12. OCHRONA

Ochrona *(protection)*: mechanizmy kontrolujące dostęp procesów lub użytkowników do zasobów.

Bezpieczeństwo *(security)*: miara zaufania, że system i jego dane pozostaną nienaruszone.

➡ SO dostarcza **środków** do realizacji przyjętej **polityki** korzystania z zasobu.

Proces musi używać zasoby systemowe zgodnie z **polityką** przewidzianą dla nich.

Ochrona może dodatkowo zwiększyć niezawodność dzięki wykrywaniu błędów.

System komputerowy jest zbiorem Procesów i Obiektów.

Obiekty *(objects)* to **sprzęt** (CPU, RAM, drukarki) oraz **oprogramowanie**.

Obiekt ma jednoznaczną **nazwę**, dostęp do Niego odbywa się za pomocą **Operacji**.

Rodzaj wykonywanych **Operacji** zależy od **Obiektu**.

Zasada: proces ma dostęp tylko do zasobów, do których został uprawniony.

Zasada wiedzy koniecznej *(need-to-know)*: proces w każdej chwili ma dostęp **tylko** do zasobów, których ***aktualnie*** potrzebuje.

Jeśli proces **P** wywoła funkcję **A**, to funkcja ta powinna mieć dostęp tylko do swoich zmiennych i parametrów formalnych; nie powinna mieć dostępu do wszystkich zmiennych procesu **P**.

12.1. Struktura domenowa

Proces działa w **domenie ochrony** *(protection domain)*, która określa dostępne dla niego zasoby.

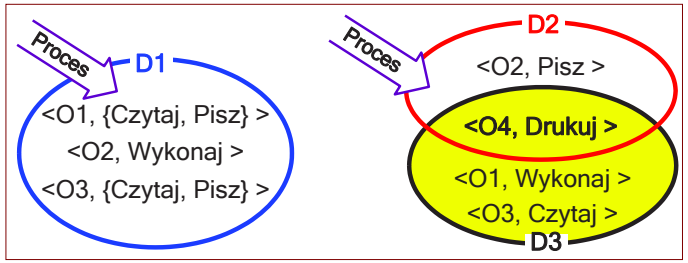
Domena definiuje Zbiór Obiektów i rodzaje Operacji, które można wykonywać dla obiektu.

Prawo dostępu *(access right)*: możliwość wykonywania **Operacji** na obiekcie.

Domena jest zbiorem praw dostępu do **O**biektu: **<nazwaObiektu, ZbiórPraw>**.

W domenie **D** istnieje prawo dostępu: **<plik F, {Czytaj, Pisz}>:**

proces działający w domenie **D** może tylko **Czytać** i **Zapisywać** plik **F**; inne operacje na pliku są dla procesu niedostępne.



◗ **Domeny mogą dzielić prawa dostępu**

Mamy trzy domeny: **D1**, **D2**, **D3**.

Prawo dostępu **<O4, Drukuj>** dzielone jest przez **D2** i **D3**;

Proces działający w każdej z dwu domen może drukować obiekt **O4**.

Czytać i **Pisać** obiekt **O1** może proces działający tylko w domenie **D1.**

Wykonywać obiekt **O1** procesy mogą tylko w domenie D3**.**

Związek między procesem a domeną może być:

-**statyczny**, ustalony zbiór zasobów dostępnych dla procesu,

-**dynamiczny**, pozwala na przełączanie procesów między domenami.

Statyczny związek proces - domena **nie** gwarantuje realizacji zasady **wiedzy koniecznej**.

Proces wykonuje się **dwufazowo**: w **1**-szej **Czyta**,

w **2**-giej **Pisze**.

☞ Statyczna domena zawiera prawo dostępu do **C**zytania i **P**isania.

☛ Domena zawiera więcej praw, niż potrzeba w każdej z dwu faz.

➡ Potrzebny jest mechanizm **zmiany zawartości domeny**.

Jeśli nie można zmienić zawartości domeny, to można utworzyć **nową** domenę ze zmienioną zawartością i **przełączać** się do niej.

Przykłady realizacji domen:

- **Użytkownik**: obiekty, do których może mieć dostęp, zależą od **identyfikacji** użytkownika.
  Przełączanie domen następuje wraz ze zmianą użytkownika.
- **Proces**: obiekty, do których ma dostęp, zależą od identyfikacji procesu.
  Przełączanie domen to wysyłanie przez proces komunikatu do innego procesu.
- **Procedura**: obiekty, do których ma dostęp to zmienne lokalne procedury.
  Przełączanie domen następuje wraz z wywołaniem procedury.

Proces działający w **domenie:**

- Użytkownika może wykonywać tylko rozkazy nieuprzywilejowane,

- Monitora może wykonywać **wszystkie** rozkazy.

W wieloprogramowym SO dwie domeny: **U**żytkownika i **M**onitora nie wystarczają, gdyż użytkownicy wymagają ochrony **wzajemnie** przed sobą.

Mechanizm ochrony powinien być odseparowany od zasad jej stosowania, gdyż umożliwia w tym samym systemie stosowanie złożonej lub prostej ochrony.

12.2. Macierz dostępów *(access matrix)*

Wiersze macierzy dostępów reprezentują **domeny**, a **kolumny** **O**biekty.
Każdy element macierzy zawiera zbiór praw dostępu.

Tabela 12.1. Macierz dostępów

Obiekt / Domena	F1	F2	F3	Printer
D_1	Czytaj		Czytaj	
D_2				Drukuj
D_3		Czytaj	Wykonaj	
D_4	Czytaj Pisz		Czytaj Pisz	

Pozycja **Operacje**(**i**, **k**) określa zbiór operacji, które Proces, działający w domenie D_i może wykonywać na obiekcie O_k.

Przykładowa macierz dostępów, zawiera cztery domeny i cztery obiekty: trzy pliki i drukarkę.

Proces działający w domenie D_1 może czytać pliki **F1** i **F3**.

Proces działający w domenie D_4 ma takie same prawa, jak w domenie D_1 i dodatkowo może Pisać do plików **F1** i **F3**.

Drukarka może być dostępna tylko dla procesu działającego w domenie D_2.

Macierz dostępów dostarcza mechanizmu dla decyzji politycznych.

Które domeny mają dostęp, do jakich Obiektów i w jaki sposób, jest decyzją polityczną.

◗ **Macierz dostępów umożliwia realizację decyzji politycznych dotyczących ochrony.**

Decyzje te dotyczą praw, które powinny znaleźć się w elemencie (**i**, **k**) tej macierzy.

Należy określić domenę działania każdego Procesu ➔ jest to na ogół w gestii SO.

◗ **Użytkownik decyduje o zawartości elementów macierzy dostępów.**

Utworzenie nowego obiektu O_k dodaje do macierzy dostępów kolumnę O_k z elementami definiowanymi przez twórcę **Obiektu**.

Macierz dostępów umożliwia implementację kontroli **statycznej** i **dynamicznej** związków między **P**rocesami a **Domenami**.

Jeśli przełączamy proces z jednej **domeny** do drugiej, to wykonujemy operację (Przełącz) na **O**biekcie, którym jest **domena**.

☛ Zaliczenie domen do Obiektów macierzy pozwala realizować przełączanie domen.

◗ Zmiana macierzy dostępów oznacza operację na obiekcie, jakim jest macierz dostępów.

Zmiany można nadzorować gdyż macierz dostępów stała się **obiektem** macierzy dostępów.

Każdy element macierzy dostępów można zmieniać z osobna, należy traktować każdy element tej macierzy, jako obiekt podlegający ochronie.

❒ **Procesy mogą przechodzić** z **jednej domeny** do **drugiej**.

Przełączenie z domeny D_i do D_k może nastąpić, gdy prawo dostępu ***Przełącz*** ∈ Operacje(**i**, **k**).

Przykładowo:

➲ Proces działający w domenie D_2 może przełączyć się do D_3 albo do domeny D_4.

Proces w domenie D_4 może przełączyć się do D_1

Proces w domenie D_1 może przejść do D_2.

Tabela 12.2. Domeny w roli obiektów

Obiekt / Domena	F1	F2	F3	Printer	D_1	D_2	D_3	D_4
D_1	Czytaj Drukuj		Czytaj			*Przełącz*↑		
➲ D_2	Wykonaj			Drukuj			*Przełącz*↑	*Przełącz*↑
D_3		Czytaj	Wykonaj					
D_4	Czytaj Pisz		Czytaj Pisz		*Przełącz*↑			

❒ **Kontrola zmian zawartości macierzy dostępów wymaga trzech dodatkowych praw**

◗ **Prawo kopiowania** ✱

Pozwala **kopiować** prawa dostępu tylko w obrębie **kolumny**, dla której dane prawo zostało zdefiniowane.

Proces działający w domenie D_2 może skopiować operację **C**zytania do dowolnego elementu macierzy związanego z plikiem **F2**.

Tabela 12.3. Macierz dostępów z prawami **Kopiowania**

Obiekt / Domena	F1	F2	F3
D_1	Wykonaj		**Pisz** ✱ ·
➲ D_2	Wykonaj	**Czytaj** ✱ ·	Wykonaj
D_3	Wykonaj		

Obiekt / Domena	F1	F2	F3
D_1	Wykonaj		**Pisz** ✱
D_2	Wykonaj		Wykonaj
D_3	Wykonaj	**Czytaj**	Pisz

Gwiazdka ✱ oznacza możliwość kopiowania prawa dostępu z jednej domeny do drugiej.

Dwa warianty schematu kopiowania:

1. Przekazanie prawa dostępu (z usunięciem)

Prawo kopiowane jest z pola **Operacji**(**i**, k) do pola Operacji(**r**, k) a następnie ***usuwane*** z pozycji **Operacje**(**i**, k)**.**

2. Ograniczenie przenoszenia prawa kopiowania.

Jeśli prawo ***R**** jest kopiowane z pola Operacje(**i**, k) do pola Operacje(**r**, k) to tworzy się tylko prawo ***R*** (a nie ***R****).*

☛ Proces działający w domenie $\mathbf{D_k}$ **nie** może dalej kopiować prawa ***R***.

Prawo właściciela

Umożliwia **dodawania nowych praw** i **usuwanie istniejących praw**.

Jeśli element Operacje(**i**, **k**) zawiera prawo ***właściciela***,

to **P**roces działający w domenie $\mathbf{D_i}$

może **dodawać** bądź **usuwać** dowolne prawa do elementów w **kolumnie k**.

Tabela 12.4. Macierz dostępów z prawami **Właściciela**

Obiekt / Domena	F1	F2	F3
D_1	**właściciel** Wykonaj		Pisz
D_2		Czytaj * **właściciel**	Czytaj * **właściciel** Wykonaj
D_3	Wykonaj		

Obiekt / Domena	F1	F2	F3
D_1	**właściciel** Wykonaj		
D_2		**właściciel** Czytaj * Pisz *	Czytaj * **właściciel** Pisz *
D_3		Pisz	Pisz

Domena D_1 jest **właścicielem** pliku **F1**, może dodać/usunąć każde dozwolone prawo w **kolumnie F1**.

Domena D_2 jest **właścicielem F2** i **F3**, może dodać/usunąć prawa wewnątrz tych dwóch **kolumn**.

Prawo kontroli

Pozwala **zmieniać elementy** w **wierszu**.

Prawo **Kontroli** jest stosowane tylko do Obiektów będących **Domenami**

Jeśli pole Operacje(**i, k**) zawiera prawo **Kontroli**,

to **P**roces pracujący w domenie $\mathbf{D_i}$,

może **usunąć** dowolne prawo dostępu z **wiersza k**.

Poniżej pole Operacji(D_2, D_4) zwiera prawo kontroli.

Tabela 13.5. Macierz dostępów z prawami **Kontroli**

Obiekt / Domena	F1	F2	F3	Printer	D_1	D_2	D_3	D_4
D_1	Czytaj		Czytaj			Przełącz		
D_2	Wykonaj			Drukuj			Przełącz	Przełącz **Kontroluj**
D_3		Czytaj	Wykonaj					
D_4	~~**Pisz**~~		~~**Pisz**~~		Przełącz			

Proces działający w domenie D_2 może **usunąć** wszystkie operacje **Pisz** działające w domenie D_4

> Prawa **Kopiowania** i **Właściciela** umożliwiają ograniczanie rozchodzenia się praw dostępu.
> Nie zapobiegają jednak rozchodzeniu się informacji (ujawnianiu).

12.2.1. Implementacja macierzy dostępów

❒ **Tablica globalna**

Tablica globalna jest zbiorem trójek <***domena***, ***obiekt***, ***zbiór_praw***>.

Wykonanie OPeracji na obiekcie O_k w domenie D_i, rozpoczyna szukanie w **Tablicy Globalnej** trójki <D_i, O_k, Z_r>, gdzie **OP** ∈ Z_r.

Odnalezienie trójki umożliwia wykonanie operacji, w przeciwnym razie sygnalizowany jest **błąd**.

Tablica często jest za duża aby przechowywać ją w PAO → **korzystanie z pamięci wirtualnej**.

❒ **Wykaz dostępów** do `Obiektów`

Każda kolumna w macierzy dostępów może być wykazem dostępu do danego obiektu.

	O_k
D_1	Czytaj, ...
D_2	Kopiuj, ...
D_3	
D_4	Pisz, Czytaj

Obiektowi odpowiadają pary <***domena***, ***zbiór_praw***>, które definiują wszystkie domeny ze zbiorami praw dostępu do danego obiektu.

Do zdefiniowanego wykazu można dodać ***domyślny*** zbiór praw dostępu.

☛ **Wykonanie OPeracji** na obiekcie O_k w domenie D_i rozpoczyna wyszukiwanie w wykazie dostępów obiektu O_k stosownej pozycji <D_i , **zbiór_praw**$_r$>.

Odnalezienie tej pozycji daje zezwolenie na wykonanie operacji;
jeżeli nie ma, przeszukiwany jest **domyślny** zbiór praw dostępu;
jeżeli też nie ma, następuje odmowa dostępu **-błąd**.

Dostęp do obiektu podlega sprawdzaniu, co wymaga przeszukiwania wykazu dostępów.
W dużych systemach, przeszukiwanie może być czasochłonne.

❒ **Wykaz uprawnień** do `domen`

Wykaz uprawnień *(capability list)* domeny jest zbiorem {**D**| **O**biekt: **OP**eracje} obiektów i dozwolonych operacji.

Obiekt identyfikuje jego **nazwa** lub adres zwany ***uprawnieniem***.

	D_i
O_1	Czytaj
O_2	Kopiuj
O_3	
O_4	Pisz Czytaj

☛ Aby **Wykonać OPerację** na obiekcie O_k w domenie D_i proces udostępnia **OP**eracji ***parametr*** (wskaźnik) określający uprawnienie do obiektu O_k.

Wykaz uprawnień związany jest z domeną, lecz nie jest dostępny bezpośrednio dla procesu działającego w tej domenie.

12.3. Schematy ochrony

Ochrona na bazie uprawnień zakłada, że uprawnienia nie przedostaną się do obszaru pamięci modyfikowanego przez procesy użytkownika.

PROCES starający się o dostęp do Obiektu MUSI wykazać, że ma uprawnienia.

System ochrony sprawdza tylko, czy uprawnienia są ważne.

Wewnętrzna ochrona wymaga odróżnienia uprawnienia od **innych danych**.

◗ Każdy Obiekt ma **znacznik** *(log)*, określający jego **typ**: uprawnienia lub inne dane (np. rozkaz).

Znacznik **nie** może być dostępny bezpośrednio dla programów użytkowych.

Ograniczenie dostępu można wymusić środkami sprzętowymi lub sprzętowo-programowymi.

Do rozróżnienia między **uprawnieniami** a innymi obiektami wystarczy **1** bit.

◗ **Przestrzeń adresową** związaną z programem można podzielić na dwie części:

1. zawiera właściwy kod oraz dane i dostępna jest dla programu,
2. zawiera **wykaz uprawnień** i dostępna jest tylko dla SO.

❒ **Schemat zamek-klucz** *(lock-key scheme)*

Każdy **Obiekt** ma wykaz jednoznacznych wzorców binarnych, zwanych **zamkami** *(locks)*.

Każda **domena** ma wykaz jednoznacznych wzorców binarnych, zwanych **kluczami**.

Proces działający w domenie może mieć dostęp do **Obiektu** tylko wtedy, gdy dana domena zawiera **klucz** pasujący do jednego z **zamków** tego Obiektu.

Wykazy kluczy domeny są zarządzane przez SO na zamówienie danej domeny.

☛ **Użytkownik nie ma prawa** bezpośrednio sprawdzać ani zmieniać zamków (lub kluczy).

Klucze można przekazywać od domeny do domeny.

Przywileje dostępów można uaktualniać zmieniając klucze przypisane do obiektu.

❒ **Kombinacja wykazów dostępów** i **uprawnień** (często stosowana przez systemy)

☛ Proces po raz **pierwszy** sięga do **O**biektu, wtedy przegląda się **wykaz dostępów**.

Jeśli dostęp jest zabroniony, to powstaje błąd.

☛ Gdy dostęp dozwolony, to tworzy się **wykaz uprawnień**, dołączany do procesu.

☛ Proces przy **następnych** odniesieniach korzysta z **wykazu uprawnień**.

Po ostatnim dostępie uprawnienia zostają cofnięte.

➡ Prawo do dostępu musi być sprawdzane przy **każdym** dostępie.

Wpis w **TablicyPlików** zawiera wykaz uprawnień do wykonywania tylko dozwolonych operacji.

Plik otworzono do **Czytania**:

-w **TablicyPlików** umieszczono uprawnienie dostępu do **Czytania**.

Próba **W**pisu do takiego pliku (naruszenie ochrony) będzie wykryta przez porównanie żądanej operacji z wykazem uprawnień w TablicyPlików.

12.4. Cofanie praw dostępu

Dynamiczna ochrona może wymagać cofnięcia praw dostępu do **O**biektów **dzielonych** przez różnych użytkowników.

- Czy cofanie praw następuje **natychmiast** lub z **opóźnieniem** (kiedy nastąpi)?
- Czy cofnięcie dotyczy **wszystkich użytkowników**, mających prawo dostępu do obiektu, lub dotyczy **grupy** użytkowników, którym prawo należy odebrać?
- Czy należy cofnąć **wszystkie prawa** dotyczące obiektu, lub unieważnić **podzbiór** praw?
- Czy dostęp jest cofany na **stałe** (cofniętego prawa dostępu nigdy nie będzie można osiągnąć), czy też dostęp można unieważnić i uzyskać go później z powrotem ?

Dla schematu **wykazu dostępów** wyszukuje się na wykazie dostępów prawo do unieważnienia i usuwa je z wykazu.

Dla schematu **wykazu uprawnień** cofanie praw jest trudniejsze, uprawnienia są rozproszone po systemie, należy je wpierw odnaleźć.

❒ Schematy realizacji cofania uprawnień

◗ **Wtórne pozyskiwanie:** uprawnienia są okresowo **usuwane** z każdej domeny.

Jeśli proces potrzebuje określonego uprawnienia, to wykrywa jego **usunięcie** i może spróbować uzyskać je na nowo.

Jeśli uprawnienie zostało **cofnięte**, to proces **nie może** uzyskać go ponownie.

◗ **Wskaźniki zwrotne: O**biekt utrzymuje wykaz **wskaźników** do związanych z nim uprawnień.

Wskaźnikami umożliwiają zmianę uprawnień stosownie do potrzeb.

◗ **Adresowanie pośrednie:** uprawnienia wskazują na **O**biekty pośrednio.

Uprawnienie wskazuje na wpis w Tablicy Globalnej, który z kolei wskazuje na Obiekt.

Aby cofnąć uprawnienia: -**przeszukuje** się Tablicę Globalną,
-**usuwa** z niej element.

☛ Gdy wystąpi próba dostępu to uprawnienie wskazuje na **nie**dozwolony element.

Elementy tablicy mogą być użyte powtórnie do innych uprawnień, ponieważ uprawnienie, jak i wpis w Tablicy zawiera jednoznaczną nazwę obiektu.

Obiekt oznaczony przez uprawnienie oraz odpowiadający mu wpis w Tablicy muszą do siebie pasować.

◗ **Klucze:** są jednoznacznymi wzorcami binarnymi i można powiązać z uprawnieniami.

Klucz definiowany jest przy **tworzeniu** uprawnień.

Proces mający dane uprawnienia **nie może** sprawdzić ani zmienić **klucza**.

Operacja **UstalKlucz** definiuje **Klucz Główny** dla każdego **O**biektu.

Tworzonym uprawnieniom przydziela się bieżącą wartość **Klucza Głównego**.

Sprawdzając uprawnienia porównuje się ich **klucz** z **Kluczem Głównym**.

Jeśli **klucze** pasują, to zezwala się na wykonanie operacji.
W przeciwnym razie następuje zgłoszenie wyjątku.

Cofnięcie uprawnień realizuje operacja **UstalKlucz**, nadającą nową wartość **Kluczowi Głównemu,** i kasująca dotychczasowe uprawnienia danego obiektu.

Metoda uniemożliwia selektywne cofanie uprawnień, gdyż z każdym Obiektem związany jest tylko jeden klucz główny.

Skojarzenie z Obiektami listy kluczy, umożliwia wybiórcze cofania uprawnień.

◗ **Grupowanie kluczy** w jednej, globalnej **Tablicy Kluczy**.

Jeden klucz można przypisać do kilku obiektów, co daje maksimum elastyczności.

Definiowanie kluczy, umieszczanie na listach oraz usuwania z list jest decyzją polityczną, której system nie powinien określać

(musi realizować, gdy zostanie określona).

12.5. Ochrona na poziomie języka programowania

Zazwyczaj ochrona organizowana jest na poziomie jądra systemu operacyjnego, które legalizuje każdy zamiar dostępu do chronionego zasobu.

Mechanizm ochron powinien uwzględniać funkcjonalną naturę danego dostępu, a nie tylko identyfikację zasobów i próby dostępu do nich.

Dobór funkcji realizujący ochronę, wykracza poza zbiór działań dostępny w SO (standardowe metody dostępu do pliku), i obejmuje też funkcje definiowane przez użytkownika.

Ochroną nie może zajmować się wyłącznie projektant systemu operacyjnego.

☛ Ochrona powinna być narzędziem dostępnym dla projektanta aplikacji.

Określenie żądanej kontroli dostępu do dzielonego zasobu w systemie sprowadza się do napisania **odpowiedniej *deklaracji*** w odniesieniu do danego zasobu.

Taką ***deklarację*** można dołączyć do **języka programowania** przez rozszerzenie jego możliwości operowania **typami**.

Deklarując ochronę przy okazji określania **typów danych**, projektant każdego podsystemu może **określić własne wymagania** w stosunku do ochrony.

Zalety podejścia programowego:

1. zapotrzebowania na ochronę są **deklarowane**, nie zaś programowym ciągiem wywołań procedur SO.
2. wymagania dotyczące ochrony mogą być **niezależnie** od środków dostarczanych przez SO.
3. projektant podsystemu nie musi dostarczać środków wymuszania ochrony.
4. notacja deklaratywna jest naturalna, ponieważ przywileje dostępów pozostają w ścisłym związku z lingwistyczną koncepcją typu danych.

Implementacja języka może udostępniać standardowe, chronione procedury do interpretacji uprawnień programowych, realizując zasady ochrony wynikające z konstrukcji języka.

Schemat umożliwia określanie zasad ochrony programistom, jednocześnie **uwalniając** ich od szczegółów związanych z **egzekwowaniem** tych zasad.

Bezpieczeństwo realizowane drogą **programową** nie będzie tak duże, jakie gwarantuje **jądro** SO, ponieważ mechanizm **programowy** przyjmuje więcej założeń odnośnie działania systemu.

Bezpieczeństwo ochrony na poziomie języka programowania zakłada, że kod wygenerowany przez kompilator nie zostanie zmieniony ani przed, ani podczas wykonania.

12.6. Metodyka realizacji ochrony

Jak realizować ochronę:

-**wyłącznie** przez **jądro systemu,**

-**głównie** przez **kompilator**

Bezpieczeństwo

Wymuszanie przez **jądro daje wyższy stopień** bezpieczeństwa samego systemu ochrony niż kod kompilatora sprawdzający ochronę.

Bezpieczeństwo **kompilatorowego** systemu ochrony zależy od:

-poprawności translatora,

-mechanizmów zarządzania pamięcią, chroniących segmenty, w których znajdują się skompilowane rozkazy wykonywanego kodu,

-bezpieczeństwa plików, z których skompilowany kod jest ładowany do pamięci.

Jądro rezyduje w ustalonych obszarach pamięci fizycznej i może być ładowane tylko ze ściśle określonego pliku.

System uprawnień znaczonych *(tagged capability system)*: obliczenia adresów wykonywane są przez sprzęt lub przez stały mikroprogram.
(zwiększa to poziom bezpieczeństwa)

Elastyczność

Ochrona przez Jądro ogranicza wdrażanie polityki użytkownika, dysponuje środkami egzekwowania zasad ochrony.

Język programowania umożliwia deklarowanie polityki ochrony, jej egzekwowanie zapewnia implementacja danego języka.

Język, który nie gwarantuje wystarczającej elastyczności można zmodyfikować lub wymienić na inny ➔ stwarza to mniej problemów w systemie niż modyfikacja **jądra** SO.

Wydajność

Egzekwowanie ochrony za pomocą sprzętu (lub mikroprogramu) daje najlepszą wydajność.

Ochrona środkami **języka programowania** realizuje **statyczną kontrolę** (podczas kompilacji).

Inteligentny kompilator może **minimalizować** koszty wywołań jądra (związane z ochroną), dopasowując mechanizmy jej wymuszania do stawianych celów.

Udostępnienie ochrony w programie użytkowym realizują ***uprawnienia programowe***.
Część oprogramowania może mieć zezwolenie do tworzenia i sprawdzania programowych uprawnień.

Program tworzący uprawnienia wykonuje operacje **markowania** struktur danych; stają się one niedostępne dla elementów oprogramowania, które nie mają prawa dostępu do chronionych struktur.

Nieuprzywilejowane programy mogą kopiować chronioną strukturę danych lub przekazywać jej adres do innych części oprogramowania, lecz **nie uzyskują** dostępu do jej **zawartości**.

Potrzebny jest dynamiczny mechanizm rozprowadzania między procesy użytkowników uprawnień do zasobów systemowych.

Dostępne są konstrukcje językowe, umożliwiające programiście deklarowanie rozmaitych ograniczeń, stosownie do specyfiki zarządzania zasobem.

➔ Konstrukcje językowe dostarczają mechanizmów do wykonywania funkcji:

1. **rozprowadzania uprawnień** między procesami konsumenckimi, gwarantując, że Proces użytkownika skorzysta z zasobu chronionego wtedy, kiedy uzyska uprawnienia.
2. **określania rodzajów operacji,** które dany proces może wykonać na przydzielonym zasobie.
   (np. proces, który ma czytać plik, powinien mieć tylko prawo czytania pliku)

   Proces nie może zwiększać swoich praw dostępu bez akceptacji mechanizmu kontroli dostępu.
3. **określania porządku**, w którym konkretny Proces może wywoływać różne operacje na zasobie.
   (plik przed czytaniem powinien być otwarty)

Należy umożliwić nadanie dwu procesom różnych ograniczeń odnośnie porządku, w którym mogą one wywoływać operacje na przydzielonym zasobie.

ANEX 12.1

❒ W systemie UNIX **domena** związana jest z **Użytkownikiem**

Przełączaniu domen odpowiada zmiana identyfikacji użytkownika, -realizuje system plików.

Z plikiem związany jest:

-identyfikator **właściciela**

- użytkownika ***(user_id)***,

-**bit domeny**, bit ustanowienia identyfikatora użytkownika ***(setuid** bit).*

Niech plik będzie własnością użytkownika **B**, zaś jego **bit domeny = 0** ←

Gdy użytkownik **A** (**user_id** = **A**) rozpoczyna wykonanie pliku Użytkownika **B**, to identyfikator **user_id** procesu określony jest jako **A**.

Jeśli **bit domeny=1**, to identyfikator przyjmuje wartość **user_id = B**, czyli właściciela pliku.

Chwilowa zmiana ident. **user_id** przestaje obowiązywać, gdy proces kończy działanie.

Domeny definiowanie identyfikatorami użytkowników, umożliwiają przyznawanie uprzywilejowania ogółowi użytkowników.

Można zezwolić użytkownikom na korzystanie z sieci, bez zezwalania na modyfikacje w niej.

Wówczas **bit domeny** (**setuid**) programu organizującego współpracę z siecią ma wartość 1, powodując zmianę identyfikatora użytkownika na czas pracy tego programu.

Identyfikator użytkownika ***(user_id)*** przyjmuje czasowo wartość identyfikatora użytkownika mającego przywilej współpracy z siecią (np.: identyfikatora korzenia drzewa katalogów).

Jeśli użytkownik utworzy plik z identyfikatorem ***user_id* = „root"** i **bitem domeny=1**, to zyskuje władzę użytkownika „**root**".

Każdemu plikowi w systemie UNIX przyporządkowany jest wykaz dostępów.

Jeśli proces otwiera plik, to przeszukuje się strukturę katalogową w celu znalezienia pliku, sprawdzenia praw dostępu i przydzielenia buforów.

Informację tę zapisuje się, jako nowy element TablicyPlików należącej do procesu.

Dalsze operacje na pliku wymagają podania indeksu do odpowiedniej pozycji TablicyPlików.

Wpis w TablicyPlików zawiera wskaźnik do pliku i jego buforów.

Zamknięcie pliku usuwa odpowiadający mu wpis z TablicyPlików.

Użytkownik ma dostęp tylko plików, które zostały otwarte (nie ma do TablicyPlików).

Ochrona pliku zapewniona, gdyż dostęp sprawdzany jest przy otwarciu pliku.

-----W innych SO uprzywilejowane programy lokowane są w specjalnym katalogu------

System operacyjny powinien zmieniać identyfikatory użytkowników dla każdego programu wykonywanego z katalogu uprzywilejowanego na:

-równoważne identyfikatorowi **„root"**,

-identyfikator właściciela tego katalogu.