Pokročilé operační systémy

I/O

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

I/O zařízení

1. samotné zařízení
2. řadič (controller)

Řadič

- rozhraní pro komunikaci se zařízením
- jeden řadič obsluhuje jedno i více zařízení
- pro komunikaci se zařízením slouží registry – čtení stavu, dat, zápis dat a řídících příkazů

Port-Mapped I/O

- registry jednotlivých zařízení mají samostatný adresní prostor (oddělený od paměti)
- přístupné přes operace `in`, `out` – čtení/zápis hodnoty z portu
  - `in al, port`
  - `out port, al`
- nevýhody: omezené řízení přístupu, omezené na speciální operace (jen zápis/čtení), které nejsou standardně k dispozici v C/C++
- lze doplnit inline assemblerem

Operace in a out v inline assembleru

```
static inline void outb(uint16_t port, uint8_t val)
{
    __asm__ volatile ("outb %0, %1" : : "a"(val), "Nd"(port));
}

static inline uint8_t inb(uint16_t port)
{
    uint8_t ret;
    __asm__ volatile ("inb %1, %0" : "=a"(ret) : "Nd"(port));
    return ret;
}
```

GCC inline assembler

```
__asm__ (kód v assembleru
        : výstupní operandy  (volitelně)
        : vstupní operandy   (volitelně)
        : změněné registry   (volitelně)
        )
```

- implicitně AT&T syntaxe
- překladač nepracuje s kódem v assembleru (nutné poskytnout dodatečné informace)
- někdy nutné doplnit `volatile`, které zabrání posunutí/odstranění kódu
- vstupní a výstupní operandy řeší, jak má být naloženo s hodnotami na hranici bloku inline assembleru
- `a` – eax, `b` – ebx, `d` – edx, `c` – ecx, `d` – edx, `N` – přímá hodnota, `r` – registr
- `"=a"(ret)` – obsah registru `eax` ulož do proměnné `ret`
- `"Nd"(port)` – hodnotu `port` použij buď jako přímou hodnotu nebo ulož do registru `edx`

Memory-Mapped I/O

- registry jednotlivých zařízení jsou namapovány do paměťového prostoru
- data se čtou/zapisují přímo na sběrnici; přesměrováno na sběrnici zařízení, ne do paměti
- výhoda: k zařízení se přistupuje jako k paměti (možné používat všechny instrukce); řízení přístupu – lze použít to, co se používá pro paměť
- problém: cache, oddělená sběrnice pro paměť
- vyčleněné oblasti paměti, např. 640 kB až 1 MB - 1 (u IBM PC)
- viz oblast 0x000b:8000, paměť pro výstup na terminál

Ilustrace adresních prostorů pro přístup k zařízením

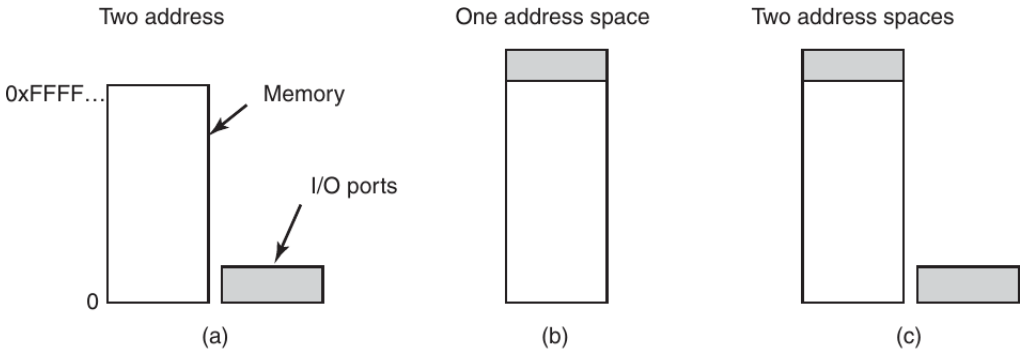

Figure 5-2. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

- Zdroj: Tanenbaum A., Bos. H. Modern operating systems.

Práce s PMIO a MMIO

- obvykle více registrů (řídící, datový)
- princip stejný, jen se liší prostředky PMIO a MMIO

Předání dat zařízení

```
*device_cmd_reg  = COMMAND;
*device_data_reg = data[0];
*device_data_reg = data[1];
*device_data_reg = data[2];
...
```

Čtění dat zařízení

```
if (*device_status_reg == READY) {
   value = *device_data_reg;
}
```

Aktivní čekaní (busy-waiting, polling)

Zápis

```
*device_cmd_reg  = COMMAND;
*device_data_reg = size;
for (int i = 0; i < BUF_SIZE; i++) {
  while (*device_status_reg != READY) { } /* loop */
  *device_data_reg = buf[i];
}
```

Čtení

```
while (*device_status_reg != READY) { } /* loop */
value = *device_data_reg;
```

Praktické příklady (1/2)

Nastavení pozice kurzoru

- 0x3d4 – řídící registr (port)
- 0x3d5 – datový registr (port)

```
uint16_t pos = row * TERM_W + column;       // pozice kurzoru
outb(0x3d4, 0x0f);                          // prikaz pro zapis LSB pozice
outb(0x3d5, (uint8_t) (pos & 0xff));        // hodnota LSB pozice
outb(0x3d4, 0x0e);                          // prikaz pro zapis MSB pozice
outb(0x3d5, (uint8_t) ((pos >> 8) & 0xff)); // hodnota MSB
```

Nastavení vzhledu kurzoru

```
outb(0x3d4, 0x0a);    // prikaz pro nastaveni velikosti kurzoru
outb(0x3d5, 0x00);    // velikost kurzoru
```

Praktické příklady (2/2)

Čtení z klávesnice

- 0x60 – datový registr (port)
- 0x64 – řídící registr (port)
- nejnižší bit řídícího registru indikuje naplnění bufferu (musí být 1, než se může číst z datového registru)

```
unsigned char keyb_scancode()
{
   while (1) {
      uint8_t status = inb(0x64);
      if (status & 0x01) {
         uint8_t scan_code = inb(0x60);
         return scan_code;
      }
   }
}
```

Úkoly

Výstup

- Vyzkoušejte příklady nastavení kurzoru s vlastními hodnotami.
- Integrujte přesun kurzoru do vašeho kódu.

Vstup

- Implementujte následující (případně další pomocné) funkce:
- `uint16_t keyb_get_stroke()`, která vrátí informace o stisknuté klávese a to tak, že odpovídající znak klávesy je ve spodních osmibitech a zbývající bity indikují, jestli byl stisknut některý z modifikátorů (ctrl, shift, případně další).
- `char keyb_getc()`, která vrací jednotlivé stisknuté znaky.

Poznámky

- Funkce `keyb_getc` a `keyb_get_stroke` implementujte jako blokující operace (vycházejte z `keyb_scancode`).
- Funkce pro práci s klávesnicí soustřeďte do samostatných souborů `keyb.c` a `keyb.h`.
- Řešení nemusí být dokonalé, ale mělo by zvládat základní operace pro vstup textu.
- Při řešení vám může být nápomocna stránka `https://wiki.osdev.org/PS/2_Keyboard`.