System operacyjny - ma za zadanie ukryć złożoność operacji wykonywanych na sprzęcie i dostarczyć użytkownikowi wygodniejszy zbiór instrukcji, jest częścią oprogramowania które może być uruchamiane w trybie jądra bądź użytkownika.

System operacyjny jako:

Maszyna rozszerzona - Architektura komputera jest trudna w obsłudze na poziomie języka maszynowego(szczególne operacje I/O), SO ma za zadanie ukryć złożoność wykonywanych operacji i dostarczyć użytkownikowi wygodny interfejs.

SO dostracza Maszynę rozszerzoną/wirtualną która jest łatwiejsza do zaprogramowania

Zarządca zasobów – W systemie mamy różne rodzaje zasobów (napędy, taśmy magnetyczne, procesory, dyski, terminale, pamięc) SO ma za zadanie udostępniać te zasoby użytkownikowi, kontrolować do nich dostęp i zapobiegać konfliktom między programami i użytkownikami.

Program sterujący – SO nadzoruje pracę programów użytkownika, przeciwdziała błędom i zapobiega niewłaściwemu użyciu komputera.

Cele SO – Wydajność i wygoda , często te 2 cele są ze sobą w konflikcie, kiedyś stawiono wydajność ponad wygodę, teraz na odwrót.

Systemy wsadowe :

Cecha charakterystyczna : Brak bezpośredniego nadzoru ze strony użytkownika podczas wykonywania jego zadania.

Struktura systemu komputerowego :

Wielkie fizyczne maszyny obsługiwane przez konsole.
Jako urządzenia wejścia **Czytniki kart** i **Przewijaki taśm**.
Jako urządzenia wyjścia **Perforatory kart**, **Przewijaki taśm**, **Drukarki wierszowe**.
Użytkownik przygotowywał zadanie (**program,dane,informacje sterujące**) na kartach i przekazywał je operatorowi.
Oczekiwanie na wyniki ~ **godziny,dni**

Struktura systemu operacyjnego :

- SO na stałe rezyduje w pamięci operacyjnej
- Obowiązkiem so jest przekazywanie sterowania od jednego zadania do kolejnego
- W celu przyśpieszenia przetwarzania zadania o podobnych wymaganiach są sortowane w **batch'e** (wsady), sortuje je operator.
- Jednostka centralna często stoi bezczynna ze względu na wolno działające mechaniczne urządzenia WE/WY

Modyfikacje :

- Wprowadzono dyski, które umożliwiły zastosowanie spoolingu, który pozwolił na jednoczesne wykonywanie obliczeń i operacji WE/WY.
- Dyski pozwoliły na czytanie z maksymalnym wyprzedzeniem z urzadzeń WE i przechowywanie plików wyjściowych aż urządzenia WY będą gotowe je przyjąć.

Wady systemów wsadowych:

- Użytkownik nie może ingerować w zadanie podczas jego wykonywania się a więc musi przygotować karty sterujące na wszystkie możliwe zdarzenia.
- Następne kroki wykonywania zadania mogą zależeć od poprzednich np. kompilacja, uruchomienie
- Testowanie – programista nie może na bierząco zmieniać programu w celu obserwacji jego zachowań.

System z podziałem czasu jest w nim wielu użytkowników i każdy z nich otrzymuje dostęp do procesora przez małą porcję czasu. Każdy użytkownik ma co najmniej jeden proces w systemie.

System interakcyjny umożliwia bezpośredni dialog użytkownika z systemem, użytkownik ma bezpośredni dostęp do plików. Zadanie interakcyjne składa się z wielu krótkich zadań a ich rezultaty w przeciwieństwie do zadań w systemach wsadowych są nieprzywidywalne.

RTOS (Real Time Operating System) Systemy czasu rzeczywistego – są stosowane tam gdzie istnieje surowe wymaganie na czas wykonania operacji lub przepływu danych.

Np. przy nadzorowaniu eksperymentów lub obrazowaniu badań medycznych sterownik pobiera dane z czujników analizuje je i na ich podstawie reguluje działanie kontrolowanego obiektu.

Wymaganie czasu rzeczywistego – przetwarzanie danych musi się ukończyć przed upływem określonego czasu.

Hard-RTOS :

- Gwarantuje terminowe wykonanie zadań krytycznych (systemy o specjalnej konstrukcji dane w pamięci o krótkim czasie dostępu lub ROM z reguły brak pamięci wirtualnej)
- Znany czas najdłuższej odpowiedzi i wiadomo, że nie zostanie on przekroczony
- Żaden z istniejących uniwersalnych SO nie umożliwia działania w czasie rzeczywistym

Soft-RTOS :

- Krytyczne zadania otrzymują pierwszeństwo przed innymi zadaniami i zachowuje je do czasu aż zostanie zakończone
- Stara odpowiedzieć się najszybciej jak to możliwe lecz nie znamy najdłuższego czasu odpowiedzi
- Unix może spełniać te wymagania

Budowa SO z jądrem warstwowym

Ułatwia modyfikacje systemu poprzez modularyzację

Sposób modularyzacji:

1. Podział na warstwy
2. Najwyższa warstwa to interfejs użytkownika a najniższa to hardware
3. Każda warstwa może wywoływać operacje dotyczące warstw niższych
4. Każda nowa warstwa jest zbudowana ponad poprzednimi
5. Każda warstwa zawiera dane i procedury , z których mogą korzystać warstwy wyższe
6. Podejście warstwowe ułatwia wyszukiwanie i weryfikacje błędów

Wady :

Mniejsza wydajność , obecnie ogranicza się ilość warstw

Przykład

0. Hardware
1. Planowanie przydziału procesora
2. Zarządzanie pamięcią
3. Program obsługi kontroli operatora
4. Buforowanie urządzeń I/O
5. Interfejs użytkownika

Mikrojądro Wady i Zalety :

+ **Jednolite Interfejsy** – procesy nie muszą odróżniać usług jądra od użytkownika gdyż wszystkie są udostępniane za pomocą przekazywania komunikatów

+ **Rozszerzalność** – możemy dodawać nowe i obsługiwać już istniejące zróżnicowane usługi, dodanie nowej funkcji wymaga jedynie stworzenia nowego serwera lub modyfikacji już istniejącego nie musimy generować nowego jądra

+ **Elastyczność** – możemy zarówno dodawać nowe usługi i rozbudowywać jądro jak i zarówno możemu usuwać już istniejące aby uzyskać mniejszą i wydajniejszą realizację

+ **Przenośność między platformami** - łatwość przenoszenia na inną architekturę gdyż trzeba zmieniać nieliczne dobrze sprecyzowane moduły

+ **Stabilność** – niewielkie mikrojądro może być dobrze przetestowane

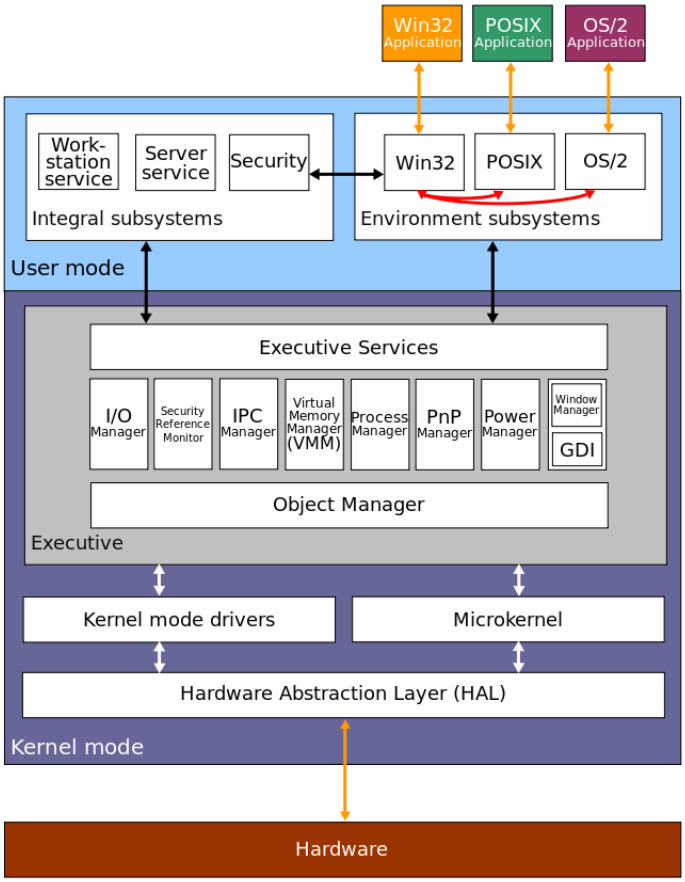
+ **Obsługa systemów obiektowych** – rozwiązania obiektowe wymuszają większą dyscypline podczas projektowania mikrojądra oraz modularnych rozszerzeń

+ **Obsługa systemów rozproszonych** – komunikaty skierowane do serwera mogą dotyczyć również serwerów zlokalizowanych na innych komputerach

- **Komunikacja** – jądro jest intensywnie wykorzystywane naprzykład komunikacja klient serwer wygląda tak : Klient > Jądro > Serwer > Jądro > Klient

- **Synchronizacja** – w przypadku synchronizacji działań dokonywanych przez serwery poziomu użytkownika

Windows NT



Hal – hardware abstraction layer warstwa abstrakcji która chowa zawiłości różnych architektur sprzętowych pod ujednoliconym api, mogą z niej korzystać zarówno jądro jak i sterowniki

Jądro odpowiada za :

- Szeregowanie zadań
- Harmonogram realizacji wątków
- Obsługa sytuacji wyjątkowych
- Synchronizacja pracy wieloprocesorowej

Kernel-mode drivers : niskopoziomowe sterowniki działające w trybie jądra

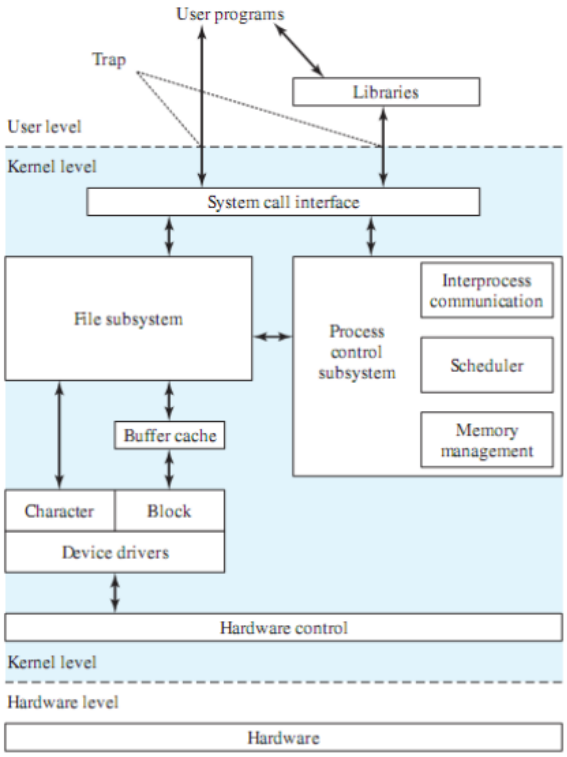
Executive Komponenty :

1. PnP Manager – odpowiada za wykrywanie podłączanych urządzeń i automatyczne ładowanie odpowiednich sterowników
2. LPC – mechanizm IPC
3. Object Manager – tak jak w linuxie zasoby reprezentują pliki tak tutaj obiekty
4. Memory Manager – zarządza pamięcią wirtualną ,ochroną pamięci i stronnicowaniem
5. Process Structure – obsługuje tworzenie i usuwanie procesów i wątków
6. Cache controller – wspólne miejsce do obsługi sterowników, I/O i pamięci podręcznej
7. GDI – framework graficzny
8. Configuration Manager – implementa rejestru

9. Power Manager – obsługuje zdarzenia związane z zasilaniem (usypia CPU, kontroluje sterowniki za pomocą specjalnych przerwań
10. I/O Manager – implementuje niezależnie od sprzętu całą obsługę I/O

User-space
Podsystemy środowiskowe - jedną z ciekawszych cech Windows NT jest natywna zdolność do udawania przeróżnych środowisk.

Jądro Linuxa



1. **Hardware** połączony z **Hardware control**
2. **Device Driver** – sterowniki do urządzeń zewnętrznych
3. **Character/Block** – rodzaj urządzenia
4. **Buffer cache** – zachowuje ostatnio pobrane dane
5. **File subsystem** - zajmuje się dostępem do pamięci
6. **Process control subsystem** – obsługuje komunikacje międzyprocesową, szereguje procesy i przydziela im pamięć
7. **System Call Interface** – udostępnia funkcje systemowe dla programistów
8. **Libraries** - niskopoziomowy kod dotyczący m. in. operacji na plikach, komunikacji międzyprocesowej, tworzenia procesów
9. **Traps/Signals** – sygnały pozwalają na komunikację asynchroniczną a pułapki na sprzątanie przed zakończeniem programu przez sygnał

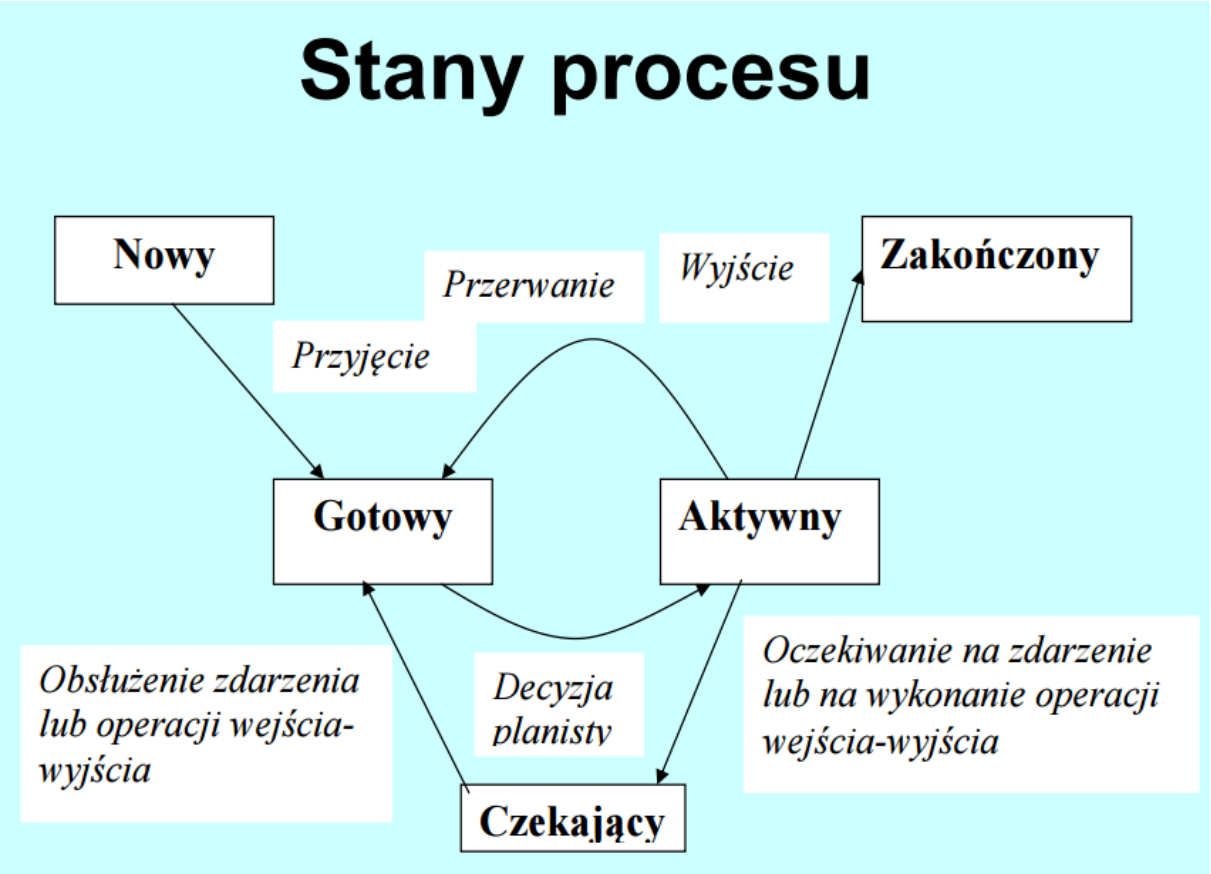
Cechy charakterystyczne jądra Linuxa

- Linux wspiera dynamiczne ładowanie modułów jądra
- Linux wspiera symetryczną wieloprocesorowość
- Jądro Linuxa jest wywłaszczalne
- Linux ma specyficzne podejście do wielowątkowości.
  Nie ma rozróżnienia wątków i zwykłych procesów.
  Dla jądra wszystkie procesy są takie same niektóre tylko dzielą zasoby.

System V Release 4 nowości

- Możliwość wykonywania rożnego rodzaju plików wykonywalnych
- Osobne sposoby obsługi procesów z podziałem czasu i procesów real-time
- Wprowadzenie vnodów co pozwoliło na obsługe wielu systemów plików
- Wprowadzono streamy

W2



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

Wątek budowa :

- Licznik rozkazów
- Zbiór rejestrów
- Obszar stosu

Wiele wątków może **współdzielić** :

- Sekcję kodu
- Sekcję danych
- Zasoby SO

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

Wątki poziomu użytkownika :

- Abstrakcja wątków jest udostępniania na poziomie użytkowniki jądro nie wie o ich istnieniu
- A więc synchronizacja szeregowanie i zarządzanie takimi wątkami odbywa się bez udziału jądra – jest bardzo szybkie
- Kontekst wątku poziomu użytkownika jest zapamiętywany i odtwarzany bez udziału jądra
- Realizowane za pomocą pakietów bibliotecznych jak pthreads
- Wątek ma własny :
  - stos,
  - przestrzeń do zapamiętywania kontekstu rejestrów z poziomu użytwkonika,
  - informacje o stanie wątki (np. maska sygnałów)

Szeregowanie wątków poziomu użytkownika :

- Jądro szereguje procesy lekkie w obrębie których wykonują się wątki poziomu użytkownika
- Proces do szeregowania swoich wątków używa funkcji bilbiotecznych
- Jeśli wywłaszczany jest proces lub proces lekki są wywłaszczane również wszystkie jego wątki
- Jeśli wątek wykona blokującą funkcję systemową to wstrzymywany jest proces lekki w obrębie którego wątek jest wykonywany

Zalety :

1. Bardziej naturalny sposób zapisu wielu programów
2. Wydajność (jeśli nie jest związany z procesem lekkim bo nie jest wtedy wspierany przez jądro więc nie zużywa zasobów jądra) wynika ona z implementacji na poziomie użytkownika i nie stosowania funkcji systemowych

Ograniczenia – wynikają głównie z braku przepływu informacji pomiędzy jądrem i biblioteka wątków

1. Jądro nie ma o nich informacji więc nie może używać swoich mechanizmów ochrony przed niedozwolonym dostępem ze strony innych wątków
2. Są szeregowane przez jądro i bibliotekę przy czym żadne nie ma pojęcia o poczynaniach drugiego
3. Wątki w przeciwieństwie do procesów działają we wspólnej przestrzeni adresowej więc biblioteka musi zapewnić mechanizmy synchronizacji
4. Nie możność jednoczesnego wykonywania wątków w przypadku maszyny wieloprocesorowej

Wątki jądra :

- Wątek jądra nie musi być związany z procesem użytkownika
- Jądro tworzy je i usuwa wewnętrznie w miarę potrzeb
- Współdzielą tekst i dane globalne jądra, są niezależnie szeregowane
- Potrzebują tylko :
  - stosu
  - przetrzeni do zapamiętywania kontekstu rejestrów
- Tworzenie i stosowanie nie jest kosztowne
- Stosowane przy:
  - asynchronicznym I/O
  - obsłudze przerwań

Proces lekki to wspierany przez jądro wątek poziomu użytkownika

- Każdy proces lekki jest związany z wątkiem jądra
- W każdym procesie może być kilka procesów lekkich, które współdzielą przestrzeń adresową i inne zasoby systemu (każdy wspierany przez oddzielny wątek jądra)
- Są niezależnie szeregowane przez systemowego planistę
- Proces działający w systemie wieloprocesorowym może uzyskać rzeczywistą równoległość wykonywania (każdy proces lekki może być uruchamiany na osobnym procesorze)

Ograniczenia procesów lekkich :

1. Większość operacji na nich wymaga użycia funkcji systemowych, które są kosztowne
2. Synchronizacja przy częstym odwoływaniu się do zasobów dzielonych może zniszczyć wszystkie korzyści wydajnościowe
3. Każdy proces lekki zużywa znaczące zasoby jądra w tym pamięć fizyczną potrzebną na stos jądra
4. System nie może wspierać dużej ilości procesów lekkich
5. Procesy lekkie nie nadają się do programów, które wymagają dużej ilośći wątków które są często tworzone i niszczone
6. Tworząc dużo procesów lekkich użytkownik może zmonopolizować procesor

Modele Wielowątkowośći

KS – Kernel Scheduler

UTS – User Threads Scheduler

Modele:

1:N - KS widzi 1 proces, UTS widzi N wątków, zarządzanie wątkami opiera się o biblioteki działające na poziomie użytkownika, nie wymaga komunikacji KS – UTS

1:1 – Wszystkie procesy i wątki z punktu widzenia KS są tym samym,
UTS nie jest potrzebny bo wszystko wykonuje KS, konieczna jest organizacja obsługi wątków za pomocą odpowiedniej bilbioteki (tworzenie, synchronizacja itp.)
planowanie zadań wykonuje tylko KS

M:N (M < N) – KS widzi M procesów lekkich , UTS widzi N wątków

PROBLEM 1 PRZY M:N – wymagana jest komunikacja między UTS a KS
przykład: UTS przełącza na bardziej znaczący wątek a w tym samym czasie KS przełącza na inny wątek jądra nie dając bardziej znaczącemu wątkowi szansy na wykonanie się

PROBLEM 2 PRZY M:N – jeżeli damy za mało wątków jądra nie wykorzystamy w pełni wielowątkowości, jeżeli za dużo będziemy tracili dużo czasu na przełączanie kontekstów

Planowanie:

Decyzje o przydziale procesora zapadają w następujących sytuacjach

Proces przeszedł ze stanu

1. **aktywność -> czekanie** (np. oczekując na zdarzenie lub operacje WE/WY)

2. **aktywność -> gotowość** (np. wskutek wystąpienia przerwania)

3. **czekanie -> gotowość** (np. po zakończeniu operacji we/wy)

4. **proces kończy działanie**

Planowowanie niewywłaszczeniowe 1 i 4

Planowanie wywłaszczeniowe 1,2,3,4 (kosztowniejsze)

Ekspedytor – przekazuje procesor do dyspozycji procesu wybranego przez planistę krótkoterminowego

Zadania Ekspedytora :

1. Przełączanie kontekstu
2. Przełączanie do trybu użytkownika
3. Wykonanie skoku do odpowiedniej komórki w programie w celu wznowienia działania

Kryteria planowania:

1. **Wykorzystanie procesora** – procesor powinien być zajęty pracą
2. **Przepustowość** – liczba procesów kończonych w jednostce czasu
3. **Czas cyklu przetwarzania** – czas od nadejścia procesu do systemu do zakończenia jego przetwarzania
4. **Czas oczekiwania** – suma okresów oczekiwania jakie proces musi odczekać w kolejce procesów gotowych do wykonania
5. **Czas odpowiedzi** – czas od wysłania żadania do pierwszej odpowiedzi

Algorytmy planowania:

(Nw) – niewywłaszczający , **W** - wywłaszczający

1. **FCFS** – **F**irst **C**ome **F**irst **S**erved (**Nw**)
   Proces który pierwszy zamówi procesor pierwszy go otrzymuje
   Wada: średni czas oczekiwania bywa b. długi
2. **SJF** – **S**hortest **J**ob **F**irst (**Nw** lub **W**)
   Daje minimalny średni czas oczekiwania dla danego zbioru procesów
   Wada: trudno określić długość następnego zamówienia na przydział procesora
3. **Planowanie Priorytetowe** (**Nw** lub **W**)
   Każdemu procesowi przydziela się priorytet a procesor przydziela się zadaniu z najwyższym priorytetem w razie równych priorytetów FCFS
   Wada : głodzenia procesy z niskim priorytetem czekają w nieskończoność na czas procesora
   Rozwiązanie : zwiększanie priorytetów długo oczekujących procesów
4. **RR** – **R**ound **R**obin (**W**)
   Planowanie rotacyjne podobne do FCFS, dodano wywłaszczanie.
   Jest to kolejka procesow gotowych do wykonania. Ustalamy jednostkę czasu i każdy wykonuje się przez ten czas jednostka czasu nie powinna być za długa bo wtedy robi nam się FCFS ani za krótka bo tracimy czas na przełączanie kontekstów
   Wada: Dość wysoki Średni czas oczekiwania
5. **Guaranteed Scheduling** – Planowanie Gwarantowane
   Mamy n użytkowników w systemie to każdy użytkownik dostaje 1/n czasu procesora
   - czas przysługujący każdemu użytkownikowi to czas jaki użytkownik jaki użytkownik jest zalogowany / ilość użytkowników
   - System musi wiedzieć ile czasu użytkownik jest już zalogowany i ile czasu CPU otrzymał dla wszystkich swoich procesów od zalogowania się
   Wyliczamy stosunek czasu przyznanego CPU do czasu przyługującego
   0.5 – zużył połowe 2.0 – zużył 2 razy więcej niż mu przysługuje gdy współczynnik jest wyższy od wpółczynników wszystkich innych użytkowników proces uruchamiany jest z najniższym priorytetem

6. **Lottery scheduling** – planowanie loteryjne
    - Jest algorytmem bardzo czułym (większa ilość losów zwiększa szanse wylosowania już w następnym losowaniu)
    - Każdy proces dostaje zadaną ilość losów i następnie losowo jest wybierany jeden który otrzyma czas CPU (im większa ilość losów tym większa szansa, że zostanie wybrany np. ważniejszy proces)
    - Procesy mogą wymieniać losy między sobą np. klient wysyła zapytanie do serwera i przekazuje mu swoje losy > serwer daje odpowiedz i zwraca mu losy
    - Niezłe wyniki przy realizacji prostszej niż planowanie gwarantowane
    - Może być używane do rozwiązania problemów trudnych do rozwiązania innymi metodami
7. **Wielopoziomowe planowanie kolejek** :
   Procesy są zaliczane do kilku grup np. pierwszoplanowe, drugoplanowe etc.
   Kolejka procesów gotowych jest rozdzielona na osobne kolejki
   Każda kolejka ma własny algorytm planujący
   Konieczny jest algorytm planowania między kolejkami lub przyznanie każdej kolejce % czasu procesora

   Ze sprzężeniem zwrotnym
   Umożliwia przemieszczanie się procesów między kolejkami.
   Idea to grupować procesy według czasu zużywania procesora np. proces zużywający za dużo czasu procesora zostaje przeniesiony do kolejki o niższym priorytecie.
   Przykład
   Mamy 3 kolejki im niższy priorytet tym na dłużej może uzyskać procesor
   1(najwyższy priorytet) algorytm z kwantem 8 (każdy proces otrzymuje procesor na 8 jednostek czasu)
   2 algorytm z kwantem 16 (każdy proces … na 16 jednostek czasu)
   3 aglorytm FCFS (proces otrzymuje czas procesora aż do swojego zakończenia)
   Zapobieganie zagłodzeniom zbyt długo oczekujący proces przenosimy do kolejki o wyższym priorytecie

Planowanie wieloprocesorowe procesów

a) każdy procesor ma swoją kolejkę
Zalety:
Wady:
b) jedna główna kolejka dla wszystkich procesorów
Zalety:
Wady:

Planowanie wieloprocesorowe wątków

Podstawowe podejścia:

1. **Współdzielenie obciążenia (Load Sharing)**
   Stosowana jest globalna kolejka gotowych do wykonania wątków gdy procesor jest wolny wybiera wątek z kolejki
   Zalety:
   - Obciążanie rozłożone równomiernie między procesory
   - Nie jest wymagany centralny planista - gdy procesor jest dostępny, procedura szeregująca systemu operacyjnego jest na nim wykonywana i dokonuje wyboru wątku
   - Globalna kolejka może być obsługiwana przy użyciu jednego z algorytmów przedstawionych dla podejść jednoprocesorowych

   Wady:
   - Dostęp do centralnej kolejki trzeba uzyskiwać stosując wzajemne wykluczanie więc dla (dziesiątek,setek) procesorów kolejka może być wąskim gardłem
   - Wywłaszczane wątki rzadko są kontynuowane na tym samym procesorze pamięć podręczna procesora traci wydajność
   - Jeżeli wszystkie wątki traktujemy jako wspólną pulę mała jest szansa, że wszystkie wątki danego procesu uzyskają dostęp do procesorów równocześnie.

2. **Szeregowanie zespołowe (Gang Scheduling)**
   Zbiór wątków tego samego procesu jest wykonywany jednocześnie na wielu procesorach
   Zalety:
   - Wszystkie wątki danego procesu otrzymują naraz dostęp do procesora
   - Bardzo korzystne w przypadku aplikacji równoległych, których wydajność znacznie spada gdy część aplikacji nie działa razem z resztą
   - Minimalizuje przełączanie procesów (wątki nie muszą oddawać procesora w oczekiwaniu na np. zwolnienie mutexu przez inny wątek)

3. **Rezerwacja procesorów (Dedicated processor assignment)**
   Przydział wątków do wykonania na konkretnych procesorach
   Zalety:
   - Całkowity brak przełączania procesów podczas życia danego procesu

   Wady:
   - Podejście zbyt rozrzutne nie ma programowania wieloprocesorowego wątek oczekujący na we/wy nadal zajmuje procesor

4. **Dynamiczne szeregowanie (Dynamic Scheduling)**
   Ilość wątków w procesach może być zmieniania podczas wykonywania

Warunki, jakie musi spełniać rozwiązanie problemu sekcji krytycznej

1. **Wzajemne wykluczanie**
   Jeżeli proces Pi działa w swej sekcji krytycznej to żaden inny proces nie działa w sekcji krytycznej
2. **Postęp**
   Jeżeli żaden proces nie działa w sekcji krytycznej i istnieją procesy, które chcą wejść do sekcji krytycznych, to tylko procesy nie wykonujące swoich reszt mogą kandydować jako następne do wejścia do sekcji krytycznych i wybór ten nie może być odwlekany w nieskończoność.
3. **Ograniczone czekanie**
   Musi istnieć wartość graniczna liczby wejść innych procesów do ich sekcji krytycznych po tym, gdy dany proces zgłosił chęć wejścia do swojej sekcji krytycznej i zanim uzyskał na to pozwolenie.
   (proces nie może czekać w nieskończoność na wejście do sekcji krytycznej)

Rozwiązanie sekcjii krytycznej dla dwóch procesów

```
{ Procesy: P0 i P1 }
j = 1-i

Proces i:

{określa, że dany proces jest gotowy do wejścia sekcji krytycznej}
var znacznik: array[0..1] of boolean;

{określa, który proces ma prawo do wejścia do sekcji krytycznej}
var numer: 0..1;

znacznik[0] := false;
znacznik[1] := false;

numer := dowolna;

repeat
    znacznik[i] := true;
    numer := j;

    { czekaj na możliwość wejścia w sekcję krytyczną }
    while (znacznik[j] and numer=j) do nic;

    {- sekcja krytyczna -}

    znacznik[i] := false;

    {- reszta -}
until false;
```