpieczny, proces P(j) musi czekać (mimo, że zasoby są wolne – jest to koszt unikania zakleszczeń)

Algorytm bankiera – obsługa żądania

Rozważmy sytuację, w której są trzy procesy *P*(1), *P*(2) i *P*(3) oraz dwa rodzaje zasobów *Z*(1) i *Z*(2), które mają po 3 egzemplarze zasobu. Stan systemu opisany tablicami: *Przydzielone*, *Max* i *Dostępne*

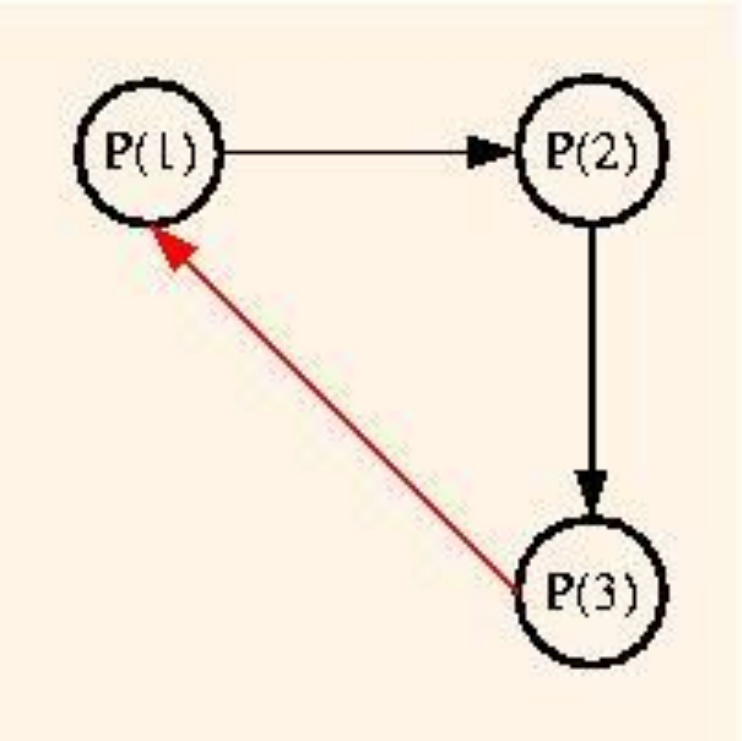
	Przydzielone		**Max**		**Dostępne**	
Proces	**Z(1)**	***Z*(2)**	***Z*(1)**	***Z*(2)**	***Z*(1)**	***Z*(2)**
P(1)	**0**	**0**	**2**	**1**		
P(2)	**2**	**1**	**2**	**3**	**1**	**1**
P(3)	**0**	**1**	**1**	**1**		

System jest w stanie bezpiecznym -> ciąg P(3),P(2), P(1): możliwe spełnienie maksymalnych potrzeb P(3); po jego zakończeniu – zwolnienie zasobu Z(2) -> spełnienie max. potrzeb P(2); z kolei możliwe spełnienie potrzeb P(1)

Wykrywanie zakleszczeń

- Jeżeli w systemie nie stosuje się algorytmów zapobiegania zakleszczeniom ani ich unikania, to może dojść do zakleszczenia. Wówczas w systemie muszą istnieć:
  - algorytm sprawdzający stan systemu, czy nastąpiło zakleszczenie
  - algorytm usuwania zakleszczenia
- algorytm wykrywania zakleszczeń przy każdym zamówieniu może prowadzić do nadmiernego czasu obliczeń
- wykonuje się ten algorytm np. raz na godzinę lub gdy wykorzystanie procesora spadnie np. poniżej 40 %

Wykrywanie zakleszczeń



- Graf oczekiwania jest **skierowany**
- **Wierzchołkami** tego grafu są procesy działające w systemie

  $P(1), P(2), ..., P(n)$
- **Krawędź oczekiwania** od procesu $P(k)$ do procesu $P(l)$ oznacza, że proces $P(k)$ czeka na zwolnienie zasobu przez proces $P(l)$.
- Istnienie cyklu w tym grafie oznacza zakleszczenie.

Wykrywanie zakleszczeń

- Wykrywanie zakleszczeń można przeprowadzić też metodą podobną do algorytmu bankiera. Wykorzystuje się w nim tablice **Dostępne, Przydzielone, Koniec i Robocze**, oraz tablicę ***Żądane[1..n, 1..m]***

Żądane[i, j] to liczba egzemplarzy zasobu Z(j), na których przydział oczekuje proces *P*(i)

- Algorytm polega na sprawdzeniu, czy istnieje taki ciąg procesów *P*(1), *P*(2), ..., *P*(*n*), że dla *i*=1, 2, ..., *n* żądanie procesu *P*(*i*) może być zaspokojone za pomocą obecnie dostępnych zasobów i zasobów będących w posiadaniu procesów *P*(1), *P*(2), ..., *P*(*i*-1)

Wykrywanie zakleszczeń

1. Zainicjuj tablicę ***Koniec*** wartościami *false* dla procesów, które coś mają przydzielone (**Przydzielone[*j*]** ≠ 0) i wartościami *true* dla procesów, które nic nie mają przydzielone (***Przydzielone*[*j*] = 0**). Tablicę **Robocze** zainicjuj zawartością tablicy Dostępne.
2. Znajdź takie *j*, że:

   Żądanie ≤ ***Robocze***

   Koniec[*j*] = *false*
3. Jeśli nie ma takiego *j*, idź do kroku 6
4. Jeśli jest takie *j*:

   Robocze:= ***Robocze*** + ***Przydzielone***

   Koniec[j] := *true*
5. Wróć do kroku 2
6. Jeśli dla każdego *j* = 1, 2, ..., *n*, ***Koniec*[*j*]** = *true*, to w systemie nie ma zakleszczenia, w przeciwnym razie jest zakleszczenie.

Usuwanie zakleszczeń

- **Usuwanie przez wywłaszczenie** - czasowe odebranie zasobów jednemu z procesów i przydzielenie ich innemu procesowi (bardziej potrzebującemu).
- **Usuwanie przez wycofywanie** - punkty kontrolne procesu – moment wpisu obrazu pamięci procesu oraz stanu przydzielonych zasobów do tego procesu (zapisywane do *dziennika systemu*). Informacje te mogą być nadpisywane w kolejnych punktach kontrolnych, jeśli system pracuje normalnie (nie zakleszcza się).
- Jeśli zostanie wykryte zakleszczenie, należy sprawdzić jakie zasoby są zablokowane. Wybrany na tej podstawie proces jest wycofany do stanu pamiętanego w ostatnim punkcie kontrolnym. Wycofywanie jest kontynuowane aż do momentu, kiedy zakleszczenie zostanie usunięte.

Usuwanie zakleszczeń

Usuwanie zakleszczenia przez usunięcie procesu

Kryteria, na podstawie których mogą być wybierane decyzje o wybraniu procesu do usunięcia:

- Priorytety procesów - usuwane są procesy o najniższym priorytecie.
- Czasy dotychczasowego działania procesów - usuwane są procesy działające najkrócej.
- Liczba i typ wykorzystywanych zasobów - usuwane są te procesy, które wykorzystywały najwięcej zasobów.
- Liczba i typ zasobów potrzebnych do zakończenia działania - usuwane są te procesy, które potrzebują najwięcej zasobów.