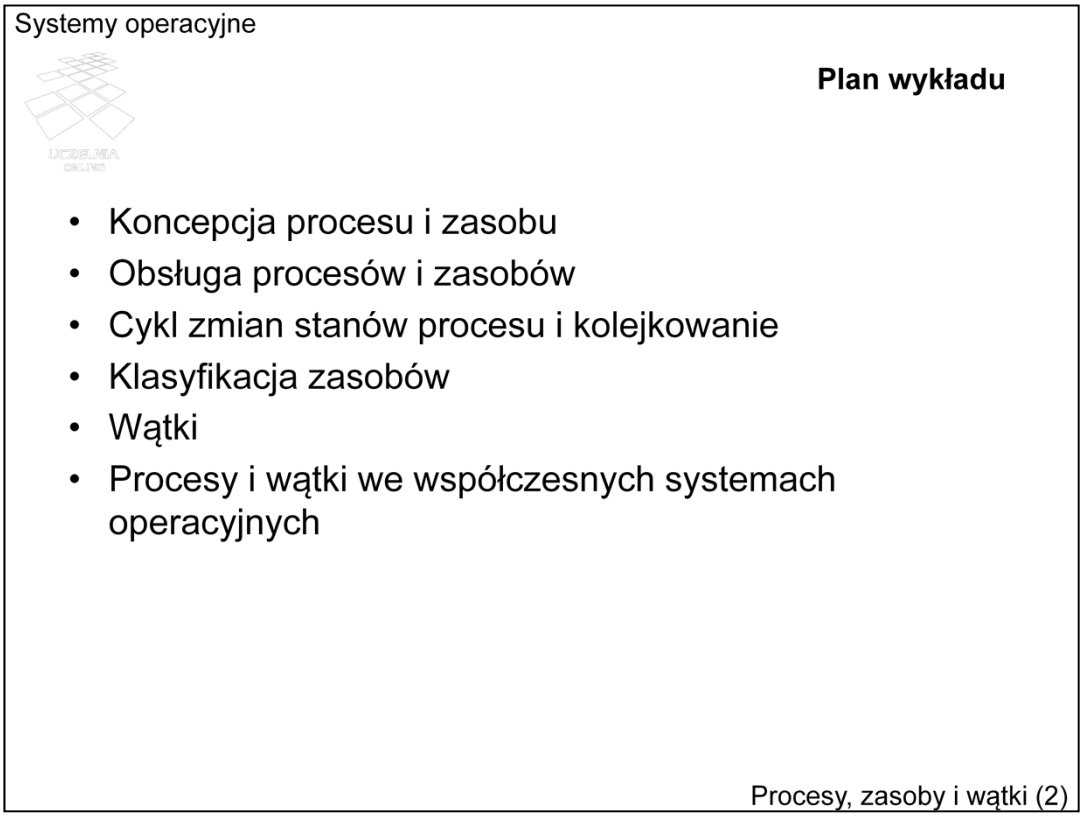
Systemy operacyjne

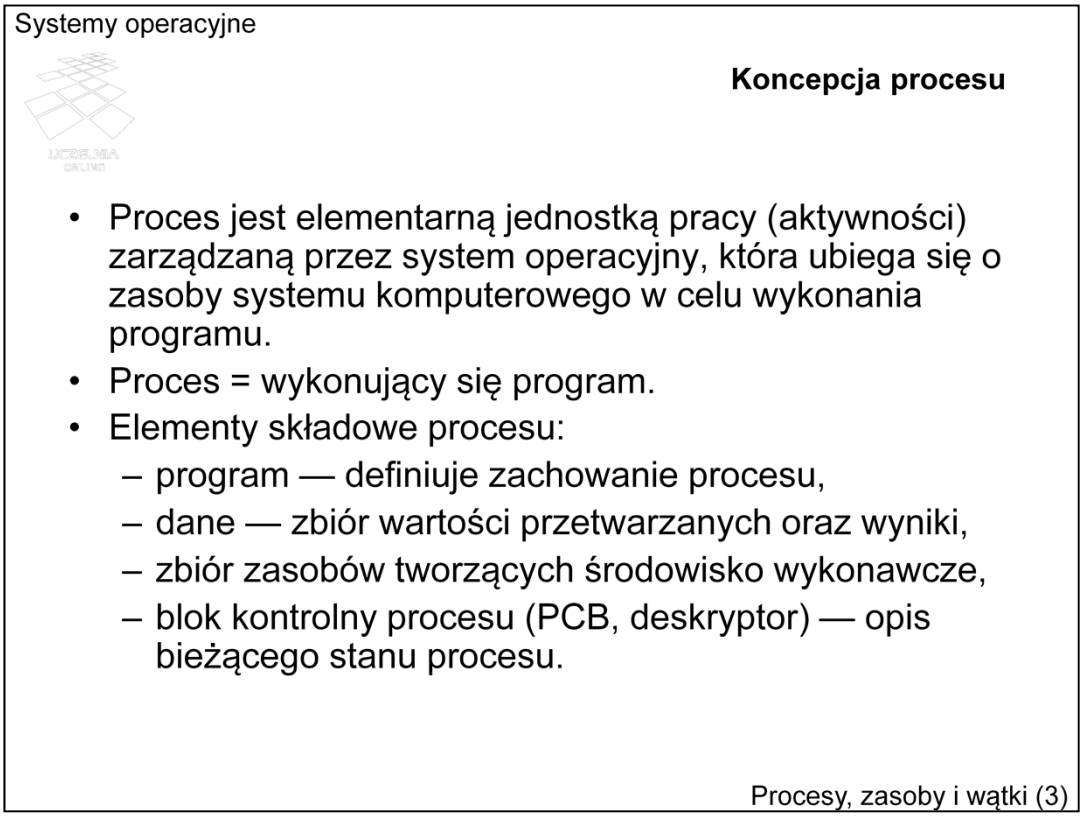
Procesy, zasoby i wątki

Wykład prowadzą:
Jerzy Brzeziński
Dariusz Wawrzyniak

Celem wykładu jest wprowadzenie fundamentalnych pojęć, integralnie związanych z systemem operacyjnym, na których opiera się przetwarzanie we współczesnych systemach komputerowych — pojęcia procesu i pojęcia zasobu. Oba pojęcia zostały zasygnalizowane we wprowadzeniu. Tutaj zostaną przedyskutowane związki pomiędzy procesami i zasobami.



Treść wykładu obejmuje przedstawienie koncepcji procesu i zasobu, ich obsługę, czyli podstawowe struktury danych oraz ogólną koncepcję zarządzania. Z relacji między procesami a zasobami wynikają stany procesu i związane ze stanem kolejkowanie procesów. Następnie przedstawiona zostanie klasyfikacja zasobów. Kolejnym istotnym zagadnieniem jest koncepcja wątku, wyodrębniona ze względu na specyfikę wykorzystywania niektórych zasobów systemu. Na koniec omówiony zostanie sposób realizacji koncepcji procesów i wątków we współczesnych systemach operacyjnych, czyli w systemie Linux oraz Windows 2000/XP.

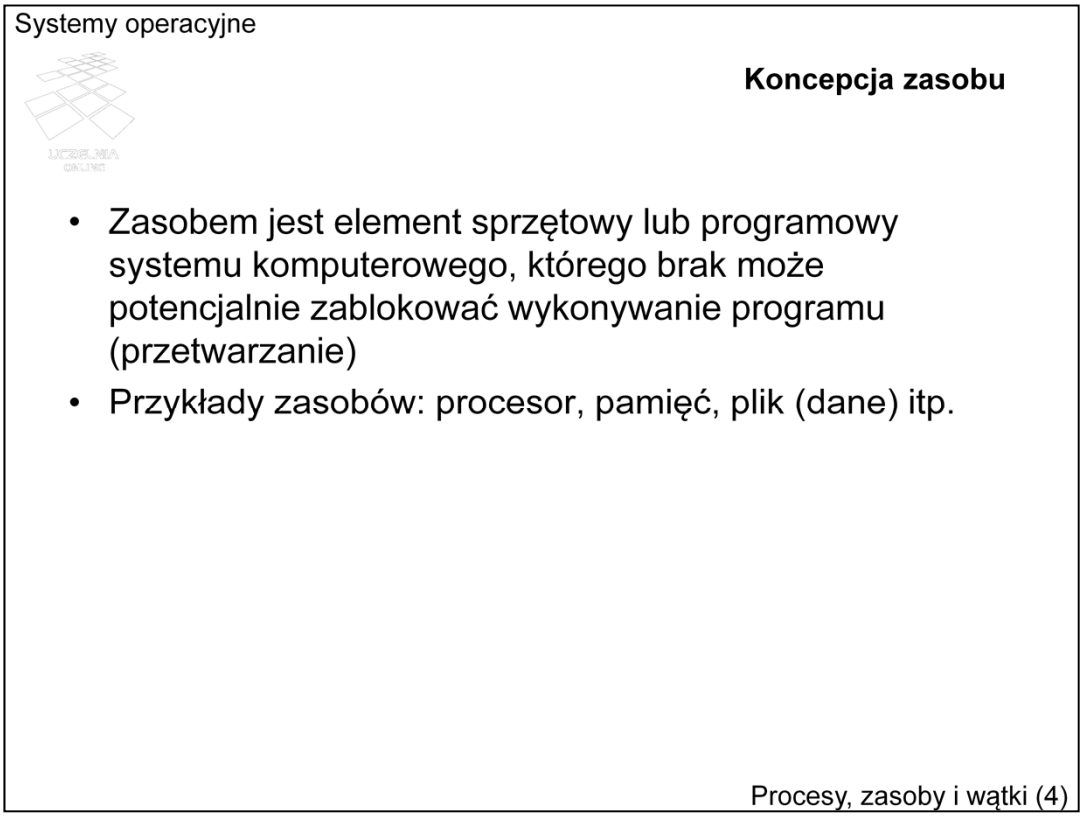


Koncepcja procesu jest jednym z najważniejszych pojęć we współczesnych systemach operacyjnych.

Proces służy do organizowania wykonywania programu w ten sposób, że stanowi on powiązanie niezbędnych zasobów systemu komputerowego i umożliwia kontrolę stanu tych zasobów, związaną z wykonywaniem programu.

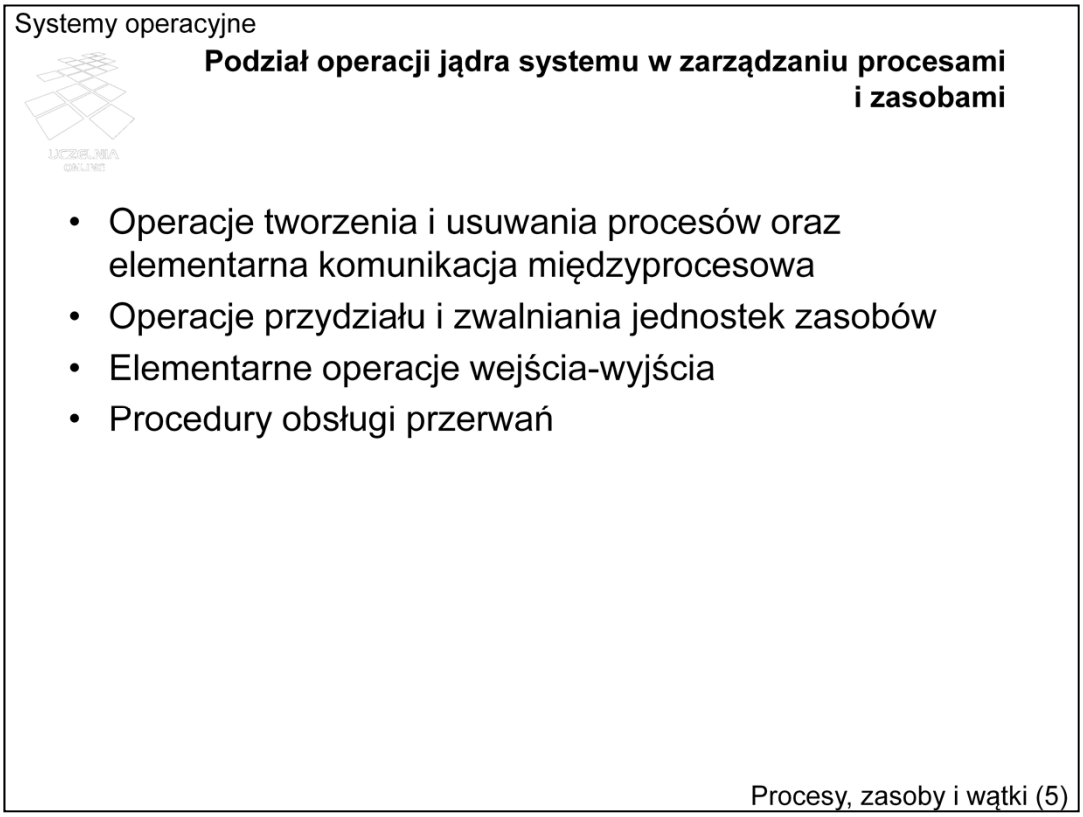
Istotne jest rozróżnienie pomiędzy procesem a programem. Program jest zbiorem instrukcji. W tym sensie jest tylko elementem procesu, znajdującym się w jego segmencie kodu (zwanym też segmentem tekstu). Poza tym do wykonania programu potrzebne są dodatkowe zasoby (procesor, pamięć itp.) Program najczęściej nie zmienia się w czasie wykonywania (nie ulega modyfikacji), podczas gdy stan procesu ulega zmianie: zmienia się **stan wykonywania** programu podobnie jak stan większości zasobów z tym związanych. Zmianie w wyniku wykonywania procesu ulega np. segment danych, segment stosu, stan rejestrów procesora itp. Procesem jest więc cały ten kontekst niezbędny do wykonania programu.

Wyodrębnienie procesu wiąże się z współbieżnością przetwarzania. W systemie może istnieć wiele procesów (wiele niezależnych przetwarzań), stąd ważne jest utrzymanie informacji o tym, które zasoby przedzielone na potrzeby każdego przetwarzania. W systemach jednozadaniowych (np. MS DOS) pojęcie procesu nie było wyodrębnione, gdyż nie było takiej potrzeby.



Zasobem jest każdy element systemu, który może okazać się niezbędny dla realizacji przetwarzania. Typowe zasoby kojarzone są z elementami sprzętowymi systemu komputerowego. Należy jednak podkreślić, że to dopiero system operacyjny definiuje taki element jako zasób, gdyż w jądrze systemu istnieją struktury do zarządzania i procedury realizacji przydziału, odzyskiwania itd.

Poza tym część zasobów tworzona jest przez jądro systemu operacyjnego. Zasoby takie często określa się jako *wirtualne*. Przykładem wirtualnego urządzenia wejścia-wyjścia jest plik. Pliki udostępnia system operacyjny. Na poziomie maszynowym pojęcie takie nie istnieje — można co najwyżej mówić o sektorach dysku, w których składowana są dane.

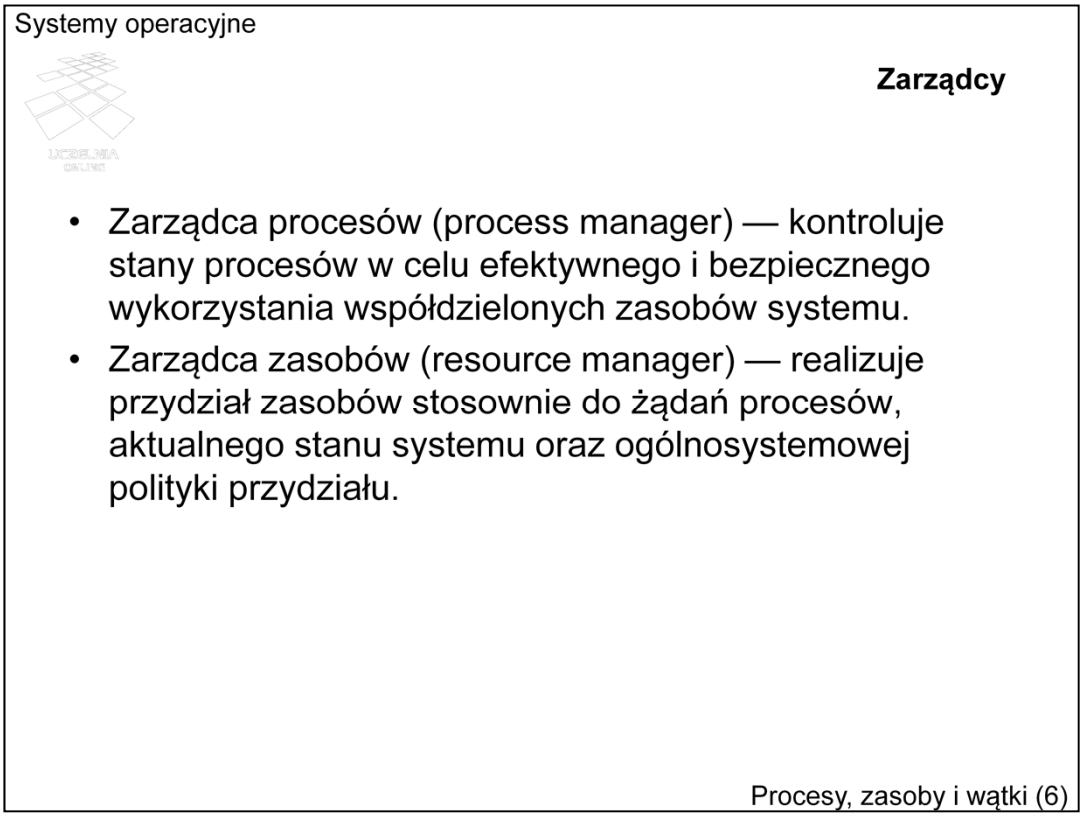


Operacje tworzenia i usuwania procesów bezpośrednio dotyczą procesów, ale pośrednio również często zasobów, gdyż utworzenie procesu wymaga przydziału pewnych zasobów. Proces tworzony jest przez inny proces i początkowo może współdzielić z nim większość zasobów.

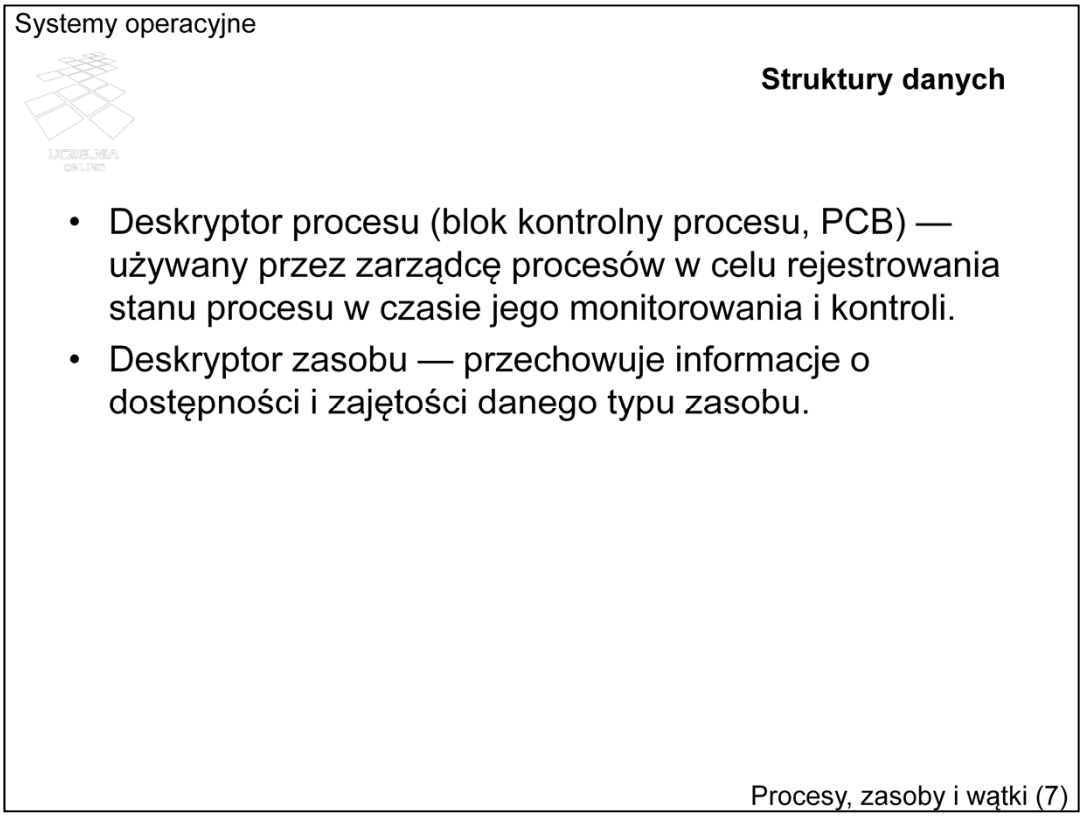
Operacje przydziału i zwalniania jednostek zasobów dotyczą tworzenia powiązań między procesami i zasobami.

Elementarne operacje wejścia-wyjścia dotyczą również zasobów, ale nie są związane z ich przydziałem, czy zwalnianiem. Są to operacje dostępu do przydzielonych zasobów. Nie wszystkie operacje dostępu do przydzielonych zasobów wymagają jednak wsparcia ze strony jądra. Operacje dostępu do pamięci na przykład realizowane są na poziomie maszynowym, jądro natomiast angażowane jest dopiero w przypadku wykrycia jakichś nieprawidłowości.

Procedury obsługi przerwań z kolei są reakcją na zdarzenia zewnętrzne lub pewne szczególne stany wewnętrzne, mogą być zatem skutkiem ubocznym realizacji dostępu do zasobów. Reakcja na przerwanie może prowadzić do zmiany stanu procesu lub zasobu, nie zawsze jednak taka zmiana jest bezpośrednio spowodowana wykonaniem procedury obsługi. Procedury obsługi przerwań wykonywane muszą być szybko, dlatego ich bezpośrednim skutkiem jest czasami tylko odnotowanie faktu zajścia zdarzenia, natomiast właściwa reakcja systemu, w konsekwencji której nastąpi zmiana stanu procesu lub zasobu, wykonywana jest później.



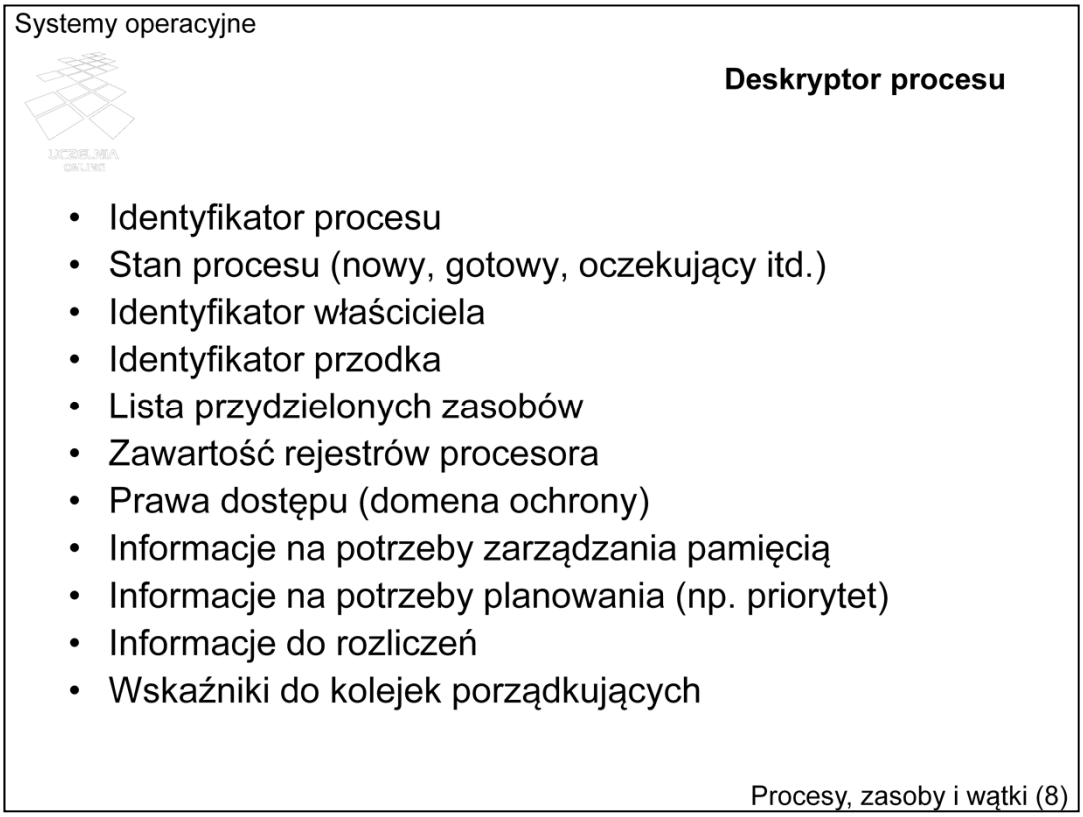
Ze względów pojęciowych lub projektowych fragmenty kodu jądra związane z obsługą procesów i zasobów wyodrębnia się postaci zarządców. Można więc mówić o zarządcy procesów, który grupuje i wykonuje funkcje obsługi procesów oraz zarządcy zasobów, którego zadaniem jest kontrola stanu zajętości zasobów i realizacja żądań procesów.



Zarządzenie wymaga odpowiednich struktur dla danych na potrzeby ewidencji stanu procesów i zasobów, ich powiązań, potrzeb zasobowych procesów itp.

Strukturę danych na potrzeby opisu stanu procesu określa się jako *deskryptor procesu* lub *blok kontrolny procesu*. Zbiór takich informacji dla wszystkich procesów określa się jako tablicę procesów. Współcześnie na potrzeby deskryptorów rzadko wykorzystywana jest rzeczywiście statyczna tablica. Podejście takie gwarantuje szybki dostęp do informacji, ale jest mało elastyczne, gdyż narzuca górny limit na liczbę procesów w systemie, a w przypadku mniejszej ich liczby oznacza marnotrawstwo pamięci.

Zasoby mogą być bardzo zróżnicowane, dlatego ogólny określenie *deskryptor zasobu* ma raczej charakter pojęciowy, a nie definicyjny. W zależności od rodzaju zasobu struktura opisu może być bardzo różna. Często jest ona narzucona przez rozwiązanie przyjęte na poziomie architektury procesora (np. w przypadku pamięci), a czasami wynika z decyzji projektowych.



Zakres niezbędnych informacji, umożliwiających odpowiednie zarządzanie procesem może być różny w zależności od celów projektowych systemu operacyjnego lub przyjętych rozwiązań implementacyjnych. Np. w przypadku systemów dla komputerów osobistych *identyfikator właściciela* lub *informacje do rozliczeń* mogą okazać się zbędne. *Prawa dostępu* mogą by różnie zaimplementowane, zależnie od przyjętej koncepcji ochrony zasobów.

Część tych informacji jest jednak niezbędna w każdym deskryptorze. *Stan procesu* potrzebny jest do podjęcia decyzji odnośnie dalszego losu procesu (np. usunięcie procesu i zwolnienie zasobów, przesunięcie procesu z pamięci fizycznej do pamięci pomocniczej lub odwrotnie itp.). *Licznik rozkazów* i *stan rejestrów* niezbędne są do odtworzenia kontekstu danego procesu. *Informacje na potrzeby planowania przydziału procesora* umożliwiają właściwe szeregowanie procesów i podejmowanie decyzji przez planistów (ang. scheduler), chociaż część tych informacji może znajdować się poza właściwym deskryptorem. *Informacje o zarządzaniu pamięcią* umożliwiają ochronę obszarów pamięci, w szczególności powstrzymanie procesu przed ingerencją w obszary poza jego przestrzenią adresową. *Informacje o stanie wejścia-wyjścia*, obejmujące dane o przydzielonych urządzeniach, wykaz otwartych plików itp., umożliwiają odpowiednie zarządzanie tymi zasobami i dostępem do nich przez system operacyjny, ich odzyskiwanie po zakończeniu procesu itp.

Systemy operacyjne

Stany procesu

- Nowy (ang. new) — proces jest tworzony.
- Wykonywany (ang. running) — wykonywane są instrukcje programu.
- Oczekujący (ang. waiting) — proces oczekuje na jakieś zdarzenie, np. na zakończenie operacji wejścia-wyjścia, na przydział dodatkowego zasobu, synchronizuje się z innymi procesami.
- Gotowy (ang. ready) — proces czeka na przydział procesora.
- Zakończony (ang. terminated) — proces zakończył działanie i zwalnia zasoby.

Procesy, zasoby i wątki (9)

W zależności od stanu wykonywania programu i dostępności zasobów można wyróżnić następujące, ogólne stany procesu:

Nowy — formowanie procesu, czyli gromadzenie zasobów niezbędnych do rozpoczęcia wykonywania procesu, z wyjątkiem procesora (kwantu czasu procesora), a po zakończeniu formowania oczekiwanie na przyjęcie do kolejki procesów gotowych.

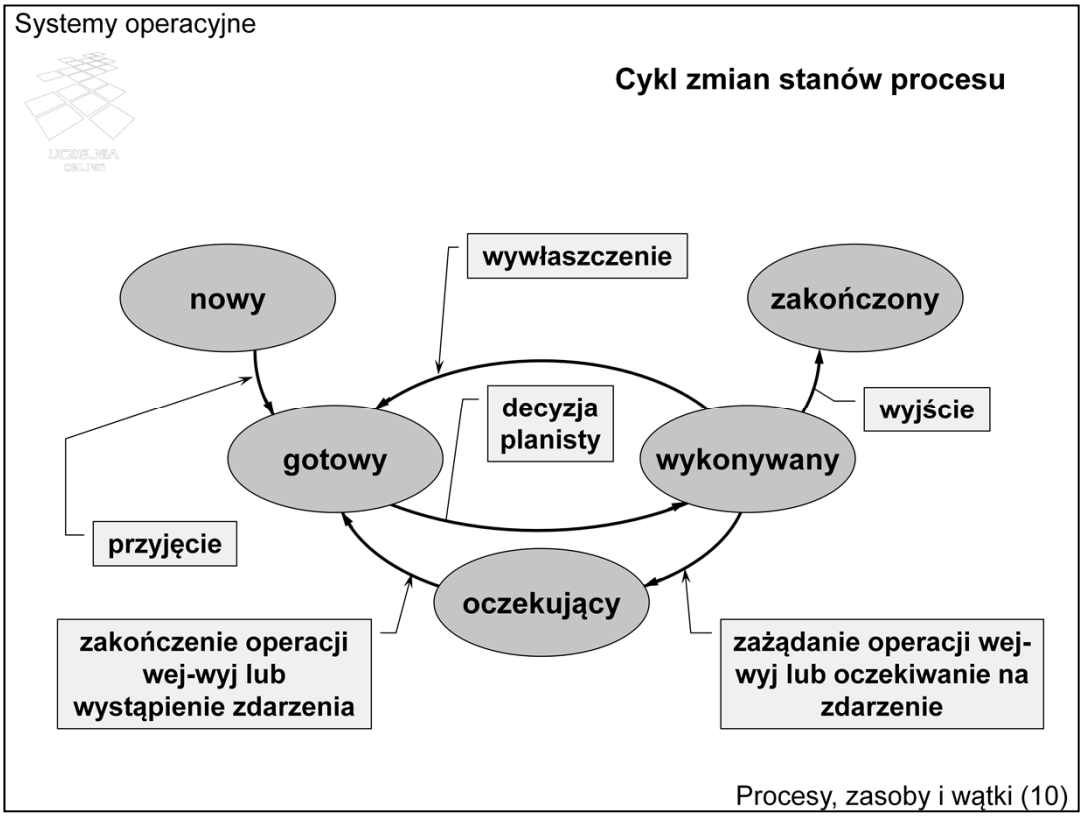
Wykonywany — wykonywanie instrukcji programu danego procesu i wynikająca z ich wykonywania zmiana stanu odpowiednich zasobów systemu.

Oczekujący — zatrzymanie wykonywania instrukcji programu danego procesu ze względy na potrzebę przydziału dodatkowych zasobów, konieczność otrzymania danych lub osiągnięcia odpowiedniego stanu przez otoczenie procesu (np. urządzenia zewnętrzne lub inne procesy).

Gotowy — oczekiwanie na przydział kwantu czasu procesora (dostępność wszystkich niezbędnych zasobów z wyjątkiem procesora).

Zakończony — zakończenie wykonywania programu, zwolnienie większości zasobów i oczekiwanie na możliwość przekazania informacji o zakończeniu innym procesom lub jądru systemu operacyjnego.

Pozostawanie procesu z stanie *zakończony* (w systemach uniksopodobnych zwany *zombi*) spowodowane jest przetrzymywaniem pewnych informacji o procesie po jego zakończeniu (np. statusu zakończenie). Całkowite usunięcie procesu mogłoby oznaczać zwolnienie pamięci i utratę tych informacji.



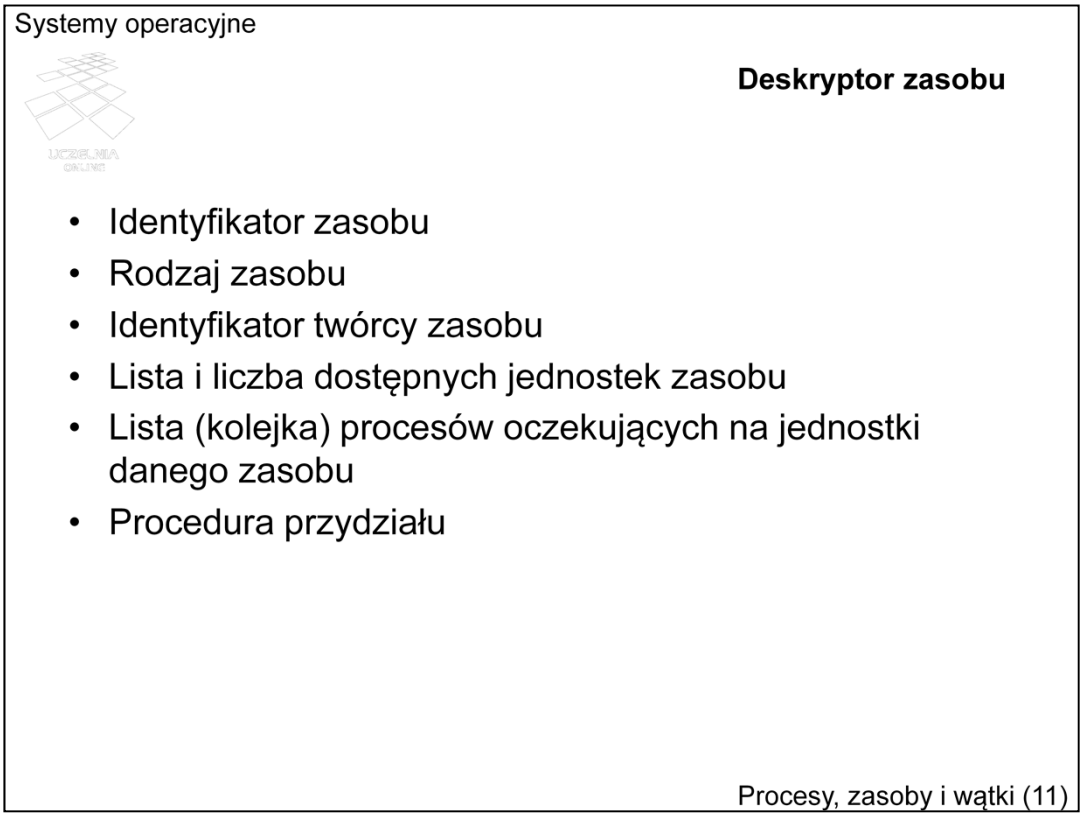
Przejście ze stanu *gotowy* do *wykonywany* wynika z decyzji modułu szeregującego procesy (planisty przydziału procesora), która oparta jest na priorytetach procesów. Jeśli w systemie jest jedna jednostka przetwarzająca (procesor), to w stanie *wykonywany* może być tylko jeden proces, podczas gdy pozostałe procesy znajdują się w innych stanach (w szczególności w stanie *gotowy*).

Przejście ze stanu *wykonywany* bezpośrednio do stanu *gotowy* oznacza wywłaszczenie procesu z procesora. Wywłaszczenie może być następstwem:

- upływu kwantu czasu w systemach z podziałem czasu,
- pojawienia się procesu gotowego z wyższym priorytetem w systemie z priorytetami dynamicznymi.

W systemie z podziałem czasu proces otrzymuje tylko kwant czasu na wykonanie kolejnych instrukcji. Upływ kwantu czasu odmierzany jest przez przerwanie zegarowe, a po stwierdzeniu wyczerpania kwantu czasu następuje przełączenie kontekstu i kolejny kwant czasu otrzymuje inny proces (rotacyjny algorytm planowania przydziału procesora).

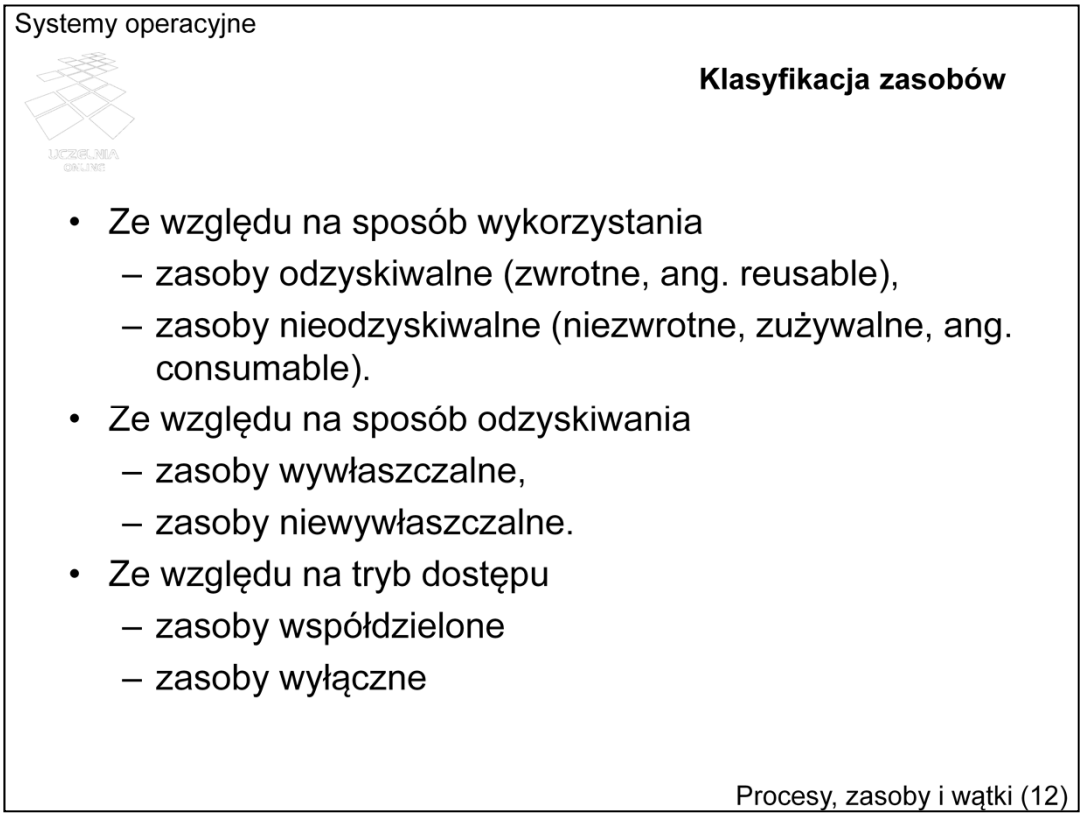
W systemie z dynamicznymi priorytetami przerwanie zegarowe lub inne zdarzenie obsługiwane przez jądro wyznacza momenty czasu, w którym przeliczane są priorytety procesów. Jeśli stosowane jest wywłaszczeniowe podejście do planowania przydziału procesora, oparte na priorytecie, proces o najwyższym priorytecie otrzymuje procesor. Więcej szczegółów zostanie omówionych w następnym module.



Informacje o zasobie mogą obejmować zróżnicowane atrybuty, których omawianie w oderwaniu od konkretnego rodzaju zasobu, czy chociażby pewnych klas nie ma sensu. Ogólnie dany zasób może być dostępny w liczbie wielu jednostek (np. pamięć) i informacje o tych jednostkach muszą być dostępne w deskryptorze.

Istotne są też powiązania pomiędzy jednostkami zasobu, a procesami, którym są przydzielone lub które na przydział oczekują. W prezentowanym modelu założono, że informacje o przydzielonych zasobach umieszczone są w deskryptorze procesu, a informacje o oczekiwaniu na przydział umieszczone są w deskryptorze zasobu, ale założenie takie ma raczej charakter poglądowy, niż implementacyjny. W rzeczywistości informacje o powiązaniach mogą być jednocześnie w obu strukturach, jeśli jest to wymagane na potrzeby zarządzania.

Innym zagadnieniem jest implementacja kolejki procesów. Procesy można powiązać w kolejkę wykorzystując odpowiednie wskaźniki w deskryptorze procesu. W deskryptorze procesu oczekującego można więc przechować wskaźnik lub identyfikator następnego (również poprzedniego) procesu w kolejce. Atrybut *lista procesów* w deskryptorze zasobu może się zatem sprowadzić do czoła takiej kolejki, czyli wskaźnika na deskryptor pierwszego proces, a deskryptor ten będzie zawierał wskaźnik na deskryptor następnego procesu.

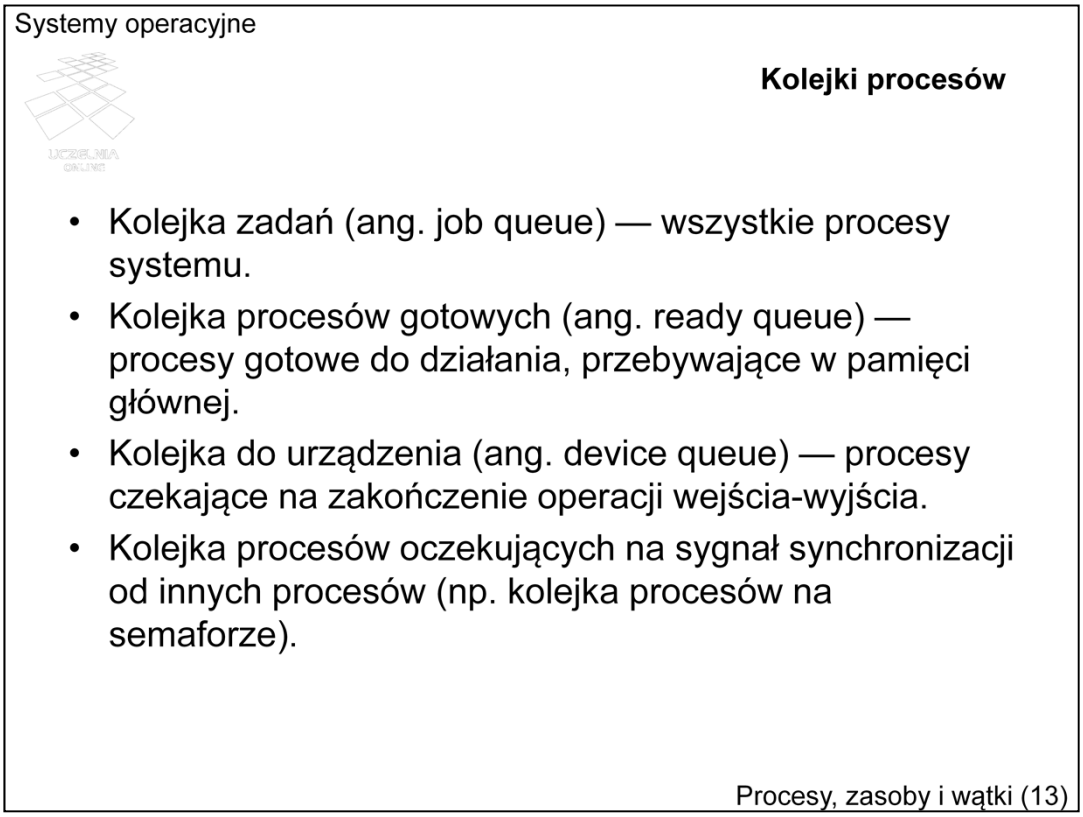


Na potrzeby wskazania ogólnych zasad zarządzania zasobami należy dokonać pewnej klasyfikacji. Sposób zarządzania w dużym stopniu zależy od przynależności zasobu do odpowiedniej klasy.

Jednostki pewnych zasobów można odzyskać po zakończonych procesach, np. pamięć. Są jednak zasoby, które proces zużywa w ramach przetwarzania, np. energia, istotna w urządzeniach przenośnych, lub czas procesora przed linią krytyczną (nie sam procesor, czy czas procesora ogólnie), istotny w systemach czasu rzeczywistego. Jednostki zasobów, podanych w przykładzie nie są wytwarzane przez procesy w systemie. Zasobem nieodzyskiwalnym wytwarzanym w wyniku przetwarzania mogą być dane lub sygnały synchronizujące.

Jeśli zasób można odzyskać, istotny z punktu widzenia pewnych problemów (np. zakleszczenia) może być sposób odzyskiwania. Zasób wywłaszczalny można odebrać procesowi (np. procesor). Natomiast jednostki zasobu niewywłaszczalnego proces sam musi zwrócić do systemu. Z punktu widzenia systemu oznacza to, że należy poczekać, aż proces, posiadający zasób, dojdzie do takiego stanu przetwarzania, w którym zasób nie będzie mu już potrzebny (np. po wydrukowaniu proces zwróci drukarkę, ale nie papier ani toner).

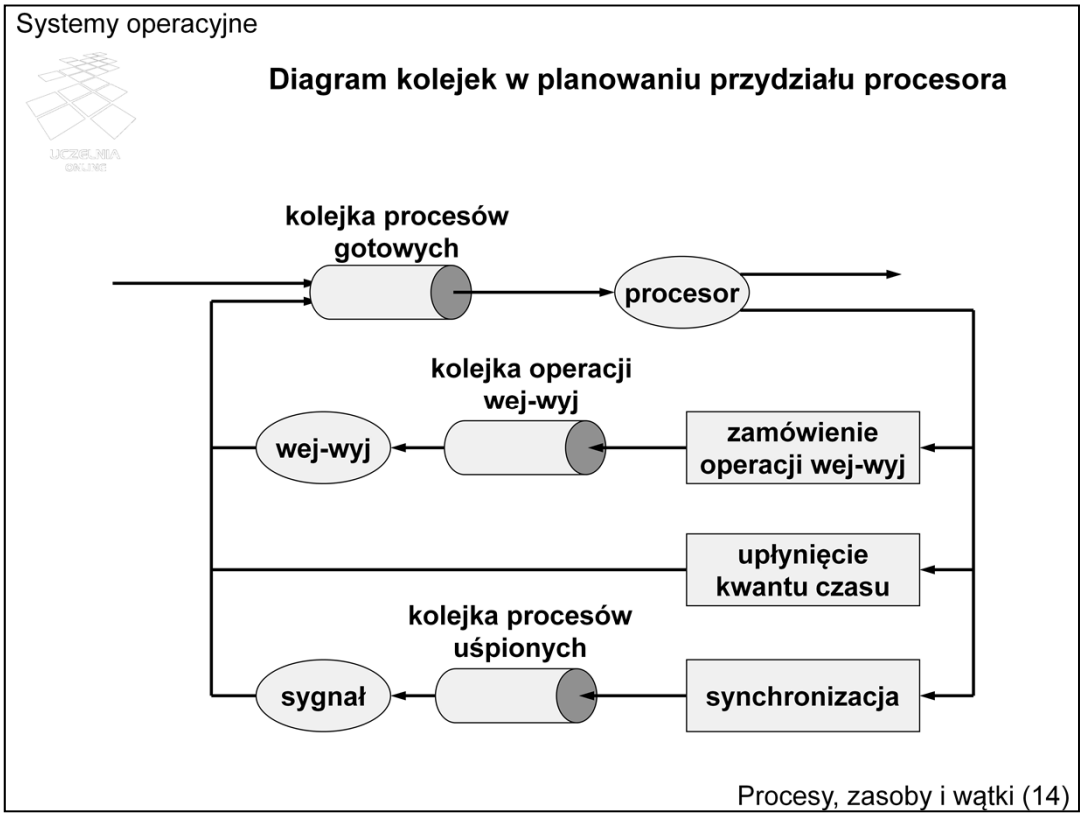
Pewne zasoby mogą być używane współbieżnie przez wiele procesów, np. segment kodu programu może być czytany i wykonywany przez wiele procesów w tym samym czasie. Są też zasoby dostępne w trybie wyłącznym, czyli dostępne co najwyżej dla jednego procesu w danej chwili czasu (np. drukarka, deskryptor procesu w tablicy procesów).



Stan procesu w dużym stopniu zależy od dostępności zasobów systemu. W zależności od tego, jakie zasoby są dostępne, proces jest albo wykonywany, albo na coś czeka (w stanach NOWY, GOTOWY, OCZEKUJĄCY). W czasie oczekiwania proces trafia do kolejki procesów oczekujących na dany zasób, a często opuszczając jedną kolejkę, trafia do innej, np. opuszczając kolejkę do urządzenia, trafia do kolejki procesów gotowych.

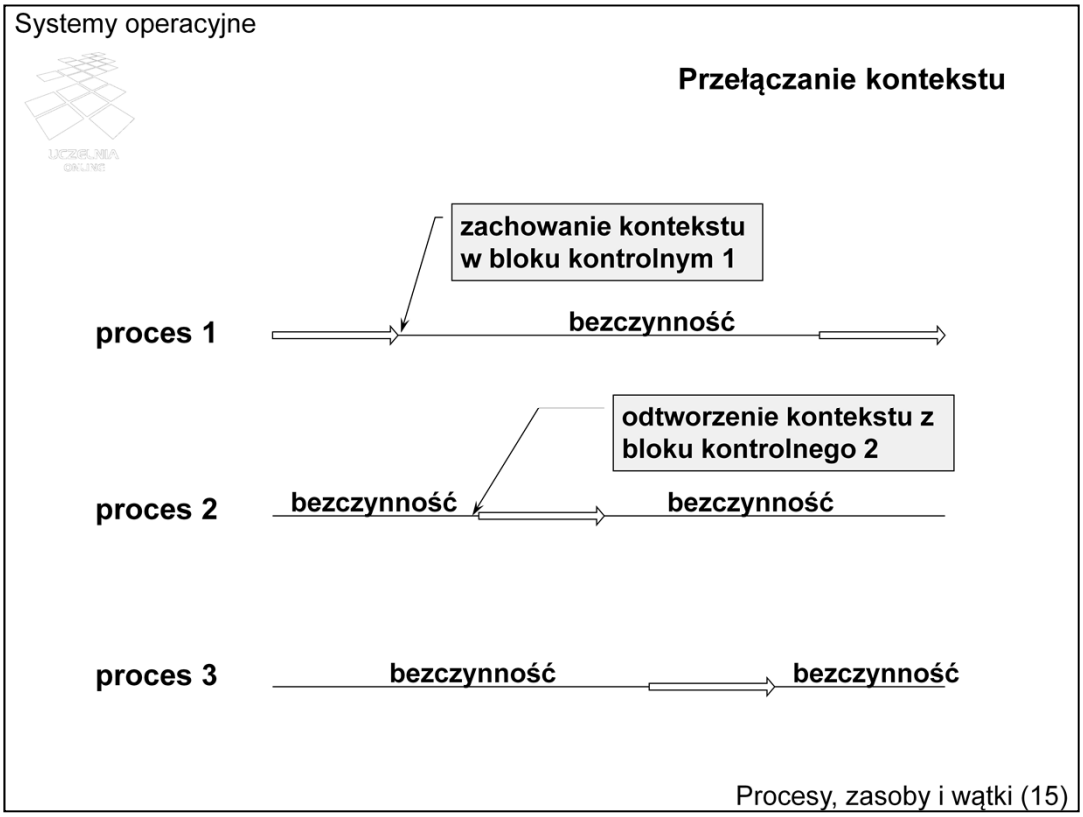
Kolejkowanie procesów polega na umieszczeniu ich na określonej liście. Jak już wspomniano, do budowy list procesów wykorzystywane są specjalne pola w tablicy procesów, w których dla każdego procesu na liście pamiętany jest identyfikator następnego procesu i/lub poprzedniego procesu.

W tym kontekście, określenie kolejka zadań może się okazać niezręczne — jest to raczej zbiór zadań lub tablica zadań.



Po wyjściu ze stanu *wykonywany* (po opuszczeniu procesora), w zależności od przyczyny zaprzestania wykonywania programu, proces trafia do jednej z kolejek. Po upłynięciu kwantu czasu proces (w stanie *gotowy*) trafia po prostu na koniec kolejki procesów gotowych, a z czoła tej kolejki pobierany jest proces do wykonywania. Następuje więc przełączenie procesora w kontekst nowego procesu. Odpowiednio częste przełączanie kontekstu przy niezbyt długim oczekiwaniu w kolejce procesów gotowych umożliwia na bieżąco (ang. on-line) interakcję procesu z użytkownikiem, dzięki czemu użytkownik ma wrażenie, że wyłącznie jego zadania są wykonywane przez system. Użytkownik nie ma zatem poczucia dyskomfortu nawet, gdy zasoby systemu są w rzeczywistości współdzielone przez współbieżnie działające procesy wielu użytkowników.

Jeśli proces opuszcza procesor z innych przyczyn, niż upłynięcie kwantu czasu, przechodzi do stanu *oczekujący*, co wiąże się z umieszczeniem go w kolejce procesów oczekujących na zajście określonego zdarzenia. Kolejek takich może być wiele, np. kolejka może być związana z każdym urządzeniem zewnętrznym, z mechanizmami synchronizacji itp.



Jak już wspomniano, opuszczenie stanu *wykonywany* przez jeden proces udostępnia procesor innemu procesowi i następuje *przełączenie kontekstu*. Przełączenie kontekstu polega na zachowaniu stanu przetwarzania procesu oddającego procesor (zachowaniu kontekstu) i załadowaniu stanu przetwarzania innego procesu (odtworzenie kontekstu). Na ogólnie rozumiany kontekst składa się stan tych zasobów, z których jeden proces w ramach przełączania rezygnuje na rzecz drugiego. W najprostszym przypadku jest to stan procesora, czyli zawartość rejestrów. W procesorze mogą być rejestry, które nie są elementem kontekstu procesu, ale przechowują wartości, stanowiące element definicji stanu całego systemu. Wartości takich rejestrów nie trzeba zachowywać ani odtwarzać, ale też nie można dopuścić do ich swobodnej modyfikacji przez proces.

Tak rozumiany kontekst zajmuje niewiele miejsca w pamięci i przechowywany jest w deskryptorze procesu. Przełączanie kontekstu sprowadza się zatem do zaktualizowania we właściwym deskryptorze informacji o kontekście procesu wywłaszczanego z procesora, a następnie załadowaniu rejestrów procesora odpowiednimi wartościami z deskryptora procesu, który ma być wykonywany jako następny.

Czas przełączania kontekstu jest marnowany z punktu widzenie wykorzystania procesora, gdyż żaden program użytkowy nie jest w tym czasie wykonywany. W celu skrócenia czasu przełączania, procesory oferują często wsparcie dla tej operacji w postaci odpowiednich instrukcji lub bardziej złożonych mechanizmów na poziomie architektury.

Systemy operacyjne

Planista

- Planista krótkoterminowy, planista przydziału procesora (ang. CPU scheduler) — zajmuje się przydziałem procesora do procesów gotowych.
- Planista średnioterminowy (ang. medium-term scheduler) — zajmuje się wymianą procesów pomiędzy pamięcią główną a pamięcią zewnętrzną (np. dyskiem).
- Planista długoterminowy, planista zadań (ang. long-term scheduler, job scheduler) — zajmuje się ładowaniem nowych programów do pamięci i kontrolą liczby zadań w systemie oraz ich odpowiednim doborem w celu zrównoważenia wykorzystania zasobów.

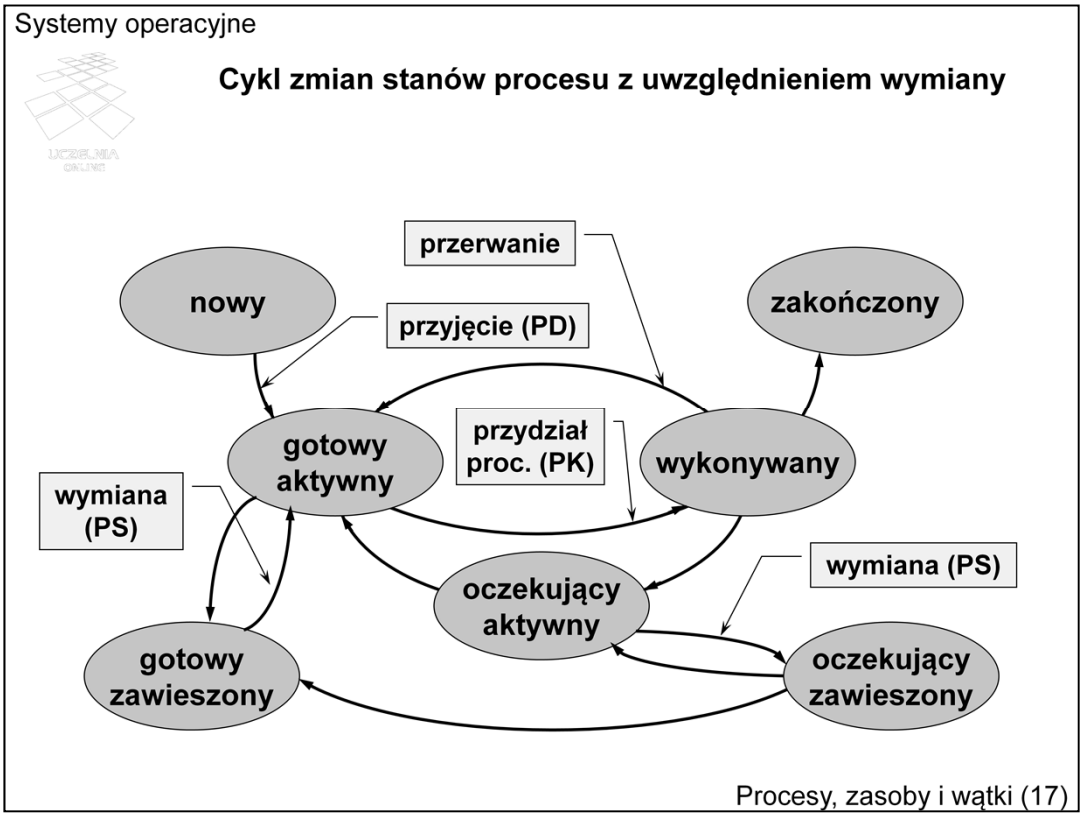
Procesy, zasoby i wątki (16)

Pytaniem, pozostawionym dotychczas bez odpowiedzi, jest: „Który proces będzie wykonywany jako następny?”, „W kontekst którego z procesów gotowych nastąpi przełączenie kontekstu?”. Wybór ten jest zadaniem planisty krótkoterminowego (ang. scheduler).

Ogólnym zadaniem planistów (programów szeregujących) jest wybieranie procesów z pewnego zbioru tak, aby dążyć do optymalizacji przetwarzania w systemie. Kryteria optymalizacji mogą być jednak bardzo zróżnicowane.

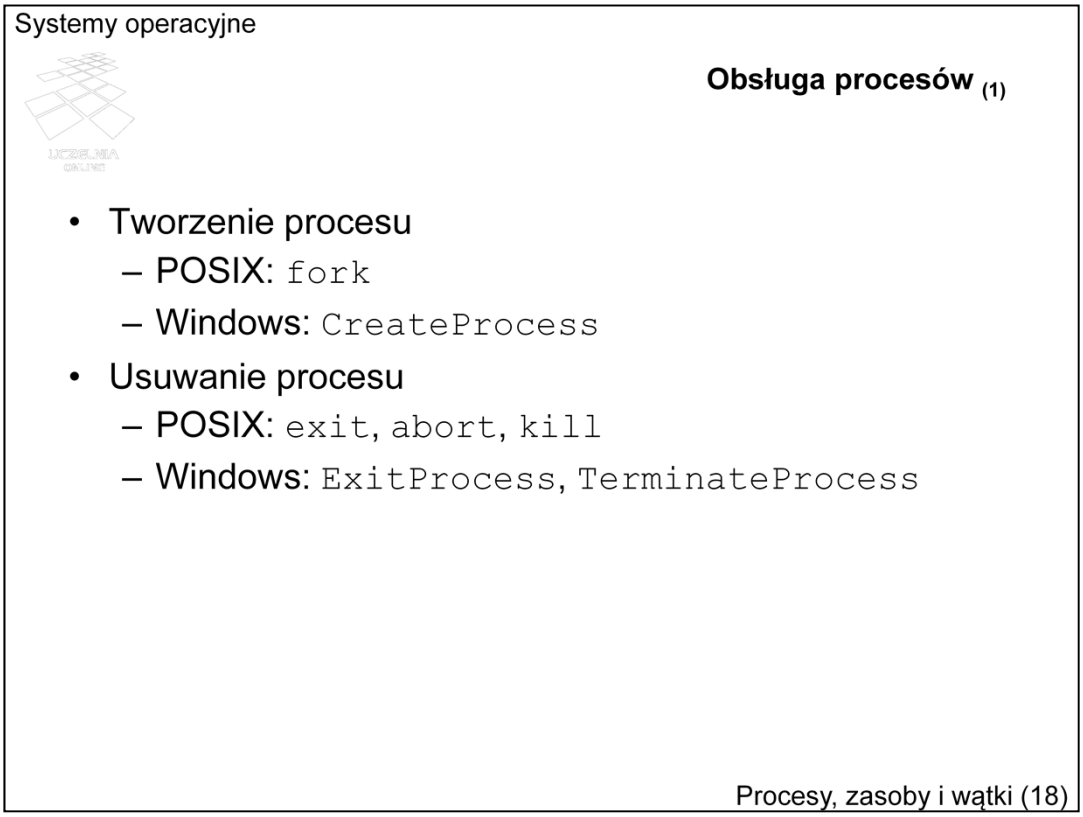
W kontekście optymalizacji wykorzystania zasobów mówi się o równoważeniu obciążenia systemu, czyli ogólnie procesora i urządzeń zewnętrznych. Zrównoważenie takie umożliwia osiągnięcie odpowiedniego poziomu wykorzystania zasobów systemu. W przeciwnym razie w systemie może powstać wąskie gardło, którym będzie procesor lub urządzenie zewnętrzne. O procesach, które wykorzystują głównie procesor (np. związane są z realizacją dużej liczby obliczeń) mówi się, że są *ograniczone procesorem*. Procesy, które większość czasu w systemie spędzają w oczekiwaniu na realizację operacji wejścia-wyjścia, określane są jako *ograniczone wejściem-wyjściem* (np. edytor tekstu). Wybór procesów tylko z jednej z tych grup może spowodować powstanie wąskiego gardła na intensywnie wykorzystywanym zasobie i tym samym zmniejszyć wykorzystania pozostałych zasobów. W systemach interaktywnych istotny jest tzw. *czas odpowiedzi*. Zbyt długi czas odpowiedzi grozi zniecierpliwieniem użytkowników i ich irracjonalnym zachowaniem.

W różnego typu systemach rola poszczególnych planistów może być większa lub mniejsza. W systemach interaktywnych zmniejsza się (lub zupełnie znika) rola planisty długoterminowego, a rośnie rola planisty krótkoterminowego. W systemach wsadowych jest dokładnie odwrotnie.



Z analizy przełączania kontekstu wynikało, że proces oddaje procesor, w związku z tym stan procesora należy zapisać w bloku kontrolnym. Wymiana procesów w pamięci oznacza usunięcie z pamięci jednego procesu, żeby zasób pamięci oddać innemu procesowi. Umożliwienie wznowienia przetwarzania uwarunkowane jest zapisaniem zwolnionych obszarów pamięci w celu późniejszego odtworzenia. Zapis wykonywany jest na dysku w specjalnie do tego przygotowanym obszarze lub pliku, zwanym obszarem (plikiem) wymiany. Z pamięci usuwany jest najczęściej jakiś proces oczekujący, ale możliwe jest też usunięcie procesu gotowego. Dla odróżnienia stanów procesu w pamięci od stanów na urządzeniu wymiany, proces w pamięci określany będzie jako *aktywny*, a proces na urządzeniu wymiany jako *zawieszony*.

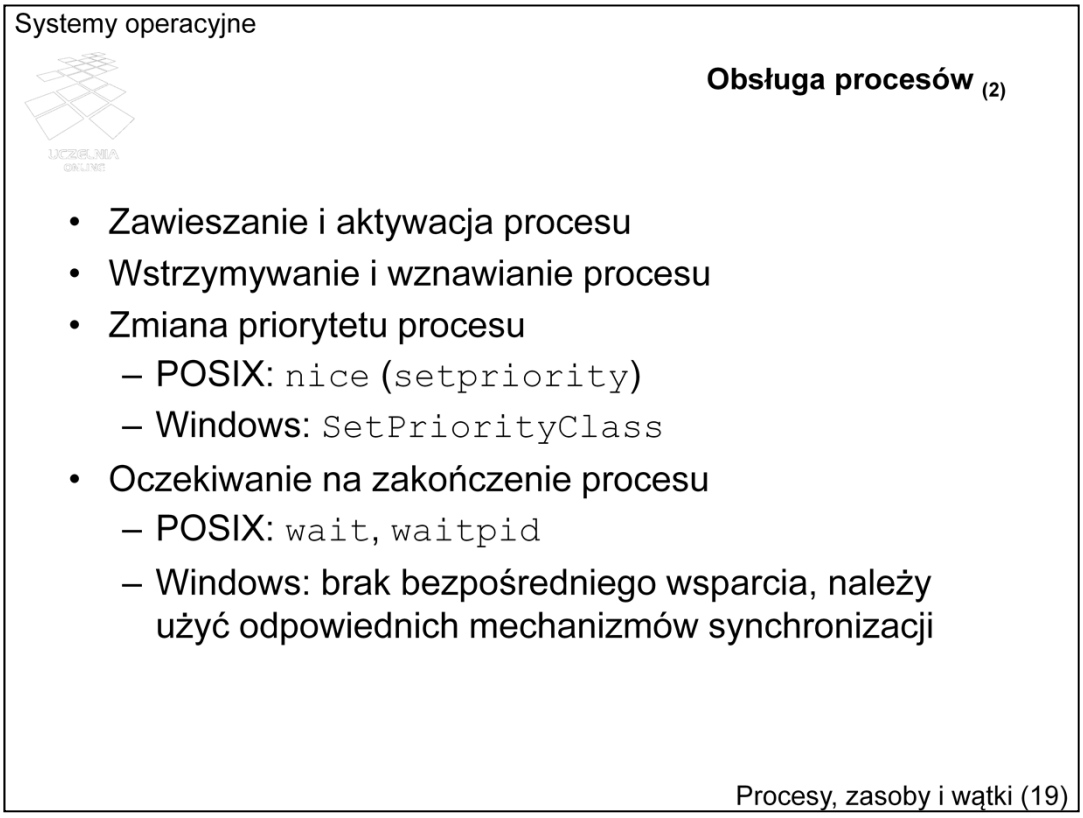
W grafie zmian stanów procesu, oprócz stanów wynikających z wymiany uwzględniono role planistów. Decyzję o przyjęciu nowego procesu do systemu podejmuje planista długoterminowy (PD). Przejście ze stanu *gotowy* do *wykonywany* wynika z decyzji planisty krótkoterminowego (PK). Planista średnioterminowy (PS) odpowiada natomiast za wymianę, czyli decyduje o tym, które procesu usunąć z pamięci, a które ponownie załadować. Uwzględniając wymianę, można powiedzieć, że pamięć jest zasobem wywłaszczalnym. W przypadku braku wymiany, odebranie procesowi pamięci oznaczałoby jego usunięcie — pamięć byłaby więc zasobem niewywłaszczalnym.



Elementarne operacje na procesach obejmują: tworzenie, usuwanie, zmianę stanu, zmianę priorytetu. Nie wszystkie operacje na procesach są dostępne dla aplikacji.

Dostępne są operacje tworzenia i usuwania. Proces jest tworzony przez inny proces, w wyniku czego tworzą się zależności przodek-potomek pomiędzy procesami. W systemach uniksopodobnych, zgodnie ze standardem POSIX, do tworzenia służy funkcja `fork`, która w Linuksie implementowana jest za pomocą bardziej ogólnej funkcji `clone`. W systemach Windows 2000/XP wykorzystywana do tego celu jest jedna z funkcji: `CreateProcess`, `CreateProcessAsUser`, `CreateProcessWithLogonW` lub (w najnowszych wersjach) `CreateProcessWithTokenW`.

Usunięcie procesu jest skutkiem zakończenia wykonywania programu, albo wynika z interwencji zewnętrznej. W systemach zgodnych z POSIX proces informuje system o swoim zakończeniu, wywołując funkcję `exit` lub `abort`. Do usuwania procesu przez inny proces, albo przez jądro systemu operacyjnego wykorzystywana jest funkcja `kill`. W systemach Windows 2000/XP dostępne są 2 funkcje: `ExitProcess`, `TerminateProcess`.



W interfejsie aplikacji nie ma funkcji umożliwiających swobodną zmianę stanu procesu. Operacje zawieszania i aktywacji (związane z wymianą) są dostępne wewnątrz zarządcy procesów. Mogą być natomiast udostępnione funkcje wstrzymania i wznawiania procesów. Ich skutkiem jest przejście odpowiednio w stan oczekiwania i gotowości. Zdarzeniem oczekiwanym po wstrzymaniu jest wywołanie funkcji wznowienia. W systemach zgodnych ze standardem POSIX efekt ten można uzyskać przez użycie funkcji `kill`, przekazując odpowiednie sygnały. W systemach Windows tego typu funkcje dostępne są dla wątków.

Nie zawsze też możliwa jest swobodna zmiana priorytetu. Funkcja `nice` w systemach standardu POSIX zmienia np. tylko pewną składową priorytetu procesu, podczas gdy właściwa wartość priorytetu zależy od kilku innych czynników. Zostanie to omówiony przy okazji szeregowania procesów.

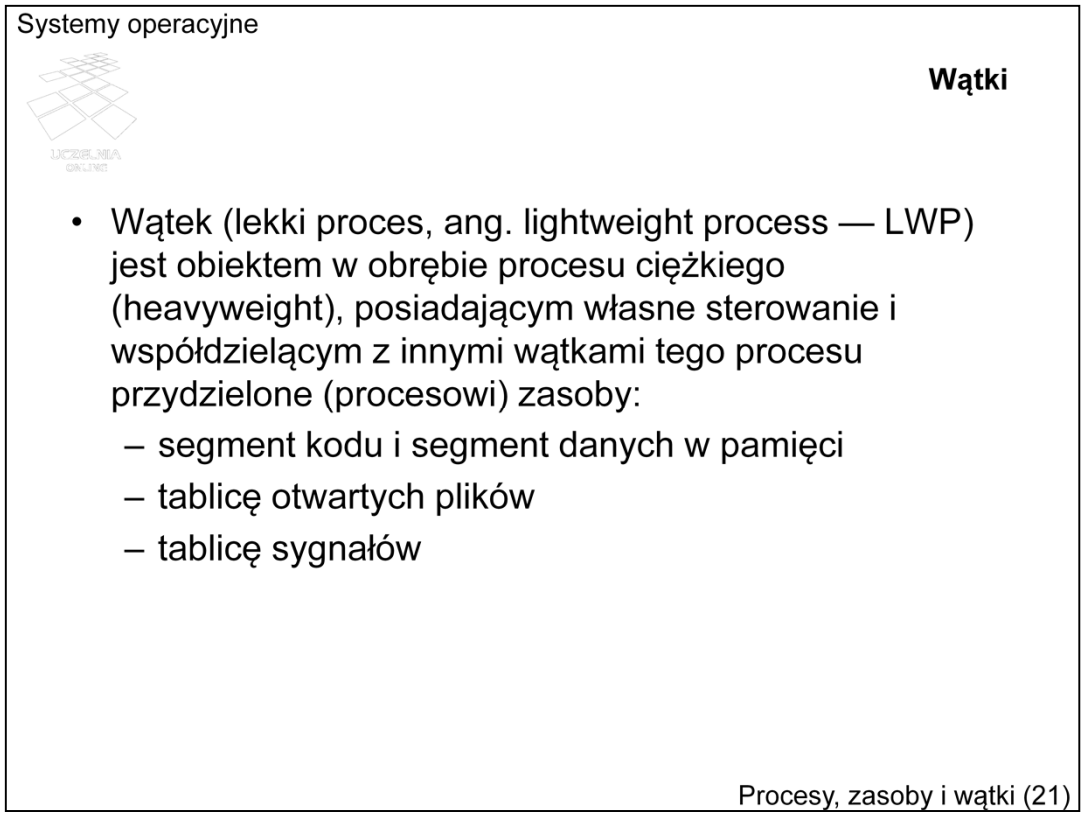
Systemy operacyjne

Elementarne operacje na zasobach

- Tworzenie deskryptora zasobu
- Usuwanie deskryptora zasobu
- Realizacja żądania przydziału jednostek zasobu
- Zwolnienie i odzyskiwanie jednostek zasobu odzyskiwalnego
- Uwolnienie (wyprodukowanie) jednostki zasobu nieodzyskiwalnego

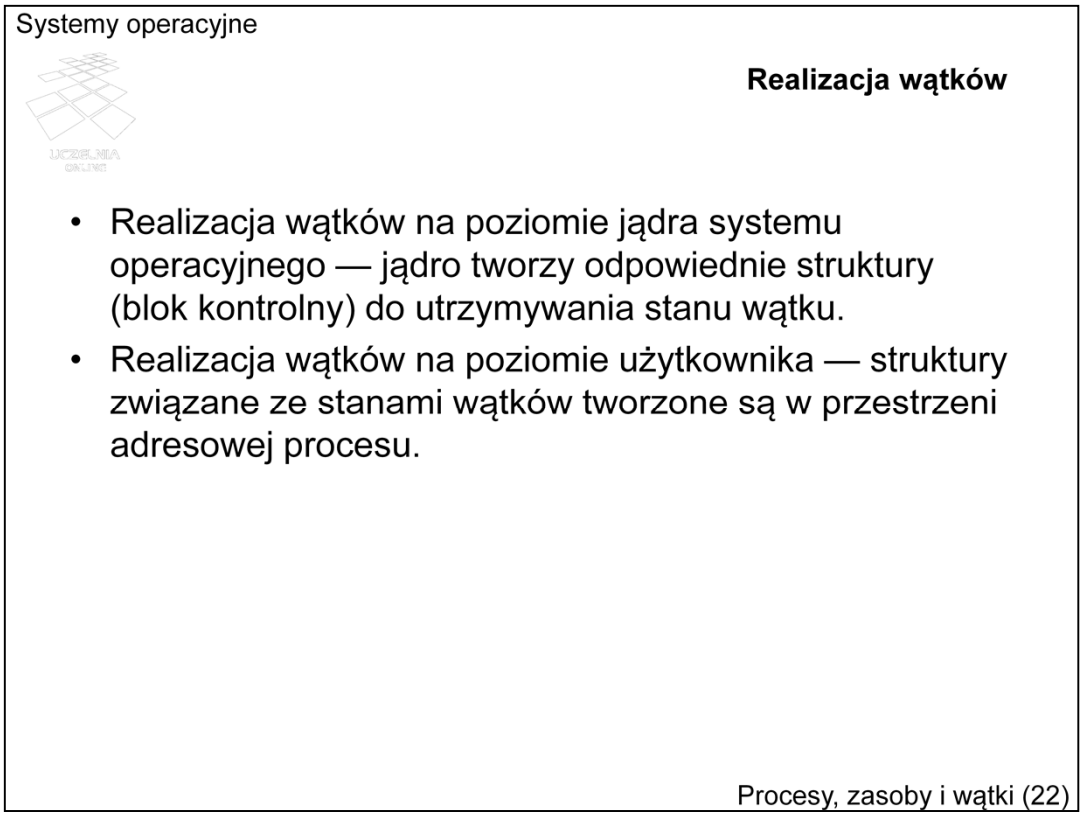
Procesy, zasoby i wątki (20)

Elementarne operacje na zasobach nie muszą być bezpośrednio dostępne dla aplikacji. Tworzenie lub usuwanie deskryptora jest wewnętrzną sprawą zarządcy. Realizacja przydziału jednostki zasobu jest skutkiem żądania ze strony procesu, ale bardzo często jest pośrednim skutkiem innych operacji (np. tworzenia procesu). Odzyskiwanie jednostek jest najczęściej skutkiem zakończeniu procesu.



Koncepcja wątku (ang. thread) wiąże się ze współdzieleniem zasobów. Każdy proces (ciężki proces w odróżnieniu od lekkiego, czyli wątku) otrzymuje zasoby od odpowiedniego zarządcy i utrzymuje je do swojej dyspozycji. Zasoby przydzielone procesowi wykorzystywane są na potrzeby **sekwencyjnego** wykonania programu, ale w wyniku wykonania programu mogą się pojawić kolejne żądania zasobowe. Niedostępność żądanego zasobu powoduje zablokowanie procesu (wejście w stan oczekiwania). W programie procesu może być jednak inny niezależny fragment, do wykonania którego żądany zasób nie jest potrzebny. Można by zatem zmienić kolejność instrukcji w programie i wykonać ten niezależny fragment wcześniej, jednak do jego wykonania może być potrzebny inny zasób. Dostępność zasobów zależy od stanu całości system i w czasie tworzenia programu nie wiadomo, które z nich będą dostępne, gdy będą potrzebne do wykonania jednego czy drugiego fragmentu.

W programie dobrze jest zatem wówczas wyodrębnić takie niezależne fragmenty i wskazać systemowi, że można je wykonać w dowolnej kolejności lub nawet współbieżnie, w miarę dostępnych zasobów. Taki wyodrębniony fragment określa się jako *wątek*. Wątek korzysta głównie z zasobów przydzielonych procesowi — współdzieli je z innymi wątkami tego procesu. Zasobem, o który wątek rywalizuje z innymi wątkami, jest procesor, co wynika z faktu, że jest on odpowiedzialny za wykonanie fragmentu programu. Wątek ma więc własne sterowanie, w związku z czym kontekst każdego wątku obejmuje licznik rozkazów, stan rejestrów procesora oraz stos. Każdy wątek musi mieć swój własny stos, gdzie odkładane są adresy powrotów z podprogramów oraz alokowane lokalne zmienne.



Tworzenie dodatkowych wątków w ramach tego samego procesu oraz przełączanie kontekstu pomiędzy nimi jest ogólnie mniej kosztowne, niż w przypadku ciężkich procesów, gdyż wymaga przydziału lub odpowiedniej zmiany stanu znacznie mniejszej liczby zasobów. Wynika to z faktu, że wątki tego samego procesu współdzielą większość przestrzeni adresowej (segment danych, segment kodu), otwarte pliki i sygnały. Nie jest zatem konieczne wykonywanie dodatkowych działań związanych z zarządzaniem pamięcią. Wątki mogą być nawet tak zorganizowane, że jądro nie jest świadome ich istnienia. Deskryptory wątków utrzymywane są w pamięci procesu (a nie jądra) i cała obsługa wykonywana jest w trybie użytkownika.

Alternatywą jest zarządzanie wątkami w trybie systemowym przez jądro, które utrzymuje deskryptory i odpowiada za przełączanie kontekstu pomiędzy wątkami.

Istotne w obsłudze wielowątkowości, niezależnie od sposobu realizacji, jest dostarczenie odpowiednich mechanizmów synchronizacji wątków wewnątrz procesu. Potrzeba synchronizacji wynika z faktu współdzielenia większości zasobów procesu. Problemy synchronizacji będą przedmiotem rozważań w innym module.

Systemy operacyjne

Realizacja wątków na poziomie jądra systemu operacyjnego

- Wątek posiada własny blok kontrolny w jądrze systemu operacyjnego, obejmujący:
  - stan licznika rozkazów,
  - stan rejestrów procesora,
  - stan rejestrów związanych z organizacją stosu.
- Własności realizacji wątków na poziomie jądra:
  - przełączanie kontekstu pomiędzy wątkami przez jądro,
  - większy koszt przełączania kontekstu,
  - bardziej sprawiedliwy przydział czasu procesora.

Procesy, zasoby i wątki (23)

Obsługa wielowątkowości na poziomie jądra systemu (w trybie systemowym) oznacza, że wszelkie odwołania do mechanizmów obsługi wątków wymagają dostępu do usług jądra, co zwiększa koszt czasowy realizacji. Jądro musi też utrzymać bloki kontrolne (deskryptory) wątków, co w przypadku wykorzystania statycznych tablic może stanowić istotny koszt pamięciowy.

Z drugiej strony, świadomość istnienia wątków procesu umożliwia uwzględnienie tego faktu w zarządzaniu zasobami przez jądro i prowadzi do poprawy ich wykorzystania.

Systemy operacyjne

Realizacja wątków w trybie użytkownika

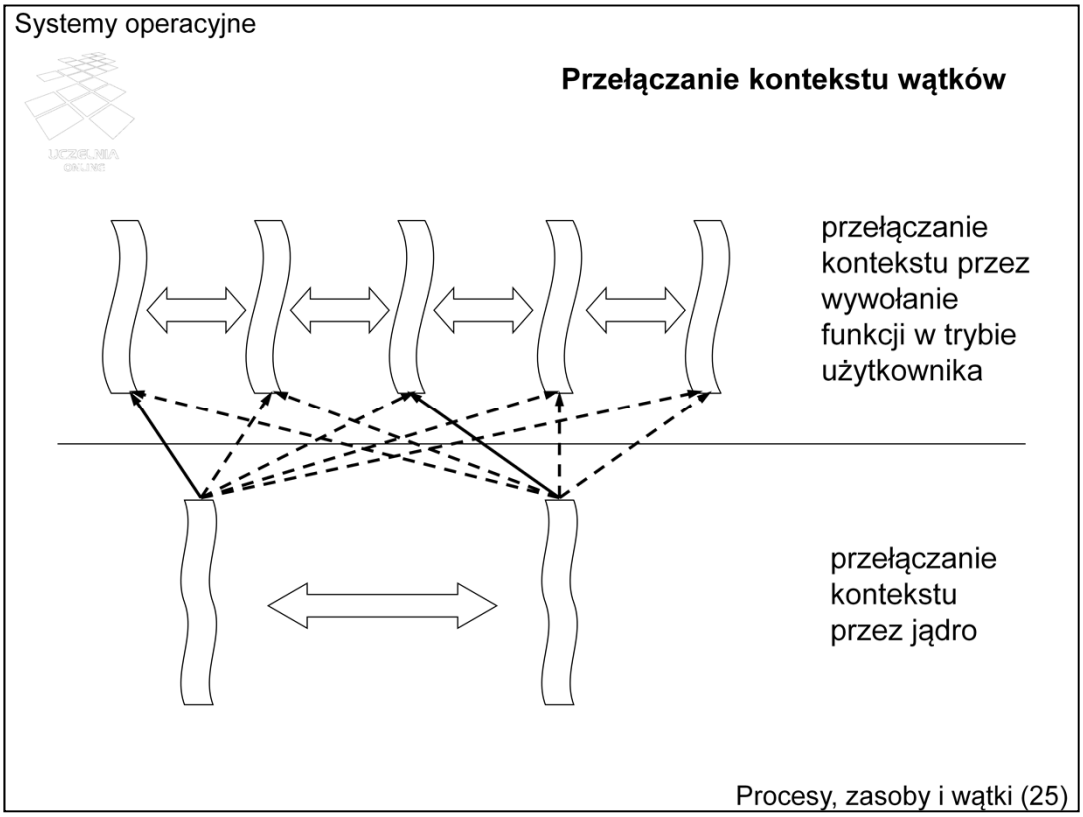
- Deskryptor wątku znajduje się w tablicy wątków w pamięci danego procesu (jądro nie wie nic o wątkach).
- Własności realizacji na poziomie użytkownika:
  - przydział czasu procesora dla procesu (nie dla wątku)
  - przełączanie kontekstu pomiędzy wątkami przez jawne odwołania do mechanizmu obsługi wątków
  - mniejszy koszt przełączania kontekstu (bez angażowania jądra systemu operacyjnego)
  - możliwość „głodzenia” wątków tego samego procesu, gdy jeden z nich spowoduje przejście procesu w stan oczekiwania

Procesy, zasoby i wątki (24)

Realizacja wątków przez odpowiednią bibliotekę w trybie użytkownika zwiększa szybkość przełączania kontekstu, ale powoduje, że jądro, nie wiedząc nic o wątkach, planuje przydział czasu procesora dla procesów. Oznacza to, że w przypadku większej liczby wątków procesu czas procesora, przypadający na jeden wątek jest mniejszy, niż w przypadku procesu z mniejszą liczbą wątków.

Problemem jest też wprowadzanie procesu w stan oczekiwania, gdy jeden z wątków zażąda operacji wejścia-wyjścia lub utknie na jakimś mechanizmie synchronizacji z innymi procesami. Planista traktuje taki proces jako oczekujący do czasu zakończenia operacji, podczas gdy inne wątki, o których jądro nie wie, mogłyby się wykonywać.

W niektórych systemach operacyjnych wyróżnia się zarówno wątki trybu użytkownika, jak i wątki trybu jądra. W systemie Solaris terminem *wątek* określa się wątek, istniejący w trybie użytkownika, a wątek trybu jądra określa się jako *lekki proces*. W systemie Windows wprowadza się pojecie *włókna*, zwanego też lekkim wątkiem (ang. fiber, lightweight thread), które odpowiada wątkowi trybu użytkownika, podczas gdy termin *wątek* odnosi się do wątku trybu jądra. Takie rozróżnienie umożliwia operowanie pewną liczbą wątków trybu jądra, a w ramach realizowanych przez te wątki programów może następować przełączanie pomiędzy różnymi wątkami trybu użytkownika bez wiedzy jądra systemu. Wątek trybu jądra można więc traktować jako wirtualny procesor dla wątku trybu użytkownika.



Kontekst pomiędzy dwoma lekkimi procesami przełączany jest przez jądro. Każdy z lekkich procesów wykonuje jakiś wątek trybu użytkownika (włókno), co obrazuje ciągła linia ze strzałką. Dla każdego lekkiego procesu istnieje zatem bieżące włókno. W ramach wykonywanego kodu takiego włókna może nastąpić wywołanie funkcji zachowania bieżącego kontekstu, a następnie funkcji odtworzenia innego (wcześniej zachowanego) kontekstu, o ile tylko w miejscu wywołania dostępny jest odpowiedni deskryptor, opisujący odtwarzany kontekst. Potencjalnie więc każdy z lekkich procesów może wykonywać dowolne z włókien, co symbolizuje przerywana linia.

Systemy operacyjne

Obsługa wątków (1)

- Tworzenie wątku
  - POSIX: `pthread_create`
  - Windows: `CreateThread`, `CreateRomoteThread`
- Usuwanie wątku
  - POSIX: `pthread_exit, pthread_cancel`
  - Windows: `ExitThread`, `TerminateThread`

Procesy, zasoby i wątki (26)

Systemy operacyjne

Obsługa wątków (2)

- Wstrzymywanie i wznawianie wątku
  - POSIX: brak
  - Windows: `SuspendThread`, `ResumeThread`
- Zmiana priorytetu wątku
  - POSIX: `pthread_setschedprio`, `pthread_setschedparam`
  - Windows: `SetThreadPriority`
- Oczekiwanie na zakończenie wątku
  - POSIX: `pthread_join`
  - Windows: brak bezpośredniego wsparcia, należy użyć odpowiednich mechanizmów synchronizacji

Procesy, zasoby i wątki (27)

Systemy operacyjne

Realizacja procesów/wątków w systemie Linux

- W jądrze systemu Linux nie odróżnia się pojęcia wątku od procesu.
- Procesy mogą współdzielić takie zasoby, jak:
  - przestrzeń adresowa,
  - otwarte pliki,
  - informacje o systemie plików,
  - procedury obsługi sygnałów.
- Deskryptory procesów (o strukturze `struct task_struct`) przechowywane są na dwukierunkowej, cyklicznej liście zadań.

Procesy, zasoby i wątki (28)

Proces potomny tworzony jest w systemie Linux poprzez wywołanie funkcji `clone`. Funkcja ta wykorzystywana jest między innymi do implementacji funkcji `fork`, ujętej w standardzie POSIX.

Tworząc nowy proces z użyciem funkcji `clone` można określić, które zasoby procesu macierzystego mają być współdzielone z potomkiem. W zależności od zakresu współdzielonych zasobów, nowo utworzony proces może być uznawany za wątek lub za ciężki proces. Typowe wątki będą współdzielić przestrzeń adresową, otwarte pliki i inne informacje związane z systemem plików (np. katalog bieżący, korzeń drzewa katalogów) i procedury obsługi sygnałów.

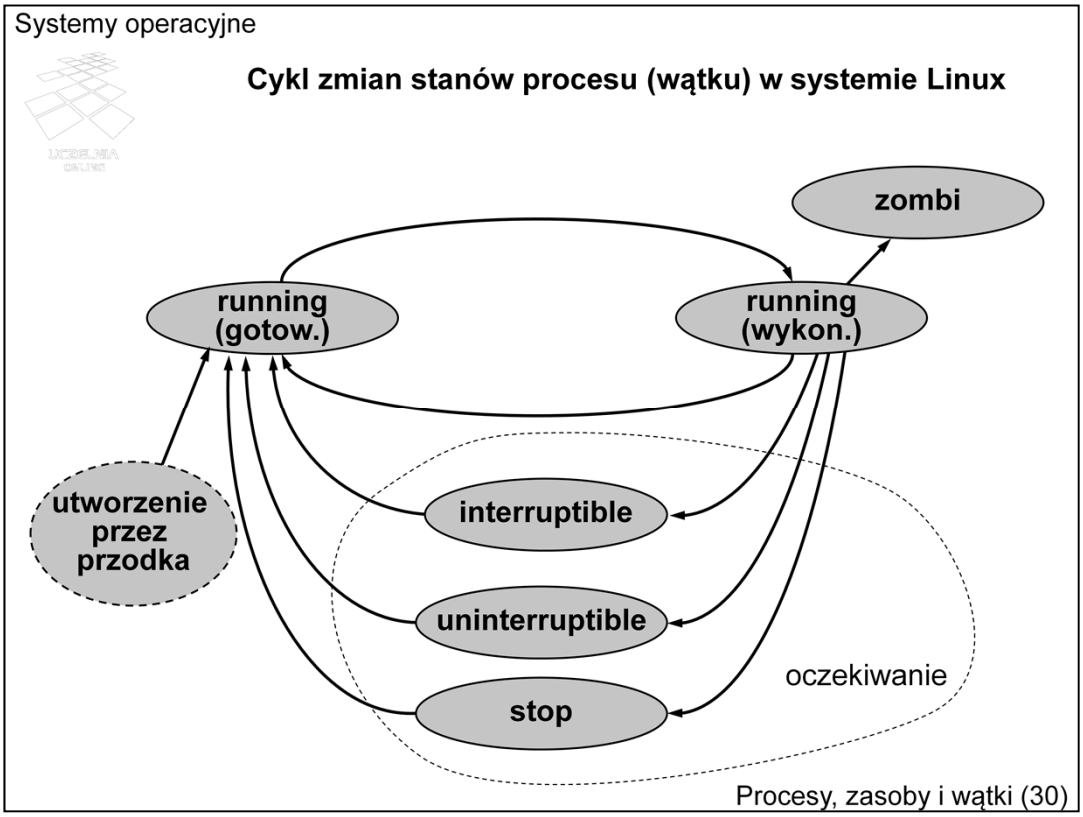
Rozróżnienie *proces ciężki – proces lekki* sprowadza się zatem do określenia zakresu współdzielenia zasobów.

Systemy operacyjne

Stany procesu (wątku) w systemie Linux

- TASK_RUNNING — wykonywanie lub gotowość (do wykonania)
- TASK_INTERRUPTIBLE — oczekiwanie na zajście zdarzenia lub sygnał
- TASK_UNINTERRUPTIBLE — oczekiwanie na zajście zdarzenia, przy czym sygnały są ignorowane
- TASK_ZOMBI — stan zakończenia utrzymywany w celu przechowania deskryptora procesu
- TASK_STOP — zatrzymanie w wyniku otrzymania sygnału (np. SIGSTOP)

Procesy, zasoby i wątki (29)



Cykl zmian stanów w Linuksie jest bardzo prosty — odpowiada dość dokładnie ogólnemu schematowi. Jedyna różnica to wyodrębnienie dwóch stanów oczekiwania — w jednym następuje reakcja na sygnały, w drugim sygnały są ignorowane. Jako specyficzny rodzaj oczekiwania można też traktować stan wstrzymania TASK_STOP. Specyficzną cechą jest również brak rozróżnienia pomiędzy stanem gotowości a stanem wykonywania. Są to oczywiście dwa różne stany, ale w deskryptorze procesu oznaczone w taki sam sposób.

Systemy operacyjne

Proces w systemie Windows 2000/XP

- Proces stanowi środowisko do wykonywania wątków.
- Struktury opisu procesu obejmują:
  - EPROCESS — blok centrum wykonawczego, opisujący proces,
  - KPROCESS — blok kontrolny procesu, część struktury EPROCESS,
  - PEB — blok środowiska procesu, dostępny w trybie użytkownika.

Procesy, zasoby i wątki (31)

Proces w systemie Windows gromadzi zasoby na potrzeby wykonywania wątków, wchodzących w jego skład. Informacje o procesie znajdują się w strukturze EPROCESS, której częścią jest właściwy blok kontrolny (KPROCESS). Zawartość obu tych struktur dostępna jest w trybie jądra. W ich skład wodzi wiele wskaźników do innych struktur (między innymi struktur opisujących wątki). Część opisu procesu — blok środowiska procesu PEB — znajduje się w części przestrzeni adresowej, dostępnej w trybie użytkownika.

Systemy operacyjne

Wątki w systemie Windows 2000/XP

- Wątki korzystają z zasobów przydzielonych procesom.
- Wątki (nie procesy) ubiegają się o przydział procesora i są szeregowane przez planistę krótkoterminowego.
- Struktury opisu wątku obejmują:
  - ETHREAD — blok centrum wykonawczego, opisujący wątek,
  - KTHREAD — blok kontrolny wątku, część struktury ETHREAD,
  - TEB — blok środowiska wątku, dostępny w trybie użytkownika.

Procesy, zasoby i wątki (32)

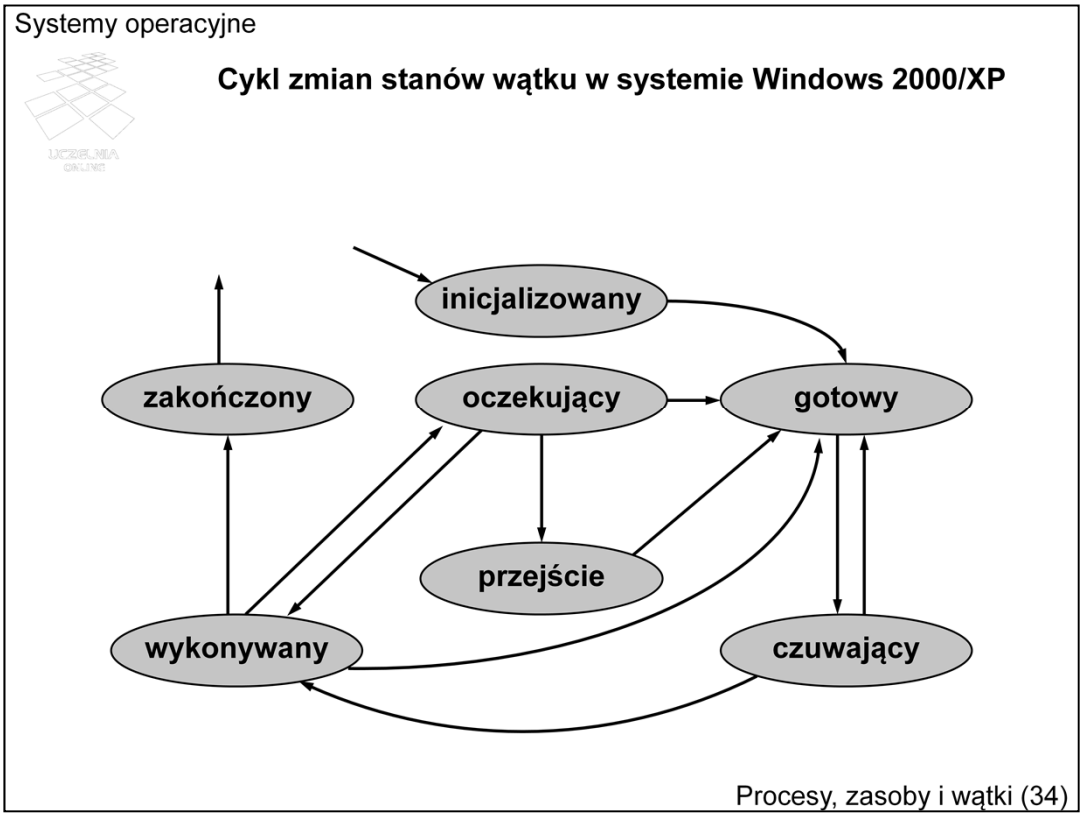
Podstawowe zasoby na potrzeby wykonania wątku (np. pamięć) przydzielone są procesowi. Są one zatem wspólne dla wszystkich wątków danego procesu. Najważniejszym zasobem przydzielanym wątkowi jest procesor. Wszelkie przetwarzanie i wynikająca stąd zmiana stanu procesu odbywa się w wątku. Struktury opisu wątku są analogiczne do struktur opisu procesu.

Systemy operacyjne

Stany wątku w systemie Windows 2000/XP

- Inicjalizowany (initialized, wartość 0) — stan wewnętrzny w trakcie tworzenia wątku,
- Gotowy (ready, wartość 1) — oczekuje na przydział procesora,
- Wykonywany (running, wartość 2),
- Czuwający (standby, wartość 3) — wybrany do wykonania jako następny,
- Zakończony (terminated, wartość 4),
- Oczekujący (waiting, wart. 5) — oczekuje na zdarzenie,
- Przejście (transition, wartość 6) — oczekuje na sprowadzenie swojego stosu jądra z pliku wymiany,
- Unknown (wart. 7)

Procesy, zasoby i wątki (33)



W systemie Windows zarówno stan gotowości, jak i czuwania odpowiada stanowi gotowości w odniesieniu do ogólnego schematu zmian stanów.