Systemy komputerowe (W08)

Sprzętowe wspomaganie systemu operacyjnego

A.Pruszkowski

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Plan wykładu
  - Dziedziny zagrożeń w oprogramowaniu
  - Metody separacji procesów
  - Wykorzystanie wirtualizacji pamięci

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice – deklaracja, inicjalizacja

```
<typ elementów> nazwa_tablicy [liczba elementów] ...
```

    - Deklaracja bez ustalenia wartości - np.:

```
int tablica_liczb[100];
unsigned char bufor[32];
double liczby_math[1024];
```

    - Inicjacja tablicy (faza deklaracji) – wielkość określa liczba podanych wartości

```
int tablica_liczb[]={10, 20, 33, 102};
```

    - Inicjacja tablicy (faza deklaracji) – podejście mieszane, podane tylko pierwsze cztery wartości

```
int tablica_liczb[100]={10, 20, 33, 102};
```

    - Inicjacja tablicy (faza używania)
      - deklaracja, offset od 0 do 99 – całość ma 100 pozycji

```
int tablica_liczb[100];
```

      - użycie – wstawienie liczby na ostatnią pozycję(!)

```
tablica_liczb[99]=1234;
```

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice - używanie
    - Podejście I

```
#define MAX_INT_ARRAY_SIZE    100
int tablica_liczb[MAX_INT_ARRAY_SIZE];
for(i=0; i<MAX_INT_ARRAY_SIZE; i++) ...
```

    - Podejście II

```
typedef int melement;
melement tablica_liczb[100];
for(i=0; i<sizeof(tablica_liczb)/sizeof(melement); i++) ...
```

    - Operator „sizeof” – zwraca wynik typu size_t (!)
      - sizeof(unsigned char) = sizeof(char) = 1 (zawsze o ile nie przeciążone!)

```
char mstring[32] = „witam";
```

      - Operator `sizeof(mstring)` zwróci 32
      - Funkcja `strlen(mstring)` zwróci 5

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablica - ułożenie w pamięci
    - Nazwa tablicy to alias dla adresu w pamięci gdzie ją umieszczono(!)
      - Odwołania realizowane są poprzez wskazanie adresu w pamięci a na niskim poziomie typowo przez instrukcje - x86: MOV, PUSH/POP, Risc-V: LD/ST
    - „Język C” nie sprawdza offsetów (!!!)
      - Takie zachowanie staje się źródłem wielu problemów, niejednokrotnie źródłem nadużyć
    - Ciekawostka mało znana
      - Odwołanie `tablica_liczb[20]` jest tożsame z `20[tablica_liczb]` (!)

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Struktury – zestaw wielu zmiennych

```
struct nazwa_definicji_struktury {
     definicje_pol_skladowych;
} nazwa_deklaracji_struktury;
```

  - Odwołania do pól – przykład

```
struct _zestaw_rejestrow{
    unsigned char A;
    ...
} zestaw_rejestrow;
zestaw_rejestrow.A = 1;
```

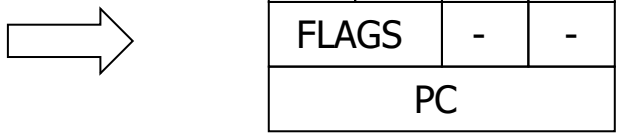
  - Nazwa struktury w przeciwieństwie do tablic nie jest aliasem adresu w pamięci
    - Adres to: `&zestaw_rejestrow`

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - **Uwaga!** nowoczesne kompilatory języka C dla zwiększenia szybkości programu wynikowego rozmieszczają pola domyślnie w adresach będących wielokrotnością słowa maszyny
    - np.: kompilacja dla X86 gdzie słowo maszyny ma 32bity
  - Przykład

```
struct _zestaw_rejestrow{
    unsigned char A;
    unsigned short FLAGS;
    unsigned long PC
} zestaw_rejestrow;
```

Możliwe rozmieszczenie pól w pamięci



  - Istnieje sposób aby upakować pola

```
struct nazwa_definicji_struktury {
   ...
} nazwa_deklaracji_struktury __attribute__((packed));
```

    - Przydatne gdy należy zachować zgodność bitową pól
      - Wymiana takich struktur z innymi maszynami
        - np.: nagłówki plików z „treścią binarną”: EXE, AVI, MP3, DOCX, ...

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Struktury z polami binarnymi

```
struct nazwa_definicji_struktury {
     definicje_pol_skladowych : liczba_bitow;
} nazwa_deklaracji_struktury;
```

  - Zastosowania
    - Interakcja ze sprzętem
    - Obsługa nietypowych typów danych
      - specyficzna kompresja danych – przykład struktury o wielkości 2B (zamiast 3B)

```
struct _czas{
   char godzina: 4; //zał. w aplikacji nie ma znaczenia czy am/pm
   char minuta: 6;
   char sekunda: 6;
} czas __attribute__((packed));
```

      - uwaga! kod wynikowy staje się większy o obsługę pól bitowych

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Unie
    - Agregacja wielu typów w jednej strukturze (Polymorphic Data Structures)

```
union nazwa_unii {
      definicje_pol_skladowych;
}
```

    - Unia zajmuje w pamięci wielkość największego pola
    - Zastosowanie - zręczne żonglowanie między typami danych
      - Przykład – użycie pól w „opcode” RISC-V

```
union risc_v_opcode{
  uint32_t raw;
  struct _u_type{
    uint32_t imm:    20;
    uint32_t rd:      5;
    uint32_t opcode:  7;
  } u_type;
  ...
};
...
```

```
uint32_t ExU2(uint32_t val){...}
...
union risc_v_opcode opcode;
...
opcode.raw=MEM[PC];
if(opcode.u_type.opcode==AUIPC){
  REG[opcode.u_type.rd]=PC+ExU2(opcode.u_type.imm);
  ...
}
```

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice i tablice struktur/unii
    - Typy takie ułatwiają organizację danych w pamięci podczas działania programu
      - Przypomnienie – „język C” nie sprawdza offsetów odwołaniach do pól tablicy(!!!)
    - Załóżmy że mamy w kodzie deklaracje:

```
#define MAX_STUDENTS 100
int tablica_ocen_lab[MAX_STUDENTS];
...
tablica_ocen_lab[100]=2;   //w tablicy ostatni element ma offset 99(!)
```

    - Błędny jest tu użyty offset 100, gdy wynikające z deklaracji to offsety od 0 do 99 włącznie

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice i tablice struktur/unii – zagrożenia: odwołania do „obcej” pamięci
    - Wyjście poza obszar przydzielonej pamięci – przypadek statyczny
      - Przypadek prosty do wykrycia

```
#define MAX_ELEMENTS 100
...
typedef struct{
   unsigned int lab, egzamin;
} student_ocena;
...
student_ocena tablica_ocen[MAX_ELEMENTS];
...
tablica_ocen[102].lab=2;
...
```

    - Tzw. ”ręczna” analiza kodu wykryje ten błąd – widać że 102 jest z poza zakresu 0…99

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice i tablice struktur/unii – zagrożenia, odwołania do „obcej” pamięci, cd.
    - Wyjście poza obszar przydzielonej pamięci – przypadek statyczny
      - Przypadek nieco bardziej skomplikowany – wymaga dokładnej analizy kodu

```
#define MAX_ELEMENTS 100
typedef struct{
   unsigned int lab, egzamin;
} student_ocena;
student_ocena tablica_ocen[MAX_ELEMENTS];
#define REFERENCE_VALUE_OFFSET    (50)
#define DEFAULT_VALUE_OFFSET      (REFERENCE_VALUE_OFFSET*2+1)
tablica_ocen[DEFAULT_VALUE_OFFSET].lab=3;
```

Ile równe jest to makro?

      - Przydatne jest tu „przejście” przez preprocesor - wywołując gcc –E main.c otrzymamy:

```
typedef struct{
   unsigned int lab, egzamin;
} student_ocena;
student_ocena tablica_ocen[100];
tablica_ocen[102].lab=3;      //tu już widać problem (!)
```

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – typy zaawansowane w języku C
  - Tablice i tablice struktur/unii – zagrożenia, odwołania do „obcej” pamięci, cd.
    - Przypadek dynamiczny – wychodzimy poza obszar przydzielonej pamięci podczas działania aplikacji (często efekt powstaje tylko przy pewnych danych wejściowych)

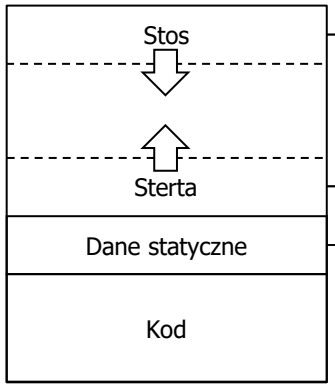
```
int correction=90; //zmienna globalna ustalana w wielu miejscach, zależnie od
                   //danych wejściowych
int func(int i){   //funkcja pomocnicza
   return i<correction ? correction*2 : i; //jakaś błędnie zapisana obróbka
}
int pos=0;
...
pos--;                  //tej operacji można w swoim kodzie „nie zauważyć”
tablica_ocen[pos].lab=2;//efekt: niejawnie wyszliśmy poza tablicę
...
pos=20;                  //pozornie wskazanie jest ok
tablica_ocen[func(pos)].lab=2;//efekt: niejawnie także wyszliśmy poza tablicę
```

    - Pomoc
      - czytanie ostrzeżenia generowane przez kompilator
      - programy do automatycznej analizy kodu np.: Valgrind [Valgrind.org]

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Rola stosu w systemie komputerowym
  - Przypomnienie – m.in. miejsce przechowywania zmiennych lokalnych, argumentów wywołania i adresów powrotu
    - Typowa organizacja pamięci (C memory layaout)

```
#include <stdio.h>

int globalna_zmienna;

int globalna_zmienna_zainicjowana=0;

void func(int parametr){
  int zmienna_lokalna; //widziana
                       //tylko w func()
  ...
}
int main(){
  int zmienna_lokalna;
  static int zmienna_statyczna;

  ...

  return 0;
}
```



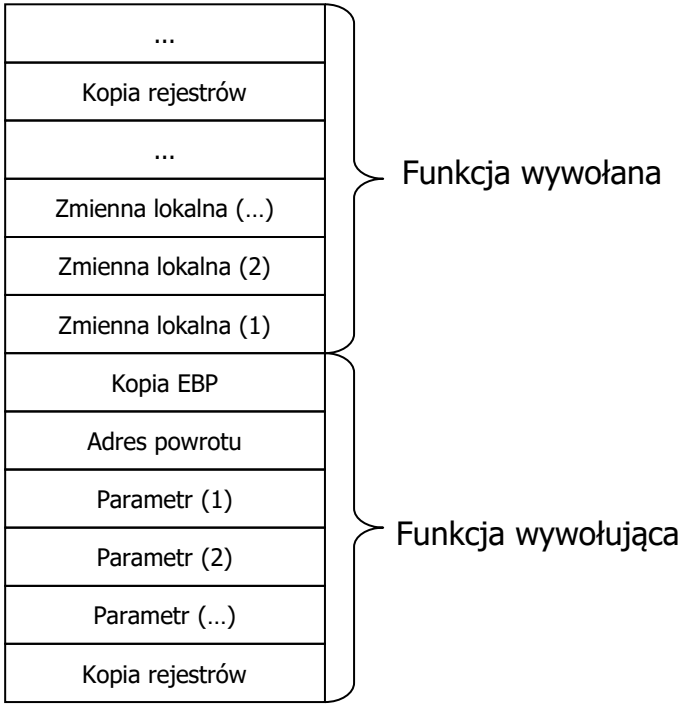
Zmienne lokalne, adresy powrotu, argumenty wywołania

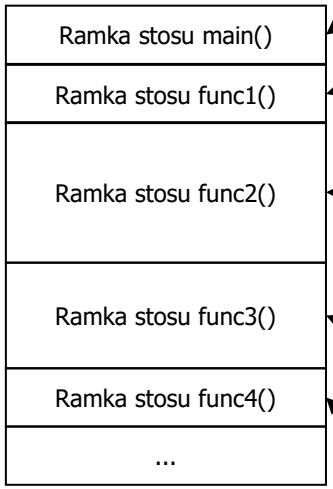
Obszar dynamicznie alokowanych obiektów/zmiennych za pomocą:
w C i C++ funkcji: malloc/free (realloc, calloc, ...),
w C++ (i tylko w tam) operatorów: new/delete

Zmienne globalne, statyczne – ich rozmiar nie ulega zmianie, są w pamięci przez cały czas działania kodu aplikacji/systemu

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Rola stosu w systemie komputerowym
  - Przypomnienie – m.in. miejsce przechowywania zmiennych lokalnych, argumentów wywołania i adresów powrotu, cd.
    - Kompilatory języka C stosują tzw. „ramkę stosu” (stack frame)
      - EBP – tzw. „Base Pointer” w X86






```
int main(){
  func1(0);
  return 0;
}
```

```
void func1(int a){
  func2(1);
}
```

```
void func2(int a){
  int tab[2];
  ...
  func3((int)'A');
}
```

```
void func3(int a){
  int x;
  ...
  func4(a);
}
```

```
void func4(int a){
  printf(".");
}
...
```

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przypomnienie – zasięg zmiennych w języku C
  - Zmienne lokalne – pomagają kompilatorowi w zarządzaniu pamięcią
    - Zmienna „znika” po wyjściu z funkcji (!)
    - Liczenie, że wskazany obszar po wyjściu z funkcji będzie zawierał tę samą wartość jest błędem

Kod błędny – czasami może działać (!)

```
int *func(void){
  int x=5;
  return &x; //pozornie poprawne
    //przekazanie adresu zmiennej x –
    //kompilator z reguły o tym ostrzeże
}
int main(){
  int *z,x,y;
  z=func();
  x=*z; //lub x=z[0];
  printf(”%d\n”, x); //tu otrzymamy 5
  y=*z;
  printf(”%d\n”, y); //wynik będzie inny(!)
  return 0;
}
```

Kod poprawny

```
void func(int *z){
  int x=5;
  *z=x; //lub z[0]=x;
}


int main(){
  int z,x,y;
  func(&z);
  x=*z; //lub x=z[0];
  printf(”%d\n”, x); //tu otrzymamy 5
  y=*z;
  printf(”%d\n”, y); //tu także otrzymamy 5
  return 0;
}
```

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Rola stosu w systemie komputerowym
  - Przypomnienie – używanie pamięci dynamicznie alokowanej
    - Alokacja

Tu może pojawić się problem jak przetestować ten fragment!

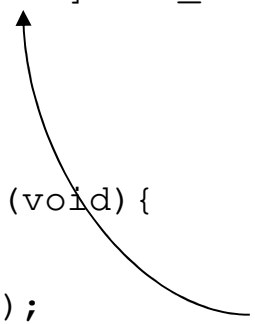
```
#define LICZBA_STUDENTOW  32
typedef struct {
   int lab, egzamin;
} ocena_t;                    //definiujemy obiekt
ocena_t *ocena;               //tu tylko deklarujemy wskaźnik
...
oceny=(ocena *)malloc(LICZBA_STUDENTOW*sizeof(ocena_t));
if(oceny != NULL){            //NULL - specjalna wartość typowo: (void*)0
   printf("Brak pamieci! (F:%s, L:%d)\n", __FILE__, __LINE__);
   ...          //zwolnienie innych dynamicznie utworzonych obiektów,
   ...          //zamknięcie plików, zamknięcie gniazd siecowych, ...
   exit(1);
}
...             //operacje na obiekcie wskazanym przez zmienną 'oceny'
free(oceny);                  //zwalniamy z pamięci objekt 'oceny'
```

    - Zwalnianie pamięci dynamicznie zaalokowanej to obowiązek programisty (!)
      - w C nie ma automatycznego Garbage Collector'a, to wiele systemów operacyjnych usuwając proces usuwa także jego struktury dynamicznie zaalokowane

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Przepełnienie stosu – niebezpieczeństwa
  - Stos ma skończoną pojemność i może jego wielkość nie być chroniona
  - Przykład błędnego kodu

```
void func(void){
   uint32_t buffer[20];
   ...
   strcpy((char*)buf, „NAPIS_DLUZSZY_NIZ_20_ZNAKOW”); //tu zostanie
                                  //zamazany adres powrotu – co powodować
                                  //będzie np.: zawieszenie programu,
                                  //niestabilność
   ...
   buf[1003]=BAD_CODE_ADDRESS;  //typowe nadużycie – tu adres
                                //powrotu podmieniany jest na złośliwy kod
   ...
}
int main(void){
   ...
   func();
   ...
}
```



Gdy ten adres będzie precyzyjnie ustalony - tak by trafić w obszar pamięci stosu gdzie składowany jest adres powrotu z funkcji main() - może ułatwić wykonanie niedozwolonych operacji po zakończeniu wykonywania tego kodu, przejmując prawa tego kodu

Adres taki może zależeć w danej aplikacji: od wersji jej kodu, od wersji użytego kompilatora i od wersji systemu operacyjnego – generalnie trudne do ustalenia ale możliwe (!)

Istnieją w systemach operacyjnych funkcję randomizacji adresów

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Funkcje „niebezpieczne” w C
  - Biblioteka LIBC zawiera funkcje obsługi ciągów tekstowych
    - Wersje niebezpieczne:

```
size_t strlen(const char *string);
int strcmp(const char *s1, const char *s2);
int strcpy(char *dst, const char *src);
int sprintf(char *str, const const char *format, ... );
char *strcat(char *dst, const char *src);
...
```

    - Wersje bezpieczne:

```
size_t strnlen(const char *s, size_t maxlen);
int strncmp(const char *s1, const char *s2, size_t maxlen);
int strncpy(char *dst, const char *src, size_t maxlen);
int snprintf(char *str, size_t maxlen, const char *format, ... );
char *strncat(char *dst, const char *src, size_t maxlen);
...
```

    - Wersje bez swojego bezpiecznego odpowiednika: atol(), atof(), atoi(), …

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Zaufanie w dostępie do danych i kodu – rozważmy przypadki:
  - Czy kod aplikacji może odwołać się do danych systemu operacyjnego?
    - Np.: miejsca gdzie przechowywane są hasła?
    - NIE !
  - Czy kod aplikacji może wywołać kod systemu operacyjnego?
    - Np.: funkcji prywatnej jądra systemu odpowiedzialnej za dostęp do pliku z hasłami?
    - Nie - jak zatem wywołać usługi systemowe?
      - Istnieją specjalne instrukcje np.: INT xx czy syscall za pomocą których przekazuje się sterowanie do systemu operacyjnego
      - Choć są od tej reguły wyjątki
  - Czy system operacyjny może wywołać funkcje z kodu aplikacji?
    - Zasadniczo jeżeli ufa aplikacji - systemy operacyjne posiadają odpowiednie mechanizmy
      - W systemie Linux rozwiązaniem jest sprawdzenie bitu „X” w prawach dostępu do danego pliku, np.:
        - -rw**x**r-**x**r-**x** 1 root root 1,1M maj 15 2020 /bin/bash
      - Domyślnie zakłada się, że funkcja main() jest miejscem przejścia między OS a aplikacją
        - funkcja main() jest naturalnym tzw. callback’iem

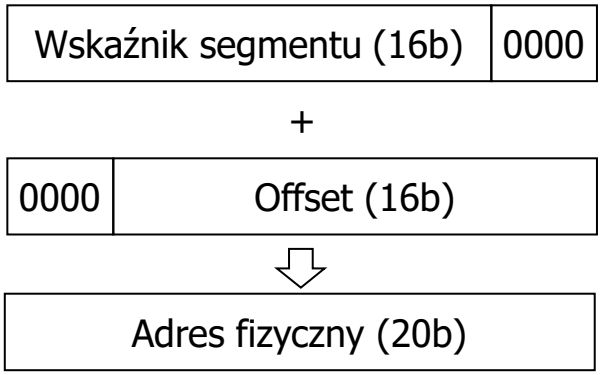
Metody separacji procesów

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Separacja procesów
  - To jedna z metod ochrony przed nadużyciami złośliwego oprogramowania
    - Aplikacje „mają wrażenie” bycia jedynymi w systemie
  - Przestrzenie adresowe mogą stosować
    - Adresy logiczne
      - Inaczej adresy tworzące przestrzeń wirtualną, pomagającą odzwierciedlić budowę aplikacji (kod, dane, stos)
    - Adresy fizyczne
      - Faktyczny adres używany do odwołania się do określonego miejsca w zewnętrznych modułach pamięci
      - W systemach bez tzw. translacji adresów – czyli bez MMU – adresy fizyczne są równoważne adresom wirtualnym
        - Mikroprocesor wyposażony w MMU także nie wspiera powyższych mechanizmów dopóki nie zostanie odpowiednio skonfigurowany jego MMU
  - Typowo systemy komputerowe wspierają translacje adresów opartą o
    - Segmentacje
      - Gdzie adres tworzą: wskaźnik segmentu i offset w jego obrębie
    - Stronicowanie
      - Gdzie przestrzeń jest podzielona na jednakowej długości strony
  - Są systemy które wykorzystują oba mechanizmy(!)

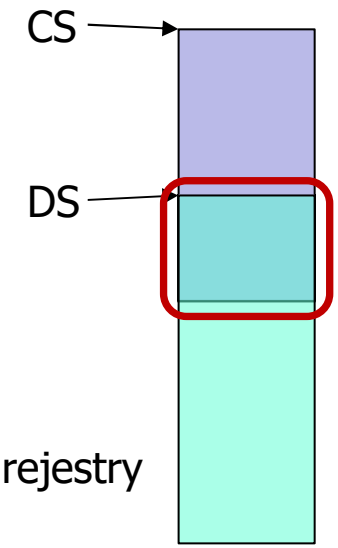
SYKO Sprzętowe wspomaganie systemu operacyjnego

- Translacja adresów
  - Translacja adresu logicznego na fizyczny - przypadek z architektury X86 pracującej w tzw. trybie rzeczywistym



SYKO Sprzętowe wspomaganie systemu operacyjnego

- Translacja adresów
  - Przypadek z architektury X86 pracującej w tzw. trybie rzeczywistym, cd.
    - Zalety
      - Prostota
    - Wady
      - Brak separacji
      - Segmenty mogą nachodzić na siebie(!)
      - Ograniczona przestrzeń adresowa
        - Adres może składać się z tylko 20bitów
    - Przykłady wskaźników segmentów – przechowywały taki wskaźnik rejestry segmentowe
      - CS – segment kodu
      - DS – segment danych
      - SS – segment stosu
      - ES, FS, GS – segmenty dodatkowe
      - Segment miał stała długość 65536B



SYKO Sprzętowe wspomaganie systemu operacyjnego

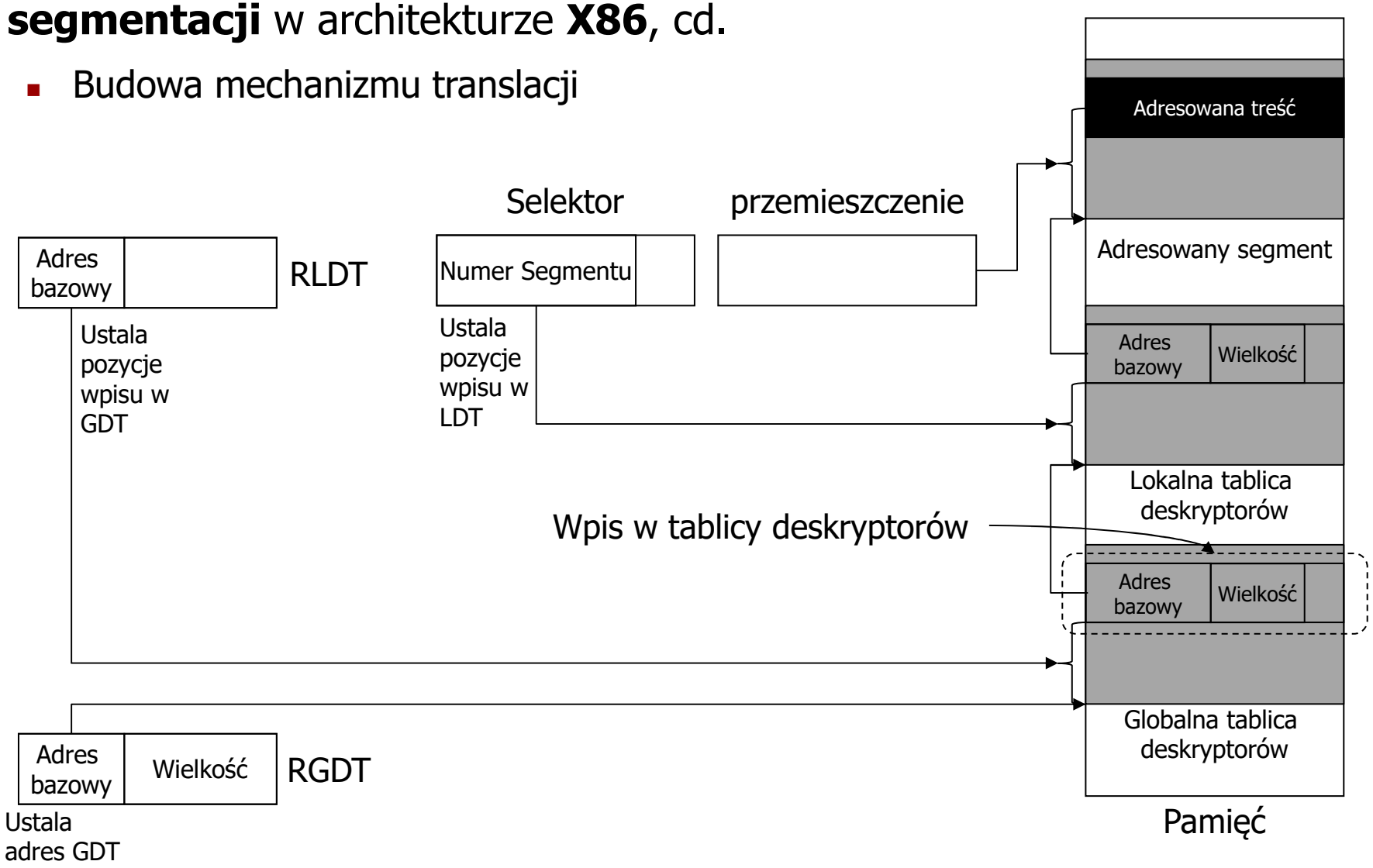
- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86**
  - Adres składa się z elementów:

    Selektor : przemieszczenie

  - Selektor to
    - 13 bitów wskazujących na numer segmentu w tablicy deskryptorów
    - 1 bit (TI) określający typ segmentu (0-globalny, 1-lokalny)
    - 2 bity (RPL) poziom żądania przywileju przy dostępie do segmentu
      - Poziomy uprzywilejowania:
        - 0 – największe uprzywilejowanie, np.: jądro systemu
        - 1 – sterowniki systemu
        - 2 – bazy danych, usługi
        - 3 – aplikacje użytkowe

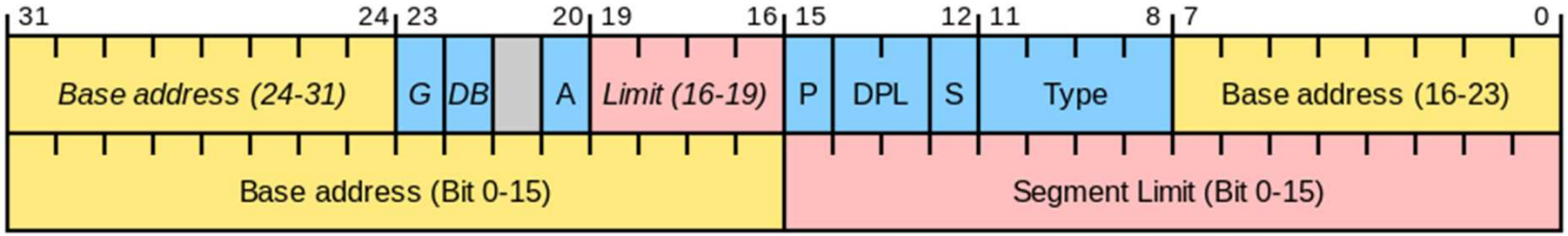
SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86**, cd.
  - Budowa mechanizmu translacji



SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **segmentacji** w architekturze **X86**, cd.
  - Pola w tablicy deskryptorów (zarówno globalnej jak i lokalnej)
    - Adres bazowy segmentu (32 bity), wielkość segmentu (20 bitów)
    - Atrybuty segmentu (12 bitów) a wśród nich najważniejsze to
      - P – (Presence) obecność segmentu w pamięci (1 bit), DPL – poziom ochrony segmentu (2 bity)
      - G – (Graininess) ziarnistości pola wielkość segmentu (1 bit), jeżeli 0 to pole wielkości opisuje jednostka 1B, jeżeli 1 to jednostką jest 4096B
        - wtedy dla wyznaczenia wielkości segmentu pole wielkości segmentu przemnaża się przez 4096
      - A – (Absence) użycie segmentu (1 bit), jeżeli 1 to segment był użyty
      - TYP – co przechowuje segment (3 lub 4 bity) i może to być segment:
        - Z danymi tylko do odczytu lub z danymi do odczytu i zapisu
        - Rozszerzalny w dół tylko do odczytu lub rozszerzalny w dół do odczytu i zapisu
        - Z kodem i być tylko do wykonania lub do wykonania i odczytu
        - Tzw. zgodny z kodem tylko do wykonania lub zgodny do wykonania i odczytu



Źródło:
wikipedia.org

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów – mechanizmy ochrony zasobów w **segmentacji** w architekturze **X86**, cd.
  - Separacji zdań
    - Zadania widzą pamięć tak jakby w systemie niebyło innych zadań
  - Kontrola przemieszczeń
    - System operacyjny przydziela zasoby pamięciowe zadaniom a mechanizmy sprzętowe kontrolują przestrzeganie tego przydziału
      - Segment jest opisany jego wielkością – łatwo wykryć odwołanie poza jego obszar
  - Kontrola typów segmentów podczas odwołań
    - Mechanizm blokuje możliwość użycia segmentu w niewłaściwy sposób
      - Segment zawiera pole określające typ - pozwala ono uniemożliwić np.: wykonywanie kodu z pamięci danych, modyfikacje segmentu kodu, ...

SYKO Sprzętowe wspomaganie systemu operacyjnego

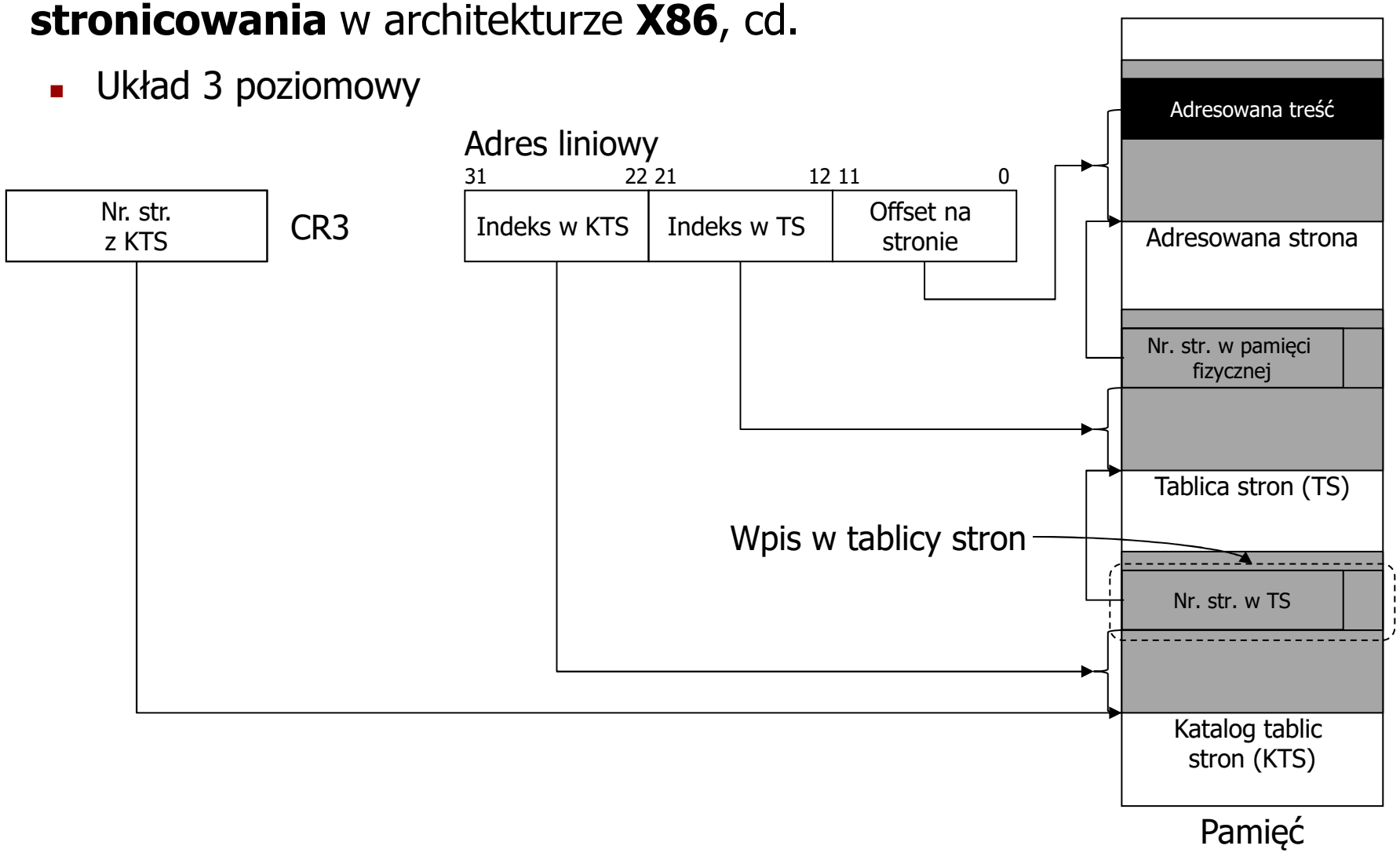
- Metody separacji procesów – mechanizmy ochrony zasobów w **segmentacji** w architekturze **X86**, cd.
  - Kontrola poziomów ochrony
    - Mechanizm określa kto z danego zasobu pamięciowego może korzystać
      - Deskryptor segmentu zawiera pole DPL (Descriptor Privilege Level)
      - Podczas działania danego zadania wykonuje się ono z określonym przez system operacyjny poziomem CPL (Current Privilege Level)
      - Zadanie ma dostęp tylko do segmentów __danych__ których DPL jest większy lub równy od CPL (co do wartości)
        - Zadanie nie może odwoływać się do danych będących własnością systemu operacyjnego
        - Zadanie może operować na danych równie ważnych co to zadanie lub mniej ważnych
      - Zadanie ma dostęp tylko do segmentów __kodu__ których DPL jest mniejszy lub równy względem CPL
        - Zadanie może swobodnie wywoływać usługi systemu operacyjnego
        - Zadnie nie może wywoływać mniej zaufanego kodu
      - Istnieje także tzw. RPL (Requestor's Privilege Level)
        - Rozwiązuje problem gdy funkcja systemowa ma odczytać dane użytkownika by np.: zapisać je na dysku (systemowa funkcja write())
        - Dzięki sprawdzeniu warunku max(CPL,RPL) <= DPL, gdzie RPL staje się CPL kodu wołającego usługę systemowa, mechanizm może dopuścić do wykonania danej operacji

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86**
  - Budowa wpisu w tablicy stron (TS) lub w katalogu tablic stron (KTS) – pola
    - Numer ramki strony w pamięci fizycznej
    - Atrybuty strony (12 bitów), najważniejsze z nich
      - D (Dirty) – czy treść strony została zmieniona (1 bit)
      - A (Accessed) – czy strona była używana (1 bit)
      - P (Present) – czy strona jest obecna w pamięci fizycznej (1 bit)
      - U/S (User/Supervisor) – czy właścicielem strony jest użytkownik czy system (1 bit)

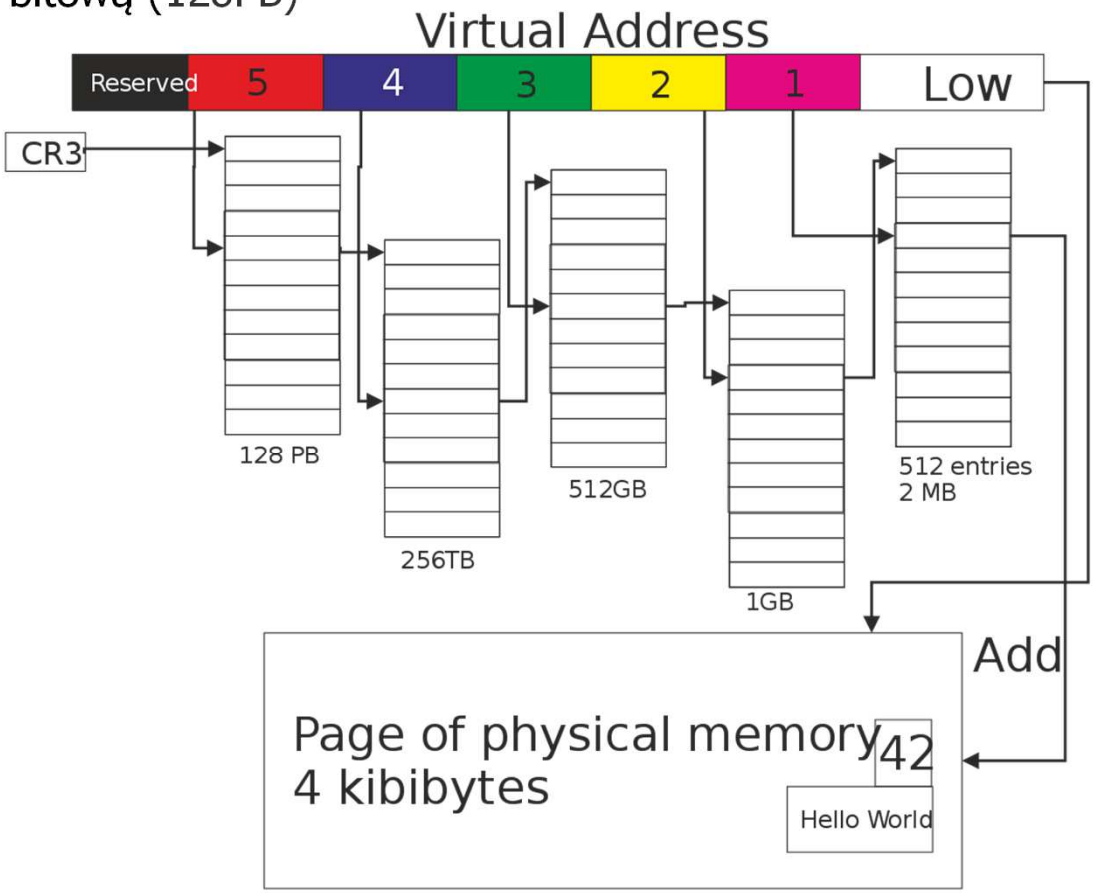
SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86**, cd.
  - Układ 3 poziomowy



SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **X86-64**
  - Układ 5 poziomowy (Intel 5-level paging)
    - Obsługuje przestrzeń 57 bitową (128PB)



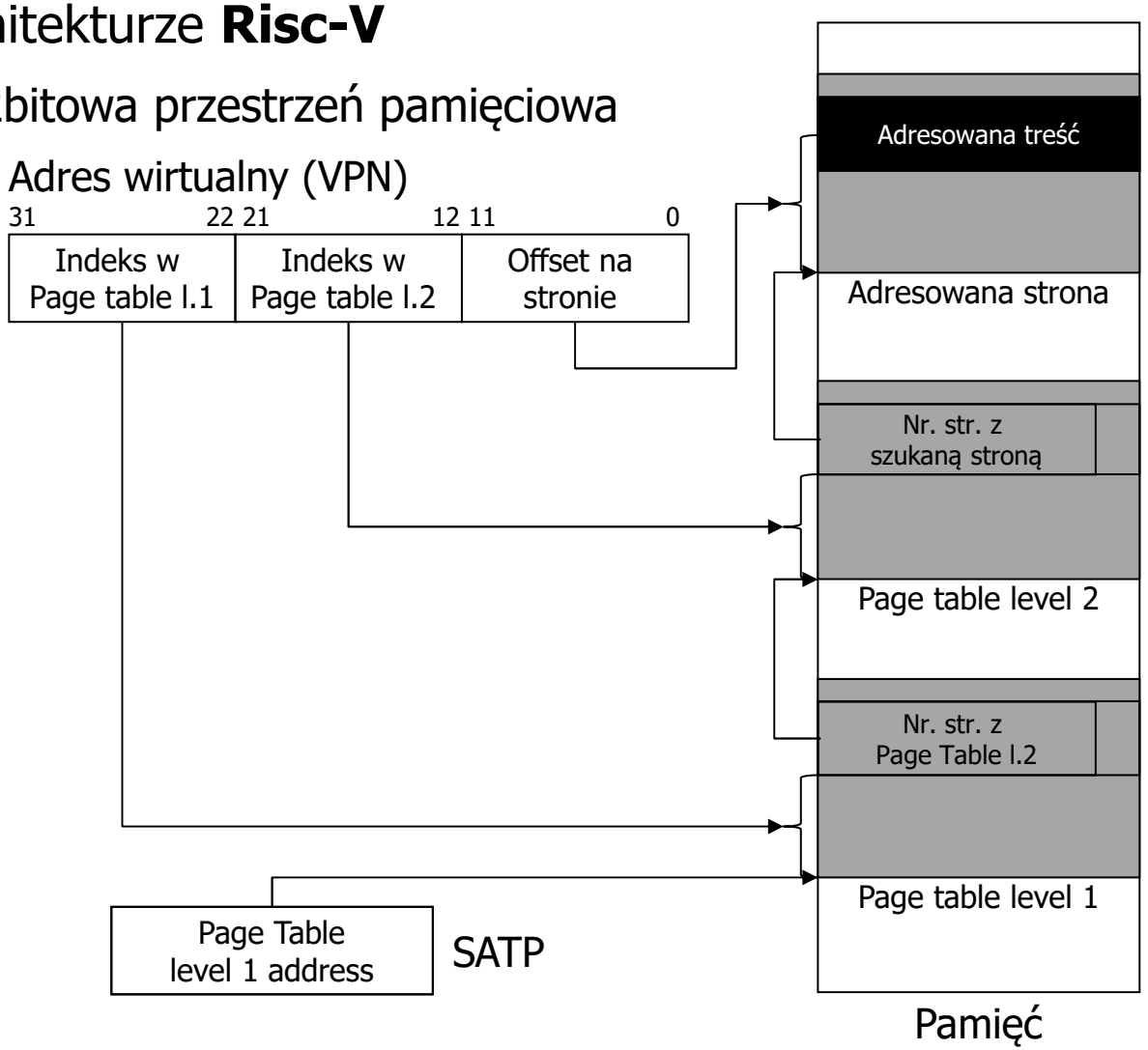
Źródło: wikipedia.org

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów – zalety i wady obu mechanizmów ochrony pamięci w **x86**
  - Zalety obu
    - Możliwość przechowywania treści segmentów w pamięci dyskowej (plik/partycja wymiany) gdy w danym momencie brak pamięci fizycznej
    - Manipulując rejestrami RGDT/RLDT lub CR3 można przełączać się między zadaniami
      - Każde zadanie ma własny zestaw segmentów/stron – więc przełącza się organizację pamięci
  - Zalety segmentacji
    - Możliwość uporządkowania przestrzeni pod względem spełnianych funkcji
      - Przestrzeń pamięci kodu odseparowana od pamięci danych, stos rozszerzający się, separacja jednej dużej tablicy z danymi od innej
  - Zalety stronicowania
    - Podział pamięci na stałej długości elementy dzięki czemu łatwo ich treść przenosić z pamięci fizycznej do pamięci wymiany
      - Tu widać wadę segmentacji – segmenty o różnych długościach trudniej przenosić z/do pamięci wymiany, pojawia się wtedy problem dopasowania i pozostających dziur w takiej pamięci

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**
  - Mechanizm **Sv32** –32bitowa przestrzeń pamięciowa



- Strony 4KB
- VPN – Virtual Page Number
- PPN – Physical Page Number
- Format wpisu w tablicy stron (page-table entries, PTE)
  - PPN[1] – bity 31…20
  - PPN[2] – bity 19…10

SYKO Sprzętowe wspomaganie systemu operacyjnego

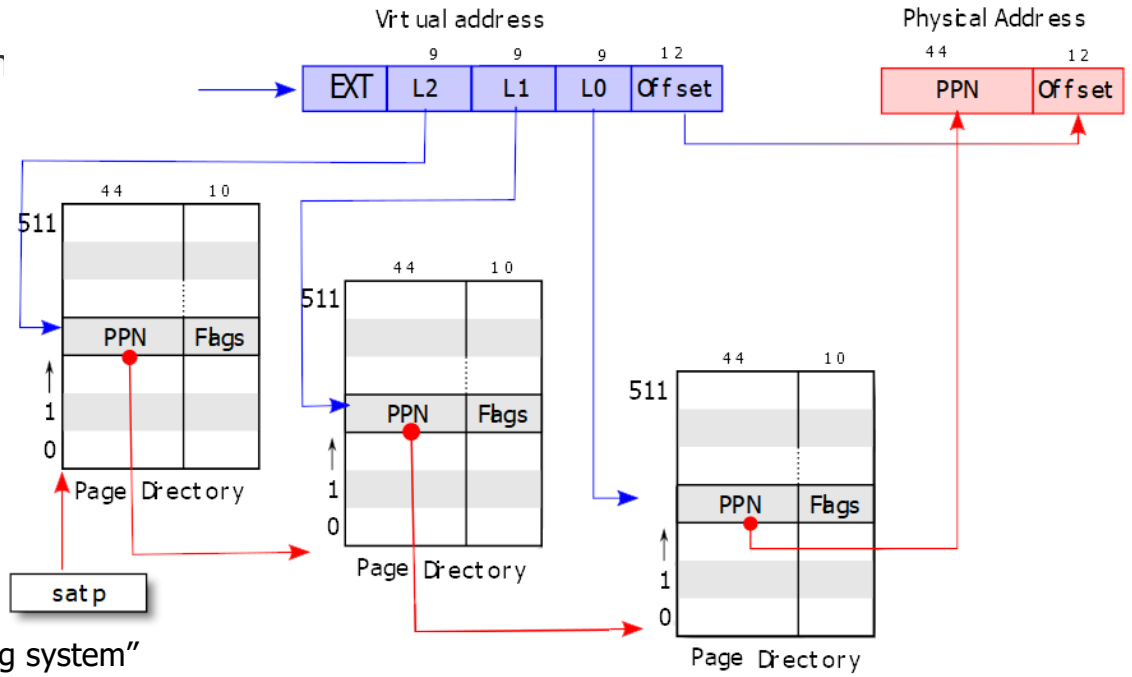
- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**, cd.
  - Format wpisu w tablicy stron (page-table entries, PTE) – pozostałe pola
    - Atrybuty strony (10 bitów), najważniejsze z nich
      - V (Valid) – czy wpis jest aktywny (1 bit)
      - X/W/R – atrybut praw dostępu wykonanie/pisanie/czytanie (3 bity)
        - Gdy te bity maja wartość 000 – wykonywane jest odwołanie od „Page Table" następnego poziomu
      - U – strona użytkowana przez system czy aplikację użytkową (1bit)
      - D (Dirty) – czy treść strony została zmieniona (1 bit)
      - A (Accessed) – czy strona była używana (1 bit)

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

Źródło: RISC-V Privileged Architectures

SYKO Sprzętowe wspomaganie systemu operacyjnego

- Metody separacji procesów - wirtualizacja adresów z wykorzystaniem **stronicowania** w architekturze **Risc-V**, cd.
  - Mechanizm translacji dostępne w architektach 64bitowych (RV64)
    - **Sv39**
      - Implementacja wirtualnej przestrzeni o adresach zapisanych na 39 bitach (czyli 512GB)
    - **Sv48**
      - Implementacja wirtualnej przestrzeni o adresach zapisanych na 48 bitach (czyli 256TB)



Źródło: „xv6: a simple, Unix-like teaching operating system”

Dziękuje za uwagę