1. WSTĘP

System operacyjny to **program komputerowy**, gwarantujący **prawidłową** i **wydajną** pracę komputera i stowarzyszonych zasobów (peryferia, pliki, oprogramowanie).

➔Udostępnia **programom** i **użytkownikom** **USŁUGI** do realizacji zadań programistycznych.

➔Jest: -**dystrybutorem zasobów** *(resorce allocator)* i arbitrem w sytuacjach konfliktowych;
-**programem sterującym** *(control program)*, nadzoruje programy użytkownika i kontroluje urządzenia peryferyjne.

Powinien **eliminować** wpływ programów **użytkowych** (kompilatory, gry, aplikacje użytkownika) na działanie systemu komputerowego.

⇨ Architektura komputera musi być wyposażona w odpowiednie mechanizmy sprzętowe.

Sens istnienia SO: -**wygoda** użytkownika,
-**wydajność** systemu komputerowego.

1.1. Tryb bezpośredni

Programista-operator obsługiwał komputery, pisząc i uruchamiając program z konsoli.

Program wprowadzano ręcznie do pamięci (rozkaz po rozkazie), ustawiając przełączniki na płycie czołowej, lub wykorzystując czytnik taśmy papierowej bądź kart dziurkowanych.

Po ustawieniu adresu startowego, rozpoczynało się wykonywanie programu.

Po **wykryciu błędu** programista wstrzymywał program, sprawdzał zawartość pamięci oraz rejestrów i wprowadzał poprawki z konsoli.

Wyniki były drukowane lub dziurkowane na taśmie / kartach.

● **Opracowano asemblery, programy ładujące** i **łączące** (konsolidatory).

Powstały biblioteki typowych funkcji, które dołączano do nowych programów bez konieczności pisania ich za każdym razem od nowa.

Powstały podprogramy obsługi każdego urządzenia We/Wy, nazywane **modułami sterującymi urządzeń** *(device driver).*
Moduł sterujący korzysta z buforów, znaczników, rejestrów, bitów kontrolnych i bitów stanu danego urządzenia.

● **Opracowano kompilatory języków Fortran, Cobol, Algol.**

Wykonanie programu w Fortranie wymagało wprowadzenia do komputera kompilatora języka.
Kompilator dostępny był na taśmie magnetycznej.
Później wczytywano programu z czytnika kart i zapisywano go na innej taśmie.
Kompilator Fortranu przekładał program na język asemblera.
Następnie wprowadzano kolejną taśmę z **asemblerem**.
Wynik asemblacji wymagał jeszcze **połączenia** z funkcjami bibliotecznymi.
W końcu powstawał program wynikowy do wykonania, w postaci kodu binarnego.

● **Czas instalowania** *(set-up time)* stanowił dużą część czasu wykonania **zadania**.

Zadanie składało się z:
-załadowania taśmy z kompilatorem,
-pracy kompilatora,
-zdemontowania taśmy z kompilatorem,
-zamontowania taśmy z asemblerem,
-pracy asemblera,
-zdjęcia taśmy z asemblerem,
-załadowania programu wynikowego uruchomienia.

◖ **Zatrudniono zawodowych operatorów.**

⇨ Programista przestał obsługiwać maszynę.
Operator szybciej obsługiwał taśmy. ⇨ Skrócono czas instalowania zadania.
Użytkownik, oprócz kart i taśm, dostarczał krótki opis wykonania zadania.

☞ Operator **nie poprawiał** błędnego programu, gdyż nie musiał rozumieć go.
Po wystąpienia błędu w programie udostępniał zawartość pamięci i rejestrów programiście.
Programista rozpoczynał usuwanie błędów, zaś operator wykonywał następne zadanie.

◖ **Zadania podobne gromadzono razem** i wykonywano grupowo, jako **wsad** *(batch).*

Operator otrzymał jedno zadanie w Fortranie, jedno w Cobolu i znowu jedno w Fortranie.
Gdyby uruchamiałby je w tej samej kolejności, to
-musiałby najpierw przygotować Fortran,
-następnie zainstalować Cobol,
-po czym znów przygotować komputer do pracy w Fortranie.
Gdy zebrał razem i wykonał oba zadania w Fortranie, dokonał jednego instalowania Fortranu.

◖ **Konsola informowała operatora o zatrzymaniu zadania.**

Musiał rozstrzygnąć, czy program zakończył się **poprawnie** czy z powodu **błędu**
Jeżeli z powodu błędu to musiał: -wyprowadzić zawartość pamięci i rejestrów,
-następnie załadować nowe zadanie i wznowić pracę.

Procesor pozostawał bezczynny podczas przechodzenia od jednego zadania do następnego
Potrzebny był mechanizm automatycznego przekazywania sterowania od zadania do zadania.

◖ **Automatyczne porządkowanie zadań** *(automatic job sequencing)* ograniczyło bezczynność.
– podstawa struktury systemów operacyjnych

➨ Powstał program zwany **monitorem rezydentnym** (*resident monitor)*, rezydujący stale w PAO.

1.2. Tryb wsadowy

Włączano komputer i wywoływano **monitor rezydentny**, który przekazywał sterowanie do programu (zadania).
Program kończył działanie i zwracał sterowanie do monitora, a ten wyznaczał kolejny program.
Monitor zapewniał automatyczne przechodzenie od jednego zadania do następnego.

Skąd monitor wiedział, który program ma wykonać?

Wprowadzono **karty sterujące** *(control cards).*

Do zadania oprócz programu i danych, dołączano karty, zawierające **dyrektywy** dla monitora i wskazujące, który program ma być wykonany.
Granice zadania określały karty: **$JOB; $END** (odpowiednio 1-sza i ostatnia karta).

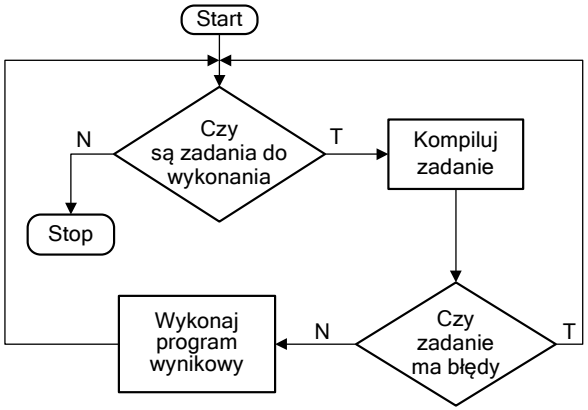
Karty sterujące wyróżniał znak dolara **$**, umieszczony w 1-szej kolumnie.
Powstał język sterowania zadaniami **JCL** (*Job Control Language)* .

$FTN - uruchomienie kompilatora Fortranu
$ASM - uruchomienie asemblera
$RUN - uruchomienie programu użytkownika

Monitory wsadowe opracowano dla komputerów bez **systemu przerwań**.

Przetwarzały sekwencje wkładów bez ludzkiej interwencji.

Oszczędzały czas, stracony podczas oczekiwania na reakcje operatorów i podjęcie wykonania nowych zadań.



Rys.1.1. Schemat działania monitora

Monitor automatyzuje zadania, zgodnie z kartami sterującymi.

Kiedy **karta sterująca** sygnalizuje utworzenie kodu programu, monitor wprowadza go do pamięci i przekazuje mu sterowanie.

Po zakończeniu pracy program zwraca sterowanie do monitora, który czyta następną kartę sterującą, ładuje nowy program itd.

↻ Cykl powtarza się aż zostaną wykonane **polecenia** ze wszystkich **kart sterujących** w zadaniu.

Wtedy monitor automatycznie przechodzi do wykonania następnego zadania.

Monitor wywoływał kompilatory jako podprogramy.

Program wynikowy także miał postać podprogramu.

Zadania przekazywano monitorowi w postaci „strumienia prac" na kartach lub taśmie papierowej.

Każda praca poprzedzona była „opisem pracy", który identyfikował ją oraz podawał niezbędne informacje, takie jak: -wymagany kompilator (jeśli było ich kilka w systemie),

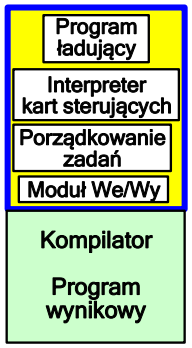
-żądanie wyprowadzenia zawartości pamięci po zaskoczeniu obliczeń.

☛ **Zadanie do wykonania** zawierało w sobie: -**program**, -**zestaw danych**, -**karty sterujące.**

Część pamięci operacyjnej była na stałe przydzielona monitorowi, pozostała wykorzystywana przez kompilator i program wynikowy.

Program ładujący i interpreter kart wymagają wykonywania operacji We/Wy.

Interpreter kart sterujących (*control-card interpreter*), odpowiadał za czytanie i wykonywanie poleceń z kart podczas wykonywania zadania.



Monitor nie miał nad wykonywaniem programu dużej kontroli.

☞ Wykonanie nielegalnego rozkazu często zawieszało cały system.

☞ Jeśli program wpadł w pętlę, to wciągał w nią cały system.

Nie było możliwości uchronienia części monitora przed zniszczeniem w wyniku działania błędnego programu wynikowego.

Pętle nieskończone były trudne do wykrycia.

Interwencje operatora w celu zatrzymania nieskończonej pętli były dość częste, a w dużym procencie monitor ulegał uszkodzeniu.

Bezpośrednia konwersacja programu z użytkownikiem, od strony technicznej, była możliwa, lecz niechętnie dopuszczana, gdyż wstrzymywała działanie całego systemu.

☞ Obszar pamięci przydzielony na MONITOR był niedostępny dla programów wynikowych.

☞ Kompilator i asembler stanowiły integralną cześć systemu dostarczonego przez producenta.

● **Wprowadzenie operacji na zbiorach to aktywizacja rozwoju MONITORÓW**

System zbiorów został udostępniony przez monitor wsadowy, gdy komputer dysponował dostatecznie dużą pamięcią zewnętrzną.

Stworzono bibliotekę podprogramów i kompletnych programów, które mogły być włączane do każdej pracy, o ile zostały wymienione przez nazwę i z zachowaniem odpowiednich konwencji.

Procedury współpracy z urządzeniami zewnętrznymi były włączone do monitora tylko raz.

➡ Programy wynikowe nie musiały odwoływać się do urządzeń zewnętrznych bezpośrednio, lecz wywoływały procedur y obsługi zawarte w monitorze jako podprogramy.

Zaoszczędzono **miejsce** w pamięci i **czas** programowania, gdyż instrukcje We/Wy miały standardową postać.

☞ Monitor musiał, gdy było to konieczne, zapewnić metodę aktualizowania biblioteki.

Szybkość całego systemu komputerowego ograniczał najwolniejszy element.

Rozwiązaniem problemu była technika **pracy pośredniej** (*off-lining*) po raz pierwszy użyta w systemie operacyjnym „PUFFTS" Uniwersytetu Purdue.

Zamiast podawać karty bezpośrednio do czytania komputerowi, **kopiowano** je wpierw na **taśmie** magnetycznej za pomocą **osobnego** urządzenia.

Kiedy program potrzebował danych z karty, dostarczano mu równoważny rekord z taśmy.

Podobnie, wyniki zapisywano na taśmie, a jej zawartość drukowano w późniejszym czasie.

➡ Czytnik kart i drukarka wierszowa były obsługiwane w **trybie pośrednim** *(off-line)*.

Off-lining polega na przenoszeniu dokumentów wejściowych z urządzeń wolnych do pamięci pomocniczej przed uruchomieniem programu oraz na kierowaniu, wygenerowanych przez program wyników do pamięci pomocniczej przed ich wyprowadzeniem na wolne urządzenia zewnętrzne.

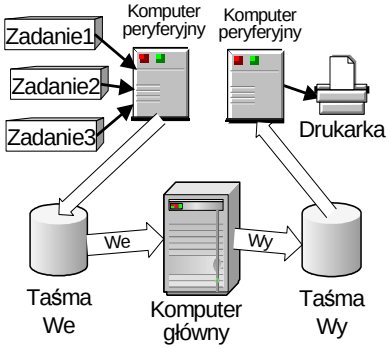
System składał się z szybkiego komputera głównego **(IBM 7090**) i jednego lub kilku wolnych komputerów peryferyjnych (IBM 1401).

Główny komputer **nie nadzorował** pracy wolnych urządzeń, lecz **współpracował** podczas wprowadzania i wyprowadzania informacji tylko z **taśmą** magnetyczną.

Na **We** taśmę magnetyczną kopiowano wsad z kart dziurkowanych

Taśma **Wy**jściowa głównego komputera była przetwarzana przez IBM 1401 i powielana na drukarce

System był efektywny, gdyż główny komputer nie był blokowany przez operacje We/Wy (szybkości transmisji taśmy magnetycznej była wyższa niż innych urządzeń).



Rys.1.2. Tryb off-lining

● **Wprowadzenie technologii dyskowej** (*dostęp swobodny)* –dalszy rozwój Monitorów

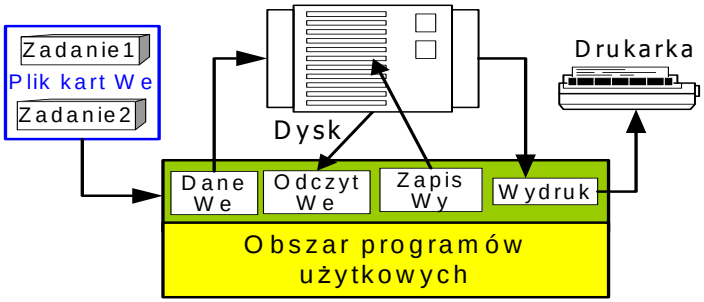
Zapis informacji na ***sekwencyjne*** pamięci taśmowe zastąpiono bezpośrednim zapisem na **dysk**.

➔ Lokalizacje obrazów kart zapisywane są w Tablicy przechowywanej przez system operacyjny.

➔ Zapotrzebowanie na dane wejściowe z czytnika realizowane są przez czytanie z **dysku**.

Gdy zadanie chce wyprowadzić dane na drukarkę są one zapisane na dysku w **buforze systemowym**.

Po zakończeniu zadania wyniki są fizycznie drukowane.



Rys.1.3. System z mechanizmem spool

Spooling „simultaneous peripheral operation on-line" -jednoczesna, bezpośrednia praca urządzeń.

Spooling używa dysku jako bufora do czytania z urządzeń We oraz do przechowywania plików Wy **do czasu**, aż urządzenia wyjściowe będą w stanie je przyjąć.

Spooling umożliwia **zdalne** przetwarzanie danych.

Monitor pobiera/wysyła dane łączem komunikacyjnym z własną szybkością bez udziału procesora, który jest tylko informowany o zakończeniu.

Kosztem pewnego obszaru pamięci na dysku i kilku Tablic, CPU mógł wykonywać **równocześnie**:

-obliczenia **jednego** zadania,

-operacje We/Wy dla innych zadań.

---------- Czy monitor wsadowy to już absolutna historia? ----------

Dostępne są **„pseudomonitory"** w ramach nowoczesnych systemów operacyjnych.

Większość dużych systemów operacyjnych oferuje użytkownikowi **"komputer wirtualny",** który może przypominać komputery wcześniejszej generacji.

Ma to zalety, ponieważ współczesne systemy (projektowane aby sprostać potrzebom bardzo skomplikowanym zadań) mogą być nieefektywne, gdy mają do czynienia *z* **prostymi** zadaniami.

1.3. Tryb wsadowy wieloprogramowy (*batch processing*)

Spooling powoduje, że pewna liczba zadań *(job pool)* jest zawczasu czytana na dysk, gdzie czeka gotowa do wykonania.

System operacyjny może tak wybierać następne zadania do wykonania, aby zwiększyć wykorzystanie CPU.

Zadania nadchodzące z taśmy magnetycznej **musiały być wykonane w kolejności nadejścia**.

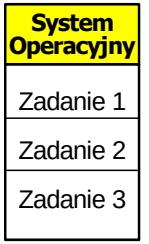
Praca pośrednia oraz spooling umożliwiający nakładanie operacji We/Wy mają ograniczenia.

☛ Jeden użytkownik **nie** zdoła zapewnić **ciągłą** aktywności CPU lub urządzeń We/Wy.

● **Planowanie zadań** (szeregowanie - *job scheduling*) to decyzja o wyborze określonego zadania. Jest możliwe, jeżeli kilka zadań umieści się na urządzeniu o dostępie bezpośrednim, jak dysk.

Najważniejszym aspektem planowania zadań jest możliwość **wieloprogramowania.**

Wieloprogramowość zwiększa wykorzystanie CPU gdyż tak organizuje zadania, aby procesor miał zawsze któreś z nich do wykonania.



Wszystkie zadania wchodzące do systemu trafiają do puli zadań, ulokowanej w **pamięci masowej** i czekają na **przydział PAO**.

Jeżeli kilka zadań jest gotowych do wprowadzenia do **PAO**, lecz brakuje dla wszystkich miejsca, to system wybiera kilka spośród nich.

SO pobiera zadanie i rozpoczyna jego wykonywanie.
Gdy zadanie zaczyna oczekiwać np.: na zakończenie operacji We/Wy, SO przechodzi do wykonania innego zadania.
Gdy pierwsze zadanie **skończy oczekiwanie** to otrzyma z powrotem CPU.

● **Planowanie przydziału procesora** *(CPU scheduling).*

Przechowywanie wiele zadań w **PAO** w **tym samym czasie** wymaga zarządzania pamięcią.

Gdy kilka zadań gotowych jest do działania, to system musi wybrać jedno z nich.

Współbieżne wykonywanie wielu zadań wymaga ***ograniczania możliwości ich wzajemnego*** zakłócania we wszystkich stadiach pobytu w systemie operacyjnym:

-w czasie planowania procesów,
-przydzielania pamięci dyskowej,
-zarządzania pamięcią operacyjną.

Wieloprogramowość sprawia, że system operacyjny decyduje za użytkownika.

1.4. Tryb wieloprogramowy z podziałem czasu

Wieloprogramowy system wsadowy jest **kłopotliwy**:

-użytkownik nie może ingerować w zadanie podczas jego wykonywania,

-musi przygotować karty sterujące na okoliczność wszystkich możliwych zdarzeń.

W zadaniu wykonywanym krok po kroku, następne kroki mogą zależeć od wcześniejszych wyników. Trudno przewidzieć, co należy robić we wszystkich przypadkach.

Podział czasu – **wielozadaniowość** *(multitasking)* stanowi rozszerzenie wieloprogramowości.

Procesor wykonuje na przemian wiele różnych zadań, przy czym przełączenia następują tak często, że użytkownicy mogą współdziałać z każdym programem podczas jego wykonania.

Interakcyjny lub **bezpośredni** (*hands-on)* system komputerowy umożliwia bezpośredni dialog użytkownika z systemem.

Użytkownik wydaje bezpośrednio instrukcje **SO** lub programowi i otrzymuje natychmiastowe odpowiedzi.

Wejściem jest zazwyczaj klawiatura, a jako wyjście służy monitor.

Użytkownik wydaje polecenie, czeka na odpowiedź i o kolejnym poleceniu decyduje na podstawie wyników poprzedniego polecenia.

Użytkownik może łatwo eksperymentować z programem i natychmiast oglądać rezultaty.

Większość systemów ma interakcyjne edytory tekstów do wprowadzania programów i interakcyjne programy uruchomieniowe, ułatwiające usuwanie błędów z programów.

Bezpośredni dostępu do **plików** *(on-line file system)* to mechanizm usprawniający częsty dostępu do danych.

Pliki zorganizowane są w logiczne niepodzielne grupy, czyli katalogi *(directories),* ułatwiające wykonywanie na nich działań.

Do plików ma dostęp wielu użytkowników, musi istnieć mechanizm nadzoru nad tym, kto i w jaki sposób z nich korzysta.

Wsadowe systemy wygodne są dla dużych zadań, których wykonanie nie wymaga stałego bezpośredniego dozoru.

Użytkownik może dostarczyć zadanie i przyjść później po wyniki.

Interakcyjne zadania składają się z wielu krótkich działań, w których rezultaty kolejnych poleceń mogą być nieprzewidywalne.

Czas odpowiedzi *(response time)* powinien być krótki.

Podział czasu *(time-sharing systems)* umożliwia interakcyjne użytkowanie systemu komputerowego po umiarkowanych kosztach.

Planowanie przydziału procesora i wieloprogramowość zapewniają użytkownikom korzystanie z małych porcji dzielonego czasu pracy komputera.

Każdy użytkownik ma przynajmniej jeden oddzielny program w pamięci.

Proces: program załadowany do pamięci operacyjnej (**PAO**) i wykonywany w niej.

➔ Wykonywanie procesu trwa zwykle niedługo i albo się kończy, albo powoduje zapotrzebowanie na operację We/Wy.

Operacje We/Wy mogą przebiegać w trybie konwersacyjnym, tzn. wyniki wyświetlane są na ekranie, a polecenia i dane - wprowadzane z klawiatury.

Szybkość interakcyjnego We/Wy zależy od człowieka, może ono zajmować sporo czasu.

System operacyjny w takich przypadkach angażuje CPU do wykonywania innego programu.

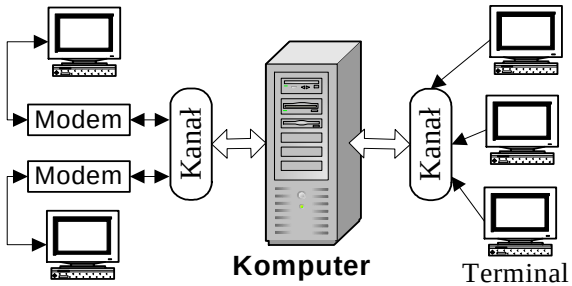
1.5. Tryb wielodostępny

System operacyjny z podziałem czasu umożliwia wielu użytkownikom korzystanie równocześnie z jednego komputera poprzez terminale, traktując je jak pulpit własnego komputera.

Przełączanie CPU od jednego użytkownika do drugiego, pozwala odnosić wrażenie, że dysponuje on własnym komputerem, choć w rzeczywistości komputer jest jeden.

Ideę podziału czasu zaprezentowano w **1960 r.**

Problemy techniczne sprawiły, że pojawiły się one na początku lat 70-tych.



Rys.1.4. Struktura systemu wielodostępnego

Z czasem zaczęto łączyć systemy wsadowe z systemami z podziałem czasu.

Wiele systemów komputerowych, pierwotnie projektowanych do pracy w trybie wsadowym, zmodyfikowano do pracy z podziałem czasu.

SO **OS/360** dla komputerów IBM, który był systemem wsadowym, poszerzono o możliwość pracy z podziału czasu.

W tym okresie systemy z podziałem czasu wzbogacano często o podsystemy wsadowe.

Systemy z podziałem czasu są bardziej złożone niż wieloprogramowe SO.

W PAO przechowuje się jednocześnie wiele aktywnych zadań wielu użytkowników, które potrzebują bezpiecznego zarządzania pamięcią i ochrony.

Działania współbieżne wymagają optymalnych metod przydziału CPU.

Muszą istnieć mechanizmy synchronizowania zadań oraz komunikacji między nimi.

Zadania nie mogą się zakleszczać, nieustannie wzajemnie na siebie czekając.

Zakleszczenie to sytuacja, w której dochodzi do blokad z udziałem wielu procesów.

● **Dysk stał się zapleczem** dla **pamięci głównej.**

Optymalizacja czasu odpowiedzi zadań, wymaga czasami odsyłania niektórych z nich na dysk i sprowadzania ich z powrotem

System plików rezyduje w zbiorze dysków ⇨ należy zapewnić zarządzanie dyskami.

● **Pamięć wirtualna** (dwie warstwy pamięci):

-pamięć operacyjna **PAO**,

-pamięć zewnętrzna (dużo większa).

➔ Programista dostaje do dyspozycji jednolicie ponumerowaną pamięć.

Obie pamięci podzielone są **na bloki** (np. 4kB).

Jej dynamiczny podział na PAO i zewnętrzną nadzoruje system operacyjny w trakcie realizacji programów.

Dla programisty pamięć jest strukturą **jednopoziomową**, adres składa się z dwu części:
-numeru strony,
-numeru komórki w obrębie strony.

Pamięć wirtualna umożliwia wykonanie **programów większych** niż **pamięć fizyczna** PAO.

Powstała **abstrakcja pamięci główna** w postaci wielkiej, jednolitej tablicy, oddzielająca pamięć logiczną - oglądaną przez użytkownika - od pamięci fizycznej.

Programiści przestali przejmować się **ograniczeniami pamięciowymi.**

System operacyjny powinien ponadto:
-obsługiwać wydajnie transmisję między terminalami a komputerem;
-weryfikować prawa dostępu użytkowników;
-po wejściu użytkownika o wysokim priorytecie i braku wolnych środków, wstrzymać wykonywanie prac o niższym priorytecie lub je usunąć, aby przyjąć nowe zadanie.

❒ **Wielozadaniowość** bez **wywłaszczenia** *(nonpreemptive multitasking)*

Kontrolę nad czasem dostępu aplikacji do procesora przejmują twórcy oprogramowania.

Scheduler może dokonywać przełączeń między procesami **tylko wtedy**, gdy aktualny **proces zwalnia** dostęp do CPU.

☛ Dobrym zwyczajem programistycznym jest używanie procedury ***yielding***, regularnie zwalniającej dostęp do procesora.

Umożliwia to udostępnienie zasobów komputerowych procesowi o wyższym priorytecie, jeżeli taki pojawi się w systemie.

⇨ Największym problemem jest zawieszenie się aplikacji, co często uniemożliwia przekazanie kontroli nad systemem innym, uruchomionym już procesom.

❒ **Wielozadaniowość** z **wywłaszczeniem** *(preemptive multitasking)*

SO zachowuje pełną kontrolę nad czasem, na jaki dana aplikacja uzyskuje dostęp do CPU.

Każdemu procesowi przypisany jest priorytet.

Jeżeli proces o wysokim priorytecie zażąda dostępu do CPU to Scheduler może przerwać aktualnie wykonywany proces i udostępnić czas CPU procesowi o **wyższym** priorytecie.

System CTSS *(Compatible Time-Sharing System)*, zaprojektowano w instytucie MIT jako system z podziałem czasu.

Zaimplementowany na komputerze IBM 7090, umożliwiał pracę konwersacyjną z udziałem **32** użytkowników.

Polecenia interakcyjne, manipulowanie plikami, kompilowanie i wykonywanie programów za pośrednictwem terminali.

IBM 7090 miał **32 K** pamięci utworzonej z 36-bitowych słów.

Monitor zużywał **5 K** słów, pozostawiając **27 K** dla użytkowników.

Zaimplementowano **wymianę** obrazu pamięci użytkowników między **PAO** a bębnem.

Planowanie CPU realizował algorytm wielopoziomowych kolejek ze sprzężeniem zwrotnym.

Kwant czasu dla poziomu **p** wynosił 2^p jednostek.

Jeśli program nie kończył swojej fazy procesora w jednym kwancie czasu, to przesuwano go w dół, na następny poziom kolejki, dając dwukrotnie więcej czasu.

1.6. Systemy wieloprocesorowe

W systemy wieloprocesorowych *(multiprocessor systems)*: pewna liczba procesorów ściśle współpracuje ze sobą, dzieląc:
-szynę komputera,
-zegar,
- urządzenia zewnętrzne.
- czasami PAO,

Systemy takie nazywa się także **ściśle powiązanymi** *(tightly coupled).*

Pozwalają zwiększy **przepustowości** (*throughput),* jednak współczynnik przyspieszenia przy **n** CPU jest mniejszy od **n**.

Kiedy kilka CPU współpracuje przy wykonaniu jednego zadania, wtedy traci się pewien czas na utrzymywanie właściwego działania wszystkich części.

Ten nakład w połączeniu z rywalizacją o zasoby dzielone zmniejsza oczekiwany zysk z zastosowania dodatkowych procesorów.

Grupa współpracujących ze sobą **n** programistów nie powoduje **n-krotnego** wzrostu wydajności pracy.

Systemy wieloprocesorowe dają możliwość wspólnego użytkowania urządzeń zewnętrznych, i zasilania ze wspólnego źródła.

Jeśli kilka programów ma działać na tym samym zbiorze danych, to taniej jest trzymać dane na jednym dysku i pozwolić na korzystanie z nich wszystkim CPU, niż rozmieszczać ich kopie na wielu komputerów.

Zwiększają niezawodność poprzez rozdzielenie zadań między pewną liczbą procesorów; awaria jednego procesora nie zatrzymuje systemu, tylko go spowalnia.

Łagodna degradacja *(graceful degradation)*: zdolność wykonywania usług proporcjonalnie do ilości ocalałego sprzętu.

Systemy tolerujące awarie - systemy przystosowane do łagodnej degradacji.

System składa się z **dwu** identycznych CPU z lokalną pamięć.

Procesory są połączone za pomocą szyny.

Jeden z nich jest procesorem podstawowym, drugi procesorem zapasowym.

☛ Każdy **Proces** ma dwie kopie: -jedną w systemie podstawowym,
- -drugą w systemie zapasowym.

Podczas działania systemu, w ustalonych okresach czasu stan informacji o każdym zadaniu (włącznie z obrazem pamięci) jest kopiowany z systemu podstawowego do zapasowego.

Wykrycie uszkodzenia uaktywnia kopię zapasową.

❒ **Wieloprzetwarzanie symetryczne** (*symmetric multiprocessing).*

Na każdym CPU działa **identyczna kopia** systemu operacyjnego.

Kopie **komunikują się** ze sobą w zależności od potrzeb.

Należy zapewnić takie wykonanie operacji We/Wy, aby dane docierały do **właściwych** CPU.

Z powodu izolacji CPU, jedne mogą pozostawać bezczynne, zaś inne będą przeciążone pracą.

Aby uniknąć niesymetrii obciążeń, CPU mogą korzystać z wspólnych struktur danych.

Pozwala to na dynamiczny podział zadań i zasobów między różne procesory i może zmniejszyć różnice między poszczególnymi systemami.

Przykładem systemu z wieloprzetwarzaniem symetrycznym była wersja Encore systemu UNIX dla komputera Multimax.

Komputer umożliwia działanie dużej liczby CPU, z których każdy pracuje pod nadzorem kopii systemu UNIX.

Równocześnie może pracować wiele procesów (**N** procesów na **N** egzemplarzach jednostki centralnej) bez pogarszania działania całego systemu.

❒ **Wieloprzetwarzanie asymetryczne** *(asymmetric multiprocessing),* każdemu procesorowi przypisano inne zadanie.

Procesor główny planuje i przydziela prace CPU podporządkowanym.

Inne CPU albo czekają na instrukcje od procesora głównego, albo zajmują się swoimi uprzednio określonymi zadaniami.

➔ Występuje w bardzo wielkich systemach, w których czynnościami zużywającymi najwięcej czasu są operacje We/Wy.

W starszych systemach wsadowych małe procesory, zlokalizowane w pewnej odległości od głównej jednostki centralnej, służyły do obsługiwania czytników kart i drukarek oraz do przekazywania zadań do i od komputera głównego.

Stacje takie nazywa się **stanowiskami zdalnego wprowadzania zadań** *(remote job entry*).

W systemach z podziałem czasu operacje We/Wy to przekazywanie znaków między końcówkami a komputerem.

Aby uniknąć angażowania procesora głównego do obsługi znaku z każdego terminala, stosuje się **procesor czołowy** *(front-end),* zajmujący się transmisjami z końcówek.

➔ Duży system komputerowy może używać minikomputera PC jako **procesora czołowego.**

Procesor czołowy działa jak bufor między końcówką konwersacyjną a procesorem głównym.

Umożliwia obsługę całych wierszy i bloków zamiast pojedynczych znaków.

Różnica między przetwarzaniem symetrycznym a asymetrycznym może wynikać albo ze sprzętu, albo z oprogramowania.

Rozróżnianie wielu CPU może być wykonywane za pomocą specjalnego sprzętu.

Można zainstalować oprogramowanie, które będzie pozwalało na istnienie tylko jednego komputera głównego i wielu komputerów podporządkowanych.

Wersja 4 systemu operacyjnego SunOS dla komputerów Sun umożliwia wieloprzetwarzanie asymetryczne, natomiast wersja 5 tego systemu (Solaris 2) jest symetryczna.

Wielu funkcji systemowych przerzuca się na podporządkowany SO mikroprocesor.

Można dodać mikroprocesor wyposażony we własną pamięć i przeznaczony do zarządzania dyskami.

Przyjmuje on polecenia od procesora głównego i realizuje własną kolejkę dyskową i algorytm planowania.

Uwalnia to procesor główny od zajmowania się planowaniem operacji dyskowych.

Komputery PC zawierają wmontowany w klawiaturę mikroprocesor.

➔ Takie zastosowanie mikroprocesorów stało się tak powszechne, że **nie jest** traktowane jako wieloprzetwarzanie.

1.7. Systemy rozproszone

W ostatnich latach pojawiła się tendencja do rozdzielania obliczeń między wiele CPU.

Procesory mają własną pamięć lokalną i komunikują się na przykład za pomocą szybkich szyn danych lub linii telefonicznych.

Systemy takie nazywa się **rozproszonymi** *(distributed systems).*

Procesory w systemach rozproszonych mogą różnić się rozmiarami i przeznaczeniem.

Mogą to być małe mikroprocesory, jak i wielkie systemy komputerowe ogólnego przeznaczenia.

❒ Celowość budowy

Podział zasobów: użytkownik jednego stanowiska może korzystać z zasobów innego.

Podział zasobów w systemie rozproszonym tworzy mechanizmy dzielonego dostępu do plików w węzłach zdalnych, przetwarzania informacji w rozproszonych bazach danych, drukowania plików w węzłach zdalnych, zdalnego użytkowania specjalizowanych urządzeń sprzętowych.

Przyspieszanie obliczeń: ***jeśli*** obliczenia można wykonywać współbieżnie, to system rozproszony umożliwia przydzielenie tych obliczeń do poszczególnych stanowisk i współbieżne ich wykonanie.

Jeżeli pewne stanowisko jest w danej chwili przeciążone zadaniami, to część z nich można przenieść do innego, mniej obciążonego stanowiska.

Niezawodność: jeżeli system składa się z dużych, autonomicznych instalacji, to awaria jednego z nich nie wpływa na działanie pozostałych.

Jeżeli system składa się z małych maszyn, z których każda odpowiada za jakąś istotną funkcję (np. za operacji We/Wy), wówczas z powodu jednego błędu może zostać wstrzymane działanie całego systemu.

Komunikacja: programy mogą wymieniać dane między sobą w ramach jednego systemu.

Wzajemne połączenie węzłów za pomocą komputerowej sieci komunikacyjnej umożliwia procesom w różnych węzłach wymianę informacji.

Użytkownicy sieci mogą przesyłać pliki lub kontaktować się ze sobą za pomocą **poczty elektronicznej** *(electronic mail).*

Przesyłki pocztowe mogą być nadawane do użytkowników tego samego węzła lub do użytkowników innych węzłów.

1.8. Systemy czasu rzeczywistego

System czasu rzeczywistego *(real-time)* stosowany jest, gdy istnieją ostre wymagania na czas wykonania operacji lub przepływu danych, ma ściśle zdefiniowane, stałe ograniczenia czasowe.

Przetwarzanie danych **musi zakończyć się** przed upływem określonego czasu, w przeciwnym razie system nie spełnia wymagań.

Czujniki dostarczają dane do komputera. a ten analizuje je i w zależności od sytuacji tak reguluje działaniem kontrolowanego obiektu, aby uzyskać pożądany wynik.

Przykładami są systemy nadzorowania eksperymentów naukowych, sterowania procesami przemysłowymi.

❒ **Rygorystyczny system czasu rzeczywistego** *(hard real-time system)*

Gwarantuje terminowe wykonanie krytycznych zadań.

Osiągnięcie tego celu wymaga ograniczenia opóźnień w **systemie**.

☛ Takie ograniczenia czasu wpływają na dobór środków wyposażenia.

Pamięć pomocnicza jest na ogół bardzo mała albo nie występuje wcale.
Dane przechowywane w pamięci o krótkim czasie dostępu lub w pamięci, z której można je tylko pobierać.

Systemy te nie mają większości cech nowoczesnych SO, które oddalają użytkownika od sprzętu, zwiększając niepewność odnośnie ilości czasu zużywanego przez operacje.

☛ Rzadko spotyka się pamięć wirtualną.

☛ Są w konflikcie z działaniem systemów z podziałem czasu i nie wolno ich ze sobą mieszać.

Żaden z istniejących, uniwersalnych SO **nie** umożliwia działania w czasie rzeczywistym.

❒ **Łagodny system czasu rzeczywistego** *(soft real-time system)*

Krytyczne zadanie do obsługi w czasie rzeczywistym **otrzymuje pierwszeństwo** przed innymi zadaniami i zachowuje je aż do swojego zakończenia.

Opóźnienia muszą być ograniczone - zadanie czasu rzeczywistego nie może w nieskończoność czekać na usługi jądra.

Łagodne traktowanie wymagań dotyczących czasu rzeczywistego umożliwia godzenie ich z systemami innych rodzajów.

Ich użyteczność jest bardziej ograniczona niż systemów rygorystycznych.

Zastosowanie ich w przemyśle i robotyce jest ryzykowne, gdyż nie zapewniają nieprzekraczalnych terminów.

W wielu dziedzinach są jednak przydatne:

-techniki multimedialne,
-kreowanie sztucznej rzeczywistości,
-badania podwodne czy wyprawy pozaziemskie.

Takie przedsięwzięcia wymagają systemów operacyjnych o rozbudowanych właściwościach, których nie mogą zapewnić rygorystyczne systemy czasu rzeczywistego.

Większość współczesnych systemów operacyjnych, łącznie z przeważającą częścią wersji systemu UNIX, spełnia wymagania łagodnego systemu czasu rzeczywistego.

1.9. Systemy operacyjne komputerów osobistych

Komputery osobiste - mikrokomputery pojawiły się w latach osiemdziesiątych.

Stosowane w nich jednostki centralne były pozbawione środków do ochrony systemu operacyjnego przed programami użytkowymi.

☛ Nie były one ani wielostanowiskowe, ani wielozadaniowe.

Położono w nich nacisk na maksimum wygody użytkowania i szybkości kontaktu z użytkownikiem.

Użytkowane są przez indywidualne osoby, a wykorzystanie CPU nie ma istotnego znaczenia.

Tworząc systemy operacyjne korzystano ze wzorów sprawdzonych dla dużych komputerów.

Pewne rozwiązania stosowane w dużych komputerach mogą wydawać się zbędne dla systemów mniejszych, jak np.: ochrona plików w komputerach osobistych.

Ochrona plików nie jest konieczna w indywidualnych komputerach osobistych.

Jednakże łączy się je w lokalne sieci komputerowych i ochrona plików znów staje się niezbędną cechą systemu operacyjnego.

Z analizy systemów operacyjnych dla dużych komputerów i minikomputerów wynika, że cechy dostępne tylko w dużych komputerach, zaadaptowano też w minikomputerach.

System operacyjny MULTICS opracowano w latach 1965-1970 w MIT dla celów obliczeniowych.
Działał na dużym komputerze głównym GE 645.

Wiele pomysłów wprowadzonych do systemu MULTICS zastosowano w firmie Bell Laboratories przy projektowaniu systemu UNIX (1970) dla minikomputera **PDP-11**.

Około 1980 r. rozwiązania z systemu UNIX stały się bazą dla UNIXopodobnych systemów operacyjnych przeznaczonych dla mikrokomputerów.

Osobista **stacja robocza** (*workstation),* taka jak Sun, HP/Apollo lub IBM RS/6000, jest wielkim komputerem osobistym, w którym adaptowano wiele cech dużych systemów operacyjnych.

➨ Komputery osobiste zmieniły pogląd na funkcje SO, przywracając pojęcie trybu bezpośredniego:

–informują użytkownika o zasobach sprzętowych i programowych,
–umożliwiają łatwą zmianę konfiguracji sprzętu, widzianej przez programy użytkowe,
–wykonują czynności organizacyjne na zlecenie użytkownika jak kopiowanie, itp.

Od pewnego czasu systemy operacyjne pisze się w językach wyższego poziomu.

Systemem Master Control Program (MCP) dla komputerów firmy Burroughs napisano w Algolu.
System MULTICS, opracowany w MIT, został napisany głównie w PL/1.
System operacyjny Primos dla komputerów Prime napisano w dialekcie Fortranu.
Systemy operacyjne: UNIX, OS/2 i Windows NT napisano niemal w całości w języku C.

Zaledwie 900 wierszy kodu oryginalnego systemu UNIX zaprogramowano w asemblerze.

Zastępując kompilator jego lepszą wersją, to zwykłe powtórne przetłumaczenie całego systemu operacyjnego poprawia jakość jego kodu.

Główne ulepszenia działania są zwykle rezultatem lepszych struktur danych i algorytmów aniżeli doskonałego kodowania w języku asemblerowym.

O efektywności SO ma istotny wpływ tylko niewielka część kodu; najważniejszymi procedurami są zarządca pamięci i planista przydziału procesora.