Wykład 3
Zarządzanie procesami. Procesy i wątki

Jarosław Koźlak

Procesy

Terminologia (niejednoznaczności historyczne):

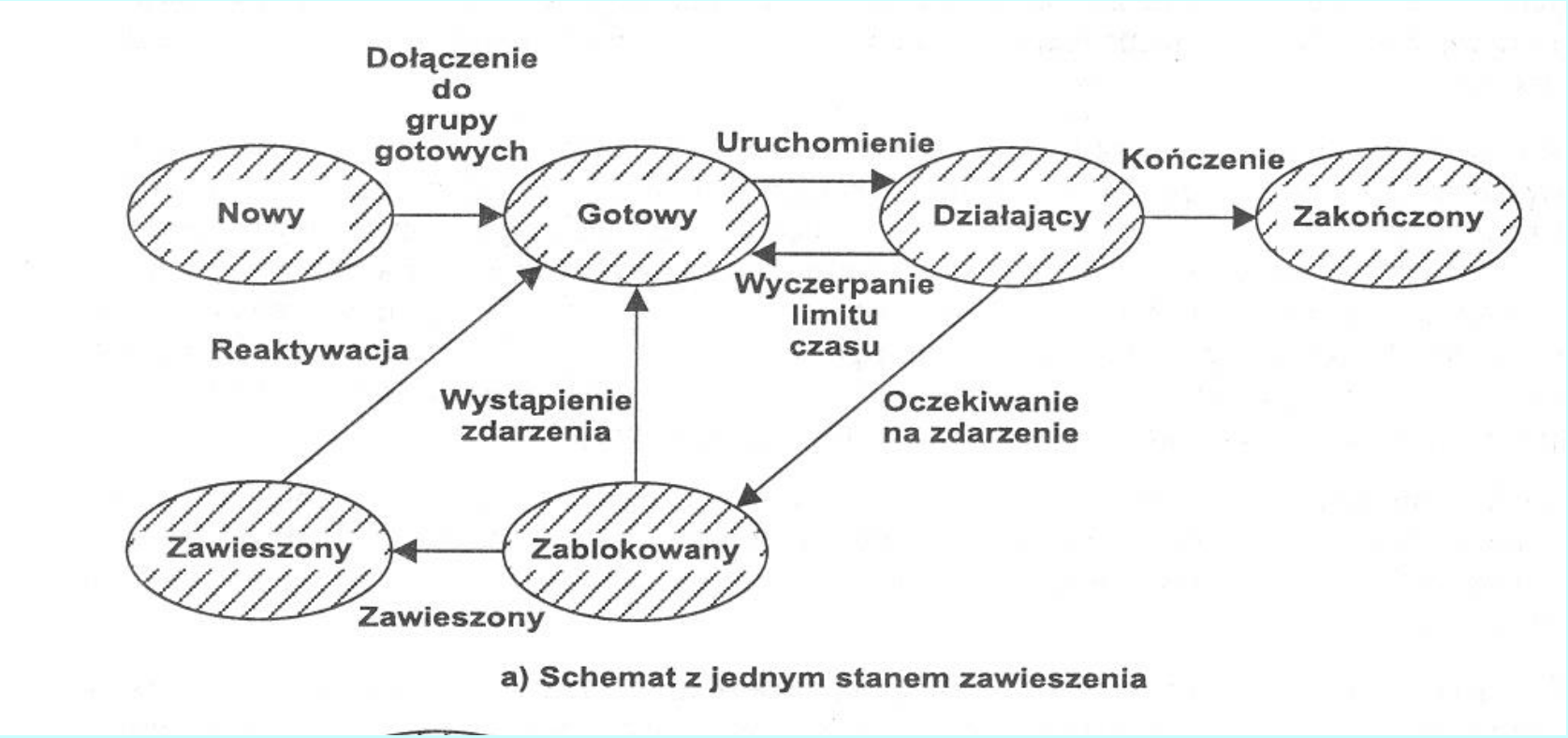
- **zadania (ang, jobs) = procesy**
- **programy użytkownika (ang. user programs) = prace (ang. tasks)**
- **Program** - obiekt pasywny **(**zawartość pliku na dysku)
- **Proces** - obiekt aktywny (zawiera licznik rozkazów określający następny rozkaz do wykonania i zbiór przydzielonych zasobów)
- **Proces --** wykonujący się program
- **Proces składa się z:**
  - kodu programu (sekcji tekstu, ang. text section)
  - odpowiedniego ustawienia licznika rozkazów (program counter)
  - zawartości rejestrów procesora
  - stosu procesu (ang. process stack) z danymi tymczasowymi (parametrami procedur, adresami powrotnymi, zmiennymi tymczasowymi)
  - sekcji danych (ang. data section) ze zmiennymi globalnymi
- **Brak odpowiedniości 1 program - 1 proces:**
  - wiele procesów może być uruchomionych w oparciu o jedną kopię programu
  - wykonywany proces może tworzyć procesy potomne

Stany procesu

Stany procesu 2

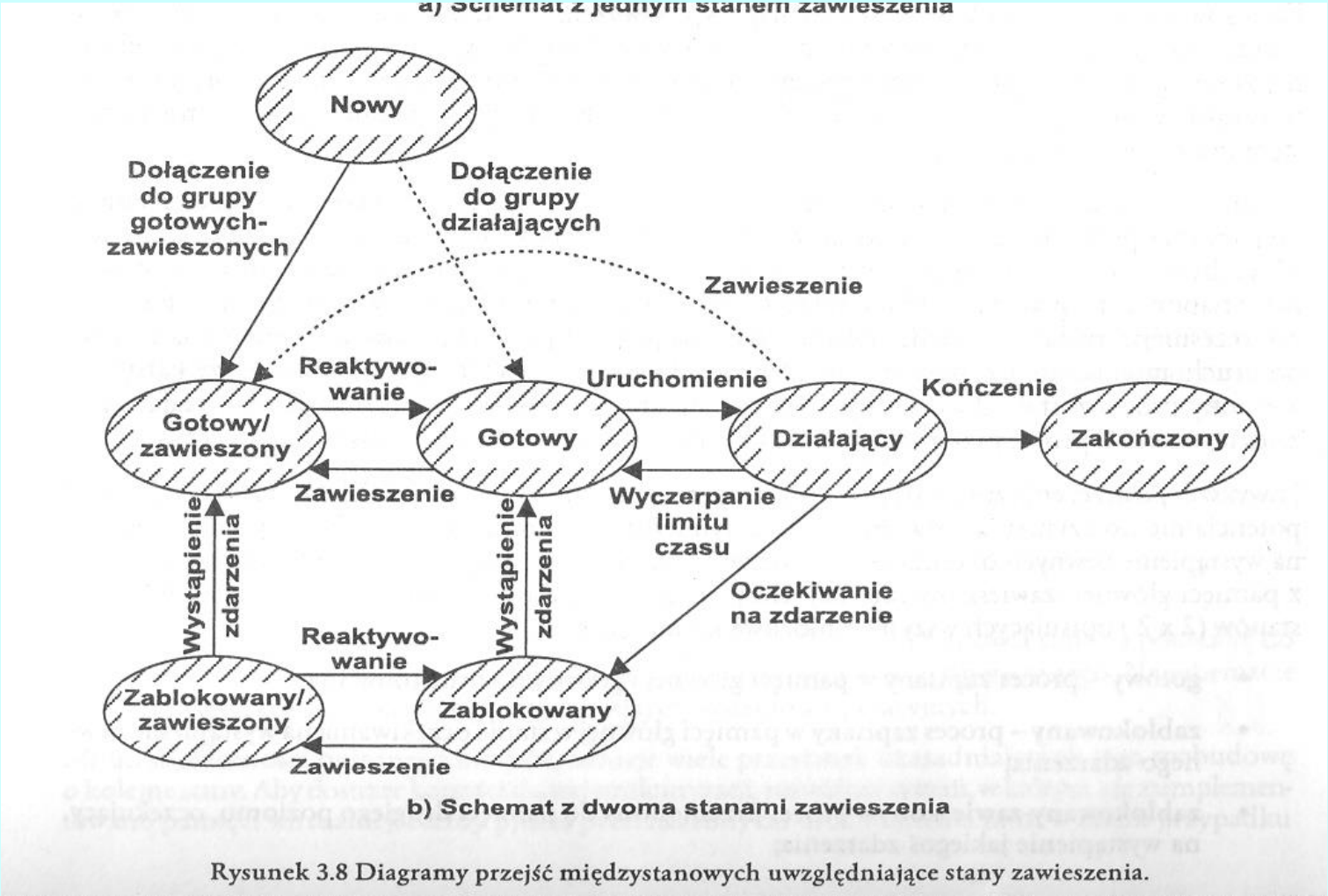
- **nowy** - proces został utworzony
- **aktywny** - są wykonywane instrukcje
- **czekający** - proces czeka na zakończenie jakiegoś zdarzenia (np. zakończenie operacji we/wy)
- **gotowy** - proces czeka na przydział procesora
- **zakończony** - proces zakończył działanie

Stany procesu: 1 stan zawieszenia



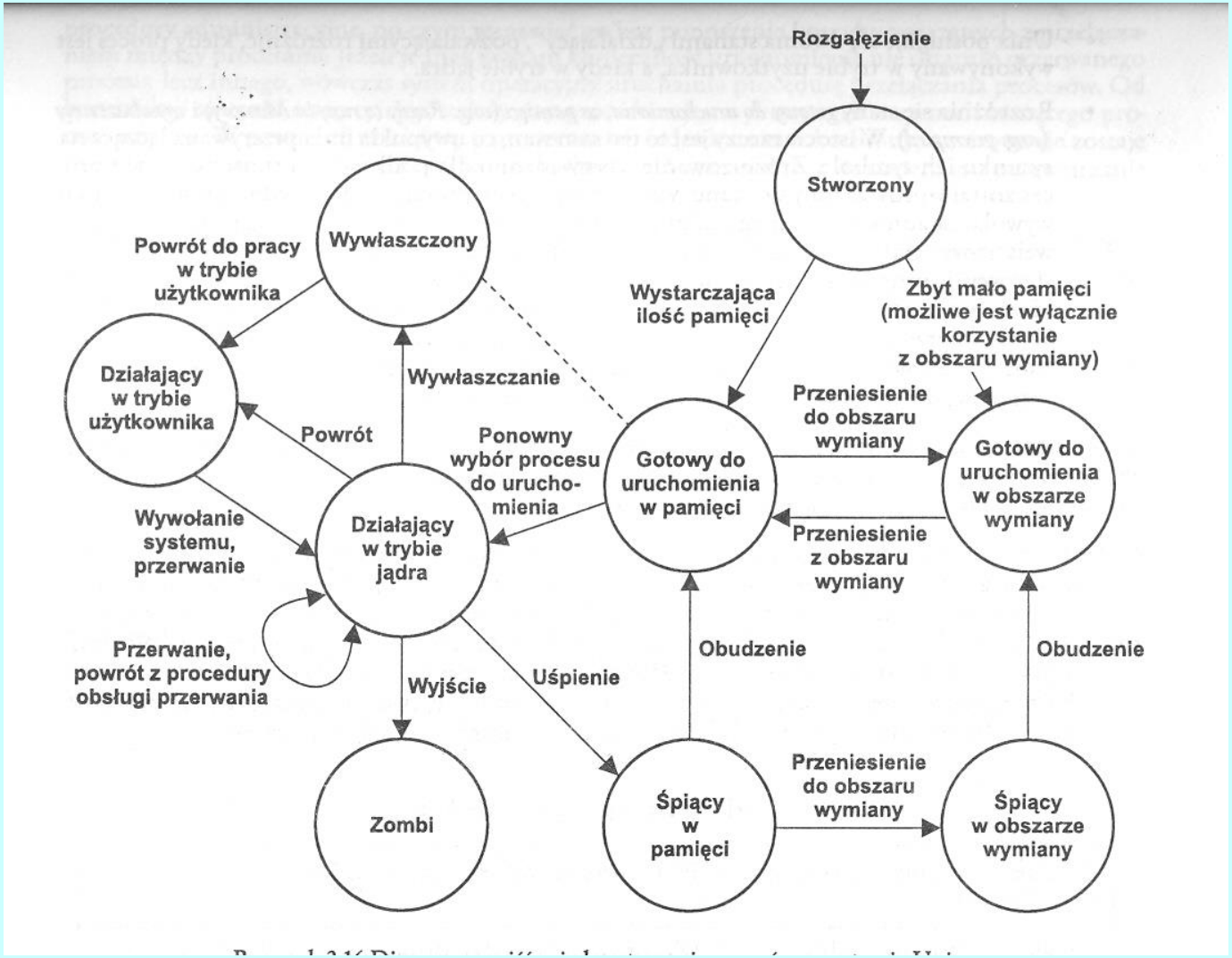
a) Schemat z jednym stanem zawieszenia

Stany procesu: 2 stany zawieszenia



Rysunek 3.8 Diagramy przejść międzystanowych uwzględniające stany zawieszenia.

Stany procesu: System Unix



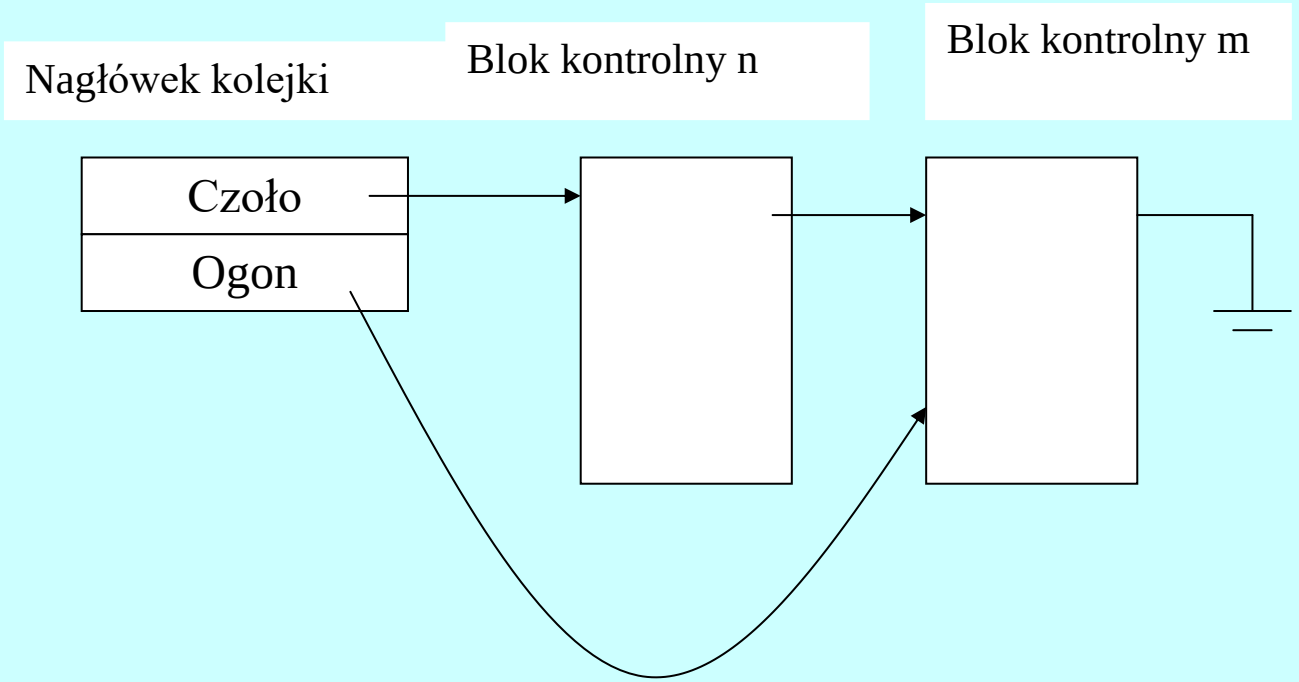
Blok kontrolny procesu

- Każdy proces jest reprezentowany w systemie operacyjnym przez blok kontrolny procesu (ang. process control block - PCB) .
- Inna nazwa bloku kontrolnego procesu - blok kontrolny zadania
- W skład bloku kontrolnego procesu wchodzą:
- **Stan procesu** : nowy, gotowy, aktywny, czekający
- **Licznik rozkazów:** z adresem następnego rozkazu do wykonania w procesie
- **Rejestry procesora:** zależne od architektury komputera np. akumulatory, rejestry indeksowe, wskaźniki stosu, rejestry ogólnego przeznaczenia, rejestry warunków
- **Informacje o planowaniu przydziału procesora:** priorytet procesu, wskaźniki do kolejek porządkujących zamówienia i inne
- **Informacje o zarządzaniu pamięcią:** zawartości rejestrów granicznych, tablice stron lub tablice segmentów
- **Informacje do rozliczeń:** ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numery procesów
- **Informacje o stanie wejścia-wyjścia:** informacje o urządzeniach we/wy przydzielonych do procesu, wykaz otwartych plików itd.

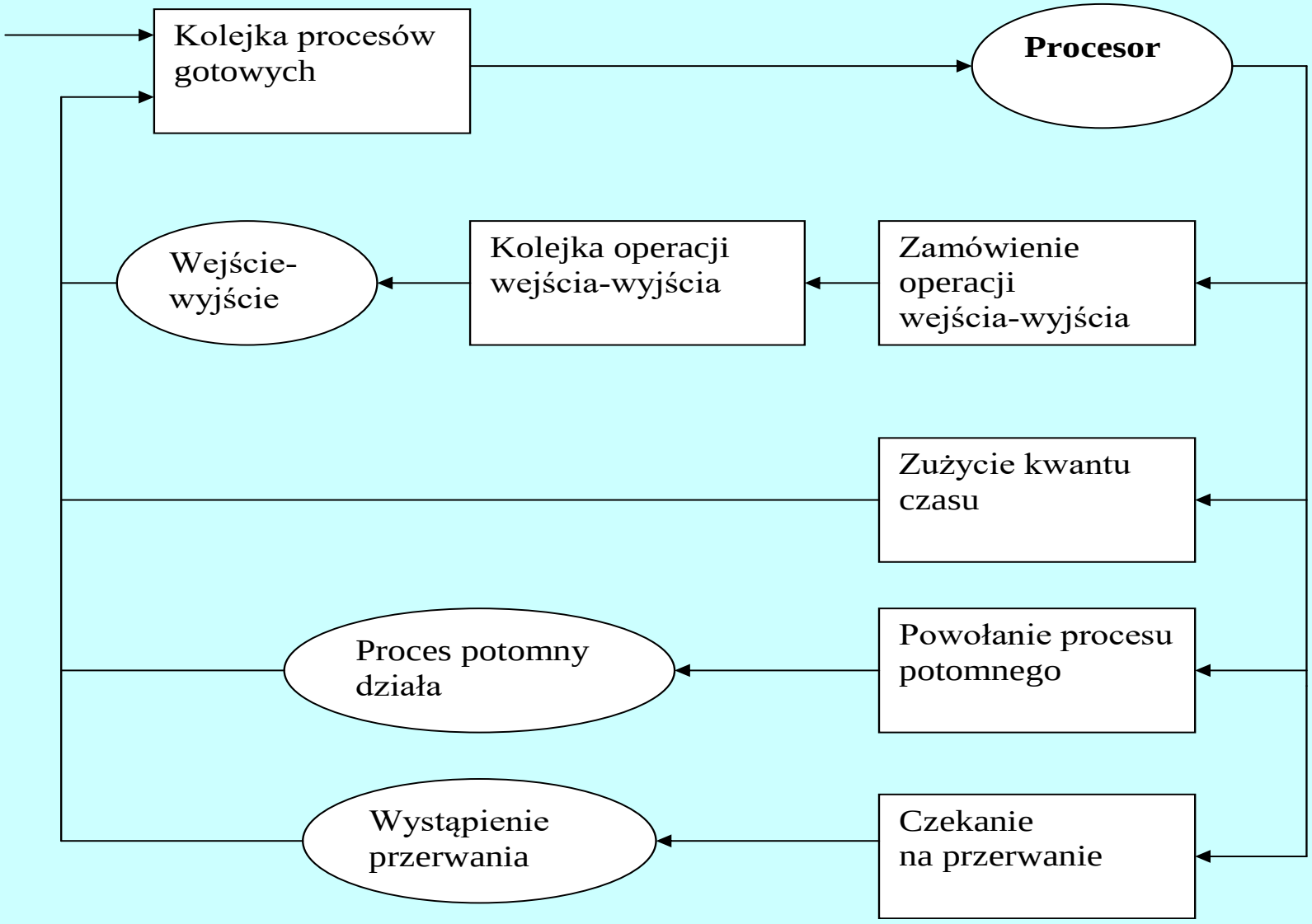
Planowanie procesów

- procesy w systemie są ulokowane w kolejkach zadań(ang. job queue)
- procesy oczekujące w pamięci głównej gotowe do działania są w kolejce procesów gotowych (ang. ready queue)
- procesy czekające na przydział konkretnego urządzenia -- w kolejkach do urządzeń (ang. device queue), każde urządzenie ma własną kolejkę
- Reprezentacja kolejek:
  - listy powiązane
  - nagłówek kolejki ma wskaźniki do pierwszego i ostatniego bloku kontrolnego procesu
  - każdy blok kontrolny ma pole wskazujący następną pozycję w kolejce procesów gotowych

Kolejka procesów



Kolejki w planowaniu procesów



Planiści

- **Planista** (program szeregujący, ang. scheduler) - proces systemowy, który odpowiada za wybór procesów z kolejek
- **Planista długoterminowy** (ang. long-term scheduler); inna nazwa - planista zadań (ang. job scheduler) - wybiera procesy z pamięci masowej i ładuje je do pamięci w celu wykonania; czas wykonywania rzędu minut (np. gdy proces opuszcza system)
- **Planista krótkoterminowy** (ang. short-time scheduler); planista przydziału procesora (ang. CPU scheduler) - wybiera jeden proces spośród procesów gotowych do wykonania i przydziela mu procesor; czas uruchamiania rzędu milisekund
- **Dwa rodzaje procesów:**
- procesy ograniczone przez we/wy - spędzają większość czasu na operacjach we/wy
- procesy ograniczone przez procesor - spędzają większość czasu na obliczeniach zajmując procesor
- **Zadanie planisty długoterminowego:** dobranie dobrej mieszanki procesów (ang. process mix) zawierającą procesy obydwu rodzajów
- **Może wystąpić dodatkowy planista średnioterminowy (ang. medium-term scheduler) -** odpowiada za usuwanie procesów z pamięci w celu zmniejszenia stopnia wieloprogramowości,
- takie postępowanie - wymiana (ang. swapping)

Przełączanie kontekstu

- przełączanie procesora dla innego procesu (koszt czasowy rzędu 1-1000us, wymaga:
- przechowania stanu starego procesu
- załadowania przechowanego stanu nowego procesu

Tworzenie procesów:

- użycie wywołania systemowego "utwórz proces"
- Proces tworzący nowy proces: proces macierzysty (ang. parent process)
- Proces tworzony przez proces macierzysty: proces potomny, potomek (ang. children)
- Powstający podproces może otrzymywać niezbędne zasoby:
- od systemu operacyjnego
- albo uzyskuje podzbiór zasobów posiadanych przez proces macierzysty
- Możliwe relacje procesu macierzystego i procesu potomka:
- proces macierzysty kontynuuje działanie współbieżnie ze swoimi potomkami
- proces macierzysty oczekuje na zakończenie działania niektórych lub wszystkich swoich procesów potomnych
- Przydział przestrzeni adresowej:
- proces potomny staje się kopią procesu macierzystego
- proces potomny otrzymuje nowy program

Tworzenie procesów. System Unix

Stworzenie procesu

- każdy proces ma jednoznaczny identyfikator procesu
- proces jest tworzony funkcją systemową **fork**
- nowy proces otrzymuje kopię przestrzeni adresowej procesu macierzystego
- oba procesy kontynuują działanie od instrukcji następującej po wywołaniu funkcji fork
- fork zwraca 0 nowemu procesowi, a identyfikator potomka - procesowi macierzystemu
- funkcja execve - zastąp. pamięci procesu przez inny program
- funcja wait umożliwia proc. macierz. czekanie na zakończenie działania potomka

Zakończenie procesu. Unix

- Proces kończy się, gdy wykona swą ostatnią instrukcję i wywołując funkcję exit poprosi SO, by go usunął
- Podczas zakończenia proces może przekazać dane do procesu macierzystego (poprzez funkcję wait w procesie macierzystym)
- Inne metody zakończenia procesu:
- poprzez wywołanie odpowiedniej funkcji systemowej można zakończyć działanie innego procesu (zazwyczaj przodek lub proces użytkownika o odp. uprawnieniach)

Wątki

Wątki

wątek (ang. thread)

Wątek - podstawowa jednostka wykorzystania procesora, obejmuje:

- licznik rozkazów
- zbiór rejestrów
- obszar stosu

Wiele wątków może współdzielić:

- sekcję kodu
- sekcję danych
- zasoby SO - pliki, sygnały itd.

- Tradycyjny proces - ciężki proces (ang. heavyweight process): równoważny zadaniu z jednym wątkiem
- Przełączanie wątków jest znacznie mniej kosztowne niż przełączanie kontekstu procesów ciężkich

Wątki 2

- **Współpracujące wątki**
  - operują na wspólnych danych
  - są w obrębie jednego procesu
  - nie ma mechanizmów ochrony, należą do tego samego użytkownika

Ograniczenia procesów

- są aplikacje, które mają do wykonania wiele niezależnych zadań, które mogą być realizowane współbieżnie, ale muszą współdzielić przestrzeń adresową i inne zasoby:
  - zarządcy baz danych pełniący rolę serwera w modelu klient-serwer
  - monitory przetwarzające transakcje
  - programy obsługi protokołów sieciowych ze środkowych i górnych warstw sieciowych
- tradycyjne procesy mogą w danej chwili używać tylko jednego procesora, dlatego nie mogą wykorzystać zalet architektur wieloprocesorowych

Rodzaje wątków

Pojęcia

- **proces** jest złożoną całością, którą można podzielić na dwie części: zbiór wątków i zestaw zasobów
- **wątek** jest obiektem dynamicznym, reprezentującym punkt sterowania wewnątrz procesu i wykonującym pewien ciąg rozkazów
- **zasoby procesu:** przestrzeń adresowa, otwarte pliki, informacje identyfikujące użytkownika, miejsce dostępne na dysku itd.
- **zasoby są współdzielone przez wszystkie wątki w procesie**
- **każdy wątek ma obiekty związane tylko z nim, jak:** licznik rozkazów, stos, kontekst rejestrów
- **właściciel zasobów** - proces

Poziomy realizacji wątków

- Wątki poziomu użytkownika
- Procesy lekkie
- Wątki poziomu jądra

Wątki poziomu użytkownika

- abstrakcja wątków jest udostępniana na poziomie użytkownika, jądro nie wie o ich istnieniu
- realizacja za pomocą pakietów bibliotecznych jak pthreads (zgodny ze standardem POSIX)
- synchronizacja, szeregowanie i zarządzanie takimi wątkami odbywa się bez udziału jądra – jest bardzo szybkie
- kontekst wątku z poziomu użytkownika jest zapamiętywany i odtwarzany bez udziału jądra
- wątek użytkownika ma własny stos, przestrzeń do zapisania kontekstu rejestrów z poziomu użytkownika a także informacje o stanie wątku jak maska sygnałów

Szeregowanie:

- jądro szereguje procesy lub procesy lekkie, w obrębie których wykonują się wątki użytkownika
- proces do szeregowania swoich wątków stosuje funkcje biblioteczne
- jeśli jest wywłaszczany proces, lub proces lekki, są wywłaszczane także jego wątki
- jeśli watek użytkownika wykona blokującą funkcję systemowa, jest wstrzymywany proces lekki, w którym wykonuje się wątek - jeśli w obrębie procesu jest tylko jeden proces lekki, to wszystkie jego wątki są wstrzymane

Wątki poziomu użytkownika: Zalety

- umożliwiają bardziej naturalny sposób zapisu wielu programów, takich jak systemy okienkowe
- wydajność - nie zużywają zasobów jądra, jeśli nie są związane z żadnym procesem lekkim
- wydajność wynika z zaimplementowania na poziomie użytkownika i z nie stosowania funkcji systemowych

Wątki użytkownika: Ograniczenia

- ograniczenia wynikają głównie z braku przepływu informacji między jądrem i biblioteką wątków
  - jądro nie ma informacji o wątkach użytkownika, nie może używać swoich mechanizmów ochrony do ich ochrony przed niedozwolonym dostępem ze strony innych wątków
  - szeregowanie przez jądro i bibliotekę, przy czym żadne z nich nie posiada wiedzy o czynnościach drugiego
- każdy proces działa we własnej przestrzeni adresowej, wątki użytkownika nie mają takiej ochrony – biblioteka wątków musi zapewnić mechanizmy synchronizacji
- niemożność jednoczesnego wykonywania wątków, w przypadku maszyny wieloprocesorowej

Wątki jądra

- wątek jądra nie musi być związany z procesem użytkownika
- jądro tworzy go i usuwa wewnętrznie w miarę potrzeb
- wątek odpowiada za wykonanie określonej czynności, współdzieli tekst jądra i jego dane globalne, jest niezależnie szeregowany i wykorzystuje standardowe mechanizmy jądra, takie jak *sleep()* czy *wakeup()*
- wątki potrzebują jedynie następujących zasobów:
  - własnego stosu,
  - przestrzeni do przechowywania kontekstu rejestrów w czasie, gdy wątek nie jest wykonywany
- tworzenie i stosowanie wątków jądra nie jest kosztowne
  - przełączanie kontekstu między wątkami jądra jest szybkie, ponieważ nie trzeba zmieniać odwzorowań pamięci
- zastosowania wątków jądra
  - wątki jądra są przydatne do wykonywania pewnych operacji takich, jak asynchroniczne wejście-wyjście
    - zamiast udostępniać osobne operacje asynchronicznego we/wy, jadro może realizować je, tworząc odrębny wątek do wykonania każdego zlecenia –
  - wątki poziomu jądra można wykorzystywać też do obsługi przerwań
- wątki poziomu jądra nie są nowym pomysłem: procesy systemowe jak demon stronicujący (pagedaemon) pełnią w tradycyjnych jądrach uniksowych te funkcje co wątki jądra

Procesy lekkie

- proces lekki - wspierany przez jądro wątek poziomu użytkownika
- wysokopoziomowe pojęcie abstrakcyjne oparte na wątkach jądra - system udostępniający procesy lekkie, musi także udostępniać wątki jądra
- każdy proces lekki jest związany z wątkiem jądra, ale niektóre wątki jądra są przeznaczone do realizacji pewnych zadań systemowych, wtedy nie przypisuje się im żadnych lekkich procesów
- w każdym procesie może być kilka procesów lekkich, każdy wspierany przez oddzielny wątek jądra, współdzielą one przestrzeń adresową i inne zasoby procesu
- procesy lekkie są niezależnie szeregowane przez systemowego planistę,
- procesy lekkie mogą wywoływać funkcje systemowe, które powodują wstrzymanie w oczekiwaniu na we/wy lub zasób
- proces działający w systemie wieloprocesorowym może uzyskać rzeczywistą równoległość wykonania, każdy proces lekki może być uruchamiany na oddzielnym procesorze
- wielowątkowe procesy są przydatne wtedy, gdy każdy wątek jest w miarę niezależny i rzadko porozumiewa się z innymi wątkami

Procesy lekkie: Ograniczenia (1)

Ograniczenia procesów lekkich:

- większość operacji na nich (tworzenie, usuwanie, synchronizacja) wymaga użycia funkcji systemowych
- funkcje systemowe są kosztowne - każde wywołanie wymaga dwóch przełączeń trybu (przy wywołaniu i zakończeniu)
- każda zmiana trybu powoduje przejście przez granice ochrony - jądro musi przekopiować argumenty funkcji systemowej z przestrzeniu użytkownika do przestrzeni jądra i sprawdzić ich poprawność
- po zakończeniu wykonania funkcji systemowej jądro musi skopiować dane z powrotem do przestrzeni użytkownika

Procesy lekkie: Ograniczenia (2)

Ograniczenia procesów lekkich 2:

- jeśli proces lekki często odwołuje się do współdzielonych danych, narzut związany z synchronizacją może niweczyć wszelkie zyski wydajnościowe
- większość systemów wieloprocesorowych udostępnia mechanizmy blokad, które można nakładać na poziomie użytkownika na niezablokowany zasób
- jeśli wątek chce uzyskać dostęp do zasobu, który jest w danej chwili niedostępny, to może:
  - aktywnie czekać (jeśli zasób jest zajęty na krótko) - co się odbywa bez udziału jądra
  - wstrzymać swoje działanie (w innych przypadkach) – wymaga zaangażowania jądra i jest operacją kosztowną czasowo
- każdy proces lekki zużywa znaczące zasoby jądra, w tym pamięć fizyczną potrzebną na stos jądra
- system nie może wspierać dużej liczby procesów lekkich
- procesy lekkie nie nadają się do programów wymagających dużej ilości wątków, które są często tworzone i niszczone
- użytkownik może zmonopolizować procesor, tworząc dużo procesów lekkich

Modele wielowątkowości

Szeregowanie wątków

Muszą istnieć dwa *schedulery:*

- jeden dla procesów/wątków systemowych - KS (ang. *kernel scheduler*) ,
- drugi dla wątków użytkownika - UTS (ang. *user threads scheduler*).

Modele wielowątkowości - 1:N

- KS widzi 1 byt/proces
- UTS widzi *N* wątków.
- kernel nie ma pojęcia o istnieniu wątków w aplikacji, natomiast zarządzanie wątkami opiera się o biblioteki działające na poziomie użytkownika
- *KS i UTS* nie muszą się ze sobą komunikować

Modele wielowątkowości - 1 : 1

- wszystkie procesy i wątki z punktu widzenia KS są jednorodnymi bytami;
- UTS nie jest potrzebny, gdyż cała praca jest wykonywana przez KS.
- konieczna jest organizacja obsługi wątków za pomocą odpowiedniej biblioteki (tworzenie, synchronizacja, itp.), ale planowanie zadań jest wykonywane tylko i wyłącznie na poziomie jądra

Modele wielowątkowości -M : N

- dla jednej aplikacji wielowątkowej KS widzi *M* bytów do zarządzania,
- na poziomie użytkownika (i UTS) istnieje *N* wątków (*M* < *N*)
- koncepcja teoretycznie najbardziej wszechstronna i daje największe możliwości
- najtrudniejsza implementacyjnie, wymagana jest komunikacji pomiędzy UTS a KS, aby przemapować M wątków użytkownika na N wątków jądra