

Operační systémy

Přednáška 12: Vstup/výstup

Hlavní funkce V/V

- OS musí **spravovat všechny V/V zařízení počítače**.
 - vyslat příkaz pro dané zařízení, reagovat na přerušení, reagovat na chyby.
 - zakázat neautorizovaný přístup.
- OS musí **poskytovat rozhraní** mezi V/V zařízeními a zbytkem systému.
 - jednotný přístup k zařízením (na úrovni API),
 - přístup k zařízením na vyšší úrovni abstrakce,
 - virtualizace zařízení (spooling),
 - jednotné pojmenování,
 - jednotné reakce na chyby.

Typy V/V zařízení

- **Blokově orientované**

- Informace je uložena ve stejně velkých blocích, každý je přímo adresovatelný.
- Lze číst/zapisovat každý blok nezávisle od ostatních bloků.
- Například: disky, diskety,...

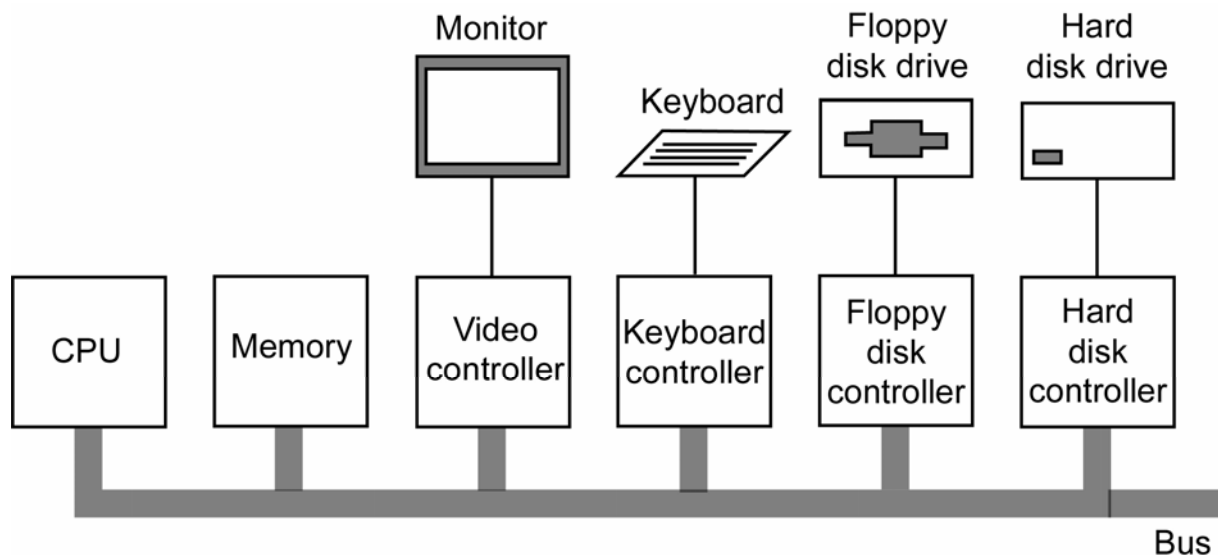
- **Znakově orientované**

- Informace je uložena jako posloupnost bytů.
- Není možné přistupovat přímo k jednotlivým znakům.
- Například: tiskárny, síťová rozhraní, myši, pásy,...

- **Některá zařízení nelze zařadit do těchto skupin**

- (např. časovače, ...).

Periferní zařízení



- V/V zařízení se obecně skládají z dvou částí:
 - **řadič zařízení,**
 - **samotné V/V zařízení.**
- Komunikace se samotným V/V zařízením probíhá na **nízké úrovni.**
- Úkolem řadičem je poskytnout **jednoduchého rozhraní** pro OS.

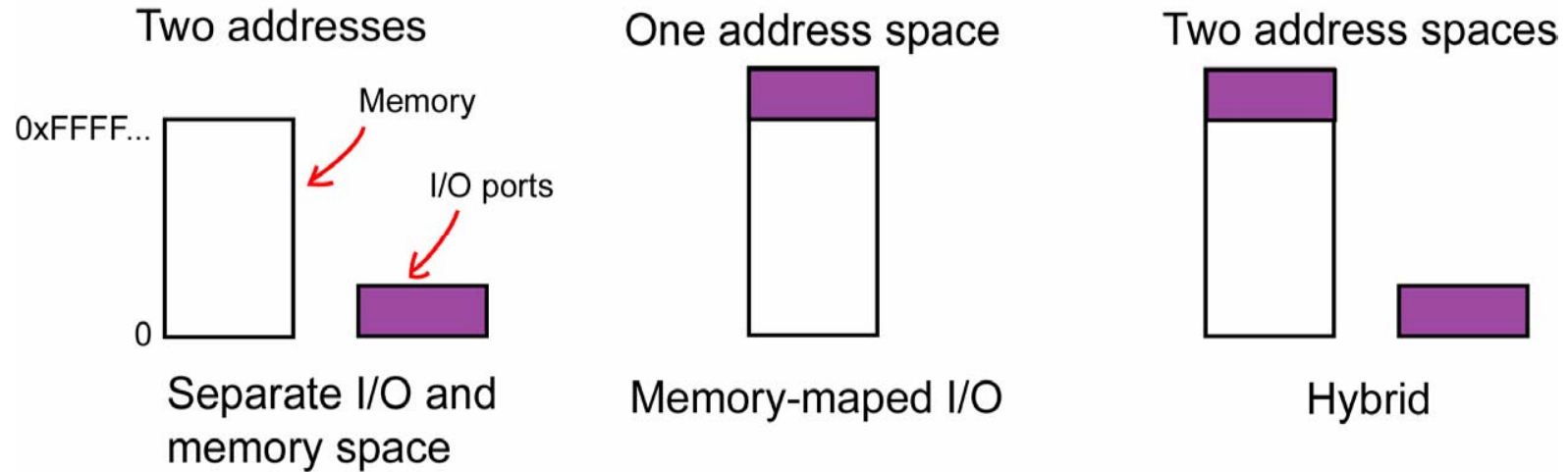
Periferní zařízení (2)

- Každý řadič má:
 - **řídící registry**
 - Pomocí zápisu do registrů, CPU může udílet příkazy V/V zařízení
 - Pomocí čtení těchto registrů, CPU může získat informaci o stavu V/V zařízení
 - **vyrovnávací paměti** (ne vždy)
 - CPU může číst/zapisovat do těchto pamětí.
- **Jak CPU komunikuje s řídícími registry a vyrovnávacími pamětmi ?**
 - Oddělený V/V prostor a paměť.
 - V/V prostor mapovaný do paměti.

Oddělený V/V prostor a paměť

- Adresový prostor paměti je jiný než V/V prostor.
- Každý řídicí registr má přiřazeno **číslo V/V portu** (8- or 16-bit integer).
- **CPU používá speciální V/V instrukce:**
 - **IN REG, PORT**
 - pro zkopírování hodnoty řídicího registru PORT do registru REG v CPU
 - **OUT PORT, REG**
 - pro zápis hodnotu z REG do řídicího registru PORT.
- Následující instrukce jsou rozdílné!!!
IN R0, 4 x MOV R0,4

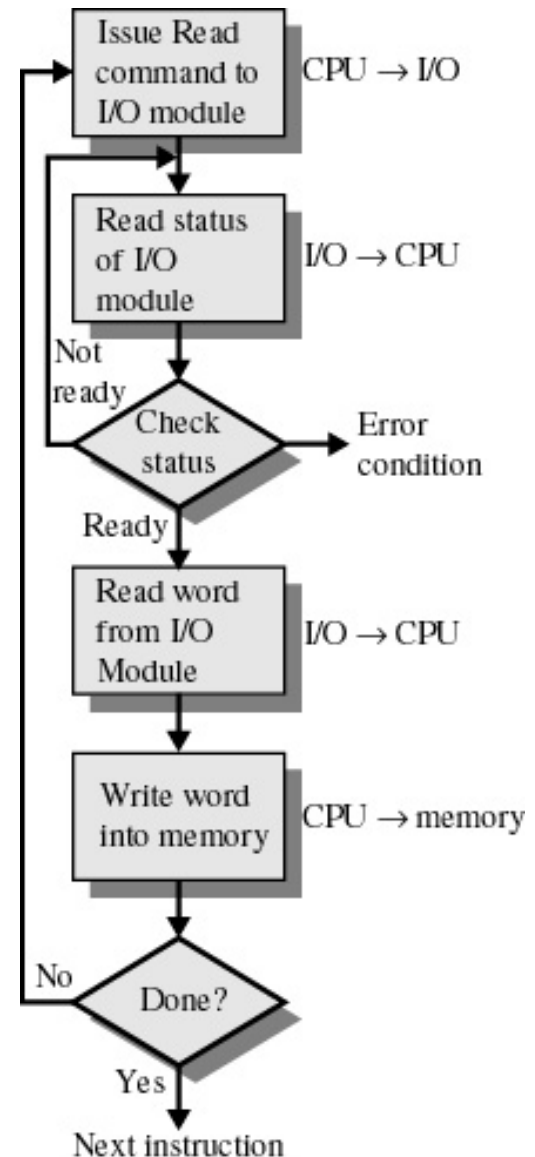
V/V prostor mapovaný do paměti



- Ve **V/V prostoru mapovaném do paměti**, každý řídicí registr má přiřazenu jedinečnou paměťovou adresu.
- V **hybridním modelu**, jsou vyrovnávací paměti mapovány do paměti, ale řídicí registry jsou odděleny pomocí čísel portů.

Programovatelný V/V

- CPU se stará o celou V/V operaci.
- CPU zahájí V/V operaci.
- Potom CPU čeká na její dokončení pomocí aktivního čekání.



Programovatelný V/V (2)

Příklad: Uživatelský proces chce vytisknout osmi znakový řetězec “ABCDEFGH” na tiskárnu. Předpokládáme V/V prostor mapovaný do paměti a tiskárnu bez vyrovnávací paměti.

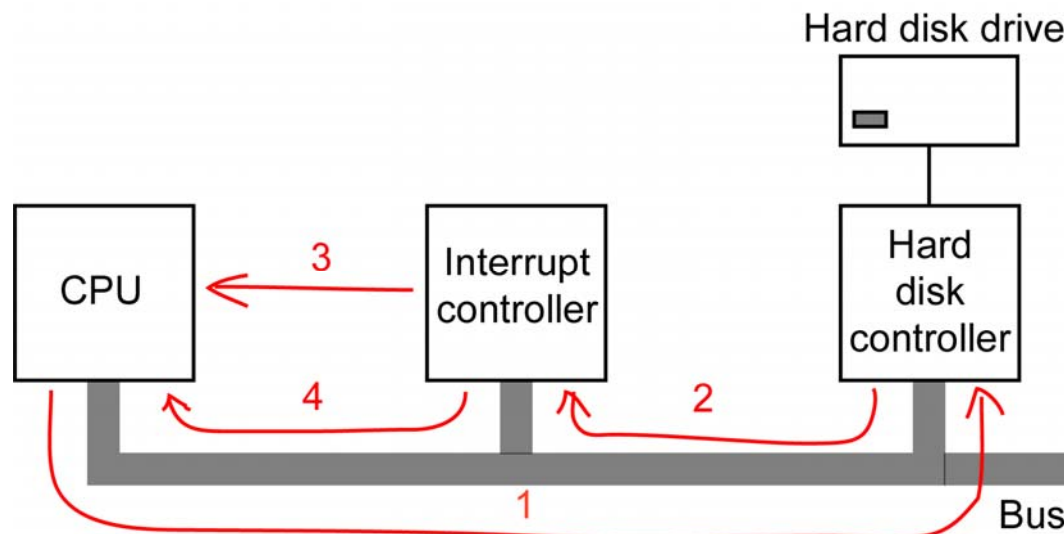
- Uživatelský proces použije systémové volání.
- OS provádí následující kód:

```
copy_from_user(buffer, p, count);          /* p is the kernel buffer */

for ( i=0; i<count; i++ ) {                /* loop on every character */
    while (*printer_status_reg != READY);   /* loop until ready */
    *printer_data_register = p[i];          /* output one character */
}

return_to_user();
```

Přerušení

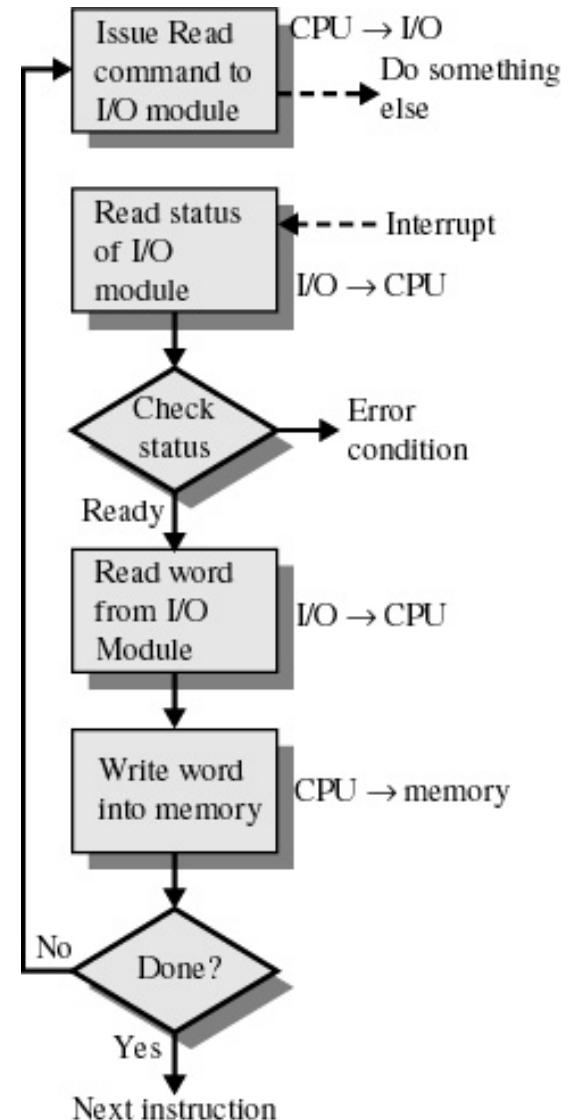


- **V/V operace pomocí přerušení**

1. **CPU řekne řadiči** co má dělat tím, že zapíše příkaz do řídicího registru.
2. Když řadič V/V zařízení dokončí operaci, **pošle signál do řadiče přerušení.**
3. Pokud je řadič přerušení připraven přijmout přerušení, **informuje o tom CPU.**
4. Řadič přerušení **pošle přes sběrnici číslo V/V zařízení do CPU.**

V/V používající přerušování

- CPU je zahájí V/V operaci a pak pokračuje v jiné práci.
- Není zde aktivní čekání.
- Když jsou data připravena je vyvoláno přerušování a CPU zkopíruje data do paměti.



V/V používající přerušení (2)

Příklad: Uživatelský proces chce vytisknout osmi znakový řetězec “ABCDEFGH” na tiskárnu. Předpokládáme V/V prostor mapovaný do paměti a tiskárnu bez vyrovnávací paměti.

Uživatelský proces použije systémové volání.
OS provádí následující kód:

```
/* Code executed when          */  
/* the print system call is made */
```

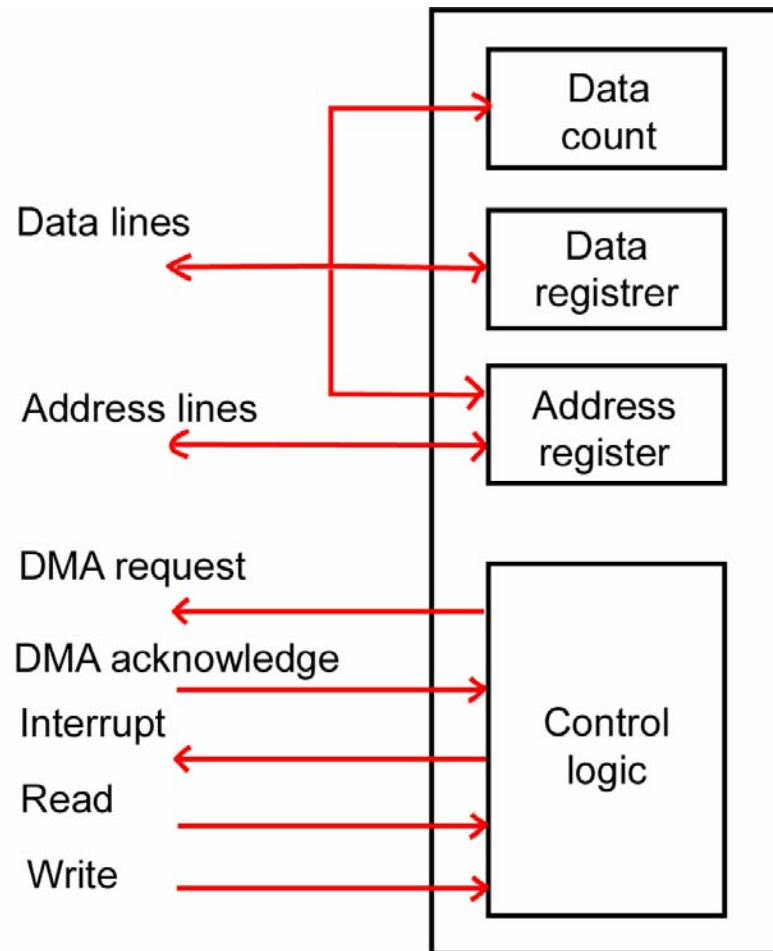
```
copy_from_user(buffer, p, count);  
enable_interrupts();  
while (*printer_status_reg != READY);  
*printer_data_register = p[0];  
scheduler();
```

```
/* Interrupt service */  
/*   procedure      */
```

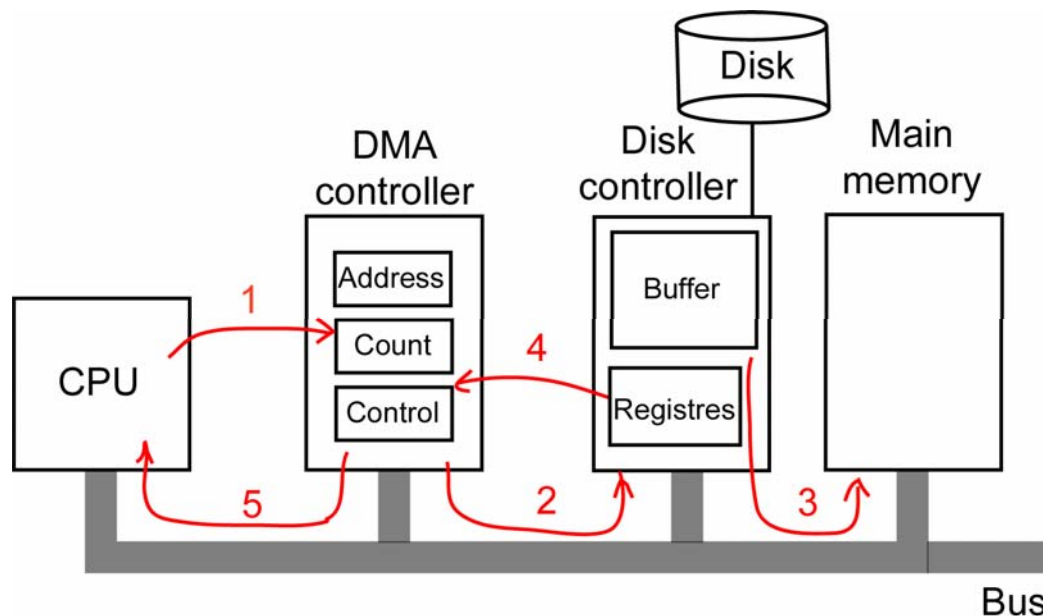
```
i = 1;  
if (count == 0) {  
    unblock_user();  
} else {  
    *printer_data_register = p[i];  
    count = count - 1;  
    i = i + 1;  
}  
acknowledge_interrupt();  
return_from_interrupt();
```

Direct Memory Access (DMA)

- **Speciální DMA chip** může řídit tok dat mezi pamětí a V/V zařízením bez asistence CPU.



DMA (2)

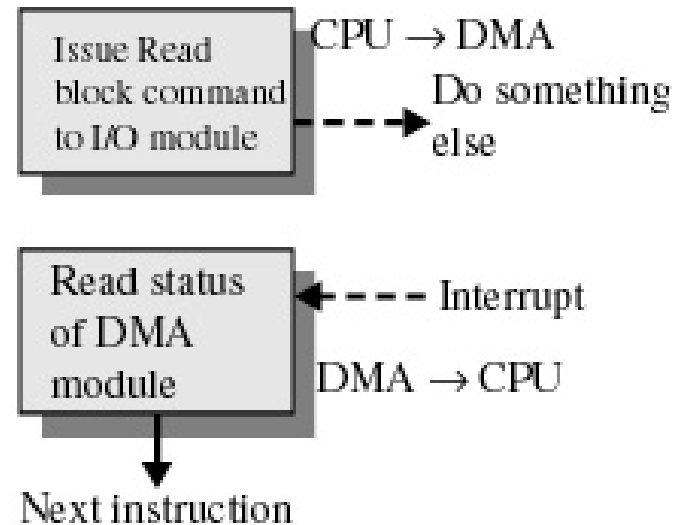


- **V/V operace pomocí DMA**

1. CPU naprogramuje DMA řadič nastavením jeho registrů.
2. DMA pošle příkaz do řadiče V/V zařízení.
3. Přenos dat do paměti.
4. Když je přenos dat dokončen, řadič disku pošle potvrzení do řadiče DMA.

V/V používající DMA

- Přesouvá data přímo do/z paměti.
- K přerušení dojde až když je V/V operace dokončena.
- CPU pouze zahájí a dokončí V/V operaci.



(c) Direct memory access

V/V používající DMA (2)

Příklad: Uživatelský proces chce vytisknout osmi znakový řetězec “ABCDEFGH” na tiskárnu. Předpokládáme V/V prostor mapovaný do paměti a tiskárnu bez vyrovnávací paměti.

Uživatelský proces použije systémové volání.
OS provádí následující kód:

```
/* Code executed when      */  
/* the print system call is made */
```

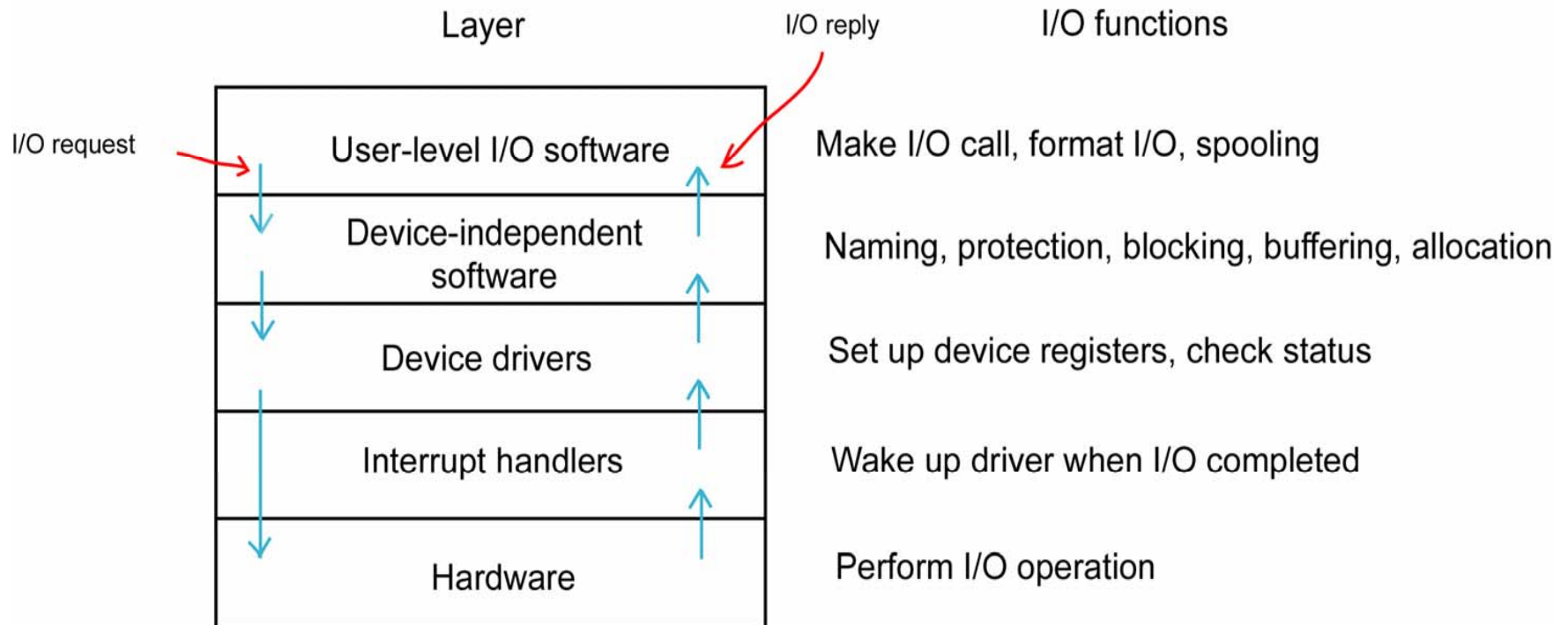
```
copy_from_user(buffer, p, count);  
set_up_DMA_controller();  
scheduler();
```

```
/* Interrupt service */  
/* procedure */
```

```
acknowledge_interrupt();  
unblock_user();  
return_from_interrupt();
```


Struktura V/V softwaru

- Typicky organizován do čtyř vrstev.
- Každá vrstva má **přesně definovanou funkci**, kterou plní, a **přesně definované rozhraní** k sousedním vrstvám.
- Funkce a rozhraní je **závislé na systému**.



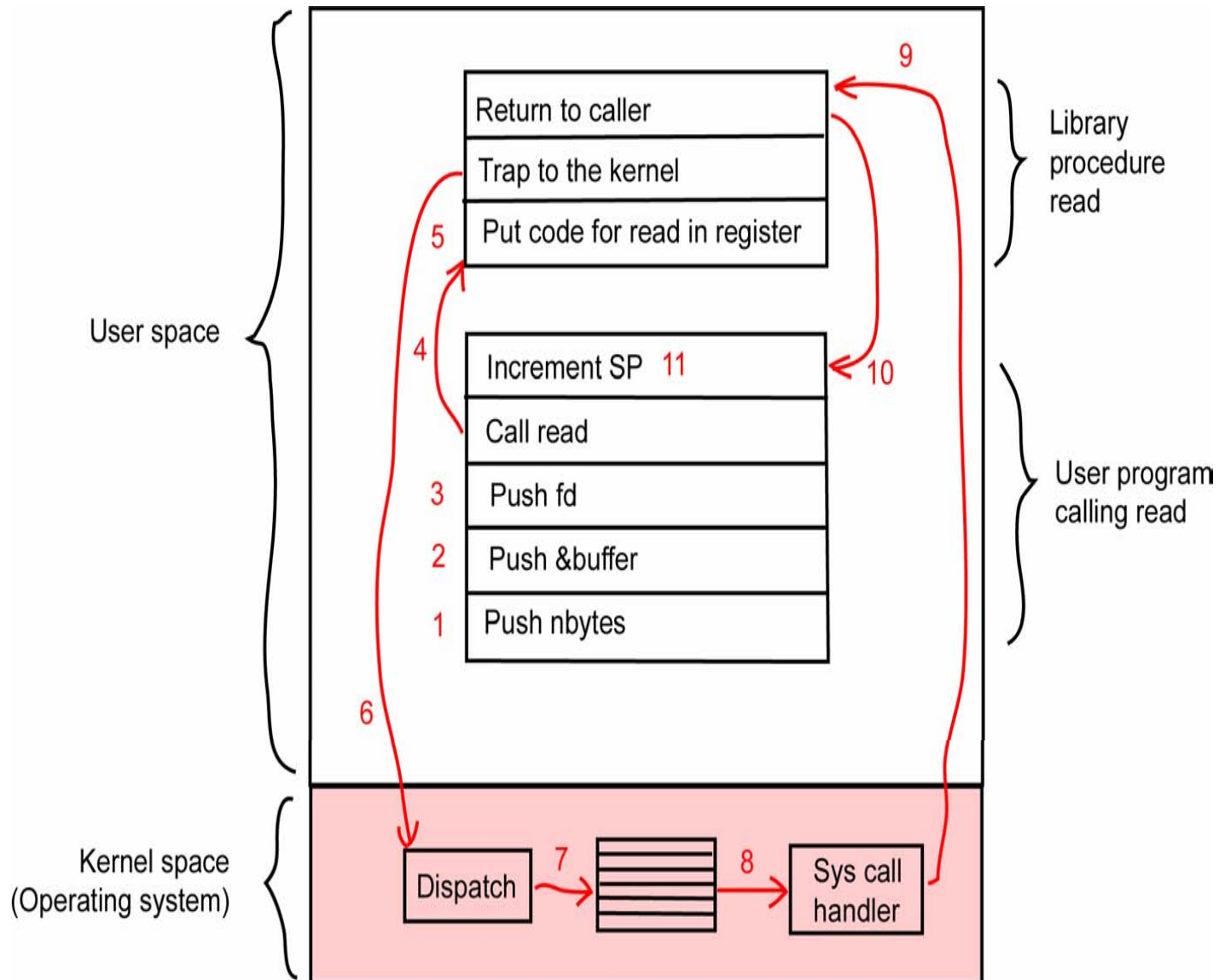
User-Space I/O Software

- Většina V/V software je součástí OS.
- Malá část je v **knihovnách**, které jsou spojené s uživatelským programem a **běží mimo jádro**.
 - **knihovní funkce**
`printf("The square of %3d is %6d\n", i, i*i);`
 - **systémová volání**
`count=write(fd, buffer, nbytes);`
- **Spooling system** = cesta umožnit přístup k jednotlivým V/V zařízením ve více úlohovém systému (e.g. printer, file transfer over a network,...).

Volání služeb jádra

- Nástroj **rozhraní mezi aplikacemi a jádrem OS**.
- **Volání jsou přímo dostupná na úrovni**
 - symbolického strojového jazyka (assembleru),
 - vyššího programovacího jazyka (C/C++).
- **Metody předávání parametrů** mezi aplikací a jádrem
 - v **registrech** (jsou dostupné aplikaci i jádru),
 - v **tabulce** uložené v hlavní paměti a adresa tabulky se umístí do
 - registru,
 - na zásobník (je dostupný aplikaci i jádru).

- **Příklad:** `count = read(fd, &buffer, nbytes)`



Volání služeb jádra (2)

Kroky systémového volání:

- 1-3 aplikace umístí parametry na zásobník,
- 4 aplikace zavolá knihovní funkci,
- 5 každé systémové volání má přiřazeno unikátní číslo,**
knih. funkce vloží toto číslo do příslušného registru,
- 6 knih. funkce provede instrukci TRAP, CPU se přepne do kernel modu
- 7 jádro zkontroluje číslo systémového volání a pak zavolá odpovídající „system call handler“,
- 8 běží „system call handler“,
- 9 CPU se přepne do uživatelského režimu a řízení se vrací knih. funkci,
- 10 knih. funkce vrací řízení aplikaci,
- 11 aplikace vyčistí zásobník a pokračuje.

Device-Independent I/O Software

- **Hlavní funkce**

- **Jednotné rozhraní pro ovladače**

- Jednoduché přidávání nových ovladačů.
 - Mapování symbolických jmen zařízení na konkrétní ovladače
(např. *major* a *minor* čísla v UNIXu).

- **Buffering**

- Vyrovnávací paměti v uživatelském prostoru, v prostoru jádra.

- **Error reporting**

- **Alokace a uvolňování jednotlivých V/V zařízení**

- **Velikost datových bloků nezávislých na jednotlivých zařízeních**

Device Drivers

- **Každé V/V zařízení** připojené k počítači **potřebuje** nějaký device-specific kód (**device driver**), který ho umí řídit.
- **Hlavní funkce**
 - **Překlad „device independent“ požadavků** pro konkrétní zařízení
(čtení datového bloku -> určení cylindru, hlavičky, sektoru)
 - **Komunikace s řadičem V/V zařízení.**
(zapsání příkazu do řídicího registru)
 - **Řazení více požadavků do fronty.**
 - **Zablokování a čekání na dokončení V/V operace**
 - **Error handling**

Interrupt Handlers

- **Po HW přerušení, SW musí provést:**
 1. **Uložení některých registrů**, které nebyly uloženy HW.
 2. **Nastavení kontextu** přerušovací rutiny (ISP), např. nastavení TLB, MMU, tabulky stránek,...
 3. **Nastavení zásobníku** pro ISP.
 4. **Potvrdit přerušení řadiči přerušení.**
 5. **Kopírovat registry** přerušeného procesu z dočasného umístění do prostoru procesu.
 6. **Spustit ISP.**
 7. Vybrat další proces, který poběží jako další.
 8. **Nastavit kontext pro další proces.**
 9. **Nahrát registry nového procesu.**
 10. **Spustit další proces.**