Obsługa urządzeń wejścia-wyjścia

Jak ma współdziałać jednostka centralna z urządzeniami wej-wyj?

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

Sterowniki i moduły sterujące urządzeń

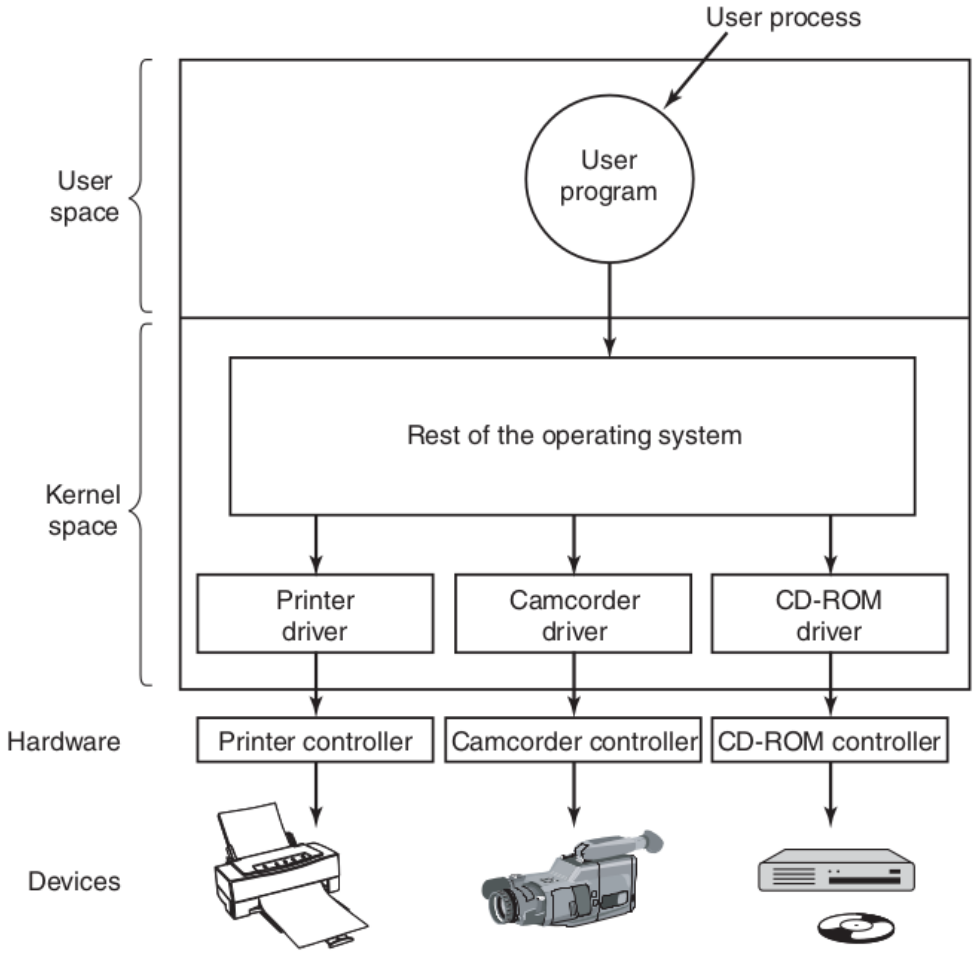


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Sterowniki i moduły sterujące urządzeń

- **Sterownik urządzenia** (*device controller*) związany jest z konkretnym urządzeniem i rozporządza 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
```

slot `00:1f.3` szyna nr 00, urządzenie nr `1f`, funkcja nr 3

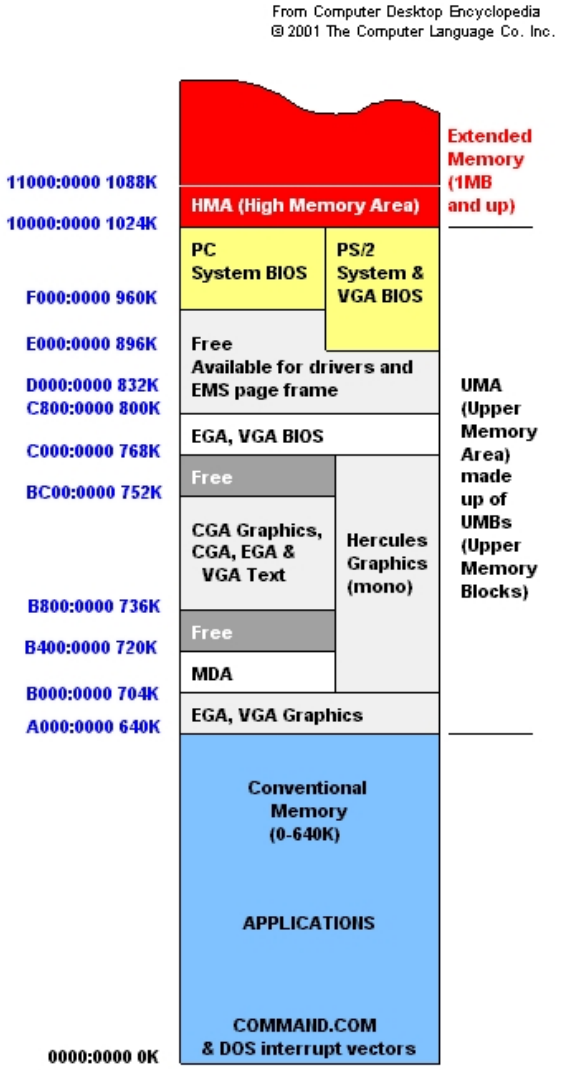
Zob.: `/proc/iomem`, `/proc/ioports`, `lspci -v`

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
  - zapis i odczyt procesora do adresów pamięci, skutkuje zapisami i odczytami do fizycznych rejestrów urządzenia
  - ochrona dostępu realizowana za pomocą mechanizmów ochrony pamięci
  - część przestrzeni adresowej jest zarezerwowana dla urządzeń - w systemach 16 i 32 bitowych z niewielką przestrzenią adresową wystepuje 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

Odpytywanie (*polling*)

Procesor regularnie sprawdza stan kolejnych urządzeń (np. dysków, portów, itp.) 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.

Obsługa przerwań

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

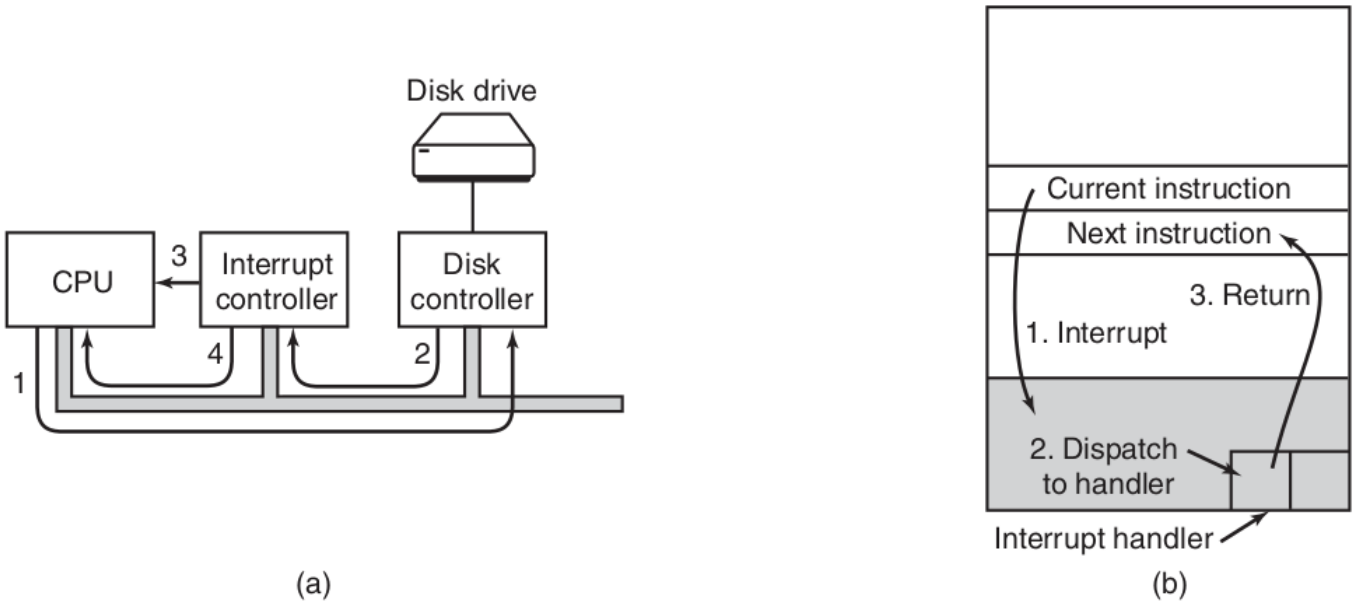


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Obsługa przerwań

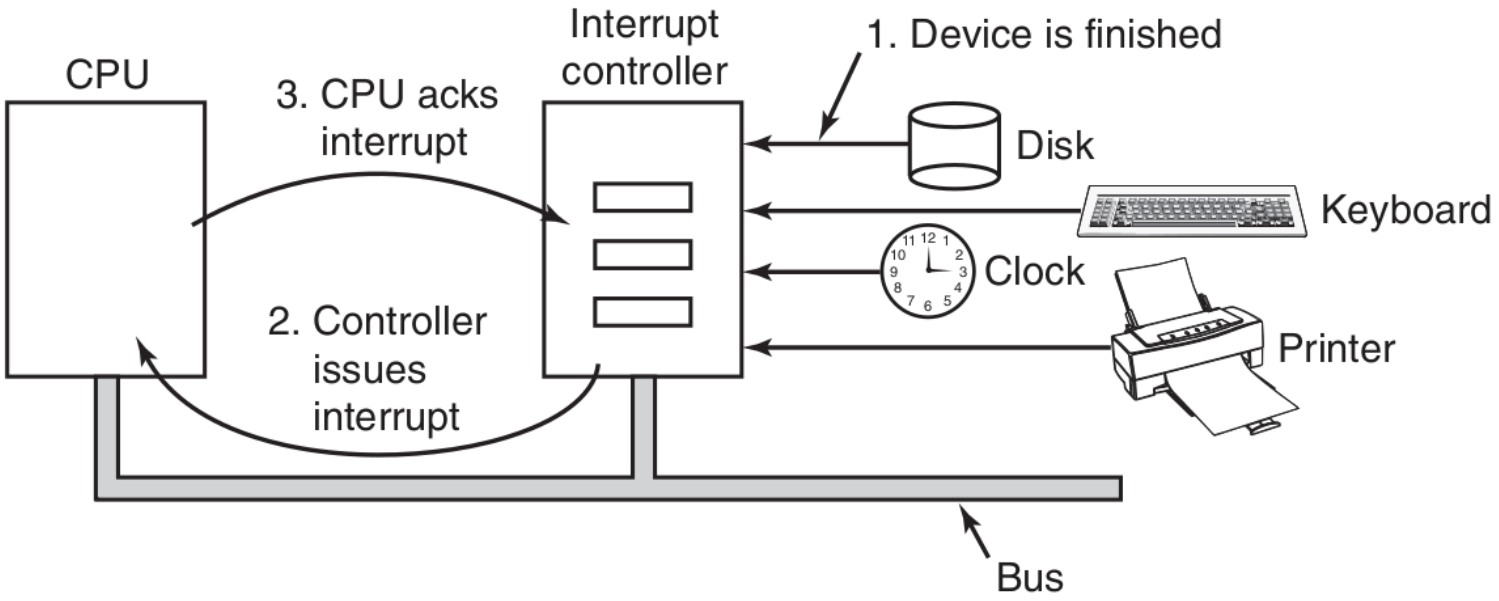


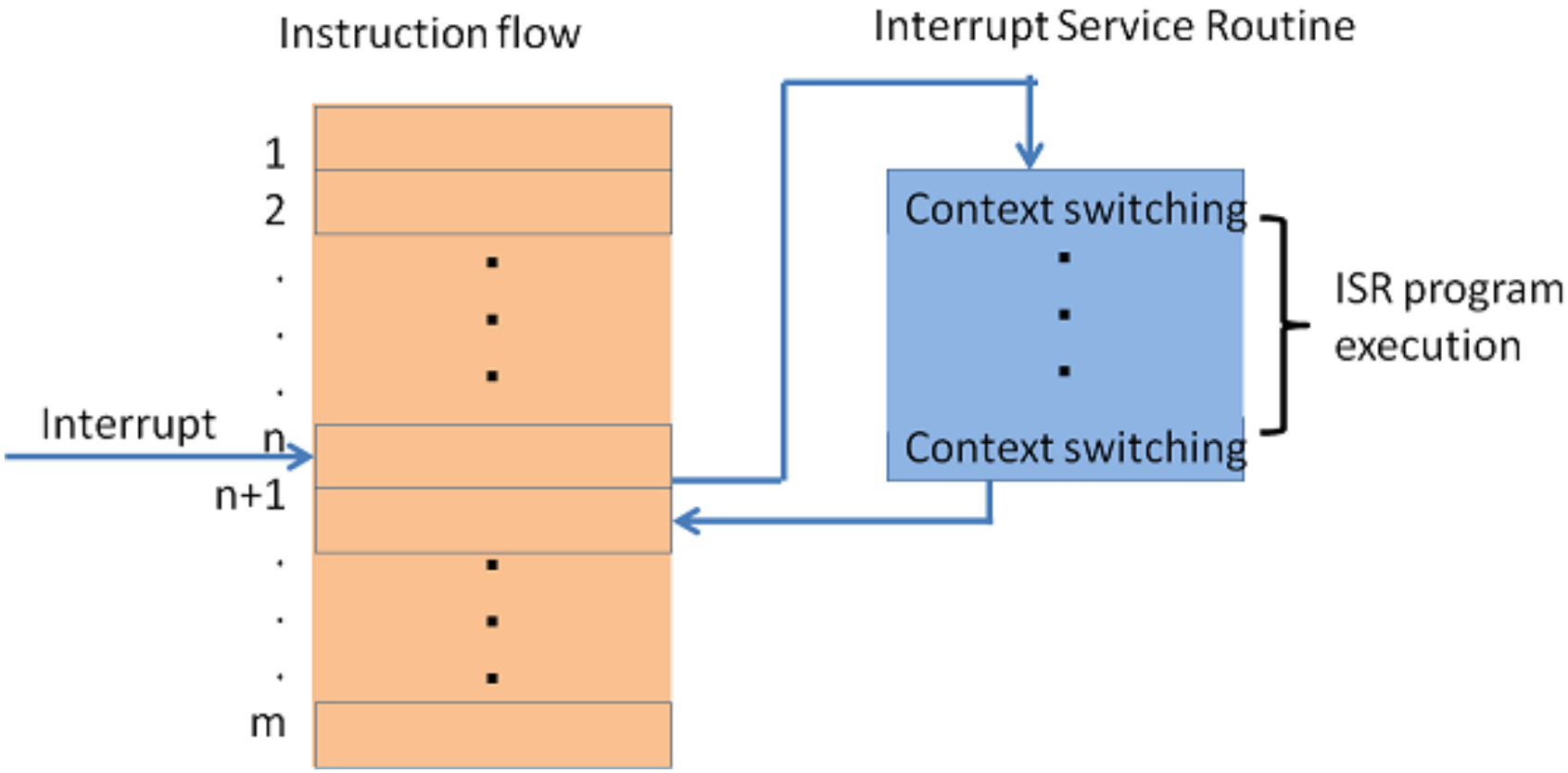
Figure 5-5. How an interrupt happens. The connections between the devices and the controller actually use interrupt lines on the bus rather than dedicated wires.

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 zostać uruchomiona jak najszybciej i nie może byc przerwana
  - wymagania urządzenia: jak najszybsza obsługa przerwania
  - wymagania systemu operacyjnego: jak najkrótsza obsługa
  - co gdy w przypadku obsługi przerwania pojawi sie nastepne?

- możliwe zagniezdzanie obsługi wywołań lub odraczanie wykonania
- w Linux obsługa przerwania podzielona na etapy:
  - górna połówka (*top half*) - niezbędne operacje
  - dolna połówka (*bottom half*) - odłożone do późniejszej obsługi

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

Przerwania: podział

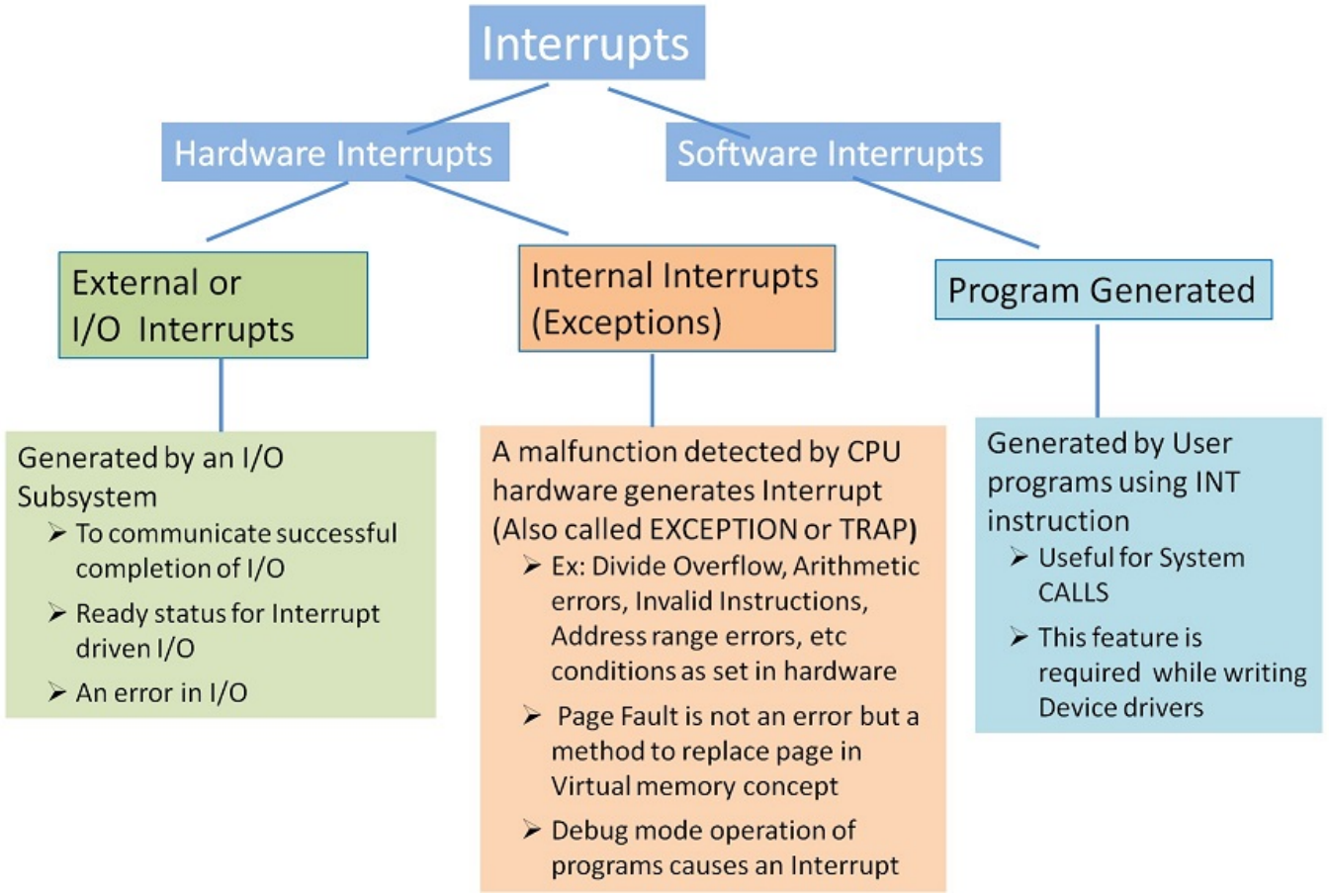
- **asynchroniczne** (przerwania) – generowane przez urządzenia sprzętowe w dowolnym czasie, niezależnie od sygnałów zegarowych procesora, np. przerwanie z karty sieciowej po odebraniu pakietu danych
- **synchroniczne** (wyjątki) – generowane w wyniku wykonania instrukcji, tworzone przez jednostkę sterowania 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
  - awaria sprzętu
- przerwania wywołane instrukcjami programów
  - błędy programów, nieprawidłowe operacje, załamania (*aborts*)
  - wyjątki (*exceptions*)
  - pułapki (*traps*)
  - przerwania programowe

Przerwania: źródła



Przerwania sprzętowe: podział wg dokumentacji Intela

- **maskowalne** – mogą być ignorowane
  - dwa stany przerwań: zamaskowany (ignorowany), nie zmaskowany
  - 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 generowane w sytuacji wystąpienia nieprawidłowości przy wykonywaniu instrukcji (IEEE 754: niedomiar (*underflow*), nadmiar (*overflow*), dzielenie przez zero, itp.

- **błędy** (*faults*) – instrukcja, która spowodowała błąd może być wznowiona. Register PC (w x86 EIC lub RIC) zapamiętany na stosie zawiera adres instrukcji, która spowodowała wyjątek (np. błąd strony)
- **pułapki** (*traps*) – zgłaszana po wykonaniu instrukcji pułapki, licznik instrukcji PC jest ustawiany na adres kolejnej instrukcji (np. śledzenie programu podczas debugowania)
- **załamania** (*aborts*) – wystąpił poważny błąd i jednostka sterowania nie może zachować adresu instrukcji; zwykle oznacza zakończenie procesu, który spowodował załamanie

Wyjątki programowe – występują na żądanie programisty (np. wywołanie funkcji systemowej, powiadomienie debuggera o zaistnieniu wyjątkowej sytuacji), są obsługiwane za pomocą pułapek

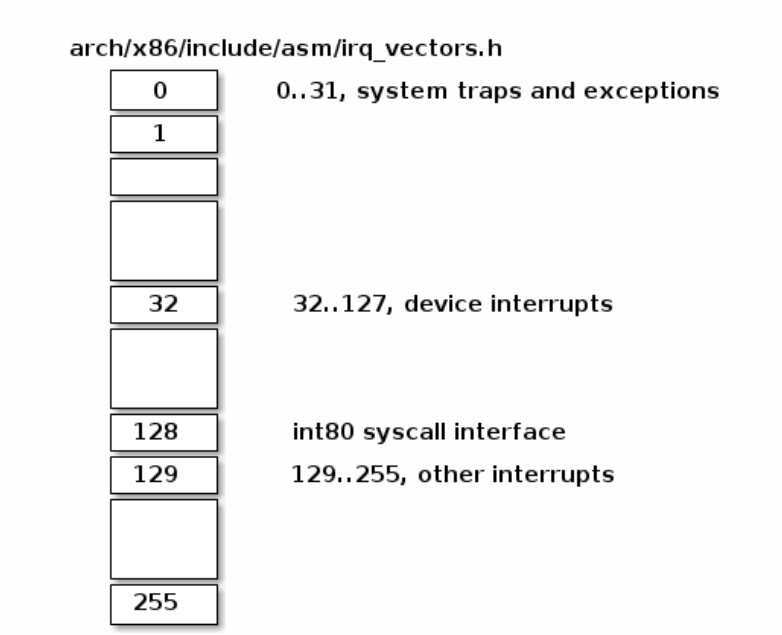
Przerwania: identyfikacja

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 rezydować w dowolnym miejscu pamięci, CPU lokalizauje ją za pomocą `IDTR`

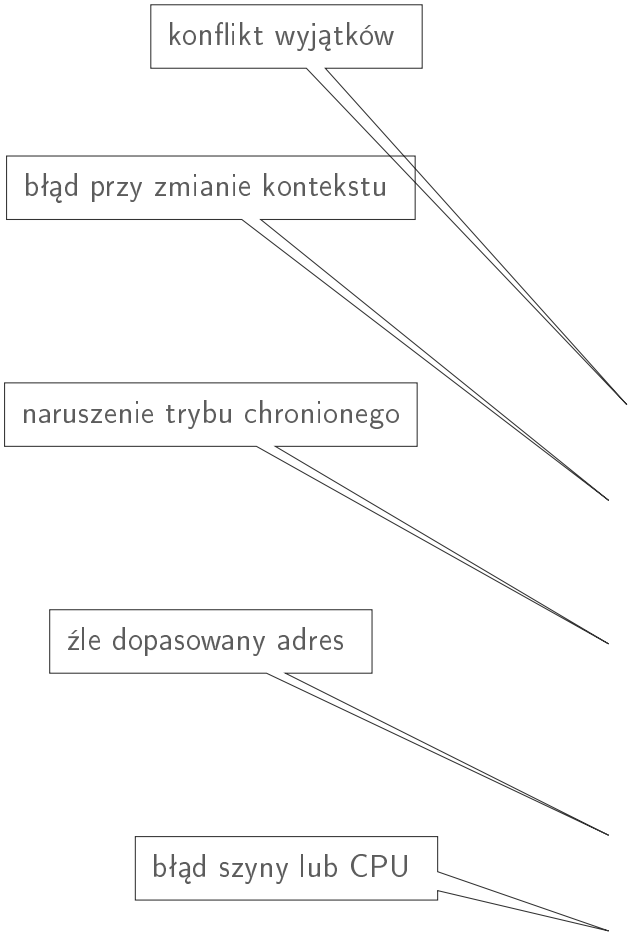


Tablica deskryptorów przerwań IDT

Rodzaje zdarzeń:

- wyjątki procesora, stałe mapowanie 0-31, niemaskowalne
- przerwania sprzętowe odwzorowują IRQ sprzętu zależnie od kontrolera przerwań 32-127
- przerwania programowe 128-255 definiowane przez BIOS i system operacyjny. Wzbudzane przez instrukcję 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



#	Wyjątek	Rodzaj	Sygnał
0	*Divide error*	fault	SIGFPE
1	*Debug*	fault or trap	SIGTRAP
2	NMI		
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 exceptoin*	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
- funkcje obsługi zazwyczaj wysyłają sygnały (w Unix/Linux) do procesu powodującego wystapienie wyjątku

Przyporządkowanie IRQ do urządzeń I/O

IRQ	INT	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	pierwszy łańcuch sterownik dysków EIDE
15	47	drugi łańcuch sterownika dysków EIDE

- PIC *Programmable interrupt controller* dla jednostek jednoprocesorowych
- chip 8259A obsługuje 8 linii IRQ, w połaczeniu kaskadowym (pin 2) do 15 linii IRQ
- PIC połaczony do pinu INTR procesora

Sterownik przerwań PIC w systemach Intel PC

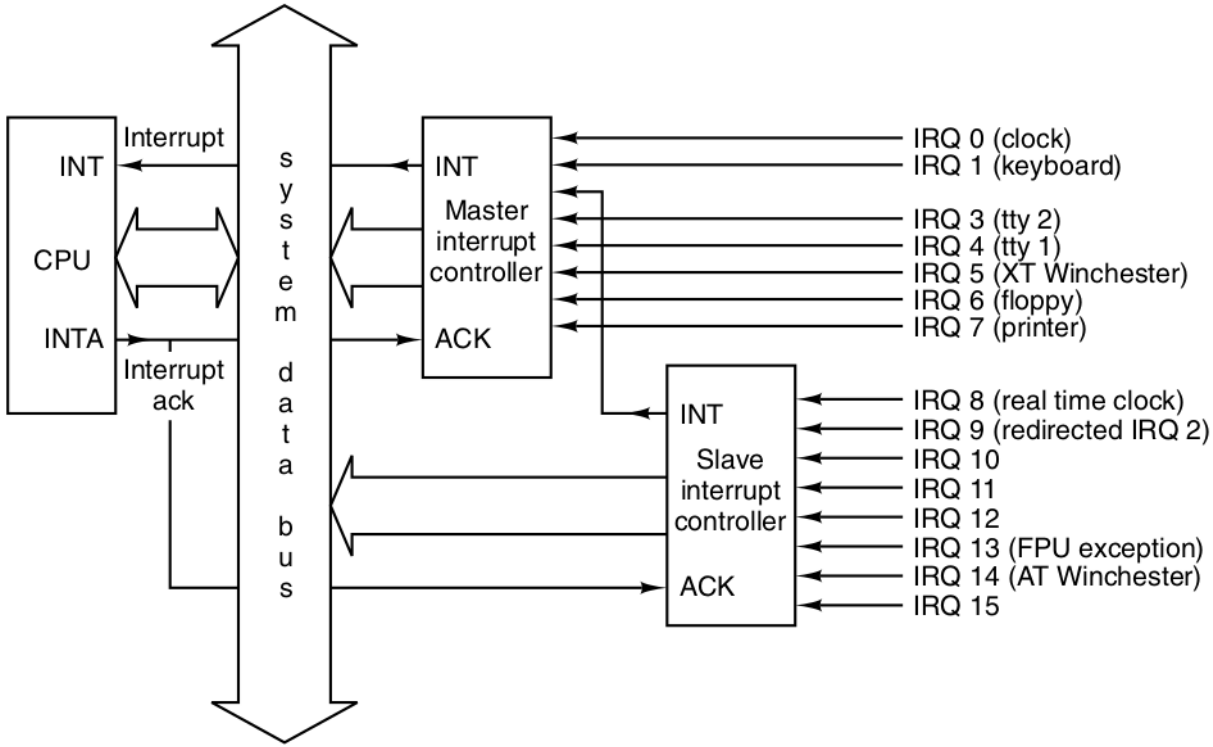


Figure 2-39. Interrupt processing hardware on a 32-bit Intel PC.

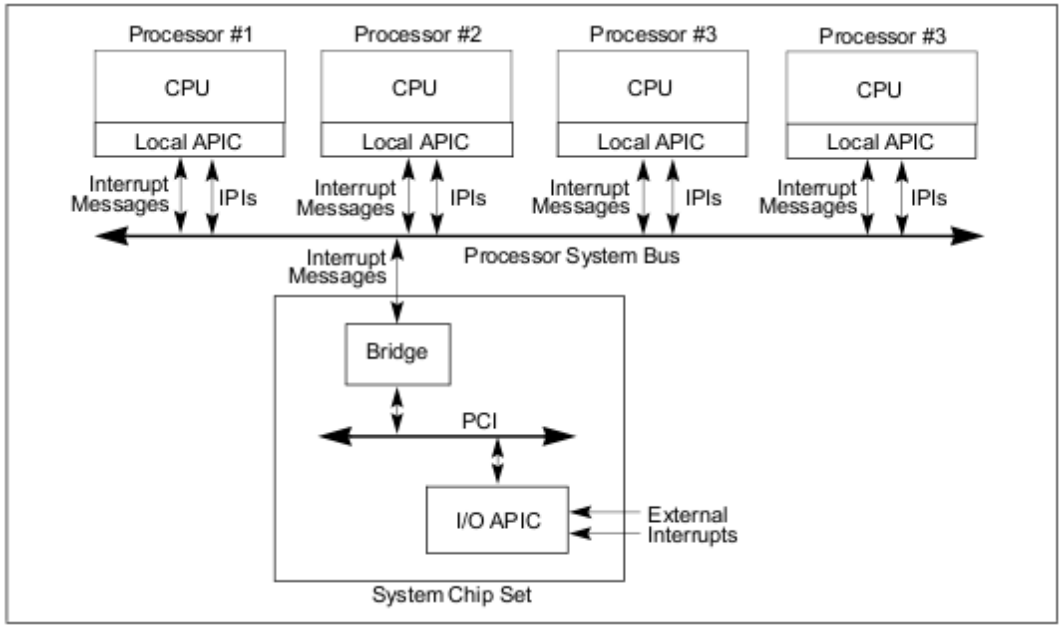
Sterownik przerwań

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

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

APIC dla procesorów Intel Xeon w systemach SMP



- IPI – *Inter-Processor Interrupt* w systemie wieloprocesorowym (SMP) procesor może zażądać działania ze strony innego procesora (tzw. klepnięcie w ramię): synchronizacja pamięci cache, zatrzymanie systemu, itp.

APIC (*Advanced Programmable Interrupt Controller*)

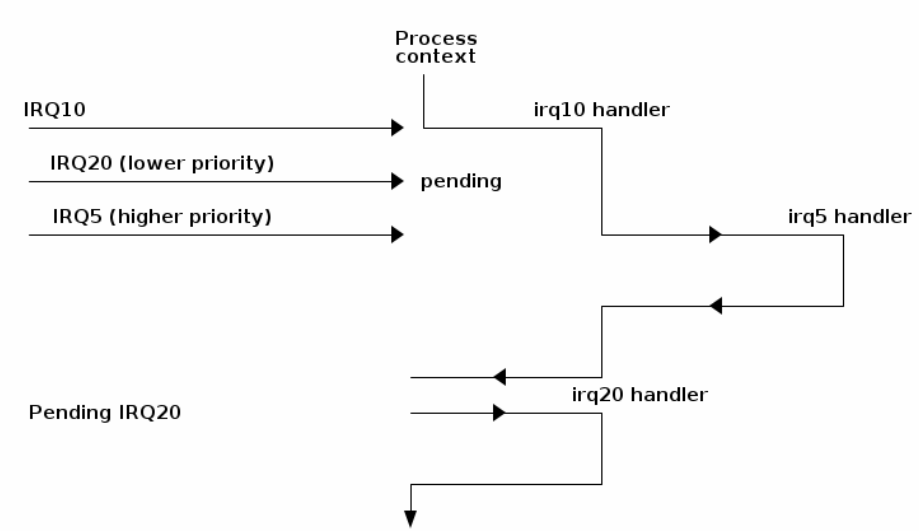
- Local APIC dla każdego CPU
  - obsługa zewnętrznych przerwań dla procesora
  - IPIs - generują i akceptują przerwania między-procesorowe
  - wektor 0-31 wyjątki x86, do 224 wektor z I/O
  - udostępnia wysokiej rozdzielczości zegar (mikrosekundy)
- I/O APIC (opcjonalnie) przy szynie systemowej (po jednym na szynę)
  - przekierowuje przerwania I/O do LAPIC
  - 24 linie IRQ, programowalne rejestry
  - programowalna tabela przekierowań (*Interrupt Redirection Table*): mapowanie IRQ na wektor przerwań, priorytet, docelowy procesor
  - przerwania zewnętrzne są tłumaczone na komunikaty i wysyłane do LAPIC

Sterowanie przerwaniami

Synchronizacja dostępu do współdzielonych danych pomiędzy procedurą obsługi przerwań i innymi potencjalnymi współbieżnymi działaniami (np. inicjalizacja sterownika, przetwarzanie danych sterownika), często wymaga włączenia i wyłączenia przerwań w kontrolowany sposób:

- z poziomu urządzenia - przez zaprogramowanie rejestrów sterujących urządzenia
- z poziomu PIC - oprogramowanie PIC włącza/wyłącza daną linię IRQ
  - żądania na wyłączonej linii nie są gubione, PIC prześle je do CPU jak tylko linia zostanie włączona
  - umożliwia szeregowe przetwarzanie przerwań danego typu
- z poziomu procesora - np. instrukcje *cli* (*CLear Interrupt flag*), *sti* (*SeT Interrupt flag*) w architekturze x86 ustawiają maskowanie przerwania, wówczas każde maskowalne przerwanie jest ignorowane (czasowo) przez procesor

Priorytety przerwań



- priorytety przerwań są wspierane przez większość architektur, choc ich obsługa jest problematyczna i rzadko wystepuje w systemach ogólnego przeznaczenia, przydatne w systemach czasu rzeczywistego
- obsługa przerwania o wyższym priorytecie może zostać wykonana w czasie obsługi przerwania o niższym priorytecie (zagnieżdżenie przerwań)
- przerwania o niższym priorytecie są odłożone do późniejszego wykonania
- wyjątki procesora x86 mają wyższy priorytet niż przerwania I/O

Przerwania: wady

- 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 wymaga dodatkowego modułu na magistrali systemowej (moduł DMAC, *DMA Controller*, część mostka południowego).

Przerwania: DMA

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

- czy wymagany jest odczyt/zapis
- adres urządzenia wej-wyj
- adres początkowej komórki pamięci z danymi (na dane)
- liczbę słów do przesłania

Procesor inicjuje przesłanie danych i kontynunuje przetwarzanie do momentu nadejścia przerwania od modułu DMAC.

Sterowanie zdarzeniami (przerwaniami) rodzi problem synchronizacji w dostępie do zasobów.

Przerwania DMA

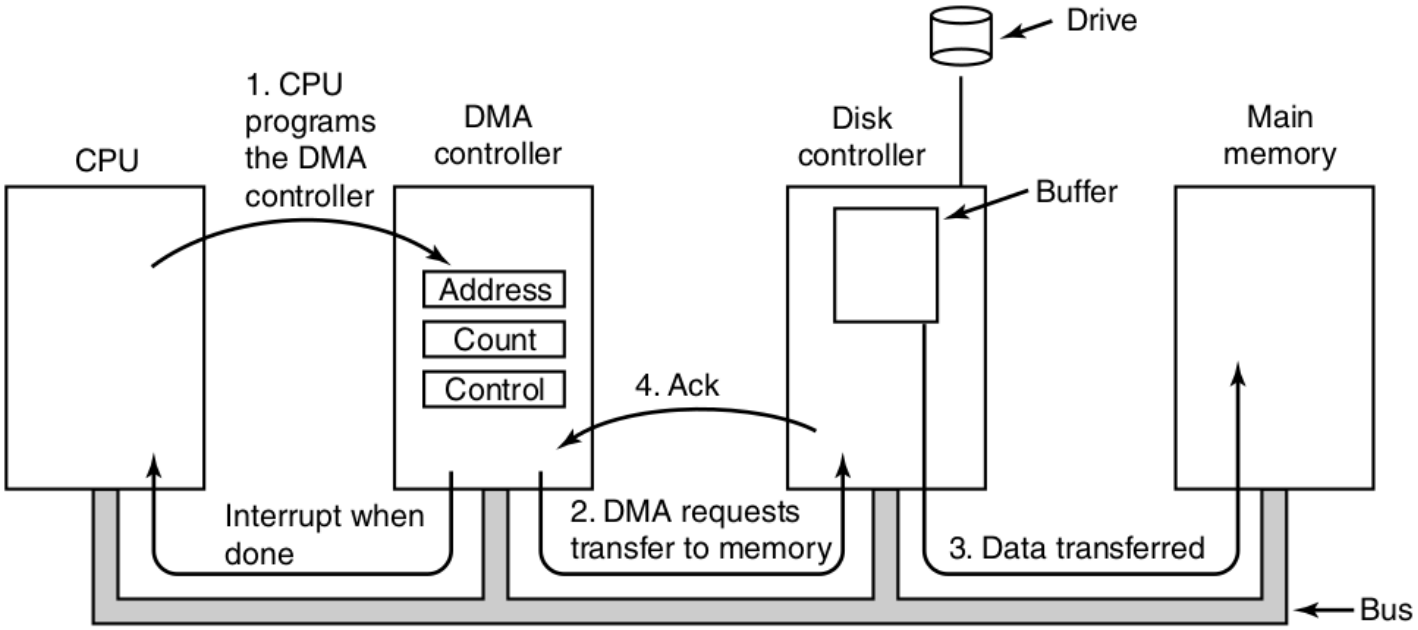


Figure 3-4. Operation of a DMA transfer.

Przerwania sygnalizowane komunikatami (MSI)

Message Signaled Interrupts

- alternatywny system przerwań wprowadzony od PCI 2.2 i PCIe
- obsługiwany przez APIC
- wykorzystuje szynę danych zamiast dedykowanych linii IRQ (PCIe nie posiada linii IRQ)
- komunikacja poprzez zapis do adresów pamięci (MMIO) magistrali PCI (mostka), komunikat nie niesie żadnych danych, jest małym pakietem identyfikującym źródło przerwania
- szybszy w działaniu (rzędy wielkości mniejsze opóźnienia), większa liczba przerwań (dziesiątki przerwan na kartę)

Przyporządkowywanie przerwań

```
$ less /var/log/syslog

Sep 22 09: ... kernel: ttyS00 at 0x03f8 (irq = 4) is a 16550A
Sep 22 09: ... kernel: PCI: Found IRQ 5 for device 00:08.0
Sep 22 09: ... kernel: maestro: Configuring ESS Maestro 2E found at IO 0xD800 IRQ 5
Sep 22 09: ... kernel: parport0: irq 7 detected
Sep 22 09: ... kernel: lp0: using parport0 (polling).
Sep 22 09: ... kernel: ttyS04 at port 0x4880 (irq = 11) is a 16550A
Sep 22 22: ... kernel: eth0: Xircom Cardbus Adapter rev 3 at 0x4800, 00:10:A4:D2:52:55, IRQ 11.
Sep 22 09: ... kernel: ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Sep 22 09: ... kernel: ide1 at 0x170-0x177,0x376 on irq 15
```

Monitorowanie przerwań sprzętowych

```
# 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
MIS:          0
```

- edge - zgłoszenie na linii IRQ przez zmianę poziomu (wymaga synchronizacji)
- `level` - zgłoszenie na linii IRQ przez podniesienie poziomu

Monitorowanie przerwań sprzętowych

```
# 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 Linux: górna i dolna połówka

- górna połówka (*top half*)
  - uruchamiany bezpośrednio po odbiorze sygnału przerwania
  - realizuje tylko te czynności, które mają kluczowe znaczenie dla funkcjonowania urządzenia (przyjęcie przerwania, restart urządzenia)
  - uruchomienie funkcji obsługi przerwań urządzeń
  - zlecenie wykoanania odroczonej akcji (dolnej połówki)
  - w tym czasie blokowana jest obsługa pozostałych przerwań
  - wykonywanie zbyt długich operacji podczas blokowaniu przerwan powoduje opóźnienia, problemy z wydajnością i gubienie przerwań
- dolna połówka (*bottom half*)
  - obejmuje czynności obsługi, które mogą być odłożone w czasie i wykonane w momencie mniejszego obciazenia systemu
  - przerwania nie są blokowane w czasie obsługi

Odroczone akcje w Linux

softIRQ

- działają w kotekście przerwania, zlecane przez funkcję obsługi przerwania
- przydzielane statycznie, użycie ograniczone do wybranych podsystemów z wymogami małych opóźnieniach i wysokiej częstotliwości
- obsługa może być zlecana równolegle na wielu CPU
- zarządzane przez wątki jądra `ksoftirq`

tasklet - szczególny rodzaj softIRQ

- działają w kontekście przerwania
- mogą być dynamicznie alokowane (np. podczasa ładowania modułu)
- ogsługa szeregowa, pojedyncza akcja może być wykonana na jednym procesorze

workqueues

- zadania odroczone wykonywane w kontekście procesu

softirq

```
$ 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
```

Monitorowanie przerwań sprzętowych: klawiatura

Działanie: naciśnięcie i przytrzymanie pojedynczego klawisza

```
root@nscobie ~$ sar -I 1 1
Linux 5.14.16-101.fc33.x86_64 (nscobie)         15.11.2021      _x86_64_        (8 CPU)

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
09:38:20             1     30,00
09:38:21             1     29,00
09:38:22             1     29,00
09:38:23             1     29,00
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
```

Monitorowanie przerwań sprzętowych: myszka/gładzik

Działanie: szybkie ruchy myszką

```
root@nscobie ~$ sar -I 12 1
Linux 5.14.16-101.fc33.x86_64 (nscobie)         15.11.2021      _x86_64_        (8 CPU)

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
09:46:02           12    469,00
09:46:03           12    472,00
09:46:04           12    469,00
09:46:05           12    473,00
09:46:06           12    349,00
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

Działanie: `ping -i 0.002 -c 1000 192.168.1.1`

```
$ sar -I 174 1
Linux 6.9.3-76060903-generic (kis)  01.11.2024  _x86_64_ (8 CPU)

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
22:44:32          174     16,00
22:44:33          174      6,00
22:44:34          174     24,00
22:44:35          174     11,00
^C

Średnia:          174    363,70
```

Wieloprgramowość i ochrona sprzętowa

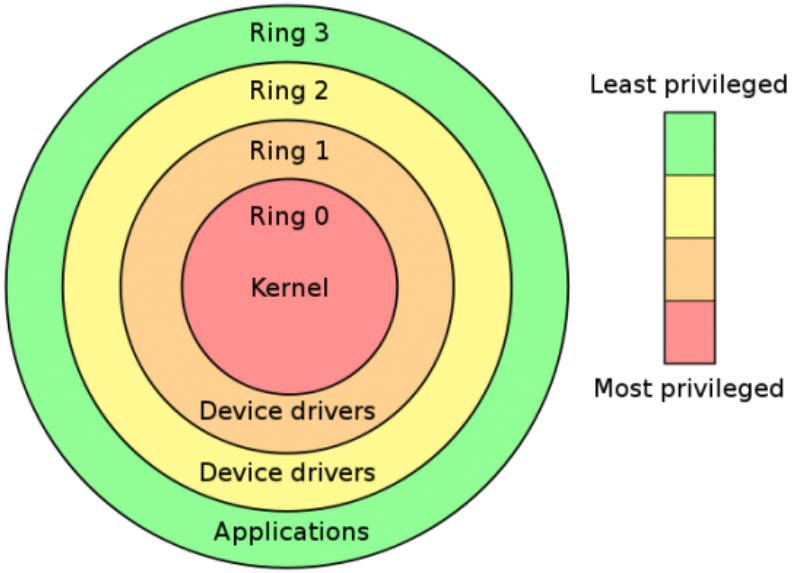
- w systemach wieloprogramowych i z podziałem czasu operacje wej-wyj nakładają się na działanie jednostki centralnej (asynchroniczne wej-wyj).
- zarządzanie wieloma programami wymaga odpowiednich mechanizmów ochrony, aby zapewnić bezpieczeństwo i stabilność systemu operacyjnego
- rolą jądra systemu operacyjnego jest nadzór nad zadobami i udostępnienie interfejsu dostępu do sprzętu
- system operacyjny i jądro systemu są programami, więc potrzebny mechanizm odróżniający kod wykonywany przez użytkonika od kodu jądra systemu

Ochrona sprzętowa

- generowanie przerwania od czasomierza (co kwant czasu) [2], poprawne działenie jest kluczowe dla stabilności działania systemu
- ochrona pamięci realizowana przez sprzęt (stronicowanie, dawniej adres bazowy i graniczny), zapewnia izolacje pamięci między procesami
- ochrona wektora przerwań, procedur wej-wyj, które umożliwiają bezpieczne i kontrolowane przerywanie działania procesora w celu obsługi zdarzeń zewnętrznych
- wszelkie rozkazy wej-wyj są uprzywilejowane, tylko system operacyjny może komunikować się z urządzeniami,
- tryby pracy procesora: tryb użytkonika, tryb jądra (pierścienie uprzywilejowania)

[2] W jądrach >= 2.6 pojawia się możliwość zastosowania tzw. *dynamic ticks*, tzn. że przerwania zegarowe są generowane w razie potrzeby zarówno wtedy, kiedy system jest zajęty, jak i wtedy kiedy pracuje w trybie "idle"(CPU idle state).

Pierścienie uprzywilejowania



- pierścień 3 - dostępny dla użytkownika
- pierścień 0 - tylko jądro systemu, ,uprzywilejowane opeacje, niedostępny bezpośrednio dla programistów i użytkowników

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

Dualny tryb pracy

- system operacyjny musi gwarantować, że niepoprawny program nie będzie mógł zakłócić działania innych programów
- niepoprawnie działający program musi generować przerwanie
- ochronie muszą podlegać wszelkie zasoby dzielone
- sprzęt pozwalający odróżnić dwa tryby pracy: **tryb użytkownika** oraz **tryb jądra** (monitora, nadzorcy, systemu, uprzywilejowany)
- każde przerwanie powoduje przejście systemu z trybu użytkownika do trybu jądra
- dualny tryb pracy jest uzupełniony listą uprzywilejowanych rozkazów maszynowych, które mogą być wykonywane tylko w trybie jądra

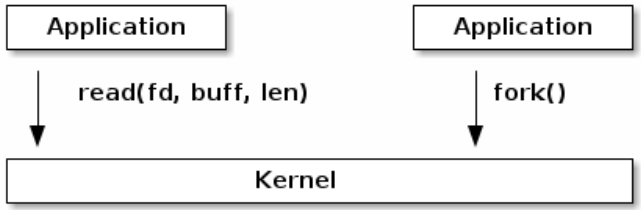
Dualny tryb pracy (cd)

- **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 użytkowe działają z ograniczonymi uprawnieniami, mogą zlecić systemowi wykonanie operacji zastrzerzonych (np. dostęp do zasobów) poprzez wywołania systemowe (*system calls*).

  Odwołanie do systemu 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/programisty wywołania systemowe są „usługami" oferowanymi przez jądro systemu aplikacjom użytkownika i przypominają API bibliotek, tj. są opisane jako wywołanie funkcji z nazwą, parametrami i wartością zwracaną[3]



Wywołania systemowe nie są w rzeczywistości wywołaniami funkcji, ale konkretnymi instrukcjami asemblera (specyficznymi dla architektury i jądra), które wykonują:

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

[3] API Linuksa jest zgodny ze standardem POSIX (*Portable Operating System Interface*).

- biblioteki systemowe udostępniają funkcje, które implementują właściwe wywołania systemowe, w ten sposób łatwo z nich korzystać w aplikacjach
- interfejs wywołań systemowych w Linuksie (większości Uniksów) stanowi częściowo biblioteka 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 zainicjować wywołania funkcji operacji 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 $0x80`
- powoduje ona przełączenie systemu do trybu jądra i rozpoczęcie wykonywania wektora przerwania 128, który wskazuje na adres procedury obsługi wywołań systemowych `system_call()` (zdefiniowana w pliku *entry.S*)
- konkretne wydołanie systemowe jest identyfikowane przez numer wywołania, który jest umieszczany w rejestrze `eax`
- rejestry (`ebx`, `ecx`, `edx`, `esi`, `edi`) są (zwykle) używane do przekazywania argumentów wywołania systemowego. Przed użyciem dokonywana jest dokładna weryfikacja przekazanych argumentów.

Przerwania programowe i wywołania systemowe

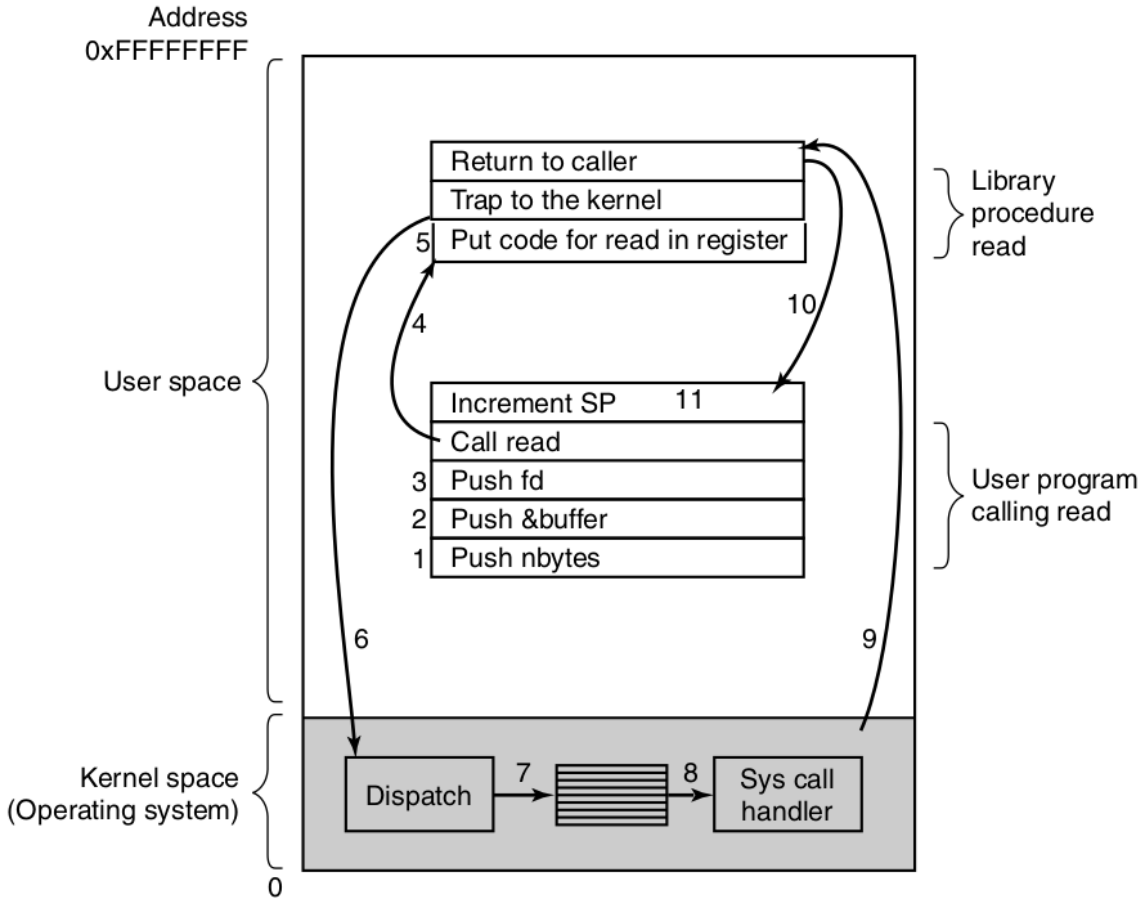


Figure 1-16. The 11 steps in making the system call read(fd, buffer, nbytes).

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_preadv2 327
#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_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
```

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
```

Przerwania programowe: śledzenie

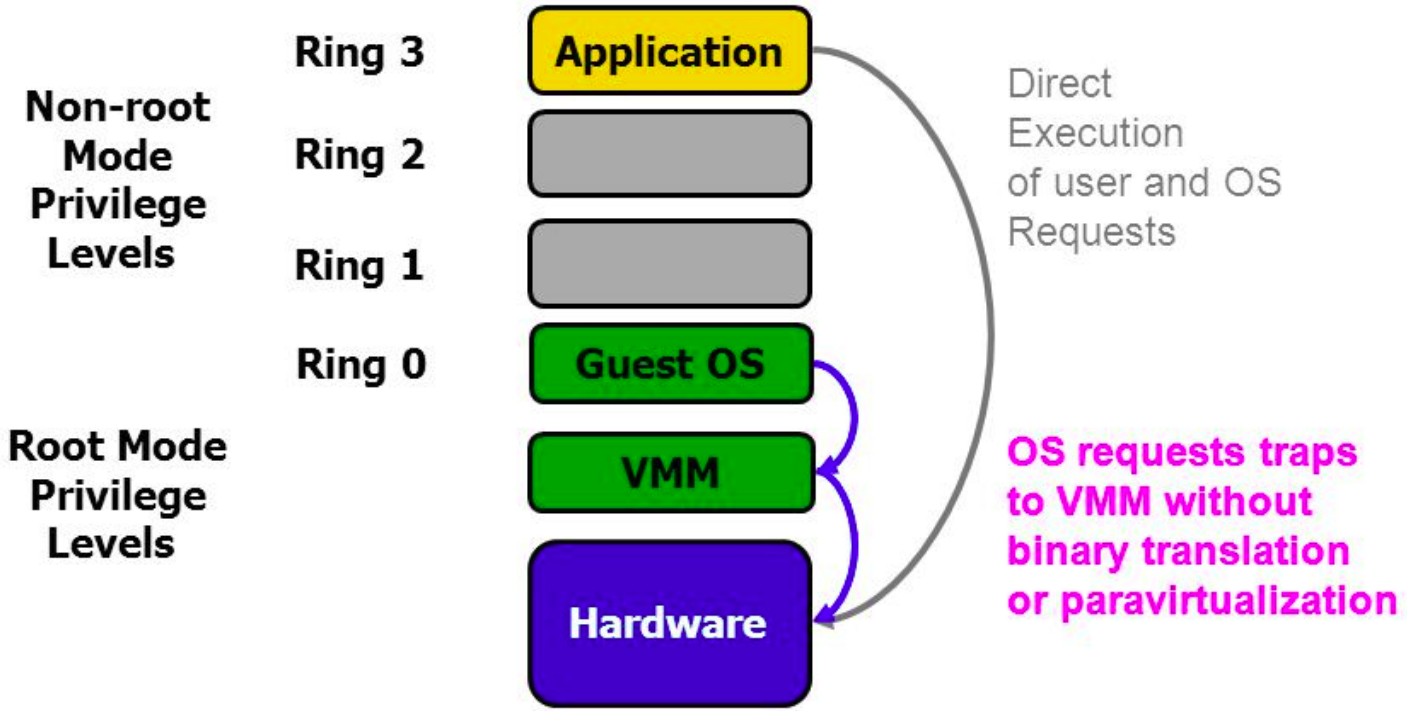
```
$ strace -c firefox
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ------------------
 26.32    0.364886          45      7963       505 futex
 13.59    0.188327          13     14104     12926 recvmsg
 11.21    0.155370          14     11038           poll
  7.90    0.109478          18      5802           write
  7.60    0.105350          18      5705         1 madvise
  5.28    0.073184        2152        34        16 wait4
  5.19    0.071983          12      5705         3 read
  4.38    0.060772          23      2565           mprotect
  3.09    0.042794          20      2041           mmap
  3.08    0.042744          41      1023           munmap
   ...
  0.00    0.000000           0         1           set_robust_list
  0.00    0.000000           0         1           inotify_init1
  0.00    0.000000           0         1           sched_setattr
  0.00    0.000000           0         1           sched_getattr
------ ----------- ----------- --------- --------- ------------------
100.00    1.386228          20     68379     14862 total
```

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
```

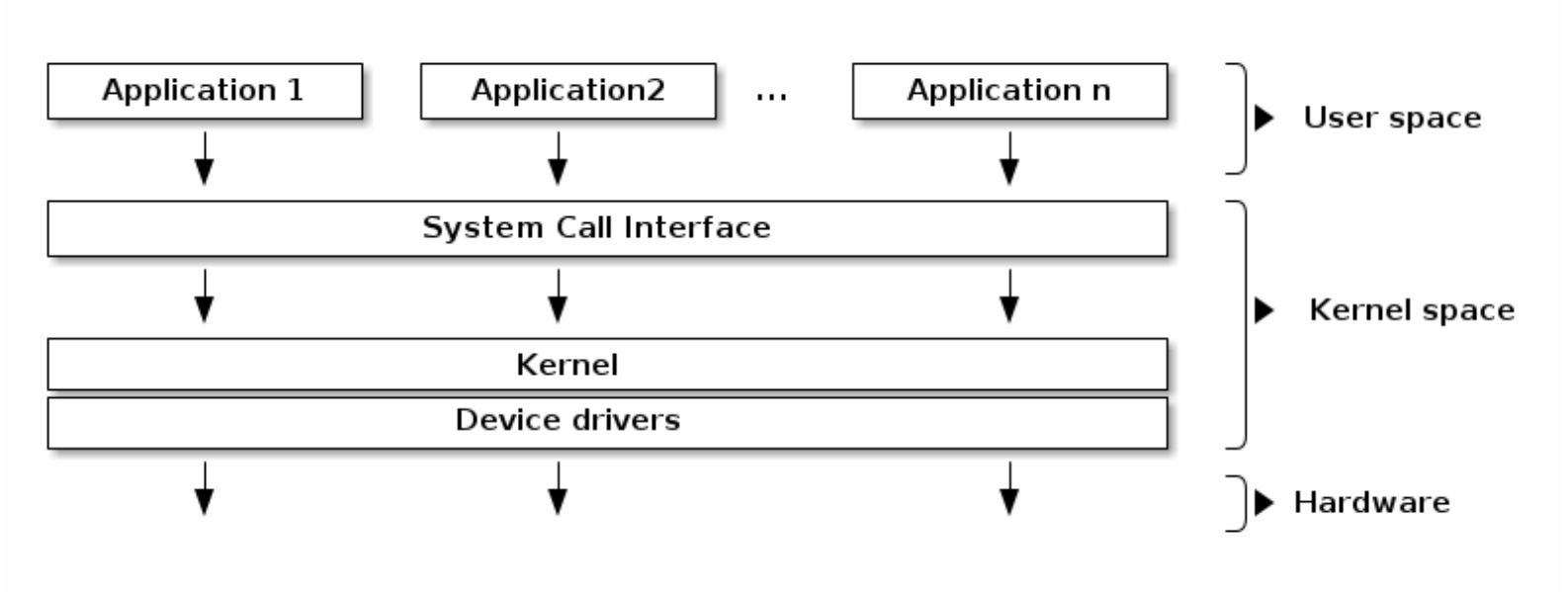
- `real` czas rzeczywisty
- `user` czas CPU spędzony w trybie urzytkownika
- `sys` czas CPU spędzony w trybie jądra

Pierścienie uprzywilejowania i wirtualizacja

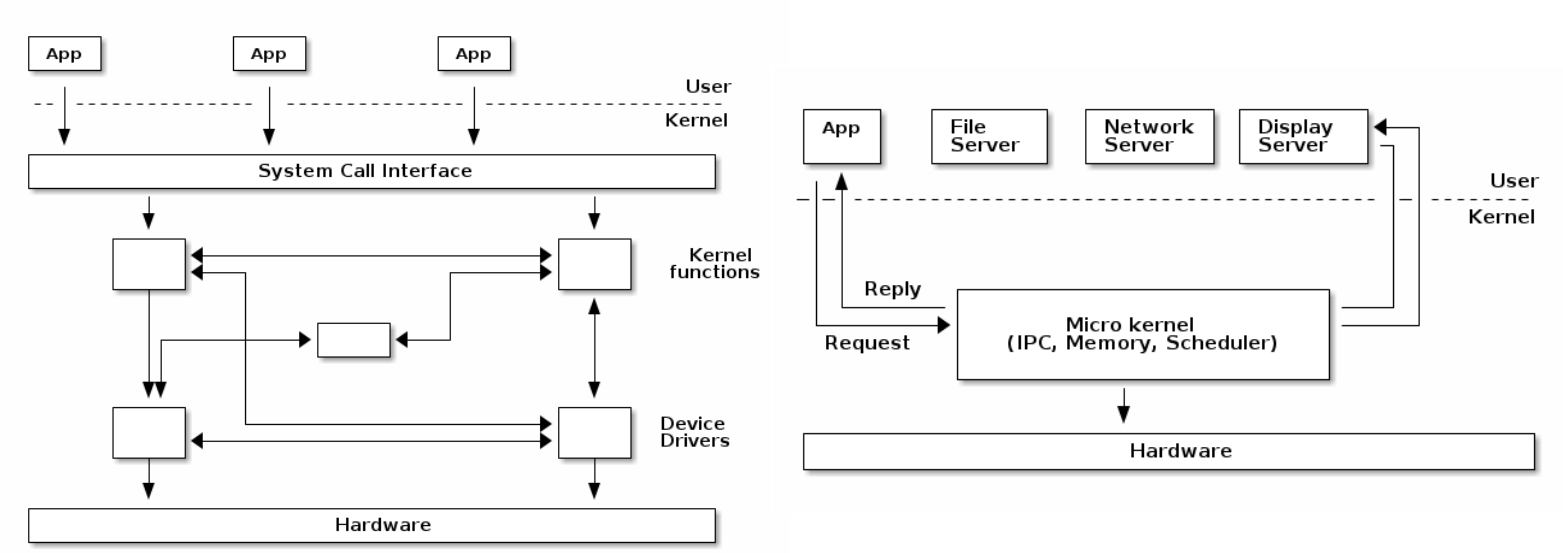


http://slideplayer.com/slide/5084764/

Typowa architektura systemu



Monolityczne jądro vs. mikrojądro



- **Jądro monolityczne** nie ma ochrony dostępu pomiędzy różnymi podsystemami jądra i w którym funkcje publiczne mogą być bezpośrednio wywoływane pomiędzy różnymi odsystemami (Linux)
- **Mikrojądro** - duże części jądra są przed sobą chronione i zwykle działają jako usługi w przestrzeni użytkownika. Ponieważ znaczna część jądra działa w trybie użytkownika, pozostały kod działający w trybie jądra jest znacznie mniejszy, stąd termin mikrojądro.
- jądra hybrydowe (Mac OS X, Windows) - jednak większość monolitycznych usług działa tu w trybie jądra, więc można kwalifikować je jako monolityczne
- Debata o wyższości mikrojądra nad monolitycznym jądrem Tanenbaum–Torvalds debate