Obsługa urządzeń wejścia-wyjścia

Jak ma współdziałać jednostka centralna z urządzeniami wejścia-wyjścia?

- sterownik urządzenia, mapowanie rejestrów urządzenia, MMIO
- aktywne czekanie
- odpytywanie
- przerwania
- bezpośredni dostęp do pamięci

Sterowniki i moduły sterujące urządzeń

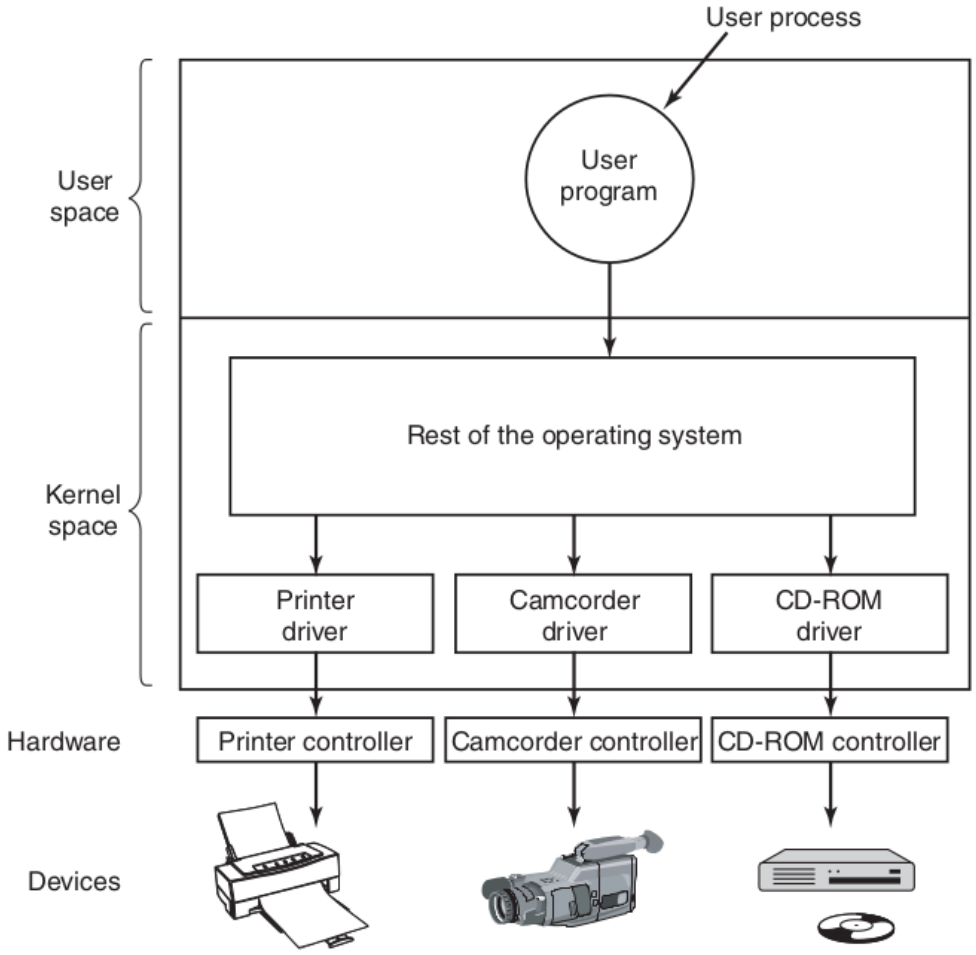


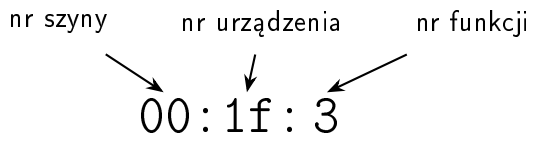
Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Sterowniki i moduły sterujące urządzeń

- **Sterownik urządzenia** (*device controller*) związany jest z konkretnym urządzeniem i dysponuje lokalnym buforem i zbiorem rejestrów o specjalnym przeznaczeniu. Odpowiada za przesyłanie danych między urządzeniem zewnętrznym a własnym buforem.
- **Moduł sterujący (obsługi) urządzenia** (*driver*) jest odpowiedzialny od strony systemu operacyjnego za komunikację ze sterownikiem urządzenia.

```
00:1f.3 SMBus: Intel Corporation 8 Series SMBus Controller (rev 04)
        Subsystem: Lenovo Device 220c
        Flags: medium devsel, IRQ 18
        Memory at e0638000 (64-bit, non-prefetchable) [size=256]
        I/O ports at efa0 [size=32]
        Kernel driver in use: i801_smbus
        Kernel modules: i2c_i801
```

Pliki `/proc/iomem`, `/proc/ioports`, polecenie: `lspci -v`

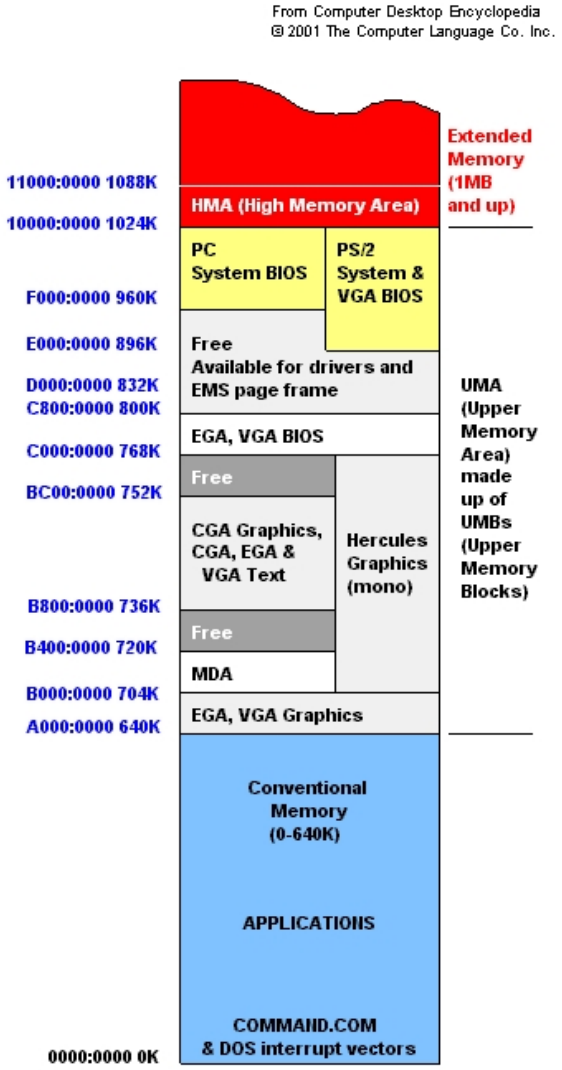


MMIO (memory-mapped input/output)

- komunikacja z urządzeniem - zapis/odczyt rejestrów urządzenia
- zbiór rejestrów urządzenia tworzy **przestrzeń portów wej-wyj**
  - potrzebne specjalne operacje dostępu `IN-OUT`
- odwzorowanie rejestrów w przestrzeni adresowej pamięci MMIO (*memory-mapped input/output*)
  - nie potrzeba dodatkowych instrukcji obsługi
  - operacja zapisu (odczytu) adresu pamięci skutkuje zapisem (odczytem) fizycznego rejestru urządzenia
  - ochrona dostępu realizowana za pomocą mechanizmów ochrony pamięci
  - część przestrzeni adresowej jest zarezerwowana dla urządzeń, nie może być użyta do komunikacji z pamięcią - zjawisko dziury w pamięci[1]

[1] https://pl.wikipedia.org/wiki/MMIO

MMIO (memory-mapped input/output)




```
       0.0000 M        0.0039 M  Reserved
       0.0039 M        0.6133 M  System RAM
       0.6133 M        0.6250 M  Reserved
       0.6250 M        0.7500 M  PCI Bus 0000:00
       0.7500 M        0.8076 M  Video ROM
       0.8125 M        0.8281 M  pnp 00:00
       0.8281 M        0.8437 M  pnp 00:00
       0.8438 M        0.8594 M  pnp 00:00
       0.8594 M        0.8750 M  pnp 00:00
       0.8750 M        1.0000 M  Reserved
       0.9375 M        1.0000 M  System ROM
       1.0000 M     2785.2656 M  System RAM
     352.0000 M      360.6260 M  Kernel code
     360.6260 M      368.3481 M  Kernel data
     370.6484 M      372.5781 M  Kernel bss
    2785.2656 M     3000.4531 M  Reserved
    3000.4531 M     3000.7656 M  ACPI Non-volatile Storage
    3000.7656 M     3020.4648 M  Reserved
    3020.4648 M     3022.4961 M  ACPI Non-volatile Storage
    3022.4961 M     3022.9961 M  ACPI Tables
...
    4077.2500 M     4077.2969 M  PCI Bus 0000:00
    4077.2500 M     4077.2695 M  TPM
    4077.2695 M     4077.2969 M  pnp 00:01
    4078.0000 M     4078.0039 M  Local APIC
    4078.0000 M     4078.0039 M  Reserved
    4092.0000 M     4096.0000 M  Reserved
```

Aktywne czekanie

Aktywne czekanie (*busy waiting*) polega na ciągłym sprawdzaniu stanu zasobów (np. rejestrów, pamięci) w celu ustalenia, czy są one dostępne do dalszej pracy.

1. sprawdź, czy drukarka jest gotowa na przyjęcie następnego znaku
2. jeśli nie jest gotowa, to idź do punktu 1
3. jeśli drukarka jest gotowa (po wydrukowaniu znaku), to sprawdź czy jest do wydrukowania nowy znak
4. jeśli jest nowy znak, to idź do punktu 1
5. jeśli nie ma więcej znaków, to drukowanie zostało zakończone

Ograniczenia:

- marnowanie cykli procesora na oczekiwanie na dostępność zasobów
- brak reakcji systemu na inne zdarzenia w czasie oczekiwania

Odpytywanie (*polling*)

Procesor regularnie sprawdza stan kolejnych urządzeń (np. dysków, portów) w celu ustalenia, czy są one gotowe do wykonania operacji

1. wybierz kolejne urządzenie wymagające obsługi
2. sprawdź, czy to urządzenie wymaga obsługi
3. jeśli tak, to uruchom procedurę obsługi urządzenia
4. jeśli nie, to przejdź do punktu 1.

Ograniczenia:

- marnowanie cykli procesora na regularne sprawdzanie stanu urządzeń
- opóźnienia w reakcji na zdarzenia w porównaniu do obsługi przerwań

Obsługa przerwań

Przerwanie to żądanie wysyłane do procesora, aby przerwał aktualnie wykonywane zadanie i zajął się obsługą zdarzenia

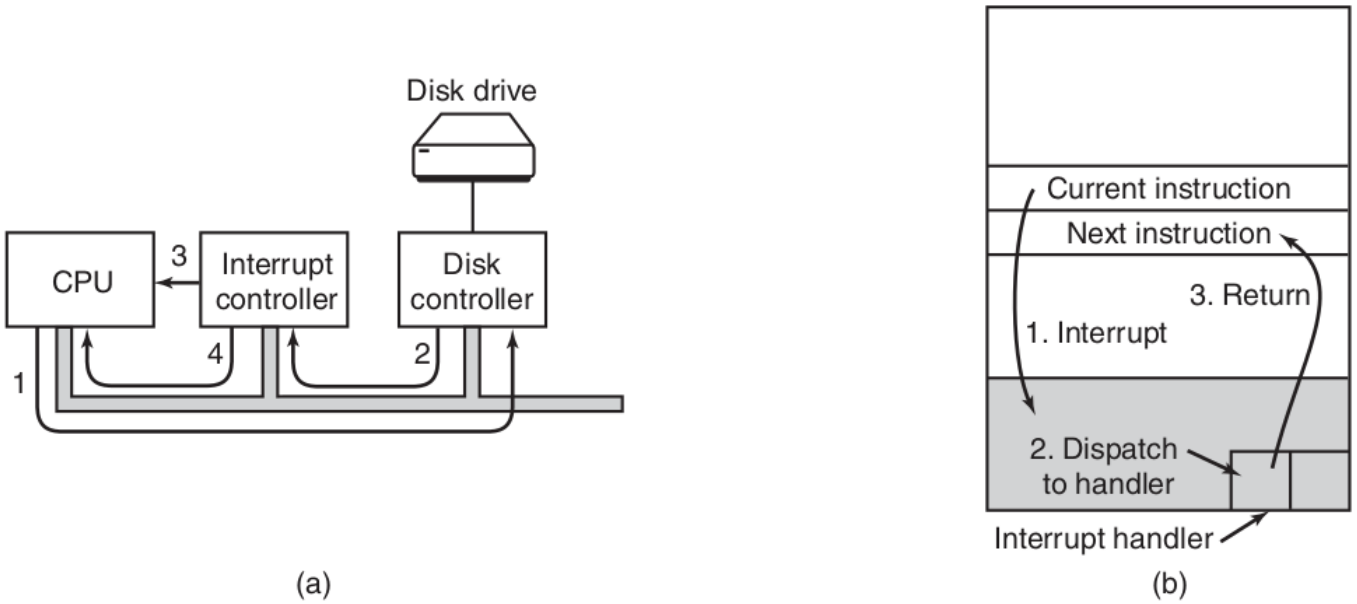


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Obsługa przerwań

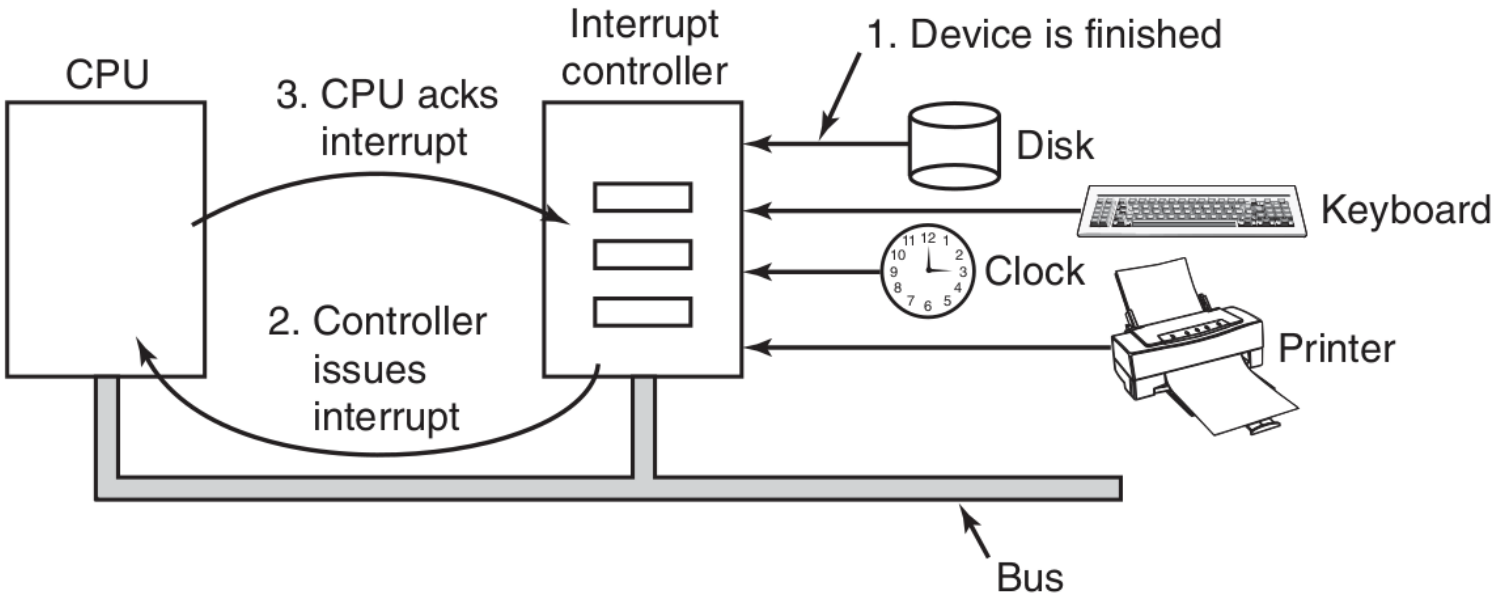


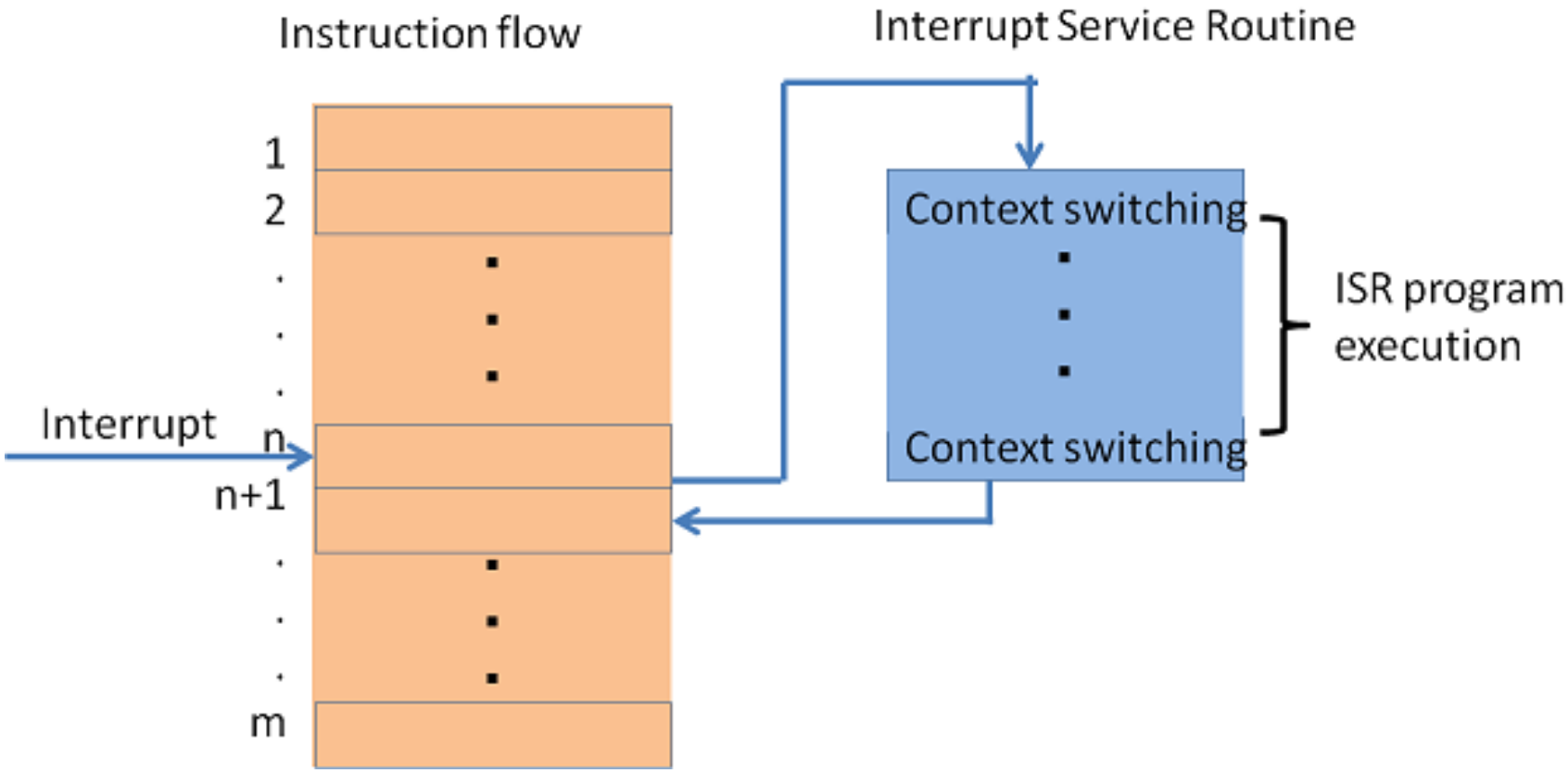
Figure 5-5. How an interrupt happens. The connections between the devices and the controller actually use interrupt lines on the bus rather than dedicated wires.

Przerwania: działanie

- jednostka centralna inicjuje przesyłanie danych przez wprowadzenie pewnych wartości do odpowiednich rejestrów sterownika urządzenia
- sterownik urządzenia rozpoczyna działanie (gromadzenie danych w buforze)
- sterownik urządzenia powiadamia procesor o zakończonej pracy, generując określone przerwanie
- procesor wstrzymuje bieżącą pracę, odkłada na stos adres przerwanego rozkazu, określa źródło przerwania i przekazuje sterowanie do procedury obsługi przerwania *(interrupt handler, interrupt service routine (ISR))* wykorzystując wektor przerwań lub odpytywanie
- przesłanie danych z bufora do programu użytkownika
- wznowienie przerwanej pracy

Procedura obsługi przerwania jest częścią modułu obsługi urządzenia, czyli kodu jądra zarządzającego urządzeniem.

Procedura obsługi przerwania



Przerwania: obsługa

- przerwanie sprzętowe może zostać zainicjowane w dowolnym momencie
- procedura obsługi przerwania powinna być krótka i wykonywana bez przerywania
  - wymagania urządzenia: jak najszybsza obsługa przerwania
  - wymagania systemu operacyjnego: jak najkrótsza obsługa
  - co zrobić, gdy podczas obsługi przerwania wystąpi następne żądanie?
- możliwe jest zagnieżdżanie obsługi przerwań lub odraczanie wykonania procedury obsługi (zależy od architektury i konfiguracji systemu)
- w Linuksie obsługa przerwania jest podzielona na etapy:
  - górna połówka (*top half*) - niezbędne operacje wykonane natychmiast po wystąpieniu przerwania
  - dolna połówka (*bottom half*) - instrukcje odłożone do późniejszej obsługi

http://www.jonmasters.org/blog/2007/12/12/everything-you-know-about-interrupts-is-wrong/

Przerwania: podział

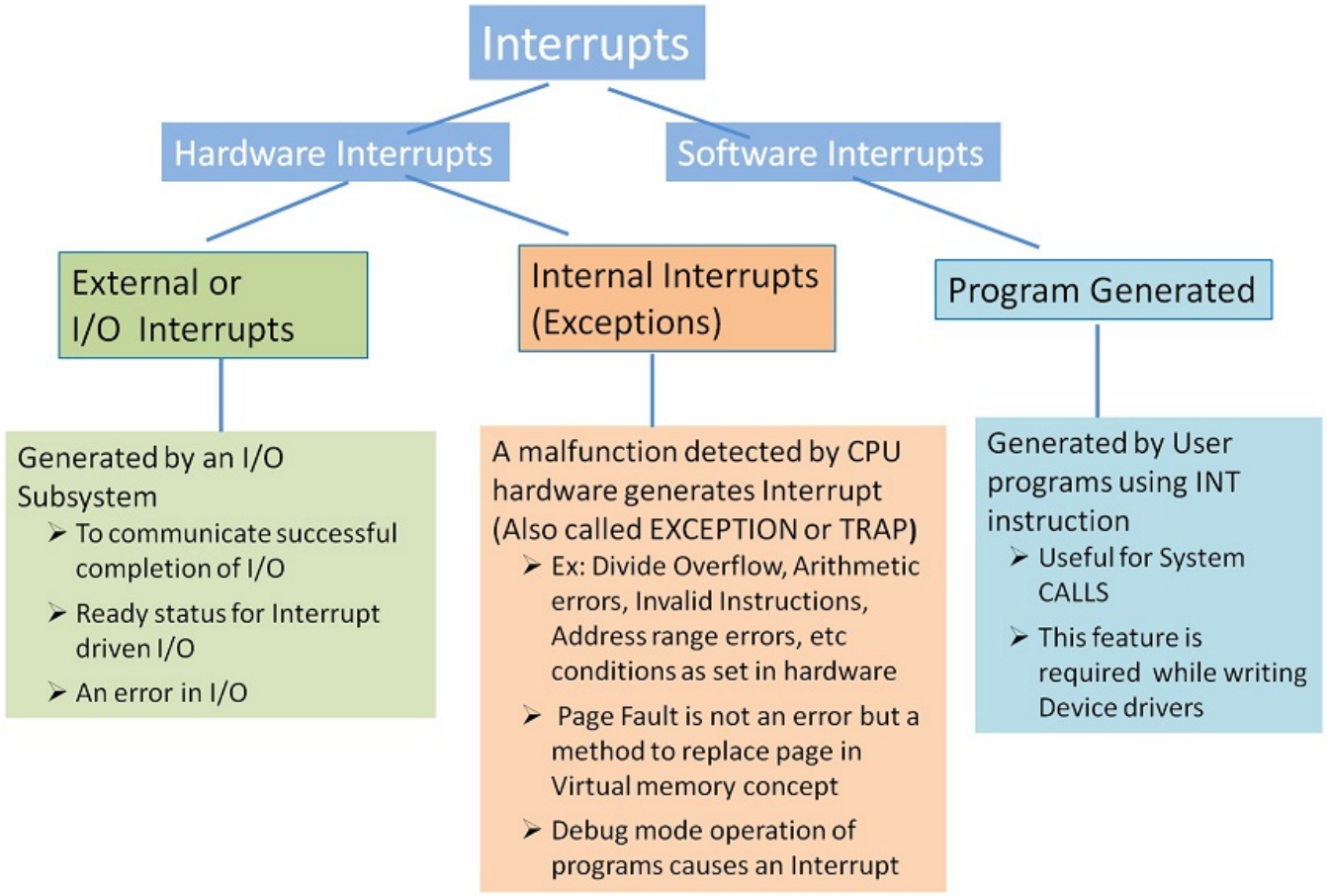
- **asynchroniczne** (przerwania) – generowane przez urządzenia sprzętowe w dowolnym czasie, niezależnie od sygnałów zegarowych procesora, np. przerwanie z karty sieciowej po odebraniu pakietu danych
- **synchroniczne** (wyjątki) – generowane przez procesor w wyniku wykonania instrukcji, tworzone przez jednostkę sterowania procesora w celu obsłużenia sytuacji wynikającej z wykonania instrukcji, np. dzielenie przez zero, wywołania systemowe

Przerwania: źródła

Systemy z obsługą przerwań są kierowane zdarzeniami generowanymi przez

- przerwania sprzętowe (*interrupts*)
  - zegar - operacje wykonywane w równych odstępach czasu
  - urządzenia I/O - generowane przez sterownik urządzenia
  - awaria sprzętu
- przerwania wywołane instrukcjami programów (*exceptions*)
  - błędy programów (*faults*), nieprawidłowe operacje,
  - załamania (*aborts*)
  - pułapki (*traps*), przerwania programowe

Przerwania: źródła



Przerwania sprzętowe: podział wg dokumentacji Intela

- **maskowalne** – mogą być ignorowane
  - dwa stany przerwań: zamaskowany (ignorowany), niezamaskowany
  - możemy tymczasowo odłożyć uruchomienie procedury obsługi przerwań do czasu, gdy przerwanie zostanie ponownie włączone
  - wszystkie IRQ (*Interrupt ReQuests*) generowane przez urządzenia I/O powodują powstanie przerwań maskowalnych
  - sygnalizowane przez pin INT procesora.
- **niemaskowalne** – nie mogą być ignorowane
  - krytyczne zdarzenia (np. awaria sprzętu) zawsze rozpoznawane przez CPU
  - sygnalizowane przez pin NMI (*NonMaskable Interrupt*)

Wyjątki wykrywane przez procesor

Wyjątki powstają, gdy podczas wykonywania instrukcji procesor wykryje nieprawidłowość.

- **błąd** (*fault*) – po obsłużeniu wyjątku, procesor może wznowić wykonanie od instrukcji, która spowodowała błąd.
  np. błąd strony, dzielenie przez zero, naruszenie ochrony pamięci
- **pułapka** (*trap*) – zgłaszana po wykonaniu instrukcji pułapki. Po obsłużeniu procesor wznawia wykonanie od następnej instrukcji.
  np. śledzenie programu podczas debugowania
- **załamanie** (*abort*) – wystąpił poważny błąd i jednostka sterowania nie może zachować adresu instrukcji; zwykle oznacza zakończenie procesu, który spowodował załamanie.
  np. krytyczny błąd sprzętu.

Wyjątki programowe – występują na żądanie programisty (np. instrukcja `INT N`), są obsługiwane za pomocą pułapek
np. wywołanie funkcji systemowej, debugowanie programu

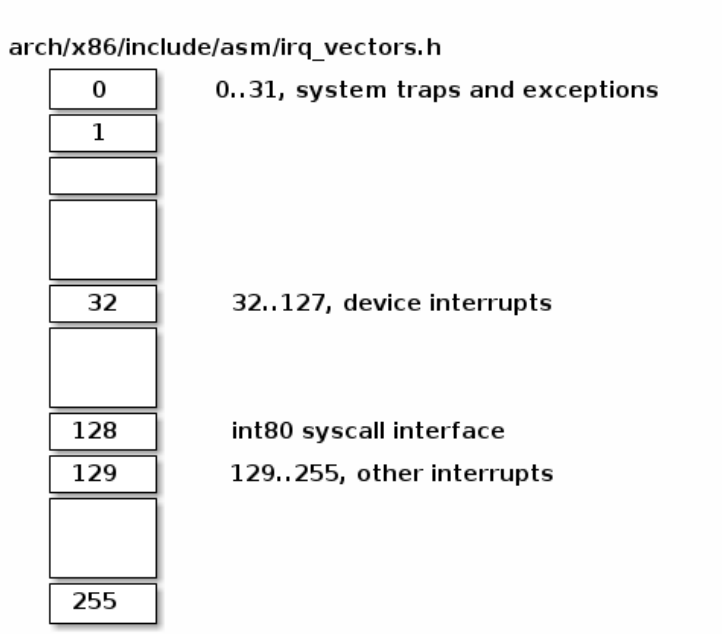
Przerwania: identyfikacja

Każde przerwanie lub wyjątek jest identyfikowane przez liczbę z zakresu od 0 do 255 (Intel nazywa tę 8-bitową liczbę bez znaku **wektorem**).

Tablica deskryptorów przerwań IDT

Interrupt descriptor table

- struktura używana w architekturach x86 do implementacji wektora przerwań
- odwzorowuje numer przerwania na adres funkcji obsługi
- tablica 256 wektorów po 8B, max. rozmiar 2048B
- może znajdować się w dowolnym miejscu pamięci, pozycję tablicy wskazuje rejestr procesora `IDTR`



Tablica deskryptorów przerwań IDT

Rodzaje zdarzeń:

- wyjątki procesora 0-31, stałe mapowanie
- przerwania sprzętowe odwzorowują IRQ sprzętu zależnie od kontrolera przerwań 32-127
- przerwania programowe 128-255 definiowane przez BIOS i system operacyjny. Wzbudzane instrukcją asemblera `INT X` przez programy, moduły obsługi urządzeń lub inne funkcje obsługi przerwań.
- w systemie Linux wektor 128 (`0x80`) wykorzystywany do wywołań funkcji systemowych

Wyjątki procesora x86

konflikt wyjątków

błąd przy zmianie kontekstu

naruszenie trybu chronionego

źle dopasowany adres

błąd szyny lub CPU

#	Wyjątek	Rodzaj	Sygnał
0	*Divide error*	fault	SIGFPE
1	*Debug*	fault or trap	SIGTRAP
2	NMI	interrupt	
3	*Breakpoint*	trap	SIGTRAP
4	*Overflow*	trap	SIGSEGV
5	*Bounds check*	fault	SIGSEGV
6	*Invalid opcode*	fault	SIGILL
7	*Device not available*	fault	
8	*Double fault*	abort	
9	*Coprocessor segment overrun*	abort	SIGFPE
10	*Invalid TSS*	fault	SIGSEGV
11	*Segment not present*	fault	SIGBUS
12	*Stack exception*	fault	SIGBUS
13	*General protection*	fault	SIGSEGV
14	*Page fault*	fault	SIGSEGV
15	zarezerowana przez Intela		
16	*Floating point error*	fault	SIGBUS
17	*Alignment check*	fault	SIGSEGV
18	*Machine check*	abort	
19	*SIMD floating point exception*	fault	SIGFPE

https://wiki.osdev.org/Exceptions

Wyjątki procesora x86

- jądro systemu dostarcza funkcje obsługi wyjątków (*exception handler*)
- wyjątki 20-31 - zarezerwowane do przyszłych zastosowań
- w UNIX/Linux funkcje obsługi wyjątku procesora (zazwyczaj) wysyłają sygnały do procesu powodującego wystąpienie wyjątku

Sterownik przerwań PIC w systemach Intel PC

Przyporządkowanie linii IRQ do urządzeń I/O i wektora przerwań w układzie 8259A PIC

IRQ	wektor	urządzenie
0	32	zegar
1	33	klawiatura
2	34	kaskada PIC
3	35	drugi port szeregowy
4	36	pierwszy port szeregowy
5	37	karta dźwiękowa
6	38	stacja dysków
7	39	port równoległy
8	40	zegar systemowy
11	43	interfejs sieciowy
12	44	mysz PS/2
13	45	koprocesor matematyczny
14	46	podstawowy kanał IDA/ATA
15	47	wtórny kanał IDA/ATA

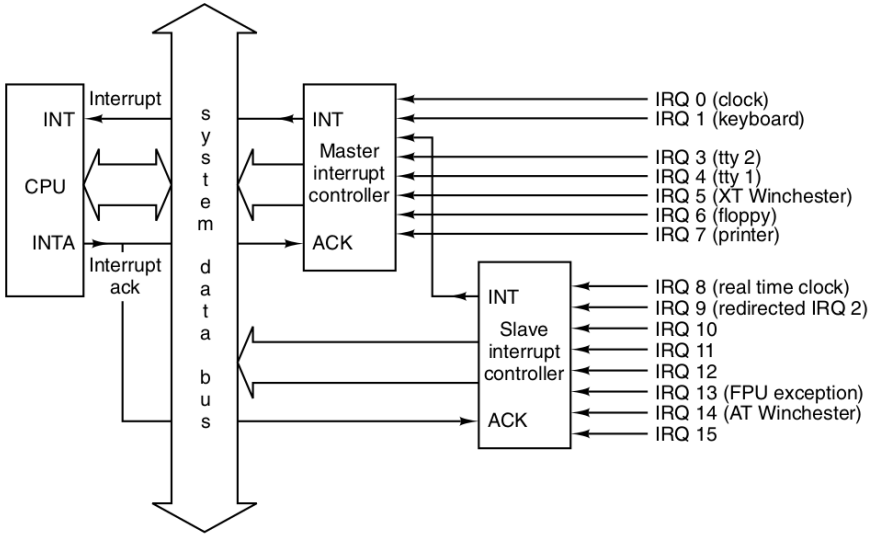


Figure 2-39. Interrupt processing hardware on a 32-bit Intel PC.

- PIC *Programmable interrupt controller* dla jednostek jednoprocesorowych
- układ 8259A obsługuje 8 linii IRQ; po połączeniu kaskadowym (przez pin 2) można obsłużyć do 15 linii IRQ

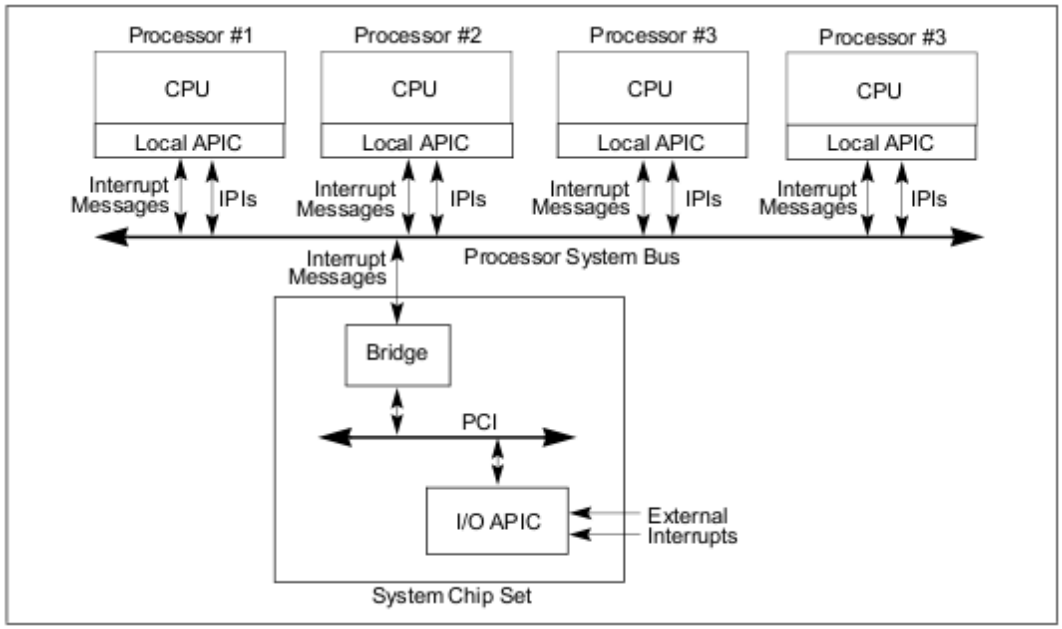
Sterownik przerwań

- linie IRQ łączą urządzenia z pinami wejściowymi programowalnego sterownika przerwań (*Programmable Interrupt Controller*, PIC)
- PIC monitoruje linie IRQ sprawdzając podnoszone sygnały na liniach
- PIC łączy się z pinami INT i NMI procesora oraz udostępnia porty (rejestry) do komunikacji z nim

Gdy urządzenie wymaga obsługi:

1. sterownik urządzenia generuje przerwanie, podnosząc sygnał na linii IRQ (*Interrupt Request*, żądanie przerwania)
2. PIC tłumaczy otrzymany IRQ na odpowiedni wektor
3. zachowuje wektor w porcie I/O sterownika przerwań, co pozwala na odczytanie go przez procesor za pomocą szyny danych
4. wysyła sygnał do pinu INT procesora (generuje przerwanie)
5. czeka, aż procesor potwierdzi otrzymanie sygnału przerwania (w tym czasie PIC nie generuje nowych przerwań)
6. CPU potwierdza przerwanie i uruchamia obsługę przerwania

Kontroler APIC w systemach wieloprocesorowych SMP



Przerwania międzyprocesorowe (IPI, *Inter-Processor Interrupt*) - w systemie wieloprocesorowym (SMP) procesor może zażądać działania ze strony innego procesora (tzw. klepnięcie w ramię), np. unieważnianie wpisów TLB na innych CPU, wykonanie funkcji na innym CPU, zatrzymanie procesora.

APIC (*Advanced Programmable Interrupt Controller*)

- **Local APIC** znajduje się w każdym logicznym procesorze
  - odbiera przerwania skierowane do tego procesora
  - generuje i akceptuje przerwania międzyprocesorowe (IPIs)
  - może generować lokalne przerwania czasowe
- **I/O APIC** przy szynie systemowej
  - odbiera przerwania I/O i kieruje je komunikatami APIC do LAPIC
  - w wielu komputerach PC ma 24 wejścia pinowe (0-23)
  - tabela przekierowań IRT (*Interrupt Redirection Table*) mapuje wejście na wektor, docelowy procesor i tryb dostarczania; system konfiguruje ją przy inicjalizacji
  - obecnie komunikaty MSI/MSI-X omijają wejścia pinowe I/O APIC kierując przerwania bezpośrednio do LAPIC

Przerwania sygnalizowane komunikatami (MSI)

Message Signaled Interrupts

- mechanizm MSI (PCI 2.2) i MSI-X (PCI 3.0, stosowane także w PCIe) zgłaszają przerwania przez zapis pod specjalnie zarezerwowany adres pamięci (w x86 jest to adres rejestru LAPIC)
- komunikat jest małym pakietem danych zawierającym informacje o przerwaniu
- MSI-X umożliwia do 2048 wektorów na funkcję PCI; wektory można niezależnie konfigurować i kierować do wybranych procesorów
- MSI/MSI-X nie korzysta z dedykowanych linii przerwań; może ograniczyć współdzielenie przerwań i ułatwiać obsługę wielu kolejek
- jądro systemu konfiguruje MSI/MSI-X i przypisanie wektorów, rejestruje procedury obsługi, a po dostarczeniu przerwania uruchamia właściwą procedurę obsługi

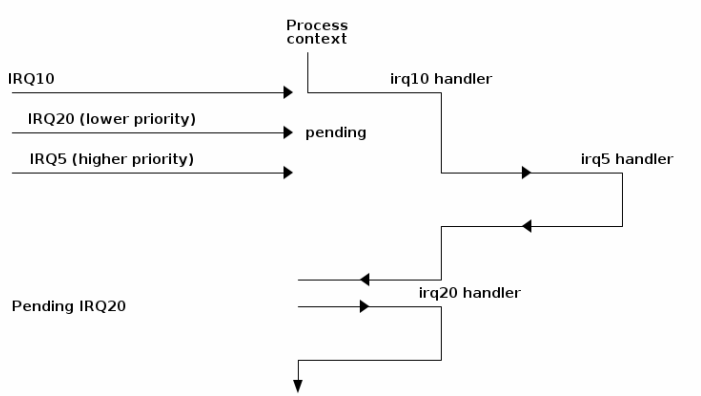
Maskowanie przerwań a synchronizacja

Synchronizacja dostępu do współdzielonych danych pomiędzy procedurą obsługi przerwań a innymi potencjalnymi współbieżnymi działaniami może wymagać włączenia i wyłączenia przerwań w kontrolowany sposób:

System może sterować przerwaniami na trzech poziomach:

- Urządzenie - sterownik może wyłączyć zgłaszanie przerwań przez rejestry urządzenia.
- Kontroler przerwań - PIC lub I/O APIC może zamaskować źródło i wstrzymać dostarczanie jego przerwań. To, czy zgłoszenie pozostanie oczekujące, zależy od urządzenia, kontrolera i trybu wyzwalania; nie każde zdarzenie musi zostać zapamiętane.
- Procesor - w architekturze x86 instrukcje `cli` i `sti` zmieniają flagę IF (*interrupt flag*) na bieżącym CPU, blokując lub dopuszczając maskowalne przerwania; nie blokują NMI ani wyjątków i nie wpływają na pozostałe CPU.

Priorytety przerwań



- IRQ o wyższym priorytecie może przerwać obsługę IRQ o niższym, jeśli kontroler i konfiguracja procesora dopuszczają zagnieżdżanie
- Niższy IRQ pozostaje oczekujący podczas obsługi wyższego i może zostać dostarczony później, o ile kontroler lub urządzenie zachowało zgłoszenie
- priorytet może być określany przez numer IRQ, konfigurację kontrolera lub ustawienia procesora
- priorytety przerwań są wspierane przez większość architektur, choć ich obsługa rzadko występuje w systemach ogólnego przeznaczenia
- priorytety są przydatne w systemach czasu rzeczywistego, pomagają ograniczać opóźnienia

Problemy operacji wejścia-wyjścia

Podczas programowej obsługi wejścia-wyjścia (aktywnego czekania lub odpytywania), a także podczas obsługi sterowanej przerwaniami, procesor jest zaangażowany w przesyłanie danych między pamięcią a urządzeniami wej-wyj:

- szybkość transferu wej-wyj jest ograniczona szybkością, z jaką procesor może testować i obsługiwać urządzenie
- procesor jest zajęty zarządzaniem przesyłem danych z wejścia i na wyjście

Szybkie urządzenia wej-wyj oraz urządzenia przesyłające duże ilości danych wymagają bezpośredniego dostępu do pamięci (DMA, *Direct Memory Access*).

Bezpośredni dostęp do pamięci (DMA)

Bezpośredni dostęp do pamięci jest realizowany przez osobny kontroler na magistrali systemowej (DMAC, *DMA Controller*) albo przez funkcję zintegrowaną z chipsetem lub kontrolerem urządzenia wej-wyj (np. karty sieciowej albo dysku SSD).

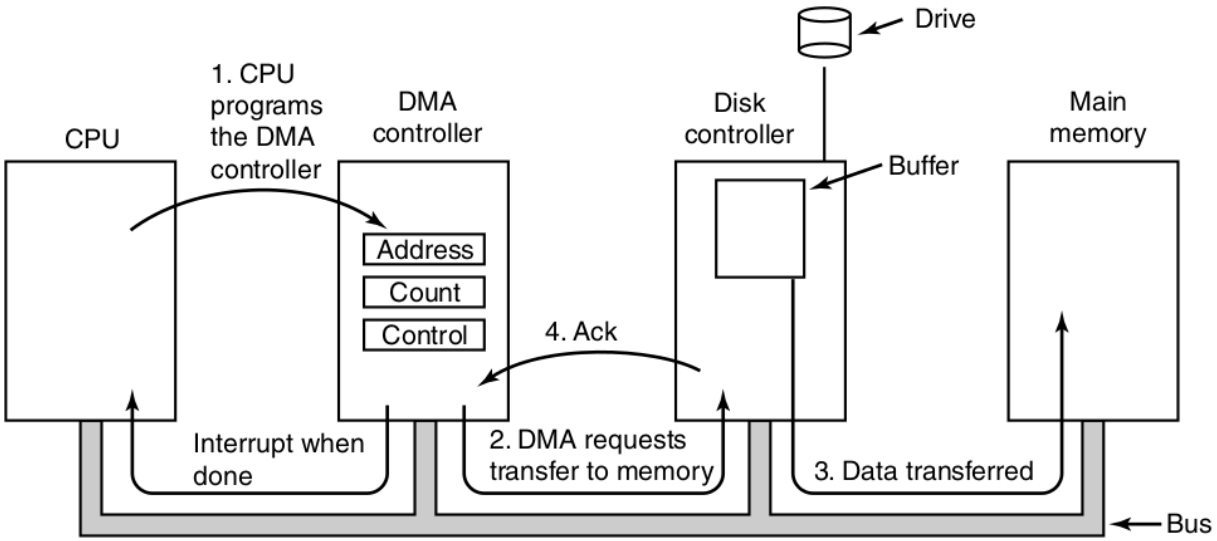


Figure 3-4. Operation of a DMA transfer.

Bezpośredni dostęp do pamięci (DMA)

Transfer danych wymaga przekazania przez procesor do modułu DMA rozkazu zawierającego następujące informacje:

- czy wymagany jest odczyt/zapis
- adres urządzenia wej-wyj
- adres początkowej komórki pamięci, z której dane mają być odczytane lub do której mają być zapisane
- liczbę słów do przesłania

Procesor inicjuje przesłanie danych i kontynuuje wykonywanie innych zadań do momentu nadejścia przerwania od modułu DMAC.

DMA ogranicza udział procesora w przesyłaniu danych, pozwala urządzeniom peryferyjnym przesyłać dane bezpośrednio między pamięcią operacyjną a urządzeniem, przyspieszając w ten sposób operacje wejścia/wyjścia.

Monitorowanie przerwań sprzętowych w Linuksie

```
$ uname -sr
Linux 2.6.18-92.1.10.el5 x86_64+

$ cat /proc/interrupts
           CPU0       CPU1
  0:  799260896          0    IO-APIC-edge  timer
  1:        140        100    IO-APIC-edge  i8042
  6:          5          0    IO-APIC-edge  floppy
  8:          0          0    IO-APIC-edge  rtc
  9:          0          0    IO-APIC-level  acpi
 12:        115          0    IO-APIC-edge  i8042
 15:    7173047       3915    IO-APIC-edge  ide1
169:          0          0    IO-APIC-level  ohci_hcd:usb1, ohci_hcd:usb2
177:     567795     125000    IO-APIC-level  ioc0
185:    6118456      40533    IO-APIC-level  eth0
NMI:       2770       1163
LOC:  799176280  799176233
ERR:          0
```

- edge - zgłoszenie na linii IRQ przez zmianę poziomu sygnału, krótkotrwałe zdarzenia (np. naciśnięcie klawisza, sygnał z timera)
- `level` – zgłoszenie na linii IRQ przez podniesienie poziomu sygnału; dłuższe zdarzenia wymagające potwierdzenia (np. transfer danych z dysku)

Monitorowanie przerwań sprzętowych w Linux

```
# uname -sr
Linux 6.9.3-76060903-generic

# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  1:      17024          0          0         12  IR-IO-APIC    1-edge      i8042
  8:          0          0          0          0  IR-IO-APIC    8-edge      rtc0
  9:      47343       2532          0          0  IR-IO-APIC    9-fasteoi   acpi
 12:     100109          0       2004          0  IR-IO-APIC   12-edge      i8042
 14:          0          0          0          0  IR-IO-APIC   14-fasteoi   INT34BB:00
 16:          0          0          0          0  IR-IO-APIC   16-fasteoi   idma64.0, i2c_designware.0, i801_smbus
120:          0          0          0          0  DMAR-MSI      0-edge      dmar0
121:          0          0          0          0  DMAR-MSI      1-edge      dmar1
122:          0          0          0          0  IR-PCI-MSI-0000:00:1c.0    0-edge      PCIe PME
...
164:      11391          0          0          0  IR-PCI-MSIX-0000:3d:00.0    1-edge      nvme0q1
...
NMI:          0          0          0          0   Non-maskable interrupts
LOC:   49586732   34698221   31900639   31100941   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:          0          0          0          0   Performance monitoring interrupts
IWI:    7828372     588193     485311     404572   IRQ work interrupts
RTR:          0          0          0          0   APIC ICR read retries
RES:     451370     469078     472718     500172   Rescheduling interrupts
CAL:    9657155    6803847    6046705    5690671   Function call interrupts
TLB:    3807589    4242390    4227898    4149771   TLB shootdowns
TRM:         16         16         16         16   Thermal event interrupts
THR:          0          0          0          0   Threshold APIC interrupts
DFR:          0          0          0          0   Deferred Error APIC interrupts
MCE:          0          0          0          0   Machine check exceptions
MCP:        356        349        349        349   Machine check polls
ERR:          0
MIS:          0
```

Przerwania w Linuksie: obsługa bezpośrednia i odroczona

- **obsługa bezpośrednia** (*top half*)
  - wykonywana w kontekście sprzętowego przerwania; powinna być krótka
  - wykonuje pilne czynności, które mają kluczowe znaczenie dla funkcjonowania urządzenia (np. odczytuje stan urządzenia lub potwierdza przerwanie)
  - może zlecić wykonanie dalszej pracy w trybie odroczonym
  - może opóźnić obsługę innych przerwań na tym samym CPU
  - wykonywanie zbyt długich operacji przy zablokowanych przerwaniach powoduje opóźnienia, problemy z wydajnością i gubienie przerwań
- **obsługa odroczona** (*bottom half*)
  - wykonuje pracę, która nie musi być zrealizowana bezpośrednio w obsłudze sprzętowego IRQ, która może być odroczona do momentu mniejszego obciążenia systemu
  - zależnie od mechanizmu działa np. jako softirq albo w wątku jądra

Mechanizmy odroczania pracy w Linux

softirq

- rodzaje softirq są zdefiniowane statycznie; zgłaszają je podsystemy jądra, m.in. obsługa przerwań (dolna połówka)
- ten sam rodzaj softirq może być obsługiwany równolegle na różnych CPU

tasklet - szczególny rodzaj softirq

- tworzone dynamicznie (np. podczas ładowania modułu)
- ten sam tasklet jest wykonywany najwyżej na jednym CPU naraz

workqueues

- zadania odroczone wykonywane w kontekście procesu
- w odróżnieniu od softirq i taskletów mogą być blokowane

softirq - statystyki przerwań

```
$ uname -sr
Linux 6.9.3-76060903-generic

$ cat /proc/softirq
                    CPU0       CPU1       CPU2       CPU3
          HI:   23555600    3297692    3334936    3477097
       TIMER:    8655197    3098823    2571420    2336278
      NET_TX:         51         10          8         14
      NET_RX:    3089136      33591      32130      78008
       BLOCK:        367        472        921        324
    IRQ_POLL:         45          0          2          4
     TASKLET:     268745       2291       2068       2142
       SCHED:   21443427   12885249   11074332   10525505
     HRTIMER:       2452         13         10          6
         RCU:    7329331    6460572    6142318    6003988
```

HI - softirq wysokiego priorytetu
TIMER - przerwania zegarowe
NET_TX - wysyłanie pakietów sieciowych
NET_RX - odbiór pakietów sieciowych
BLOCK - operacje blokowe (np. kończenie operacji I/O)
IRQ_POLL - odpytywanie urządzeń o zakończenie operacji
TASKLET - tasklety
SCHED - prace związane z planistą (np. bilansowanie obciążenia)
HRTIMER - wysokorozdzielcze timery
RCU - mechanizm synchronizacji jądra RCU (Read-Copy Update)

Monitorowanie przerwań sprzętowych: klawiatura i mysz

Fragment wpisu /var/log/syslog dotyczący przerwań klawiatury i myszy:

```
kernel: i8042: PNP: PS/2 Controller [PNP0303:KBD,PNP0f13:MOU] at 0x60,0x64 irq 1,12
kernel: serio: i8042 KBD port at 0x60,0x64 irq 1
kernel: serio: i8042 AUX port at 0x60,0x64 irq 12
```

Przytrzymanie klawisza IRQ=1

```
$ sar -I --int=1 1

09:38:15         INTR    intr/s
09:38:16            1      1,00
09:38:17            1      0,00
09:38:18            1      1,00
09:38:19            1     24,00
...
09:38:24            1     29,00
09:38:25            1     30,00
09:38:26            1     29,00
09:38:27            1     29,00
09:38:28            1      3,00
^C
Średnia:            1     20,23
```

Ruch gładzika (touchpad) IRQ=12

```
$ sar -I --intr=12 1

09:45:56         INTR    intr/s
09:45:57           12      0,00
09:45:58           12     71,00
09:45:59           12    469,00
09:46:00           12    471,00
09:46:01           12    471,00
...
09:46:07           12    356,00
09:46:08           12    472,00
09:46:09           12    102,00
09:46:10           12      0,00
^C
Średnia:           12    273,02
```

Monitorowanie przerwań sprzętowych: wlan0

Wpis z `lspci` dotyczący karty sieciowej wlan0:

```
$ lspci -v
00:14.3 Network controller: Intel Corporation Cannon Point-LP CNVi [Wireless-AC] (rev 30)
        Subsystem: Intel Corporation Cannon Point-LP CNVi [Wireless-AC]
        Flags: bus master, fast devsel, latency 0, IRQ 173, IOMMU group 6
        Memory at b3938000 (64-bit, non-prefetchable) [size=16K]
        Capabilities: <access denied>
        Kernel driver in use: iwlwifi
        Kernel modules: iwlwifi, wl
```

Strumień pakietów na interfejsie wlan0

```
$ ping -i 0.002 -c 1000 192.168.1.1
```

Przerwania karty sieciowej wlan0

```
$ sar -I --intr=173 1

22:44:25         INTR    intr/s
22:44:26          174      8,00
22:44:27          174      6,00
22:44:28          174     67,00
22:44:29          174    980,00
22:44:30          174   1845,00
22:44:31          174    674,00
...
Średnia:          174    363,70
```

Wieloprogramowość i ochrona sprzętowa

- w systemach wieloprogramowych i z podziałem czasu operacje wejścia-wyjścia nakładają się na działanie jednostki centralnej (asynchroniczne wej-wyj).
- procesy współdzielą procesor, pamięć i urządzenia wejścia-wyjścia
- błąd programu nie może pozwolić mu zmienić kodu jądra ani pamięci innych procesów
- zarządzanie wieloma programami wymaga odpowiednich mechanizmów ochrony, aby zapewnić bezpieczeństwo i stabilność systemu operacyjnego
- system operacyjny i jego jądro są programami, dlatego potrzebny jest mechanizm odróżniający kod wykonywany przez użytkownika od kodu jądra systemu
- ochrona sprzętowa: izolacja zasobów, ochrona pamięci, tryb uprzywilejowany procesora

Przerwanie przekazuje sterowanie do jądra

- Procedury obsługi przerwań są wykonywane w trybie uprzywilejowanym i mają dostęp do zasobów jądra.
- Gdy procesor wykonuje kod użytkownika i przyjmuje przerwanie lub wyjątek, przekazuje sterowanie do procedury obsługi w trybie uprzywilejowanym.
- Procesor zachowuje stan potrzebny do obsługi zdarzenia i późniejszego wznowienia pracy.
- Po obsłudze jądro może wznowić przerwany kod, wybrać inne zadanie albo zakończyć proces, który spowodował wyjątek.
- Jeśli zdarzenie wystąpi podczas wykonywania kodu jądra, procesor pozostaje w trybie uprzywilejowanym.

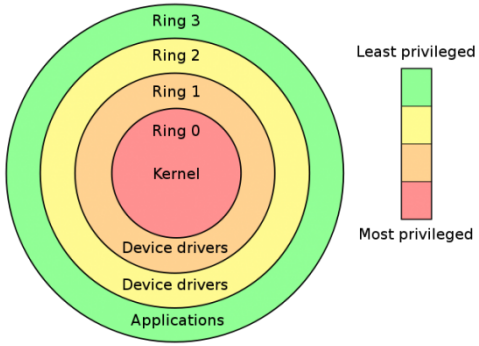
Sprzętowe mechanizmy ochrony

- Procesor rozróżnia poziomy uprawnień; kod użytkownika nie może swobodnie wykonywać operacji uprzywilejowanych
  $\longrightarrow$ dualny tryb pracy procesora: użytkownik i jądro.
- Dostęp do portów wejścia-wyjścia jest kontrolowany przez mechanizmy uprawnień procesora i system operacyjny. Rozkazy wejścia-wyjścia są uprzywilejowane, dlatego tylko system operacyjny może komunikować się z urządzeniami.
- W x86 są cztery poziomy Ring 0-3 - tzw. pierścienie uprzywilejowania. Systemy operacyjne (Linux/Windows) używają zwykle Ring 0 dla jądra i Ring 3 dla programów użytkownika; poziomy 1 i 2 są na ogół niewykorzystywane.

Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, oficjalna dokumentacja.

Pierścienie uprzywilejowania

Mechanizm sprzętowy (zaimplementowany w procesorze) służący do izolacji i kontroli dostępu do zasobów systemu.



Cztery pierścienie (*Ring* 0-3) w architekturze x86:

- pierścień 3 - najniższe uprawnienia, dostępny dla użytkownika
- pierścienie 1 i 2 - rzadko używane, czasem dla modułów urządzeń wejścia-wyjścia
- pierścień 0 - najwyższe uprawnienia; uprzywilejowane operacje może wykonywać tylko jądro systemu, niedostępne bezpośrednio dla programistów i użytkowników

http://en.wikipedia.org/wiki/Ring_(computer_security)

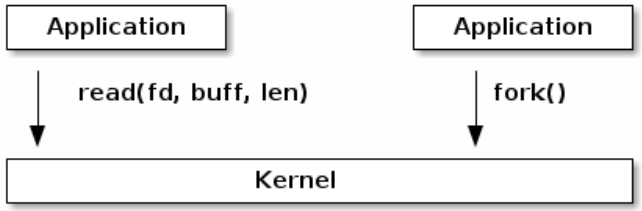
Dualny tryb pracy

- **tryb jądra (nadzorcy, monitora)** – w tym trybie wykonywane są rozkazy uprzywilejowane, np. rozkazy wejścia-wyjścia, zmieniające stan rejestrów zarządzających pamięcią i czasomierzem, rozkaz *halt*, rozkazy włączania/wyłączania przerwań
- **tryb użytkownika** – aplikacje działają z ograniczonymi uprawnieniami i mogą zlecić systemowi wykonanie zastrzeżonych operacji (np. dostępu do zasobów) za pośrednictwem wywołań systemowych (*system calls*).

  Wywołanie systemowe (*system call*) jest traktowane przez sprzęt jak przerwanie programowe. Poprzez wektor przerwań sterowanie przekazywane jest do odpowiedniej procedury obsługi w systemie operacyjnym.

Przerwania programowe i wywołania systemowe

Z punktu widzenia użytkownika lub programisty wywołania systemowe są „usługami" oferowanymi aplikacjom przez jądro. Przypominają funkcje API bibliotek: mają nazwę, parametry i wartość zwracaną.[2]



Wywołanie systemowe to żądanie usługi jądra. Programy zwykle korzystają z funkcji opakowujących udostępnianych przez bibliotekę C, które wykonują:

- identyfikują wywołanie systemowe i jego parametry
- przełączają tryb wykonania z trybu użytkownika do trybu jądra
- odbierają rezultat wywołania systemowego

[2]Biblioteka C w Linuksie udostępnia wiele interfejsów zgodnych z POSIX (*Portable Operating System Interface*); Linux oferuje również interfejsy specyficzne dla systemu.

- biblioteki systemowe udostępniają funkcje implementujące wywołania systemowe, dzięki czemu można z nich łatwo korzystać w aplikacjach
- interfejs wywołań systemowych w Linuksie (podobnie jak w większości systemów Unix) jest częściowo udostępniany przez bibliotekę C (`glibc`)

Program:	wywołanie funkcji printf()
Biblioteka C:	printf() $\longrightarrow$ write()
Jądro:	wywołanie systemowe write()

- wywołania systemowe są niezbędne, gdyż programy działające w przestrzeni użytkownika nie mogą wykonać kodu jądra
- programy nie mogą bezpośrednio wywoływać funkcji jądra, ponieważ jądro jest umieszczone i działa w chronionym obszarze pamięci operacyjnej.

Przerwania programowe i wywołania systemowe

- przerwanie programowe realizuje w procesorach rodziny x86 instrukcja `INT n`, gdzie `n` jest numerem wektora przerwania
- `INT $0x80` powoduje przełączenie systemu do trybu jądra i rozpoczęcie wykonywania wektora przerwania 128, który wskazuje na adres procedury obsługi wywołań systemowych `system_call()`
- konkretne wywołanie systemowe jest identyfikowane przez numer umieszczany w rejestrze `eax`
- argumenty wywołania systemowego umieszczane są w osobnych rejestrach (np. `ebx`, `ecx`, `edx`, `esi`, `edi` w architekturze 32-bitowej x86)

Przerwania programowe i wywołania systemowe

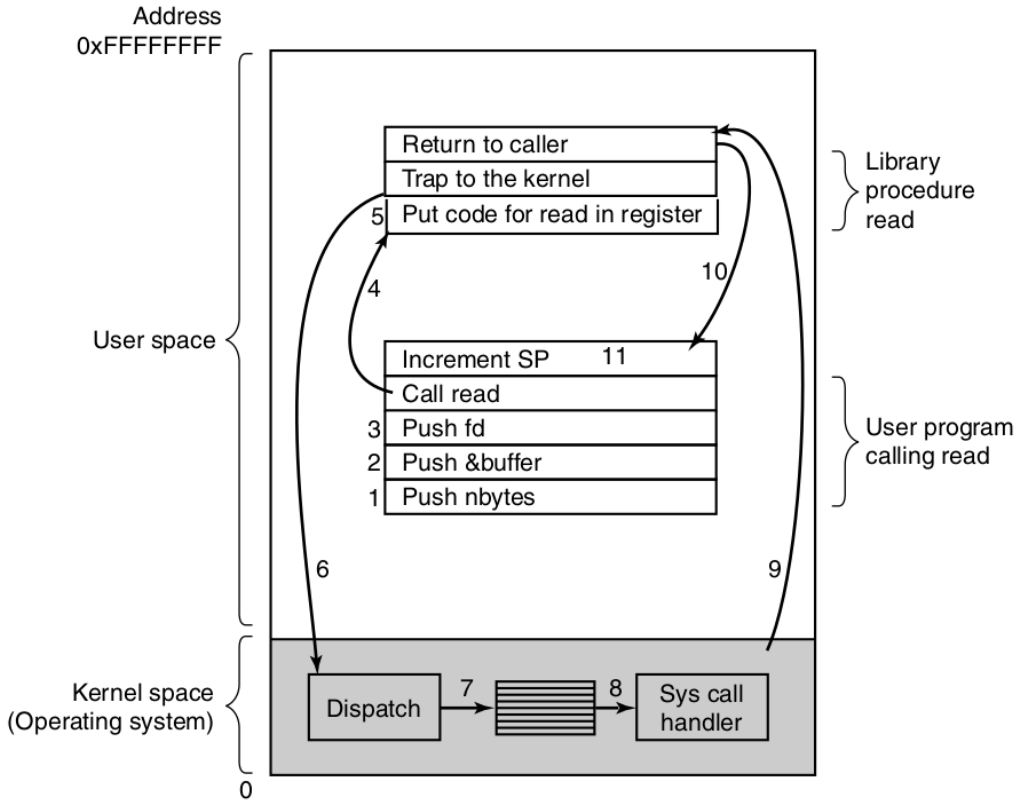


Figure 1-16. The 11 steps in making the system call read(fd, buffer, nbytes).

Przykład wywołania funkcji `read(fd, buffer, nbytes)`

Przerwania programowe i wywołania systemowe

Fragmenty pliku /usr/include/asm/unistd_64.h

Fedora 24

```
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
...
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
...
#define __NR_access 21
#define __NR_pipe 22
#define __NR_select 23
#define __NR_sched_yield 24
...
#define __NR_clone 56
#define __NR_fork 57
#define __NR_vfork 58
#define __NR_execve 59
#define __NR_exit 60
#define __NR_kill 62
#define __NR_uname 63
...
#define __NR_pwritev2 328
```

Fedora 33

```
...
#define __NR_pkey_free 331
#define __NR_statx 332
#define __NR_io_pgetevents 333
#define __NR_rseq 334
...
#define __NR_pidfd_send_signal 424
#define __NR_io_uring_setup 425
#define __NR_io_uring_enter 426
...
#define __NR_mount_setattr 442
#define __NR_quotactl_fd 443
#define __NR_landlock_create_ruleset 444
#define __NR_landlock_add_rule 445
#define __NR_landlock_restrict_self 446
#define __NR_memfd_secret 447
```

Opis funkcji systemowych Linux dostępny jest w podręczniku `man` (np. `man 2 read`).

Przerwania programowe: śledzenie

```
$ strace -n echo "Witaj Świecie"
[  59] execve("/usr/bin/echo", ["echo", "Witaj \305\232wiecie"], 0x7ffc6d7a05e0 /* 82 vars */) = 0
[  12] brk(NULL)                        = 0x636ecc694000
[   9] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7d38fe8d6000
[  21] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Nie ma takiego pliku ani katalogu)
[ 257] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[ 262] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=147643, ...}, AT_EMPTY_PATH) = 0
[   9] mmap(NULL, 147643, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7d38fe8b1000
[   3] close(3)                         = 0
[ 257] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[   0] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
...
[ 257] openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
[ 262] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18671040, ...}, AT_EMPTY_PATH) = 0
[   9] mmap(NULL, 18671040, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7d38fd400000
[   3] close(3)                         = 0
[ 262] newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
[   1] write(1, "Witaj \305\232wiecie\n", 15) = 15
[   3] close(1)                         = 0
[   3] close(2)                         = 0
[ 231] exit_group(0)                    = ?
[ 231] +++ exited with 0 +++
```

Przerwania programowe: śledzenie

```
$ strace -c cat /etc/hosts > /dev/null
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 35.84    0.000944         944         1           execve
 11.92    0.000314          28        11         6 openat
 11.20    0.000295          29        10           mmap
  6.42    0.000169          24         7           close
  5.58    0.000147          36         4           brk
  5.28    0.000139          23         6           fstat
  4.78    0.000126          31         4           mprotect
  4.71    0.000124          24         5           read
  3.83    0.000101          50         2         1 arch_prctl
  3.49    0.000092          46         2           munmap
  3.26    0.000086          21         4           pread64
  2.01    0.000053          53         1         1 access
  0.91    0.000024          24         1           write
  0.76    0.000020          20         1           fadvise64
------ ----------- ----------- --------- --------- ----------------
100.00    0.002634          44        59         8 total
```

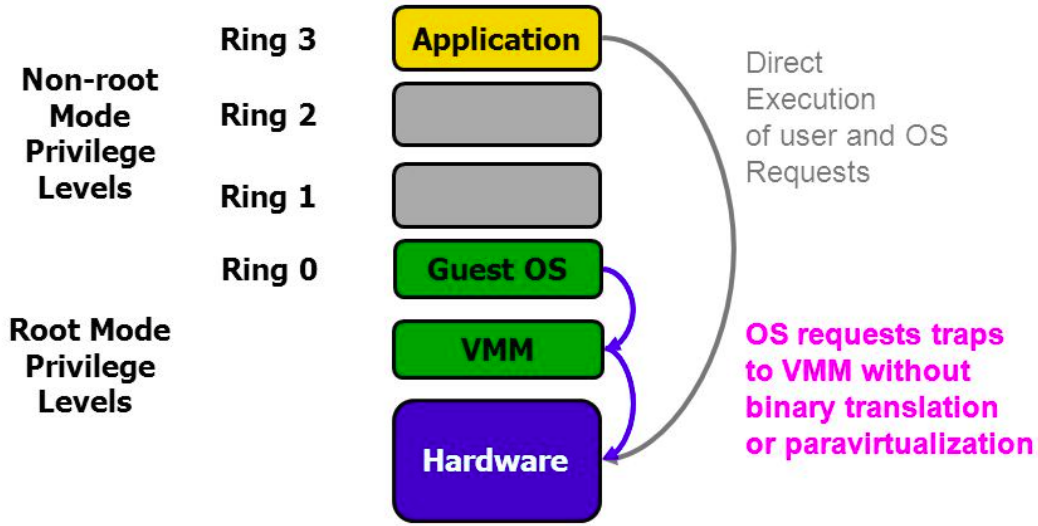
`time` - udział w łącznym czasie zarejestrowanym w wywołaniach, `seconds` - łączny zmierzony czas, `usecs/call` - średni czas jednego wywołania, `calls` - liczba wywołań, `errors` - zlicza wywołania zakończone błędem

Czas spędzony w trybie jądra

```
$ /bin/time -p ping -i 0.002 -c 500  127.0.0.1 > /dev/null
real 0.99
user 0.19
sys 0.80
```

- `real` czas rzeczywisty wykonania polecenia
- `user` czas CPU spędzony w trybie użytkownika
- `sys` czas CPU spędzony w trybie jądra

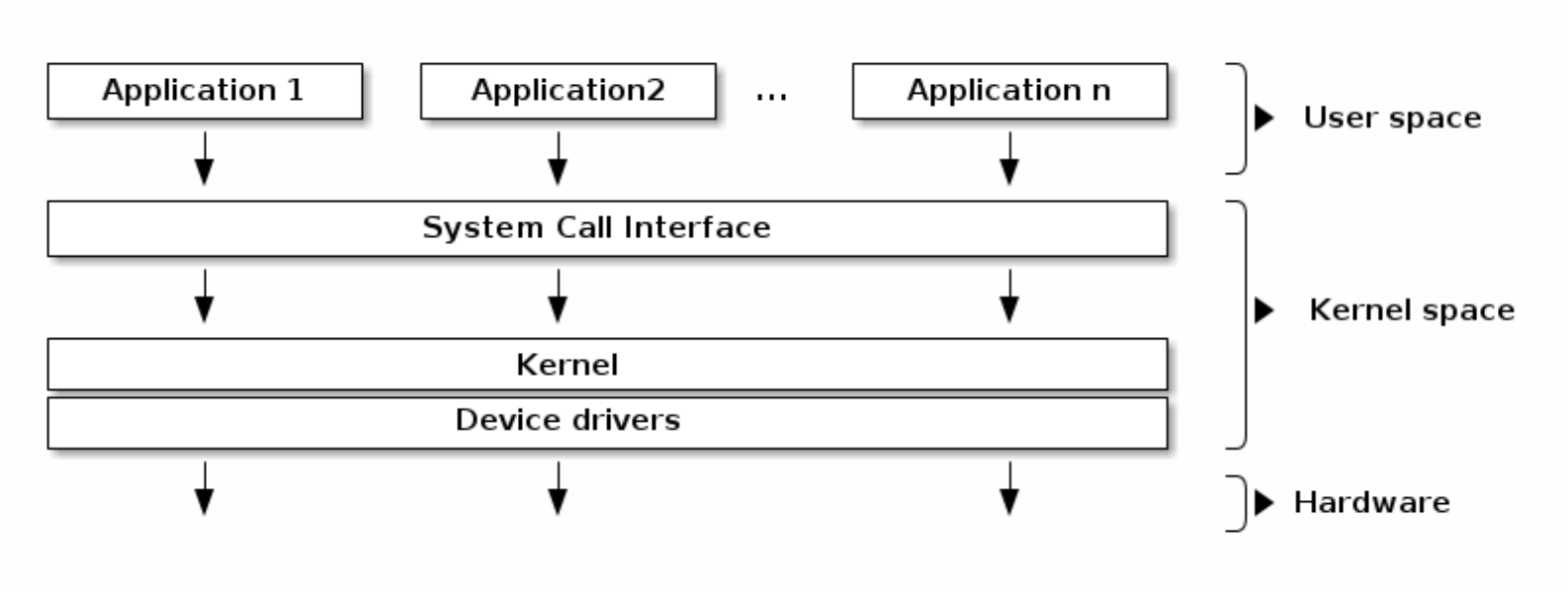
Pierścienie uprzywilejowania i wirtualizacja



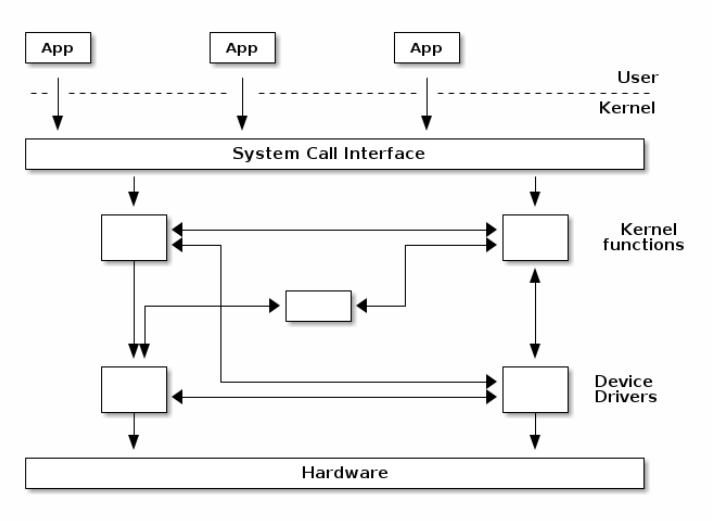
Mechanizmy sprzętowej wirtualizacji, takie jak Intel VT-x, wprowadzają tryby VMX root i VMX non-root. Hiperwizor (np. KVM) zarządza przejściami między trybem hosta i gościa; system gościa może działać w Ring 0 w trybie VMX non-root.

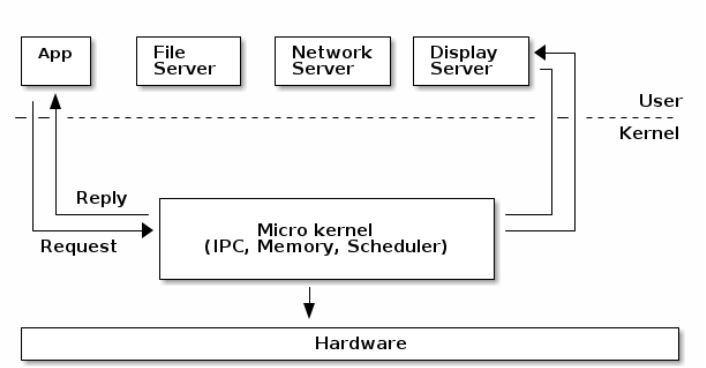
http://slideplayer.com/slide/5084764/

Typowa architektura systemu



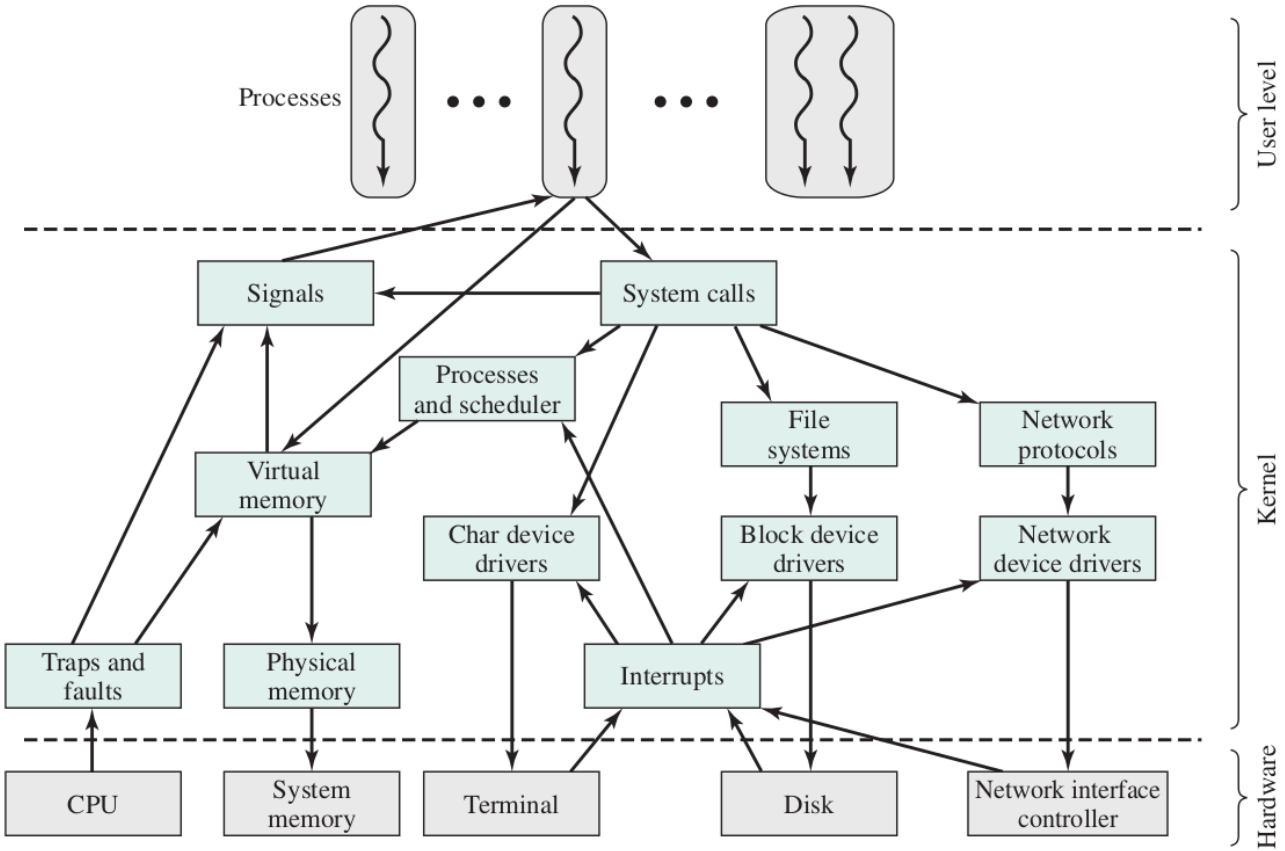
Monolityczne jądro vs. mikrojądro





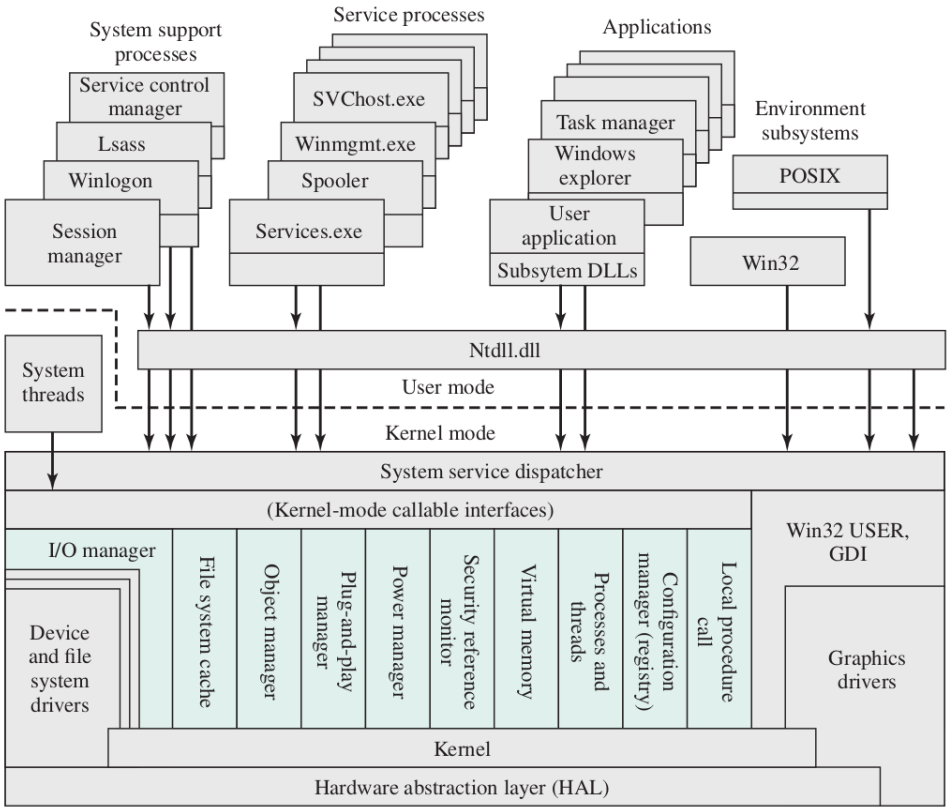
- **Jądro monolityczne** nie ma ochrony sprzętowej dostępu pomiędzy różnymi podsystemami jądra. Podsystemy mogą wywoływać swoje funkcje bezpośrednio, co ogranicza koszt komunikacji (np. Linux)
- **Mikrojądro** utrzymuje w trybie jądra tylko podstawowe mechanizmy. Wiele usług systemu działa w odizolowanych procesach w przestrzeni użytkownika. Taki podział zwiększa izolację: awaria usługi nie musi zakończyć się awarią całego systemu. Wymaga jednak komunikacji między procesami, co może zwiększać narzut i złożoność systemu (np. Minix)
- jądra hybrydowe - systemy Mac OS X i Windows wykonują znaczną część usług i sterowników w trybie jądra. Czasem kwalifikuje się je jako monolityczne (Mac OS X, Windows)
- Debata o wyższości mikrojądra nad monolitycznym jądrem Tanenbaum–Torvalds debate

Architektura jądra Linux



W. Stallings, *Operating Systems: Internals and Design Principles*, 9th Edition, Pearson, 2018.

Architektura jądra Windows



Egzekutor (*Executive*) – zbiór modułów jądra Windows (zaznaczony na rysunku kolorem), który udostępnia API oprogramowaniu działającemu w trybie użytkownika.

W. Stallings, *Operating Systems: Internals and Design Principles*, 9th Edition, Pearson, 2018.

Pytania sprawdzające

- Co to jest przerwanie sprzętowe i jak jest obsługiwane przez system operacyjny?
- Wyjaśnij różnicę między trybem uprzywilejowanym a trybem użytkownika.
- Porównaj jądro monolityczne i mikrojądro.