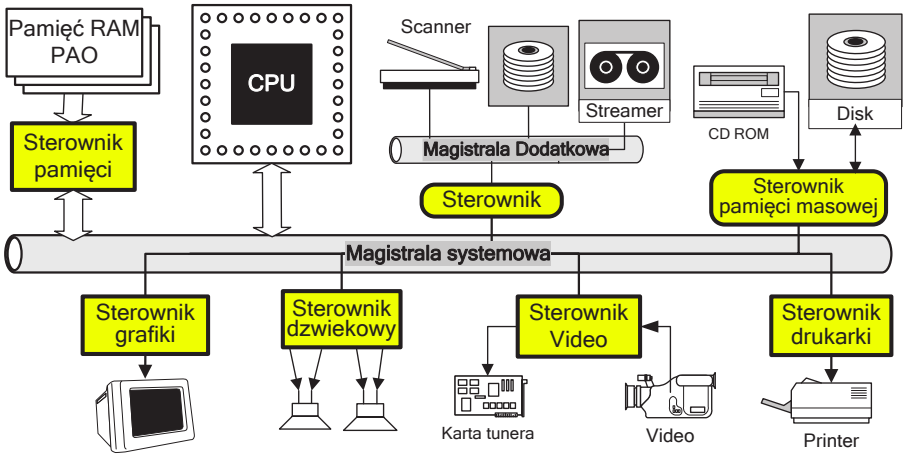
2. SYSTEM KOMPUTEROWY

Z czego System Operacyjny może korzystać?

System operacyjny powiązany jest z architekturą komputera ⇒korzysta z jego mechanizmów.



Rys. 2.1. Schemat systemu komputerowego

System komputerowy to: -**jednostka centralna** (**CPU**)
-**sterowniki urządzeń**, połączone wspólną szyną, z dostępem do PAO
-**pamięć operacyjna**(**PAO**).

Każdy sterownik urządzenia odpowiada za określony typ urządzenia.

CPU oraz **sterowniki urządzeń mogą działać współbieżnie** i **rywalizować o cykle pamięci**.

➔ **Sterownik pamięci** zapewnia uporządkowany, synchroniczny dostęp do wspólnej pamięci.

Sprzętowy sterownik urządzenia zarządza:
-lokalną pamięcią buforową,
-**zbiorem** specjalizowanych **rejestrów**.

Odpowiada on za przemieszczanie danych między urządzeniami zewnętrznymi, nad którymi sprawuje nadzór, a swoją lokalną pamięcią buforową.

Start komputera uruchamia **program rozruchowy** (*bootstrap)*, który ładuje jądro systemu operacyjnego (**SO**) do pamięci i rozpoczyna jego działanie.

➔ **SO** rozpoczyna wykonanie pierwszego procesu typu **Init** i czeka na **Zdarzenie**

Wystąpienie **Zdarzenia** sygnalizuje **przerwanie** *(interrupt)* od sprzętu lub od oprogramowania.

☛ Każdemu **przerwaniu** odpowiada procedura jego obsługi.

Różne **zdarzenia** mogą generować **przerwania** w **dowolnej chwili**, poprzez wysłanie sygnału do CPU.

Oprogramowanie wykonując operację **wywołanie systemu** *(system call)* też generuje **przerwanie**.

Pojęcie przerwanie pojawiło się w sprzęcie pod **koniec** lat **50-**tych.

Procesor po otrzymaniu przerwania wstrzymuje aktualnie wykonywane zadanie i przechodzi do ustalonego miejsca w pamięci, które zawiera adres startowy procedury obsługi przerwania.

Po wykonaniu procedury obsługi przerwania, jednostka centralna wznawia przerwane zadanie.

Przykłady przerwań
-nieprawidłowy rozkaz We/Wy,
-kanał We/Wy zakończył operację,
-urządzenie We/Wy zakończyło operację,
-błędny rozkaz procesora centralnego,
-arytmetyczny nadmiar stałoprzecinkowy,
-dzielenie przez zero,
-naruszenie ochrony pamięci,
-przepełnienie zegara interwałowego,
-przerwanie przyciskiem przez operatora,
-zapotrzebowanie na usługę systemową,
-przerwanie w komunikacji między CPU,
-wykrycie możliwości uszkodzenia sprzętu.

Dwa mechanizmy przekazywania sterowania do **procedury obsługi przerwania**:

❶Wywołanie **ogólnej procedury:**
-identyfikuje tylko informacje **opisujące** przerwanie;
-wymaga wywołania szczegółowej procedury **obsługującej** przerwanie.

❷ **Tablica wskaźników** zwana **wektorem przerwań** *(interrupt vector)*

Zawiera adresy procedur obsługujących przerwania, przy założeniu, że liczba możliwych przerwań jest **z góry znana**.

float **(*wFa[3])**(int)

0	adr0	→ **F1Int**
1	adr1	→ **F2Int**
2	adr2	→ **F3Int**

Tablica indeksowana jest jednoznacznym **numerem urządzenia**, które wysłało przerwanie.

Zapewnia to natychmiastowe pobranie adresu do procedury obsługi przerwania, zgłoszonego przez urządzenie.

Współczesne systemy operacyjne obsługują przerwania poprzez **wektor przerwań**.

● **Przechowywanie adresu przerwanego zadania**:

- w ustalonej komórce lub w komórce indeksowanej numerem urządzenia - starsze rozwiązanie.
- na stosie systemowym.

Jeśli procedura obsługi przerwania będzie zmieniać np.: wartości rejestrów procesora, to musi zapamiętać ich stan bieżący, aby przy końcu swojego działa odtworzyć je.

Po obsłużeniu przerwania następuje pobranie do licznika rozkazów zapamiętanego adresu powrotnego i wznowienie przerwanych zadań, tak jakby przerwania nie było.

● **Maskowanie przerwań**

Podczas obsługi jednego przerwania inne są wyłączone *(disabled).*

Nowe przerwanie jest **opóźniane** do czasu zrealizowania obsługi bieżącego przerwania.

Inaczej przetworzenie drugiego przerwania - przy niedokończonym pierwszym - mogłoby zniszczyć dane pierwszego przerwania → spowodować jego utratę *(lost interrupt).*

➔ Obecne rozwiązania zezwalają na obsługę nowego przerwania przed zakończeniem obsługi innego.

Korzysta się ze schematu **priorytetów**, w którym poszczególnym **typom żądań** nadaje się priorytety według ich względnej ważności.

Informacje związane z przerwaniami pamięta się w **osobnym** miejscu dla każdego **priorytetu**.

Przerwanie o **wyższym priorytecie** będzie obsłużone nawet wtedy, gdy jest aktywne przerwanie o niższym priorytecie.

Przerwania **tego samego** lub niższego poziomu będą **maskowane** tj. selektywnie wyłączane.

● **System operacyjny sterowany przerwaniami** *(interrupt driven)* –współczesne rozwiązanie.

SO czeka na jakieś ZDARZENIE jeżeli: - nie ma procesów do wykonania,
-urządzenia We/Wy nie wymagają obsługi,
-użytkownicy nie oczekują odpowiedzi, itd.

Zdarzenia są prawie zawsze sygnalizowane za pomocą **przerwań** lub tzw. **pułapek**.

Pułapka *(trap),* czyli wyjątek, to rodzaj przerwania generowany przez oprogramowanie:

- błąd (dzielenie przez zero, próba niewłaściwego dostępu do pamięci),
- zamówienie z **programu użytkownika**, wymagające obsługi przez SO.

Po wykryciu przerwania (lub pułapki) sprzęt przekazuje sterowanie do SO.

System operacyjny w pierwszej kolejności przechowuje bieżący stan CPU.

Następnie ustala rodzaj powstałego przerwania:

- poprzez **odpytywanie** *(polling),* tj. badania stanu wszystkich urządzeń We/Wy w celu wykrycia potrzebującego obsługi,
- jako wynik zadziałania wektorowego systemu przerwań.

Każdy typ przerwania ma w SO oddzielny segment kodu, odpowiedzialny za jego obsługę.

2.1. Przerwania We/Wy

❒ **Zamawianie** przez **proces użytkownika operacji We/Wy**

Rozpoczynając operację We/Wy CPU ustawia zawartość rejestrów w sterowniku urządzenia.

Sterownik sprawdza dane w tych rejestrach i określa rodzaj działania.

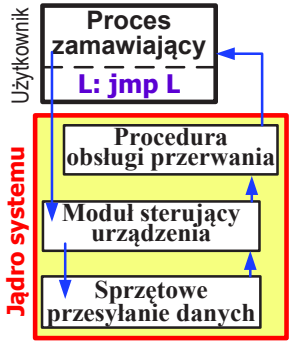
Jeśli sterownik wykryje np.: zamówienie czytania, to rozpocznie przesyłanie danych z urządzenia do swojego lokalnego bufora.

Po przesłaniu danych **sterownik urządzenia** wysyła **przerwanie,** informując CPU, że skończył operację.

Po **rozpoczęciu** operacji We/Wy możliwe są dwa mechanizmy jego realizacji.

❶ **Synchroniczne We/Wy** *(synchronous I/O):*

Przesyłanie danych **rozpoczyna** się i twa aż do **zakończenia**, po czym sterowanie wraca do procesu użytkownika.



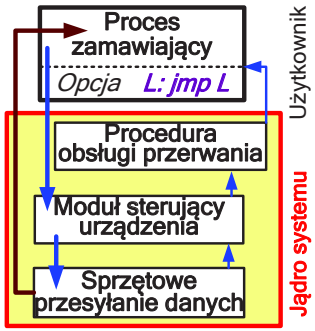
Rys. 2.2a. Synchroniczne

❷ **Asynchroniczne We/Wy** *(asynchronous I/O):*

Następuje **oddanie sterowania** do programu użytkownika **bez czekania** na zakończenie operacji We/Wy.

Operacja We/Wy może być kontynuowana wraz z innymi działaniami systemu.

Zwiększa to wydajność systemu.



Rys. 2.2b. Asynchroniczne

❒ **Czekanie** na **zakończenie transmisji We/Wy**

Synchroniczne We/Wy:

- specjalny rozkaz **Wait** powoduje bezczynność CPU aż do chwili wystąpienia przerwania.
- wykonywanie pętli czekania: ***Loop:*** **jmp** ***Loop*** , powtarzanej do nadejścia sygnału przerwania, który przekaże sterowanie do innej części SO.

W pętli tego rodzaju może powstać **konieczność odpytywania** urządzeń We/Wy, które **nie powodują** przerwań, lecz określają **znacznik** w jednym z ich własnych rejestrów i oczekują, że SO **zauważy zmianę** jego wartości.

Jeżeli CPU czeka na koniec operacji We/Wy, to takie podejście **wyklucza**:

- równoczesną pracę kilku urządzeń,
- możliwość zachodzenia na siebie w czasie: **obliczeń** i **operacji** We/Wy.

Asynchroniczne We/Wy (rozpoczęcie operacji We/Wy i przekazanie sterowania zadaniu)**:**

-jeżeli po rozpoczęciu operacji We/Wy program użytkownika **nie będzie gotowy** do działania, a SO nie będzie ma nic do roboty, to **potrzebny jest** rozkaz czekania lub pętla bezczynności.

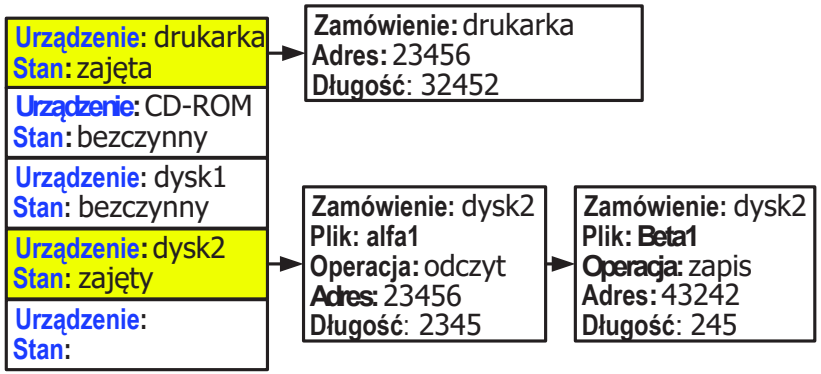
-należy odnotowywać wiele zamówień na operacje We/Wy w tym samym czasie.

System operacyjny posługuje się **Tablicą Stanów Urządzeń** *(device status table)*, której elementy odnoszą się do poszczególnych urządzeń.

Tablica zawiera typ urządzenia, jego adres i stan (odłączone, bezczynne, zajęte).

Jeżeli urządzenie jest zajęte z powodu przyjęcia zamówienia, to odpowiadający mu element **Tablicy Stanów Urządzeń** zawiera rodzaj zamówienia i inne parametry.

Różne procesy mogą składać zamówienia do tego samego urządzenia, SO będzie utrzymywał dla każdego urządzenia kolejkę (listę) oczekujących zamówień.



Rys. 2.3. Tablica stanu urządzeń

Jeśli urządzenie We/Wy wymaga obsługi to wysyła przerwanie:

- Po wystąpieniu przerwania SO określa, które urządzenie spowodowało przerwanie.
- Pobiera z **Tablicy Stanu Urządzeń** informacje o stanie danego urządzenia i zmienia je, odnotowując wystąpienie **przerwania**.
- Zakończenie operacji urządzenia We/Wy również jest sygnalizowane **przerwaniem**.
- Jeśli są następne zamówienia oczekujące na dane urządzenie, to SO rozpoczyna ich realizację.
- Na koniec procedura obsługi przerwania urządzenia We/Wy zwraca sterowanie.

Jeśli na zakończenie działania procedury obsługi przerwania czekał jakiś program (co zostało odnotowane w **Tablicy Stanów Urządzeń**), to oddaje mu sterowanie.

W przeciwnym razie następuje powrót do tego, co było robione przed przerwaniem:

- do **wykonywania** programu użytkownika (program rozpoczął operację We/Wy, operacja ta się zakończyła, a program nie zaczął jeszcze na nią czekać);
- do **pętli czekania** (program zapoczątkował **kilka** operacji We/Wy i czeka na zakończenie **jednej** z nich, lecz to przerwanie pochodziło **od innej**).

W systemie z podziałem czasu SO może rozpocząć wykonywanie innego procesu.

Wiele systemów interakcyjnych umożliwia użytkownikom terminali **pisanie z wyprzedzeniem**, tzn. wprowadzenie danych zanim będą one potrzebne.

Przerwania mogą wtedy sygnalizować nadchodzenie znaków z terminalu, choć blok kontrolny urządzenia będzie wykazywał brak zamówienia na wejście z danego urządzenia ze strony programu.

W takich SO należy zastosować **bufor** na przechowywanie przekazywanych z wyprzedzeniem znaków, w którym będą one czekały na zapotrzebowanie.

2.2. Kanał DMA

Urządzenia takie jak **dyski** mogą przesyłać informacje z dużą szybkością.

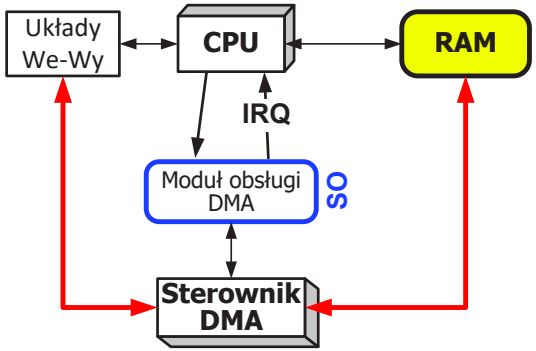
Niech CPU potrzebuje 2μs na obsługę każdego przerwania, a nadchodzą one, co 4μs.

Nie zostaje dużo czasu na wykonywanie procesu.

Problem rozwiązuje **bezpośredni dostęp do pamięci operacyjnej** (**D***irect* **M***emory* **A***ccess*).

Po ustawieniu buforów, wskaźników i liczników sterownik urządzenia przesyła bezpośrednio cały **blok danych** między własnym buforem a PAO - bez udziału CPU.

☛ Przerwanie generowane jest raz na **cały blok danych**, a nie po przesłaniu każdego znaku.



Rys. 2.4. Kanał DMA

Przesłania danych może **zażądać program użytkownika** lub **system operacyjny.**

System operacyjny wybiera bufor (pusty We lub pełny Wy) z kolejki buforów do przesłania.

Moduł obsługi urządzenia z SO, ustawia w rejestrach sterownika DMA adresy źródła, miejsca docelowego i długość transmisji.

Sterownik DMA inicjuje operację We/Wy.

Po przesyłaniu sterownik DMA wysyła CPU przerwanie.

Gdy sterownik DMA jest zajęty przesyłaniem danych,

CPU może wykonywać inne zadania.

2.3. Struktura pamięci

Pamięć operacyjna jest obszarem pamięci bezpośrednio dostępnym dla *CPU*.

Współpraca z PAO odbywa się za pomocą ciągu rozkazów **load** / **store**.

Load pobiera **słowa** z PAO do wewnętrznego rejestru CPU.

Store umieszcza zawartości **rejestru *CPU*** w PAO.

CPU automatycznie pobiera z PAO rozkazy do wykonania.

Cykl rozkazowy w systemie o architekturze von Neumanna:
-pobranie rozkazu z PAO i przesłanie go do rejestru rozkazów,
-dekodowanie rozkazu i realizowanie (może pobrać argumenty z PAO),
-wynik wykonaniu rozkazu z powrotem przechowywany jest w PAO.

Jednostka pamięci „widzi" tylko strumień adresów pamięci.

. Nie jest jej znany sposób, w jaki one powstały (licznik rozkazów, indeksowanie, itp.).
. Nie wie czemu służą (rozkazy lub dane).

➔ **Istotny jest ciąg adresów pamięci generowanych przez wykonywany program.**

Życzenie: program i wszystkie dane stale pozostają w PAO**.**
Na razie jest to niemożliwe, PAO jest za mała.
PAO jest tzw. pamięcią ulotną *(volatile storage)*, zależną od stabilności zasilania.

PAO można **rozszerzyć** stosując dodatkowa pamięć trwałą (dysk magnetyczny).
CPU **nie ma** rozkazów posługujących się adresami dyskowymi.

➔ **Dysk jest źródłem i miejscem przeznaczenia przetwarzanych informacji.**

Rejestry CPU są na ogół dostępne w jednym cyklu zegara.

Dostęp do PAO odbywa się za pośrednictwem transakcji z szyną pamięci.

Może zajmować wiele cykli, wtedy CPU **utyka**, gdyż brakuje danych do zakończenia rozkazu.
Ratunkiem jest wstawienie między CPU a PAO szybkiej **pamięci podręcznej** (***cache***).

● **Plik mapowany** w **PAO**

Pozwala zarezerwować logiczny **obszar przestrzeni adresowej**, przydzielić do niego **pamięć fizyczną** i odwzorować fragment pliku (lub cały plik)

Pamięć fizyczną można odnieść do dowolnego pliku na dysku.

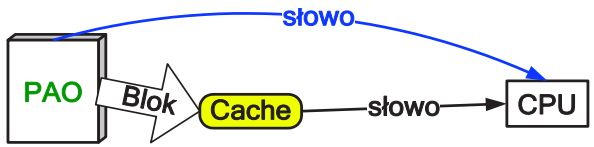
Po zmapowaniu pliku korzysta się z niego tak, jakby dane pliku znajdowały się wyłącznie w PAO.

☛ System używa **plików mapowanych** do ładowania i wykonywania plików .exe i DLL.

☛ Można mapować na przestrzeń adresową procesu pliki danych na dysku.

☛ Dzielenie danych przez wiele procesów działających na tym samym komputerze.

❒ **Pamięć podręczna** *(cache)*



Staranne zarządzanie pamięcią podręczną *(cache management)* jest istotne ze względu na jej ograniczone rozmiary.

Jej wielkość i właściwe algorytmy zastępowania w niej bloku danych może spowodować, że ponad 80% wszystkich dostępów będzie odnosić się do **cache**.

Dane przechowywane są w **PAO** lub na **dysku.**

Przed ich użyciem kopiowane są na okres przejściowy do **cache**.

☛ Gdy potrzebny jest fragment danych, sprawdza się najpierw, czy nie ma go w pamięci cache.

☛ Jeśli nie ma, to pobiera się dane z PAO i umieszcza ich kopię w **cache** przy założeniu, dużego prawdopodobieństwa, że będą one znów potrzebne.

Programista (kompilator) **decyduje, które dane umieści** w **rejestrach** lub w **PAO.**

Istnieją pamięci podręczne zrealizowane w całości sprzętowo, poza kontrolą *SO*.

Większość systemów ma ***podręczną pamięć rozkazów*** do przechowywania następnego rozkazu, przewidywanego do wykonania.

Bez tej pamięci podręcznej CPU musiałoby czekać przez kilka cykli na pobranie rozkazu z PAO.

Pamięć operacyjną można uważać za **szybką pamięć podręczną** dla **pamięci masowej.**

Przesyłanie danych z pamięci podręcznej do ***CPU* i rejestrów jest zwykle funkcją sprzętową, niewymagającą udziału systemu operacyjnego**.

❒ **Dyski magnetyczne**

Szybkość transmisji dyskowej określają dwa czynniki.

❶ **Tempo przesyłania** *(transfer rate):* szybkość przepływu między napędem a komputerem.

❷ **Czas ustalania położenia głowicy** *(positioning time)* lub *(random access time)* składają się z:

czasu wyszukiwania *(seek time)* -czas przesuwania głowicy do odpowiedniego cylindra;

opóźnienie obrotowe *(rotational łatency)* -czas, w którym sektor przejdzie pod głowicą.

Przesyłanie danych szyną We/Wy nadzorują **sterowniki** z wbudowaną pamięcią podręczną.

Sterownik **macierzysty** ***(host** controller)* to sterownik przylegający do szyny w komputerze.

Sterownik **dysku** *(disk controller)* jest wbudowany w każdy napęd dyskowy.

Wykonując dyskową operację We/Wy, komputer umieszcza rozkaz w sterowniku **macierzystym**.

Sterownik **macierzysty** wysyła polecenie do sterownika **dysku**, a ten uruchamia napęd dysku.

Dane w sterowniku **dysku** przesyłane są między pamięcią **podręczną** a powierzchnią dyskową.

Dane po stronie komputera przesyła się między pamięcią **cache** a sterownikiem **macierzystym**.

❒ **Zgodność** i **spójność pamięci** w **hierarchicznej strukturze pamięci**

Te same dane mogą być na różnych poziomach.

0: Registers
1: Cache
2: PAO
3: Disk Cache
4: Magnetic Disk
5: Removable Media

Liczba ***k1***, zawarta w **PLIKU dyskowym** będzie zwiększona o 1.

☛ Operację rozpoczyna kopiowanie bloku dyskowego z liczbą ***k1*** do **PAO**.

☛ Następnie kopiuje się ***k1*** do **pamięci podręcznej**.

☛ Stamtąd kopiuje się do **rejestru** wewnętrznego CPU.

Kopia liczby ***k1*** pojawia się w kilku miejscach.

Po wykonaniu operacji w **rejestrze,** wartość ***k1*** będzie różna w różnych systemach pamięci.

➔ **Stanie się ona taka sama po przekopiowaniu** z **powrotem na dysk.**

Nie ma problemu w środowisku jednozadaniowym, ponieważ dostęp do liczby ***k1*** będzie dotyczyć zawsze jej kopii na **najwyższym** poziomie hierarchii pamięci.

W środowisku wielozadaniowym kilka procesów może sięgać po ***k1***.
Należy zapewnić, że każdy z nich otrzyma jej najnowszą wartość.

W środowisku wieloprocesorowym każde CPU zawiera lokalną ***cache***.

Kopia liczby ***k1*** może istnieć jednocześnie w wielu pamięciach ***cache***.

Różne CPU mogą działać jednocześnie, należy zapewnić, że uaktualnienie wartości ***k1*** w jednej ***cache*** znajdzie natychmiast odbicie innych ***cache***, przechowującym zmienną ***k1***.

Jest to problem **zgodnością pamięci podręcznej** i zazwyczaj rozwiązywany jest sprzętowo.

2.4. Ochrona sprzętowa

W systemie bez podziału, błąd mógł powodować zawieszenie aktualnie wykonywanego programu.

W systemie z podziałem czasu błąd w jednym programie może zagrozić wielu procesom.

W systemach wieloprogramowych błędny program może pozamieniać dane innych programów lub uszkodzić rezydentną część systemu operacyjnego.

Bez ochrony przed błędami komputer musi wykonywać w danej chwili tylko jeden proces albo wszystkie wyniki należy uznać za podejrzane.

SO musi gwarantować, że błędny program nie będzie zakłócać działania innych programów.

Wiele błędów programowania wykrywa sprzęt.

⇐ Tymi błędami zajmuje się na ogół SO.

Gdy program użytkownika, próbuje wykonać niedozwolony rozkaz lub sięgnąć do pamięci nienależącej do jego przestrzeni adresowej, ***wpadnie w pułapkę*** zastawioną przez sprzęt, co oznacza przejście do SO za pomocą wektora przerwań.

Gdy wystąpi błąd w programie, SO wymusza nienormalne zakończenie programu.

Zdarzenie takie obsługuje ten sam kod co żądanie nienormalnego zakończenia programu pochodzące od użytkownika.

Pojawia się komunikat o błędzie, po czym następuje składowanie pamięci programu.

Obraz pamięci programu jest zazwyczaj zapisywany w pliku.

❒ **Dualny tryb operacji**.

Ochroną muszą być objęte wszystkie wspólnie wykorzystywane zasoby systemu.

➔ **Sprzęt rozróżnia dwa oddzielne tryby pracy:**

-tryb **użytkownika** *(user mode)*

-tryb **monitora** *(monitor mode)* zwany **trybem systemu** (*system mode*).

W sprzęcie istnieje **bit trybu** *(mode bit),* którego stan wskazuje bieżący tryb pracy:

monitor = 0;

użytkownik = 1.

Odróżnia się działania wykonywane na zamówienie SO od działań na zamówienie użytkownika.

Rozruch systemu przebiega w trybie monitora.

Załadowany system operacyjny uruchamia procesy użytkowe w trybie użytkownika.

☛ Każdorazowo po wystąpieniu pułapki lub przerwania sprzęt zmienia tryb pracy z trybu **użytkownika** na tryb **monitora** (zmienia wartość bitu).

Gdy SO przejmuje sterowanie komputerem jest on w trybie monitora.

Przejście do programu użytkownika poprzedzone jest zawsze przełączeniem przez system trybu pracy na tryb **użytkownika** (ustawiając bit trybu na 1).

➔ **Rozkazy uprzywilejowane** *(privileged instructions)* to potencjalnie niebezpieczne rozkazy kodu maszynowego.

● Ich wydzielenie daje dalszą ochronę.

Przykładowe rozkazy: -włączanie i wyłączanie systemu przerwań,

-modyfikacja rejestrów zarządzania pamięcią lub rejestrów czasomierza,

-rozkaz **halt**.

Sprzęt pozwala wykonywać rozkazy uprzywilejowane tylko w trybie monitora.

Próba wykonania rozkazu uprzywilejowanego w trybie użytkownika **nie jest** realizowana.

Sprzęt potraktuje rozkaz, jako niedopuszczalny i spowoduje **awaryjne przejście do SO**.

Już procesor Intel **80486** posiadał dualny tryb operacji.

❒ **Ochrona We/Wy**

Program użytkownika może zakłócić działanie systemu, wydając niedozwolony rozkaz We/Wy:

- poprzez dotarcie do komórek pamięci w obrębie samego SO,
- nie zwalniając procesora.

Wszystkie rozkazy We/Wy są uprzywilejowane.

Separują programy użytkownika od wykonywania jakichkolwiek operacji We/Wy - zarówno niedozwolonych jak i dozwolonych.

Jak program użytkownika może wykonać operację We/Wy?

Użytkownik musi **poprosić *system***, aby wykonał operację **We/Wy**.

Jest to **wywołanie systemowe** *(system call)* lub **wywołanie funkcji systemu operacyjnego**.

Wywołanie **systemowe** realizowane jest zależnie od właściwości CPU:

-przejście do określonej komórki w wektorze przerwań, `

-za pomocą ogólnego rozkazu **trap**,

-niektóre systemy zawierają rozkaz **syscall**.

Wywołanie systemowe traktowane jest **jak przerwanie programowe.**

Obsługa wywołania systemowego jest częścią systemu operacyjnego.

Za pośrednictwem wektora przerwań sterowanie przekazywane jest do odpowiedniej procedury obsługi w SO, a bit trybu zostaje przełączony w tryb **Monitora**.

W trybie **Monitora** sprawdzany jest rozkaz przerywający, aby określić, które wywołanie systemu miało miejsce.

Rodzaj usługi zgłaszanej przez użytkownika określa parametr wywołania systemowego.

Dodatkowe informacje związane z usługą mogą być przekazane za pomocą rejestrów lub pamięci .
(za pomocą umieszczonych w rejestrach wskaźników do komórek pamięci).

Tryb monitora wykonuje zamówienie i przekazuje sterowanie do rozkazu po wywołaniu systemowym.

Należy zapewnić, że **program użytkownika nigdy nie wejdzie w tryb pracy monitora.**

Niech komputer pracuje w trybie użytkownika.

Przechodzi on w tryb monitora przy każdym wystąpieniu przerwania lub pułapki, wykonując skok pod adres określony w wektorze przerwań.

. Załóżmy, że program użytkownika umieścił nowy adres w wektorze przerwań.

. Ten nowy adres mógłby wskazywać miejsce w programie użytkownika.

Wówczas po wystąpieniu przerwania sprzęt przełączyłby komputer w tryb monitora i przekazał sterowanie według zmienionego wektora przerwań do programu użytkownika.

Program użytkownika przejąłby kontrolę nad komputerem w trybie monitora.

❒ Ochrona pamięci

- Należy chronić wektor przerwań przed zmianami, które mógłby wprowadzić program użytkownika.
- Należy chronić systemowe procedury obsługi przerwań.

Inaczej program użytkownika mógłby przechwycić sterowanie od procedury obsługi przerwania, pracującej w trybie monitora.

Nawet, jeśli użytkownik nie uzyskałby możliwości sterowania pracą komputera, to zmiany w procedurach obsługi przerwań zakłóciłyby właściwe działanie systemu.

Aby oddzielić od siebie obszary pamięci **każdego programu**, należy ustalić zakresy **dopuszczalnych** adresów programu i chronienia pamięci poza tymi adresami.

Ochronę tego rodzaju uzyskuje się za pomocą rejestrów: **bazowego** i **granicznego**.

Rejestr bazowy: **najmniejszy** dopuszczalny **adres** pamięci.

Rejestr graniczny: rozmiar obszaru pamięci.

Rejestr **bazowy** = **33333**,

Rejestr **graniczny** = **99999**;

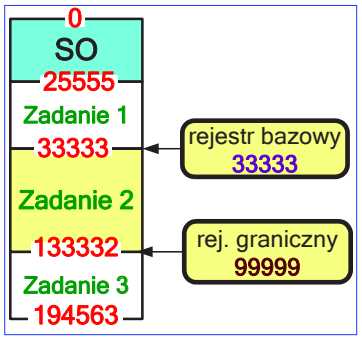
Jako poprawne w programie mogą wystąpić odniesienia do adresów od **33333** do **133332**.

Ochronę realizuje **sprzęt CPU**:

porównuje **każdy** **Adres** powstały w trybie pracy z zawartością w/wm **rejestrów**.

BAZA ≤ Adres ≤ BAZA + Graniczny → OK

Próba programu pracującego w trybie **użytkownika** uzyskania dostępu do pamięci SO lub do programu innego użytkownika kończy się przejściem do SO, i wygenerowanie błędu.



Zawartości rejestru bazowego i granicznego określa SO poprzez **rozkazy uprzywilejowane**.

Rozkazy **uprzywilejowane** można wykonać tylko w trybie **monitora**.

Tylko system operacyjny może załadować rejestr **bazowy** i **graniczny**.

SO działając w trybie monitora, ma nieograniczony dostęp do swojej pamięci, jak i do pamięci użytkowników; w razie wystąpienia błędów może składować te obszary.

❒ Ochrona jednostki centralnej

Należy mieć pewność, że SO utrzymuje stałą kontrolę nas systemem komputerowym.

Program użytkownika nie może wpaść w nieskończoną pętlę, gdyż grozi to odebraniem sterowania SO na zawsze.

Osiąga się to przez zastosowanie czasomierza, ustawionego tak, aby generował w komputerze przerwanie po wyznaczonym okresie.

Okres ten może być stały lub zmienny.

Odmierzanie zmiennych okresów implementuje się za pomocą zegara stałookresowego i licznika.

System operacyjny ustawia **licznik**.

Przy każdym tyknięciu zegara następuje zmniejszenie licznika.

➔ Z chwilą **wyzerowania** **licznika** powstaje przerwanie.

Dziesięciobitowy licznik z jednomilisekundowym zegarem umożliwia przerwania w odstępach od 1 do 1024 ms, z przyrostem co 1 ms.

Przed oddaniem sterowania do programu użytkownika SO ustawia czasomierz na przerwanie.

Gdy czasomierz wygeneruje przerwanie, sterowanie wraca automatycznie do SO, który może uznać:

- przerwanie za błąd,
- przyznać programowi więcej czasu.

Rozkazy modyfikujące działanie czasomierza są zastrzeżone na użytek monitora.

- **Czasomierz zapobiega** zbyt **długiemu działaniu programu użytkownika.**

W liczniku pamięta się, ile czasu zostało przydzielone programowi na wykonanie.

Program z siedmiominutowym przydziałem czasu może mieć licznik zainicjowany na 420.

Co sekundę czasomierz generuje przerwanie, a licznik **zmniejsza** się o 1.

Dopóki licznik jest dodatni, dopóty sterowanie powraca do programu użytkownika.

Gdy licznik stanie się **ujemny**, SO zakończy program z powodu ***przekroczenia*** przydzielonego limitu czasu na jego realizacje..

● Czasomierz realizacje podział czasu.

Czasomierz można nastawić na przerywanie co każde **N** ms,

gdzie **N** jest **kwantem czasu,** przydzielanym każdemu użytkownikowi, bądź programowi (procesowi) na działanie.

Przełączanie kontekstu *(context switch):* system operacyjny odświeża stany rejestrów, zmiennych wewnętrznych i buforów oraz zmienia inne parametry, przygotowując następnemu programowi przestrzeń.

Po zmianie kontekstu następny program kontynuuje swoją pracę od miejsca, w którym został przerwany (po wyczerpaniu ostatniego przydzielonego kwantu czasu).

SO jest wywoływany (po upływie kwantu czasu) w celu wykonania prac administracyjnych, jak dodanie wartości **N** do rekordu określającego ilość czasu (w celach rozliczeniowych), którą program użytkownika zużył do tej pory.

● Czasomierz oblicza bieżący czas.

Przerwania czasomierza sygnalizują upłynięcie pewnej jednostki czasu, co pozwala SO na obliczanie bieżącego czasu w odniesieniu do pewnej wartości początkowej.

Jeśli przerwania następują, co 1 s i wystąpiło ich 1427, od 13^{00} to obecnie jest godzina 13:23:47.

☛ Wskazywanie **dokładnego** czasu wymaga bardziej precyzyjnych obliczeń, ponieważ **czas przetwarzania** przerwań **powoduje opóźnianie** takiego zegara.

Większość komputerów ma sprzętowy zegar czasu rzeczywistego, na który SO nie ma wpływu.

ANEX 2.1. Przerwania w Windows

Obsługą pułapek powodowanych wyjątkami i przerwaniami generowanymi przez sprzęt lub oprogramowanie zajmuje się **jądro** SO.

System Windows określa kilka wyjątków niezależnych od architektury, m.in.:

-niedozwolona próba odwołania do pamięci,
-nadmiar lub niedomiar zmiennopozycyjny,
-dzielenie zmiennopozycyjne przez zero,
-złe przyleganie danych,
-błąd czytania strony,
-przekroczenie rozmiaru zbioru stronicowania,
-osiągnięcie punktu kontrolnego przez program uruchomieniowy (debuger)
-nadmiar stałopozycyjny,
-dzielenie całkowite przez zero,
-niedozwolony rozkaz,
-rozkaz uprzywilejowany,
-próba naruszenia ochrony strony,

Dyspozytor przerwań w jądrze obsługuje przerwania, wywołując procedury obsługi przerwań (np. moduły obsługi urządzeń) lub wewnętrzne procedury jądra.

Przerwanie jest reprezentowane przez obiekt przerwania, który zawiera wszystkie informacje potrzebne do obsługi przerwania. Posługiwanie się obiektem ułatwia kojarzenie procedur obsługi przerwania z przerwaniem, eliminując konieczność bezpośredniego dostępu do sprzętu powodującego przerwanie.

Ze względu na przenośność SO dyspozytor przerwań odwzorowuje przerwania sprzętowe w standardowy zbiór. Przerwania mają priorytety i są obsługiwane w ich kolejności.

W systemie Windows XP są 32 poziomy przerwań (IRQL).

Osiem poziomów jest zarezerwowanych do użytku jądra; na pozostałych 24 poziomach są rozlokowane przerwania sprzętowe warstwy HAL.

Przerwania systemu Windows XP zdefiniowane są w poniższej tabeli.

Poziomy przerwań	Rodzaje przerwań
31	sprawdzanie procesora lub błąd szyny
30	awaria zasilania
29	powiadomienie międzyprocesorowe (żądanie uaktywnienia innego procesora, np. wyekspediowanie do niego procesu lub aktualizacja TLB)
28	zegar (używany do pomiaru czasu)
27	profil
3-26	typowe przerwania sprzętowe komputera PC
2	ekspedycja i opóźnione wywołanie procedury (kernel), (DPC – Dispatch Procedure Call)
1	asynchroniczne wywołanie procedury (APC - Asynchronous Procedure Call)
0	pasywne

Aby powiązać przerwanie dowolnego poziomu z procedurą obsługową, jądro korzysta z tablicy przerwań.

W komputerze wieloprocesorowym system Windows przechowuje oddzielną tablicę przerwań dla każdego procesora, a poziom przerwań (IRQL) może być w celu ich maskowania ustalany niezależnie dla każdego procesora.

Przerwania występujące na poziomie równym lub niższym niż IRQL danego procesora będą blokowane aż do obniżenia IRQL przez wątek z poziomu jądra.

Windows wykorzystuje ten mechanizm do stosowania przerwań programowych w celu wykonywania funkcji systemowych.

Jądro dla przykładu używa przerwań programowych, gdy :

-należy rozpocząć ekspedycję wątku,
-do obsługi czasomierzy,
-umożliwiania działań asynchronicznych.