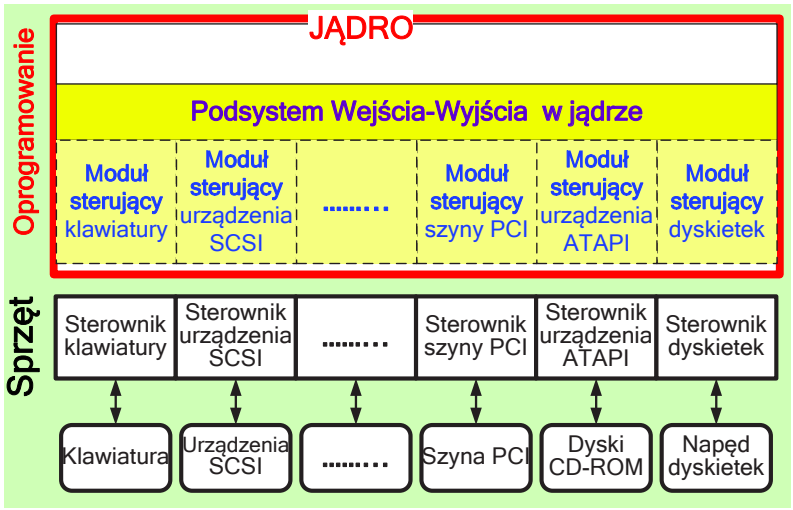
13. SYSTEM WEJŚCIA-WYJŚCIA

Zadania komputera: -przetwarzanie informacji,
-obsługa urządzeń We/Wy.

Metody sterowania urządzeń tworzą w jądrze *Podsystem Wejścia/Wyjścia (I/O subsystem),* który oddziela resztę jądra od samego zarządzania We/Wy.

Moduły sterujące *(device drivers)* -zawarte w jądrze SO- odzwierciedlają cechy indywidualne różnych urządzeń i tworzą jednolity interfejs dostępu do **Podsystemu We/Wy**.



Rys. 13.1. Struktura We/Wy

Warstwa modułów sterujących ukrywa różnice między sterownikami urządzeń przed Podsystemem Wejścia/Wyjścia w jądrze.

Uniezależnienie podsystemu Wejścia/Wyjścia od sprzętu upraszcza projektowanie SO.

Dostęp aplikacji do modułów sterujących odbywa się wg **pewnego standardu**, charakterystycznego dla określonych typów urządzeń (dostęp: blokowy, znakowy, do gniazd sieciowych).

Większość SO zawiera **system obejścia** *(escape system)* standardu, przekazujący w sposób przezroczysty polecenia od aplikacji do modułu sterującego.

W UNIX funkcja systemowa **ioctl** (*I/O control),* umożliwia aplikacji dostęp do funkcji modułu sterującego, ***bez konieczności*** nowego **odwołania** do **systemu**.

Argumenty funkcji systemowej **ioctl**:

1. **deskryptor pliku** –łączy aplikację z modułem sterującym; bezpośrednie odniesienie do urządzenia sprzętowego zarządzanego przez moduł.
2. **liczba całkowita** –identyfikator poleceń dostępnych w module sterującym.
3. **wskaźnik** –na strukturę danych w PAO do składowania/pobierania informacji.

❒ Urządzenia blokowe

Interfejs urządzenia blokowego rozpoznaje polecenia typu:

Czytaj **Pisz** **Szukaj**.

Dostęp do urządzenia w aplikacjach odbywa się za pomocą **Interfejsu Systemu Plików**.

- **Surowe Wejście/Wyjście**: aplikacje specjalne (np. bazy danych), mogą preferować dostęp do urządzenia blokowego widzianego, jako Liniowa Tablica Bloków.
- **Mechanizm** plików **odwzorowywanych** w **pamięci** może tworzyć warstwę **powyżej** modułów sterujących urządzeń blokowych.
  - Operację Czytaj/Pisz zastępuje mechanizm dostępu do pamięci dyskowej poprzez Tablicę Bajtów w PAO.
  - Wywołanie systemowe, odwzorowujące plik w PAO przekazuje adres wirtualny Tablicy Bajtów zawierającej kopię pliku.

> We/Wy odwzorowywane w PAO jest efektywne, gdyż przesyłanie odbywa się za pomocą mechanizmu dostępu do pamięci Wirtualnej stronicowanej na żądanie.

❒ Urządzenia sieciowe

Adresowanie sieciowe We/Wy różni się od We/Wy dyskowego.

SO udostępnia **interfejs gniazda sieciowego**.

Różni się od interfejsu **Szukaj** – **Czytaj** - **Pisz** odnoszącego się do dysków.

Funkcja **Wybierz***(selekt)* -z interfejsu gniazda sieciowego- zarządza zbiorem gniazd.

Zwraca informację, które z gniazd mają nieodebrane pakiety, a które mogą przyjąć nowy pakiet.

Funkcja **Wybierz** uwalnia od konieczności odpytywania i aktywnego czekania.

❒ Zegary i czasomierze

Podstawowe funkcje sprzętowych zegarów i czasomierzy:

1. podawanie bieżącego i upływającego czasu,
2. uruchomienie wykonania operacji **X** *w* chwili **T**

Czasomierz programowalny *(programmable interval timer)*, sprzęt służący do:

-pomiaru czasu,
-uruchomiania wykonywania operacji.

Z mechanizmu czasomierza korzysta:

planista przydziału CPU, generując przerwania wywłaszczające proces po wyczerpaniu kwantu czasu (przerwania administracyjne).

podsystem dyskowy We/Wy, okresowo kopiujący na dysk bufory podręczne,

podsystem sieciowy kasujący operacji, które są wykonywane za wolno.

System operacyjny może udostępniać interfejs czasomierza procesom użytkowym.

Symulując zegary wirtualne, SO może obsługiwać więcej zamówień związanych z pomiarem czasu niż wynosi liczba dostępnych kanałów sprzętowych czasomierza.

Jądro (moduł sterujący czasomierza) utrzymuje wykaz przerwań zgłoszonych przez własne procedury i zamówienia użytkowników, uporządkowany w kolejności od najwcześniejszego.

Jądro nastawia czasomierz na najwcześniejszy termin z wykazu.

Jądro sygnalizuje wystąpienie przerwania zamawiającemu i przeładowuje rejestry czasomierza na następny, najbliższy termin.

❒ **We/Wy** z **blokowaniem** oraz **bez blokowania** (asynchroniczne).

◗ **Blokowane wywołanie systemowe:** program użytkowy wykonujący ten typ operacji zostaje wstrzymany, i przeniesiony do kolejki procesów **Czekających**.

Po zakończeniu funkcji systemowej program wraca do kolejki procesów **Gotowych** i z chwilą podjęcia działania otrzymuje wartości od funkcji systemowej.

W interfejsach użytkowych SO często występują odwołania do systemu z blokowaniem.

Fizyczne czynności wykonywane przez urządzenia We/Wy są z reguły asynchroniczne.

➔ Niektóre procesy użytkowe wymagają We/Wy **bez** blokowania:

-interfejs użytkownika, w którym sygnały z klawiatury przeplatają się z wyświetlaniem danych na ekranie,

-aplikacja czytająca z pliku, a jednocześnie wyświetla informacje na ekranie.

➔Aplikacje wielowątkowe umożliwiają zachodzenia na siebie obliczeń i operacji We/Wy.

Część wątków będzie się odwoływać do systemu w sposób **blokowany**, inne będą kontynuować działanie.

◗ **Asynchroniczne wywołanie systemowe** (bez blokowania).

Powrót po asynchronicznym odwołaniu do systemu następuje natychmiast, **bez** czekania na zakończenie operacji We/Wy.

➔ Aplikacja kontynuuje wykonywanie swojego kodu.

Zakończenie operacji We/Wy (w jakiś czas potem) sygnalizowane jest aplikacji przez:

-ustawienie pewnej zmiennej w przestrzeni adresowej aplikacji,

-przekazanie odpowiedniego sygnału,

-przerwania programowe,

-przywołanie *(callback routine),* wykonywane poza porządkiem obowiązującym w aplikacji.

◗ **Zwielokrotnione czytanie** z **blokowaniem**.

W **jednym** odwołaniu do systemu określa się **potrzebną** liczbę operacji Czytania odnoszących się do **kilku** urządzeń,

powrót z wywołania następuje wówczas, gdy **którakolwiek** z tych operacji zakończy działanie.

13.1. Podsystem **W**ejścia-**W**yjścia w ***jądrze***

Jądro SO udostępnia usługi dotyczące We/Wy takie jak:

-planowanie We/Wy,

-buforowanie,

-przechowywanie podręczne,

-spooling,

-rezerwowanie urządzeń i obsługę błędów.

❒ **Planowanie Wejścia/Wyjścia** określa optymalny porządek ich wykonywania.

Cel: -poprawić wydajność ogólną systemu,

-polepszyć korzystanie wspólne z urządzeń,

-zmniejszyć średni czas oczekiwania na zakończenie operacji We/Wy.

Ramię dysku jest na początku dysku, trzy procesy zamawiają Czytanie z blokowaniem.

P1 żąda bloku na końcu dysku,

P2 na początku,

P3 w środku dysku.

Obsługując procesy w kolejności **P2, P3, P1** SO może zmniejszyć drogę ramienia dysku.

Proces wywołujący **blokowaną** operację We/Wy jest umieszczony w **kolejce** do urządzenia.

☛ Planista We/Wy może **zmienić porządek** w kolejce, aby polepszyć sprawności systemu.

➔ SO może dbać, aby: -każdy proces był obsługiwany **sprawiedliwie**,

-w pierwszej kolejności obsługiwane były **pilne** zamówienia.

❒ **Buforowanie**

Bufor to obszar pamięci, w którym przechowuje się dane przesyłane między dwoma urządzeniami lub między urządzeniem a aplikacją.

Trzy Powody buforowania

❶ **Dysproporcje** między **szybkościami producenta** i **konsumenta strumienia danych.**

Podwójne buforowanie:

-urządzenie zapisuje napływające dane do ***pierwszego*** bufora;

-po jego zapełnieniu następuje **zamówienie** operacji **Pisania** na dysku.

Urządzenie zapełnia teraz ***drugi*** bufor.

Gdy operacja Pisania z ***1-go*** bufora na dysku zakończy się, następuje przełączenie do ***1-go*** bufora, zaś dysk zapisuje informacje z ***drugiego*** bufora.

❷ **Dopasowanie urządzeń** o różnych rozmiarach przesyłanych jednostek danych.

❸ **Zapewnienie semantyki kopii** na **Wejściu** i **Wyjściu aplikacji.**

Aplikacja chce zapisać **bufor** danych na dysku.

Wywołuje funkcję systemową **Pisz**, podając wskaźnik do **bufora** i liczbę bajtów.

Co się stanie, jeśli **po wywołaniu** funkcji systemowej aplikacja zmieni zawartość bufora **?**

Semantyka kopii gwarantuje, że wersja danych zapisana na dysku będzie wersją ***z chwili odwołania*** się przez aplikację do **systemu**, niezależnie od późniejszych zmian w buforze aplikacji.

Można przekopiować dane aplikacji przez funkcję systemową **Pisz** do **bufora** w **jądrze**, zanim nastąpi przekazanie sterowania do aplikacji.

☛ Pisanie na dysku wykorzystuje **bufor** w **jądrze**, zatem zmiany w buforze aplikacji nie wywołają żadnych skutków.

❒ **Pamięć podręczna** → przechowuje **kopię** obiektu danych, który przebywa **gdzie indziej**.

☛ Bufor może zawierać ***jedyną***, istniejącą kopię danych.

SO stosuje **bufory** w PAO (traktując je jak pamięci podręczną) aby:
-realizować wymogi semantyki kopii,
-skutecznie planować operacje dyskowe.

Bufory polepszają wydajność operacji We/Wy na plikach:
-dzielonych przez procesy,
-zapisywanych i zaraz potem czytanych.

☛ Gdy jądro otrzyma zamówienie na plikową operację We/Wy, wówczas sprawdza:

czy w buforze pamięci podręcznej w PAO nie ma potrzebnego fragmentu pliku, jeśli jest, to **unika** się fizycznej transmisji dyskowej.

Pamięć podręczna **buforów** gromadzi na **kilka** sekund wyniki dyskowych operacji Pisania, aby zebrać dużą porcje informacji do przesłania.

❒ **Spooling** i **rezerwowanie urządzeń**

Spooling *(**S**imultaneous **p**eripheral **o**peration **o**n-**l**ine)* to użycie **bufora** do przechowania danych, gdy urządzenie, nie dopuszcza przeplatania danych w kierowanym do niego strumieniu.

System operacyjny przechwytuje wszystkie informacje kierowane na drukarkę.

Dane wyjściowe każdej aplikacji gromadzone są w **osobnym** **pliku-buforze**.

Gdy aplikacja skończy drukować, system spoolingu ustawia plik z **obrazem** drukowanych wyników w kolejce do drukarki.

Pliki są kopiowane na drukarce po kolei.

◗ W jednych **SO** **spooling** **zarządzany jest** przez **proces systemowego demona.**

Demon (*daemon*): trwały proces systemowy, na przemian pozostający w uśpieniu lub pobudzany do działania przez zdarzenia systemowe.

◗ W innych **SO** **spooling** **jest obsługiwany przez wątek** w **jądrze.**

W obu przypadkach SO dostarcza interfejsu sterującego, umożliwiającego użytkownikom i administratorom wyświetlanie kolejki, usuwanie niepotrzebnych zadań przed wydrukowaniem, itp.

Windows udostępnia funkcje systemowe umożliwiające czekanie na urządzenia.

Wywołanie systemowe **Open** otwierające plik ma parametr określający rodzaj dostępu, na który zezwala się innym, współbieżnym wątkom.

☛ Unikanie zakleszczeń należy w tych systemach do obowiązków aplikacji.

❒ **Obsługa błędów**

Systemowe wywołanie We/Wy powoduje zwrotne przekazanie jednego bitu informacji, określającej skutek wywołania: sukces lub niepowodzenie.

W UNIX zastosowano dodatkowo zmienną całkowitą o nazwie **errno,** za pośrednictwem, której jest przekazywany kod błędu (około stu wartości informujących o charakterze błędu).

Awaria urządzenia standardu SCSI jest komunikowana za pomocą protokołu SCSI w formie **„klucza diagnostycznego"** (*sense key),* który identyfikuje ogólny rodzaj błędu, jak błąd sprzętowy lub niedozwolone zamówienie.

Dodatkowy kod diagnostyczny *(additional sence code)* informuje o kategorii błędu, np. zły parametr polecenia.

Dalsze kody diagnostyczny zawierają więcej szczegółów.

❒ **Struktury danych jądra**

Uniksowy system plików udostępnia różne obiekty np.: -przestrzenie adresowe procesów,
-pliki użytkowników,
-surowe urządzenia.

☛ Każdy z tych obiektów realizację operacji **Czytaj**, lecz o ***różnej semantyce***.

Czytając plik **Użytkownika**, **jądro** **sprawdza** bufor pamięci podręcznej, przed wykonaniem operacji czytania z dysku.

Czytając z **dysku**, **jądro** musi mieć **pewność**, że rozmiar zamówienia jest wielokrotnością rozmiaru sektora dyskowego i czy leży w granicach sektorów.

Czytając obraz Procesu wystarczy przekopiować dane z pamięci operacyjnej.

Różnice w semantyce maskuje jednolita struktura metod obiektowych.

W Windows We/Wy zrealizowane jest na bazie **przekazywania** komunikatów.

Zamówienie operacji We/Wy **przekształca** się na **komunikat** wysyłany za pośrednictwem **jądra** do zarządcy We/Wy

→ stamtąd do modułu sterującego,

→ każdy z pośredników może zmienić zawartość komunikatu.

Dla operacji **Wy**jścia komunikat zawiera **dane** do pisania.

Komunikat na **We**jściu zawiera **bufor** przeznaczony na przyjęcie danych.

Podsystem **W**ejścia-**W**yjścia nadzoruje:

- zarządzanie przestrzenią nazw plików i urządzeń;
- przebieg dostępu do plików i urządzeń;
- poprawność operacji (drukarka nie może składować);
- przydzielanie miejsca w systemie plików oraz przydział urządzeń;
- buforowanie, przechowywanie podręczne oraz spooling;
- planowanie operacji We/Wy;
- doglądanie stanu urządzeń, obsługę błędów, czynności naprawcze po awarii;
- konfigurowanie i wprowadzanie w stan początkowy modułu sterującego.

13.2. Przekształcanie zamówień We/Wy na operacje sprzętowe

Czytanie pliku z dysku.
Program użytkowy odwołuje się do danych przez **nazwę pliku**.

W MS-DOS **nazwa** jest odwzorowywana na **liczbę** wskazującą pozycję w **Tablicy** dostępu do pliku (FAT), a zawarty w tej pozycji wpis określa, które bloki dyskowe są przydzielone do pliku.

W UNIX nazwa jest odwzorowywana na numer **i-węzła**, który zawiera informacje o przydziale miejsca na dysku.

❒ **Powiązanie nazwy pliku** ze **sterownikiem dysku**

MS-DOS.

Pierwsza część nazwy pliku jest napisem identyfikującym konkretną jednostkę sprzętu.
To, że **c:** *oznacza* podstawowy dysk twardy, jest wbudowane w SO;
napis ten jest odwzorowany na określony adres portu za pomocą **Tablicy urządzeń**.
Dwukropek oddziela **przestrzeń nazw** urządzeń od **przestrzeni nazw** systemu plików.

UNIX.

Nazwy urządzeń są pamiętane w przestrzeni nazw systemu plików.
Żadna część nazwy ścieżki nie jest przeznaczona na nazwę urządzenia.
Dostępna jest **Tablica Montaży**, która wiąże przedrostki nazw ścieżek z nazwami urządzeń.
UNIX dopasowuje do nazwy ścieżki w **Tablicy Montaży** najdłuższy przedrostek;
odpowiadający mu wpis w **T**ablicy **M**ontaży zawiera nazwę urządzenia.

Otrzymana nazwa również ma postać nazwy z przestrzeni systemu plików.
Poszukiwania jej w strukturze katalogowej systemu plików daje **numer urządzenia** w postaci pary <***starszy, młodszy***>.

Starszy numer identyfikuje **moduł sterujący**, który należy wywołać w celu obsługi operacji We/Wy tego urządzenia.

Młodszy numer przekazywany jest do modułu sterującego, i jest **indeksem** w **Tablicy Urządzeń**. Wpis w **Tablicy Urządzeń** zawiera poszukiwany adres portu lub odwzorowany w pamięci adres sterownika urządzenia.

◗ Niektóre systemy operacyjne ładują moduły sterujące na życzenie.

Podczas rozruchu system sprawdza najpierw szyny sprzętowe, aby określić, jakie urządzenia są obecne;
-po, czym wprowadza do pamięci niezbędne moduły sterujące;
-robi to natychmiast lub przy pierwszym wystąpieniu zamówienia na operację We/Wy.

◗ System UNIX w wersji V używa mechanizmu strumieniowego.

Umożliwia on aplikacji dynamiczne zestawianie kodu modułów obsługi w potoki.

Strumień to połączenie między modułem sterującym a procesem poziomu użytkownika.
Składa się z:
-czoła strumienia *(stream head),* będącego interfejsem z procesem użytkownika,
-**zakończenia sterującego** *(driver end)* nadzorującego urządzenie.
Między nimi może występować kilka **modułów strumienia** *(stream modules).*

Moduły można umieszczać w strumieniu, aby dodawać do nich funkcje warstwowo.

☛ Proces może **Otworzyć** port szeregowy urządzenia za pośrednictwem strumienia, do którego **Wstawiono** moduł analizy nadchodzących danych.

13.3. Wydajność

Procesy We/Wy mają istotny wpływ na wydajność systemu.

-procesor **wykonuje** kod modułu sterującego, realizuje planowanie, blokowanie i odblokowanie procesów ➔ zmiany **kontekstów obciążają** CPU i jego pamięć podręczną.

-**operacje** We/Wy **spowalniają** kopiowanie danych szyną między sterownikami a PAO oraz kopiowanie między buforami jądra a przestrzenią danych aplikacji.

-obsługa przerwania jest zadaniem względnie kosztownym; przerwanie powoduje zmianę stanu systemu, wykonanie procedury obsługi przerwania i odtworzenie stanu.

-zakończeniu operacji We/Wy towarzyszy odblokowanie procesu z przełączaniem kontekstu.

Programowanie We/Wy może być wydajniejsze niż sterowane przerwaniami, jeżeli liczba cykli zużywanych na aktywne czekanie nie jest zbyt duża.

✹ **Ruch** w **sieci może powodować częste zmiany kontekstu.**

Zdalne rejestrowanie się na odległym komputerze poprzez inny komputer.

Napisano znak na lokalnej maszynie: -przerwanie klawiaturowe,
-procedura obsługi przerwania przekazuje znak do modułu sterującego w **jądrze**, i dalej do procesu użytkownika.

Proces użytkownika wywołuje systemową operację ***sieciowego*** *We*/Wy.

Znak przechodzi do **lokalnego jądra**, a stamtąd, przez warstwy ***sieciowe*** trafia do modułu obsługi urządzenia sieciowego.

Moduł sterujący urządzenia sieciowego przekazuje pakiet do sterownika sieci, który wysyła znak i generuje przerwanie.

Przerwanie przekazywane jest z powrotem przez **jądro**, aby zakończyć systemową operację ***sieciowego*** We/Wy.

Sprzęt sieciowy zdalnego systemu odbiera pakiet i powoduje przerwanie.

Znak zostaje wydobyty z protokołów sieciowych i przekazany do **demona sieci**.

☛ W/wym przepływowi informacji towarzyszą zmiany kontekstów i stanów.

W Solaris wykorzystano **wątki jądra** do implementacji demona usługi ***Telnet***, aby wyeliminować przełączeń kontekstu przy każdym przeniesieniu znaku między demonami a jądrem.

W innych systemach do terminalowych operacji We/Wy używa się oddzielnych ***procesorów czołowych*** *(front end processors)* co zmniejsza obciążenie CPU wywołane obsługą przerwań.

Koncentrator terminali *(terminal concentrator)* może obsługiwać w jednym porcie komputera dane pochodzące z setek zdalnych terminali.

Kanał We/Wy jest specjalizowanym procesorem, występującym w komputerach głównych, który przejmuje od procesora głównego obciążenia związane z operacjami We/Wy.

Kanał We/Wy może wykonywać bardziej ogólne programy aniżeli sterowniki urządzeń i sterowniki bezpośredniego dostępu do pamięci

➔ **Kanały We/Wy dają się stroić.**

Metody poprawy wydajności We/Wy:

zmniejszyć liczbę przełączeń kontekstu.

zmniejszyć liczbę kopiowań danych podczas przekazywania ich od urządzenia do aplikacji.

zmniejszyć częstość występowania przerwań przez stosowanie dużych przesłań.

zwiększyć współbieżność za pomocą sterowników pracujących w trybie DMA lub kanałów w celu uwolnienia CPU od zwykłego przesyłania danych.

realizować elementarne działania za pomocą sprzętu i pozwalać na ich współbieżne wykonywanie w sterownikach urządzeń, z jednoczesnym działaniem szyny i procesora.

równoważyć wydajność CPU, podsystemów pamięci, szyny i operacji We/Wy.

Gdzie należy implementować funkcje We/Wy: -w **sprzęcie,**
-w **module sterującym,**
-w **oprogramowaniu aplikacji**.

- **Eksperymentalne** algorytmy We/Wy można implementować w aplikacji.

  Kod aplikacji jest elastyczny, a występujące w nim błędy nie powodują awarii systemu, pozwalając na łatwą modyfikację kodu.

  Może być mało wydajny gdyż aplikacja nie może korzystać z wewnętrznych struktur danych jądra i jego funkcji (komunikaty wewnątrz jądra, stosowanie wątków, blokowanie zasobów).

- **Przetestowany** algorytm na poziomie aplikacji można zaimplementować w jądrze.

  Polepsza to wydajność, lecz w jego opracowanie wymaga dużego wysiłku, gdyż jądro SO jest skomplikowanym modułem oprogramowania.

  Implementacja w jądrze wymaga starannego przetestowania załamań systemu.

- **Implementacje sprzętowe** specjalizowane pozwalają uzyskać największą wydajności.

  Wady implementacji sprzętowej:

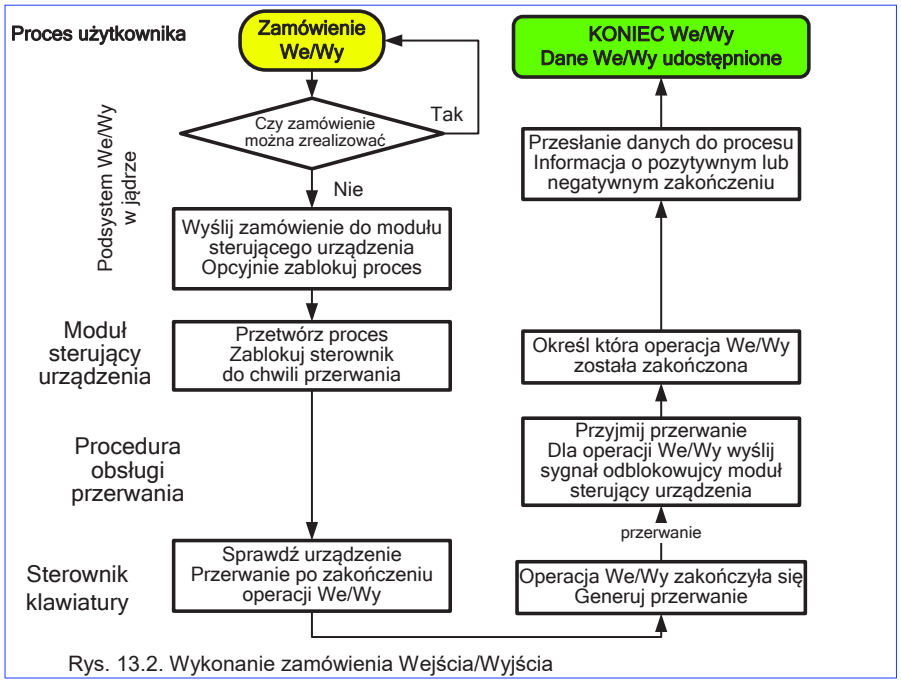
  -koszty dalszych ulepszeniami lub usuwania błędów,

  -zwiększony czas opracowywania,

  -zmniejszoną elastyczność.

Sprzętowy sterownik RAID może **ograniczać** możliwość wpływania **jądra** na kolejność i położenie czytanych/zapisywanych bloków, mimo że **jądro** dysponuje danymi umożliwiającymi poprawianie wydajności operacji We/Wy.

Operacja We/Wy jest **wielokrokowa** i **zużywa dużo cykli CPU.**



Rys. 13.2. Wykonanie zamówienia Wejścia/Wyjścia

1. Proces zamawia operację We/Wy (blokującą go), powołując się na deskryptor otwartego pliku.
2. **Jądro** funkcji systemowej sprawdza **poprawności parametrów**.
   Jeżeli zamawiana operacja dotyczy Wejścia i potrzebne dane znajdują się w buforze pamięci podręcznej, to następuje przekazanie ich do procesu i zamówienie We/Wy zostaje spełnione.
3. W przeciwnym razie następuje uruchomienie procedury realizacji operacji We/Wy.
   Proces zostaje **usunięty** z kolejki procesów **Gotowych** i umieszczony w kolejce **procesów Czekających** na urządzenie.
   Potrzebna operacja We/Wy podlega **zaplanowaniu**.
   Podsystem We/Wy pośle zamówienie do modułu sterującego. Zamówienie zostaje przesłane za pomocą wywołania podprogramu lub za pośrednictwem wewnętrznego komunikatu **jądra**.
4. Moduł sterujący rezerwuje miejsce na przyjęcie danych w buforze jądra i planuje operację; następnie wysyła polecenia do sterownika, zapisując je w rejestrach sterujących urządzenia.
5. Sterownik urządzenia nakazuje sprzętowi wykonanie przesłania danych.
6. Moduł sterujący może odpytywać o stan urządzenia i dane lub zorganizować przesyłanie w trybie DMA do pamięci jądra.
7. Procedura obsługi przerwania odbiera przerwanie, zapamiętuje niezbędne dane, przekazuje sygnał do modułu sterującego i kończy obsługę przerwania.
8. Moduł sterujący urządzenia określa, która operacja We/Wy skończyła się, określa stan zamówienia i powiadamia podsystem We/Wy w **jądrze**, że zamówienie zostało zakończone.
9. **Jądro** przesyła dane lub przekazuje umowne kody do przestrzeni adresowej procesu, który złożył zamówienie, i przemieszcza proces z kolejki **Oczekującej** do kolejki **Gotowych**.
10. Przeniesienie procesu do kolejki procesów **Gotowych** powoduje jego **Odblokowanie**.
    ☛ Planista **przydziela** CPU, i wznawiana jest praca po zakończonym odwołaniu do systemu.