Laborator 6 - Întreruperi

Obiectivele laboratorului

- înţelegerea modului de comunicare cu dispozitivele periferice
- deprinderea de cunoştinţe de implementare a rutinelor de tratare a întreruperilor (pe Linux şi Windows)
- înţelegerea particularităţilor în sincronizarea cu rutinele de tratarea a întreruperilor

Cuvinte cheie

- IRQ
- port I/O
- adresă I/O
- adresă de bază
- UART
- request_region / release_region
- inb / outb
- READ_PORT_UCHAR / WRITE_PORT_UCHAR
- request_irq / free_irq
- HalGetInterruptVector
- IoConnectInterrupt / IoDisconnectInterrupt
- spin_lock_irqsave / spin_unlock_irqrestore
- KeSynchronizeExecution

Materiale ajutătoare

- Slide-uri de suport pentru laborator
- SO2 Reference Card

Noţiuni generale

Comunicaţia cu hardware-ul

Un dispozitiv periferic este controlat prin scrierea şi citirea registrelor sale. De cele mai multe ori, un dispozitiv are mai multe registre, care pot fi accesate la adrese consecutive, fie în spaţiul de **adrese din memorie**, fie în spaţiul de **adrese I/O**. Fiecare dispozitiv conectat la magistrala I/O are un set de adrese I/O, numite porturi I/O. Porturile I/O pot fi mapate la adrese din memoria fizică, astfel încât procesorul va putea realiza comunicarea cu dispozitivul prin instrucţiuni care operează direct cu memoria. Din motive de simplitate, vom folosi porturile I/O pentru comunicarea cu dispozitivele fizice.

Porturile I/O ale fiecărui dispozitiv sunt structurate într-un set de registre specializate, pentru a oferi o interfaţă uniformă pentru programare. Astfel, majoritatea dispozitivelor vor avea următoarele registre:

- registrul de **control**, care primeşte comenzi pentru dispozitiv
- registrul de **stare**, care conţine starea interna a dispozitivului
- registrul de **intrare**, din care se preiau datele de la dispozitiv
- registrul de **ieşire**, în care se scriu datele pentru a le transmite către dispozitiv

În majoritatea cazurilor, porturile fizice sunt diferenţiate după numărul de biţi: pot fi porturi pe 8, 16 sau 32 de biţi.

Spre exemplu, portul paralel are 8 porturi de I/O pe 8 biţi, începând de la adresa de bază `0x378`. Registrul de date se găseşte la adresa de bază (`0x378`), registrul de status la adresa de bază + 1 (`0x379`), iar cel de control la adresa de bază + 2 (`0x37a`). Registrul de date este atât registru de intrare cât şi de ieşire.

Tratarea întreruperilor

Deşi există echipamente care pot fi controlate în întregime folosind porturi I/O sau zone de memorie speciale există şi situaţii în care acest lucru este insuficient. Principala problemă care mai trebuie tratată e faptul că anumite evenimente au loc la intervale de timp nedeterminate şi este ineficient ca procesorul (CPU) să interogheze repetat echipamentele despre starea lor. Modalitatea de rezolvare a acestei probleme o reprezintă **IRQ** (Interrupt ReQuest) prin care procesorul este anunţat de apariţia unui anumit eveniment extern.

Pentru ca IRQ-urile să fie utile trebuie să existe secvenţe de cod care să le trateze. Deoarece, în multe situaţii, numărul de întreruperi disponibile este limitat, un device driver trebuie să aibă un comportament cât mai civilizat relativ la ele: întreruperile trebuie să fie cerute înainte de a fi utilizate şi eliberate atunci când nu mai este nevoie de ele. În plus, în anumite situaţii, device driver-ele trebuie să folosească în comun o întrerupere sau să se sincronizeze cu întreruperile. Despre toate acestea vom discuta în continuare.

Locking

Atunci când trebuie să accesăm resurse partajate între o rutină de tratare a întreruperii (A) şi cod ce rulează în context proces sau într-o rutină de tratare a unei acţiuni amânabile (B), trebuie să folosim un mod special de sincronizare. În (A) trebuie sa folosim o primitivă de tip spinlock, iar în (B) trebuie să dezactivăm întreruperile **ŞI** să folosim o primitivă de tip spinlock. Dezactivarea întreruperilor nu este suficientă pentru că rutina de întrerupere poate rula pe un alt procesor decât cel pe care rulează (B).

Folosirea doar a unui spinlock poate duce la deadlock. Exemplul clasic de deadlock în acest caz este:

- rulăm în context proces, pe procesorul X, şi luăm lock-ul
- înainte de a lăsa lock-ul, se generează o întrerupere pe procesorul X
- rutina de tratare a întreruperii va încerca să ia şi ea lock-ul

Comunicaţia cu hardware-ul în Linux

În Linux, conceptul de porturi I/O este implementat pe toate platformele, chiar şi pe cele care folosesc un singur spaţiu de adresă.

Alocarea porturilor I/O

Înainte de a putea lucra cu porturile I/O, trebuie să ne asigurăm ca avem acces exclusiv la ele. Pentru a obţine porturile dorite, se foloseşte funcţia `request_region`:

```
#include <linux/ioport.h>

struct resource *request_region(unsigned long first, unsigned long n, const char *name);
```

Parametrul `first` specifica adresa de bază pentru dispozitiv, iar `n` numărul de porturi dorite. Adresa de bază este începutul zonei contigue de adrese (sau de porturi I/O) asociate dispozitivului şi trebuie să fie unică pentru fiecare dispozitiv. Parametrul `name` reprezintă numele dispozitivului.

Pentru eliberarea porturilor rezervate se foloseşte funcţia `release_region`:

```
void release_region(unsigned long start, unsigned long n);
```

Spre exemplu, portul serial `COM1` are adresa de bază 0x3F8 şi deţine 8 porturi. Secvenţa de cod pentru alocarea porturilor asociate acestuia este următoarea:

```
#include <linux/ioport.h>
#define MY_BASEPORT 0x3F8
#define MY_NR_PORTS 8

if (! request_region(MY_BASEPORT, MY_NR_PORTS, "com1")) {
    /* handle error */
    return -ENODEV;
}
```

iar cea pentru eliberare:

```
release_region(MY_BASEPORT, MY_NR_PORTS);
```

De cele mai multe ori, alocarea porturilor se realizează la iniţializarea driver-ului, în funcţia `init_module`, iar eliberarea acestora la deiniţializare, în funcţia `cleanup_module`.

Toate porturile alocate apar în `/proc/ioports`:

```
$ cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-005f : timer
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
0378-037a : parport0
037b-037f : parport0
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial
...
```

Operaţii de scriere şi citire a porturilor I/O

După ce un driver a obţinut intervalul de porturi I/O dorite, trebuie să realizeze operaţii de citire sau scriere pe aceste porturi. Întrucât porturile fizice sunt diferenţiate după numărul de biţi (8, 16 sau 32 de biţi), există diferite funcţii de acces a porturilor în funcţie de dimensiunea lor. În `asm/io.h` sunt definite următoarele funcţii de acces a porturilor:

- `unsigned inb(int port)`, citeşte porturi de dimensiune un octet (8 biţi)
- `void outb(unsigned char byte, int port)`, scrie porturi de dimensiune un octet (8 biţi)
- `unsigned inw(int port)`, citeşte porturi de dimensiune doi octeţi (16 biţi)
- `void outw(unsigned short word, int port)`, scrie porturi de dimensiune doi octeţi (16 biţi)
- `unsigned inl(int port)`, citeşte porturi de dimensiune patru octeţi (32 biţi)
- `void outl(unsigned long word, int port)`, scrie porturi de dimensiune patru octeţi (32 biţi)

Argumentul port specifică adresa portului de unde se citeşte sau se scrie, iar tipul său este dependent de platformă (poate fi `unsigned long` sau `unsigned short`).

Anumite platforme pot avea probleme atunci când procesorul încearcă să transfere date prea rapid către şi de la dispozitiv. Soluţia este inserarea unei întârzieri după fiecare instrucţiune de I/O, în cazul în care urmează o altă instrucţiune de acelaşi tip. În cazul în care dispozitivul pierde date, se pot folosi funcţii care introduc această întârziere; numele acestora este similar cu cele descrise mai sus, cu deosebirea că se termină în `_p`: `inb_p`, `outb_p`, etc.

Spre exemplu, următoarea secvenţă scrie un octet pe portul serial `COM1` şi apoi îl citeşte:

```
#include <asm/io.h>
#define MY_BASEPORT 0x3F8

unsigned char value = 0xFF;
outb(value, MY_BASEPORT);
value = inb(MY_BASEPORT);
```

Deşi funcţiile descrise mai sus sunt definite pentru device drivere, ele pot fi folosite şi din user-space, prin includerea header-ului . Pentru a putea fi folosite, vor trebui apelate mai întâi funcţiile `ioperm` sau `iopl` pentru obţinerea permisiunii de a realiza operaţii cu porturile. Funcţia `ioperm` obţine permisiunea pentru porturi individuale, în timp ce `iopl` pentru întregul spaţiu de adrese I/O. Pentru a putea folosi aceste funcţii utilizatorul trebuie să fie `root`.

Următoarea secvenţă obţine permisiunea pentru primele 3 porturi ale portului serial, şi apoi le eliberează:

```
#include <sys/io.h>
#define MY_BASEPORT 0x3F8

if (ioperm(MY_BASEPORT, 3, 1)) {
    /* handle error */
}
if (ioperm(MY_BASEPORT, 3, 0)) {
    /* handle error */
}
```

Al treilea parametru al funcţiei `ioperm` este 1 pentru obţinerea permisiunii şi 0 pentru eliberare.

Întreruperi în Linux

Obţinerea unei întreruperi

La fel ca şi în cazul celorlalte resurse, un driver trebuie să obţină accesul la o linie de întreruperi înainte de a o putea utiliza şi să o elibereze la finalul execuţiei.

În Linux, cererea de obţinere şi respectiv eliberare a unei întreruperi se face cu ajutorul funcţiilor `request_irq` şi `free_irq`:

```
#include <linux/interrupt.h>

int request_irq(unsigned int irq_no,
                irqreturn_t (*handler)(int irq_no, void *dev_id),
                unsigned long flags, const char *dev_name, void *dev_id);

void free_irq(unsigned int irq_no, void *dev_id);
```

Se observă că pentru obţinerea unei întreruperi utilizatorul trebuie să specifice **numărul întreruperii** (`irq_no`), un **handler** ce va fi chemat în momentul generării întreruperii (`handler`), **flag**-uri ce vor instrui kernelul despre comportamentul dorit (`flags`), **numele dispozitivului** ce foloseşte această întrerupere (`dev_name`), şi un pointer ce poate fi setat de către utilizator la orice valoare, şi care nu are semnificaţie globală (`dev_id`). De cele mai multe ori, `dev_id` va fi setat la pointerul către datele private ale dispozitivului. În schimb, la eliberarea întreruperii, utilizatorul trebuie să paseze aceeaşi valoare a pointerului (`dev_id`), împreună cu numărul întreruperii (`irq_no`). Numele dispozitivului (`dev_name`) este folosit pentru afişarea de statistici în `/proc/interrupts`.

Valoarea pe care o întoarce `request_irq` este 0 în cazul în care înregistrarea s-a efectuat cu succes sau un cod de eroare negativ care indică motivul eşecului. O valoare uzuală este `-EBUSY` care este întoarsă atunci când întreruperea este ocupată deja de un alt echipament.

Flag-urile ce pot fi pasate la obţinerea unei întreruperi sunt `IRQF_SHARED`, `IRQF_SAMPLE_RANDOM` şi `IRQF_DISABLED`:

- `IRQF_SHARED` anunţă kernelul că întreruperea poate fi partajată cu alte dispozitive. Dacă acest flag nu este setat, atunci, dacă există deja un handler asociat cu întreruperea cerută, cererea de obţinere a unei întreruperi va eşua. O întrerupere partajată este tratată prin execuţia tuturor rutinelor înregistrate. Această abordare duce la o situaţie interesantă: cum îşi poate da seama un device driver dacă rutina de tratare a întreruperii a fost activată de o întrerupere generată de dispozitivul pe care îl gestionează? Răspunsul este simplu: toate dispozitivele care oferă suport pentru întreruperi au asociate un registru de stare, care poate fi interogat în rutina de tratare pentru a afla dacă întreruperea a fost sau nu generată de dispozitiv (în cazul portului serial, acest registru de stare este `IIR`). La cererea unei întreruperi partajate cu `request_irq`, argumentul `dev_id` trebuie să fie unic în kernel; poate fi setat la un pointer către datele private ale modulului, dar nu poate fi `NULL`.
- `IRQF_SAMPLE_RANDOM` anunţă kernelul că poate folosi întreruperea ca o sursă de evenimente asincrone ce pot fi folosite pentru a genera numere aleatoare. Din motive de securitate, utilizatorul nu trebuie să folosească acest flag decât dacă întreruperile au o rata de generare variabilă şi nu pot fi influenţate de către un atacator din exterior. Exemple de dispozitive a căror întreruperi sunt potrivite pentru acest scop: discul, mouse-ul, tastatura. Exemple de dispozitive a căror întreruperi nu pot fi folosite pentru acest scop: placa de reţea, ceasul.
- `IRQF_DISABLED` anunţă kernelul că handler-ul se va executa cu toate întreruperile dezactivate pe procesorul local; un device driver nu trebuie să folosească acest flag, el este folosit doar în cazuri speciale - de exemplu la tratarea întreruperii de ceas.

Obţinerea întreruperii se poate realiza fie la iniţializarea driver-ului, în funcţia `init_module`, fie atunci când dispozitivul este deschis prima dată, în funcţia `open`.

Următorul exemplu realizează aceste operaţii pentru portul serial `COM1`:

```
#include <linux/interrupt.h>

#define MY_BASEPORT 0x3F8
#define MY_IRQ 4

struct my_device_data *my_data;
int err;

if ((err = request_irq(MY_IRQ, my_handler, IRQF_SAMPLE_RANDOM | IRQF_SHARED,
                       "com1", my_data))) {
    /* handle error*/
    return err;
}
```

După cum se poate observa, IRQ-ul pentru portul serial `COM1` este 4, care este folosit în mod partajat (`IRQF_SHARED`), iar kernel-ul este informat că poate folosi întreruperile apărute ca sursă de entropie (`IRQF_SAMPLE_RANDOM`).

Atenţie! La cererea unei întreruperi partajate (`IRQF_SHARED`) cu `request_irq`, argumentul `dev_id` nu poate fi `NULL`.

Pentru eliberarea întreruperii asociate portului serial se va executa următoarea secvenţă:

```
free_irq(MY_IRQ, my_data);
```

In funcţia de iniţializare, `init_module` sau în funcţia de deschidere a dispozitivului, `open`, trebuie activate întreruperile pentru a putea fi generate de către dispozitiv. Această operaţie este dependentă de dispozitivul folosit, dar de cele mai multe ori presupune setarea unui bit din registrul de control.

Pentru portul serial trebuie realizate două operaţii pentru activarea întreruperilor:

- se activează toate întreruperile prin setarea bitului 3 (`Aux Output 2`) în registrul `MCR`
- se activează întreruperea dorită (`RDAI`, `THREI`) prin setarea bitului corespunzător în registrul `IER`.

În exemplul de mai jos se activează întreruperea `RDAI` pentru portul `COM1`:

```
#include <asm/io.h>
#define MY_BASEPORT 0x3F8

outb(0x08, MY_BASEPORT+4);
outb(0x01, MY_BASEPORT+1);
```

Implementarea rutinei de tratare a întreruperii

Să examinăm acum signatura funcţiei de tratare a întreruperii:

```
irqreturn_t (*handler)(int irq_no, void *dev_id);
```

Funcţia primeşte ca parametri numărul întreruperii pe care rutina o tratează şi pointer-ul trimis la cererea de obţinere a întreruperii. Rutina de tratare a întreruperii trebuie să întoarcă o valoare cu tipul `irqreturn_t`. Pentru versiunea curentă de kernel există doar două valori valide: `IRQ_NONE` şi `IRQ_HANDLED`. Device driverul trebuie să întoarcă `IRQ_NONE` dacă observă că întreruperea nu a fost generată de dispozitivul pe care îl comandă. În caz contrar, device driverul trebuie să întoarcă `IRQ_HANDLED`.

Un handler de întrerupere va avea următoarea structură:

```
irqreturn_t my_handler(int irq_no, void *dev_id)
{
    struct my_device_data *my_data = (struct my_device_data *) dev_id;

    /* if interrupt is not for this device (shared interrupts) */
        /* return IRQ_NONE;*/
    /* clear interrupt-pending bit */
    /* read from device or write to device*/

    return IRQ_HANDLED;
}
```

De obicei, primul lucru executat în rutina de tratare a întreruperii este să se determine dacă întreruperea a fost generată de dispozitivul comandat de driver. Pentru aceasta, de obicei se citesc informaţii din registrul de control al dispozitivului, care indică daca acesta a generat întreruperea. Al doilea lucru constă în resetarea bitului `interrupt-pending` pe dispozitivul fizic; cele mai multe dispozitive nu vor mai genera întreruperi până când acest bit nu a fost resetat. Acest pas depinde doar de dispozitiv, şi poate şi lipsi (cum e cazul portului paralel, care nu are un astfel de bit). Portul serial are un astfel de bit: bitul 0 din registrul `IIR`.

Locking

Deoarece rutinele de tratare a întreruperilor se execută în context întrerupere, acţiunile care se pot efectua sunt limitate: nu se poate accesa memoria din user-space, nu se pot apela funcţii blocante, nu se poate face sincronizare doar cu spinlock-uri deoarece acest lucru ar duce la deadlock în cazul în care spinlock-ul este deja obţinut de către un proces care a fost întrerupt.

Totuşi, există cazuri în care device driverele trebuie să se sincronizeze cu întreruperile. În aceste situaţii este necesară dezactivarea întreruperii cu care dorim să ne sincronizăm (nu se pot dezactiva întreruperile partajate) cu ajutorul funcţiilor `disable_irq`, `disable_irq_nosync` şi `enable_irq`. Dezactivarea are loc la nivelul tuturor procesoarelor din sistem şi apelurile pot să fie imbricate: dacă se apelează de două ori `disable_irq` vor fi necesare tot atâtea apeluri `enable_irq` pentru activarea ei. Diferenţa dintre `disable_irq` şi `disable_irq_nosync` e că prima din ele va aştepta terminarea handler-elor aflate în execuţie. Din această cauză `disable_irq_nosync` este în general mai rapidă.

Spre exemplu, următoarea secvenţă dezactivează şi apoi activează întreruperea pentru portul serial `COM1`:

```
#define MY_IRQ 4

disable_irq(MY_IRQ);
enable_irq(MY_IRQ);
```

Este posibilă şi dezactivarea tuturor întreruperilor pentru procesorul curent. Dezactivarea tuturor întreruperilor de către device drivere pentru sincronizare este total neadecvată; nu trebuie folosită niciodată această abordare. Funcţiile care dezactivează / reactivează întreruperile pe procesorul local sunt `local_irq_disable` şi `local_irq_enable`. În general nu puteţi folosi aceste funcţii pentru sincronizare datorită problemelor ce apar în sisteme multiprocesor.

Pentru sincronizare, uneori este necesar să se folosească un spinlock şi să se dezactiveze şi întreruperile. În acest scop, există funcţii de locking care dezactivează, respectiv reactivează întreruperile: `spin_lock_irqsave`, `spin_unlock_irqrestore`, `spin_lock_irq` şi `spin_unlock_irq`.

```
#include <linux/spinlock.h>

void spin_lock_irqsave(spinlock_t *lock, unsigned long flags);
void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags);

void spin_lock_irq(spinlock_t *lock);
void spin_unlock_irq(spinlock_t *lock);
```

Funcţia `spin_lock_irqsave` dezactivează întreruperile pentru procesorul local înainte de a obţine spinlock-ul; starea anterioara a întreruperilor este salvată în `flags`. În cazul în care sunteţi siguri că întreruperile pe procesorul curent nu au fost deja dezactivate de altcineva (deci sunteţi siguri că trebuie să activaţi întreruperile când eliberaţi spinlock-ul), puteţi folosi funcţia `spin_lock_irq`.

Pentru spinlock-urile de tip read / write există funcţii similare: `read_lock_irqsave`, `read_unlock_irqrestore`, `read_lock_irq`, `read_unlock_irq`, `write_lock_irqsave`, `write_unlock_irqrestore`, `write_lock_irq` şi `write_unlock_irq`.

Astfel, pentru a folosi o resursă partajată atât în context proces cât şi în rutina de tratare a întreruperii, se vor folosi funcţiile descrise mai sus în context proces astfel:

```
unsigned long flags;
DEFINE_SPINLOCK(lock);

spin_lock_irqsave(&lock, flags);
/* critical region ? access shared resource */
spin_unlock_irqrestore(&lock, flags);
```

În rutina de tratare a întreruperii, se pot folosi funcţiile `spin_lock` şi `spin_unlock` pentru acces la resursa partajată.

Statistici despre întreruperi

Statistici despre întreruperile din sistem pot fi găsite în `/proc/interrupts` sau `/proc/stat`. În fişierul `/proc/interrupts` apar numai întreruperile din sistem care au câte un handler asociat:

```
 # cat /proc/interrupts
                CPU0
  0:         7514294       IO-APIC-edge   timer
  1:            4528       IO-APIC-edge   i8042
  6:               2       IO-APIC-edge   floppy
  8:               1       IO-APIC-edge   rtc
  9:               0       IO-APIC-level  acpi
 12:            2301       IO-APIC-edge   i8042
 15:              41       IO-APIC-edge   ide1
 16:            3230       IO-APIC-level  ioc0
 17:            1016       IO-APIC-level  vmxnet ether
NMI:               0
LOC:         7229438
ERR:               0
MIS:               0
```

În prima coloană se specifică numărul IRQ-ului asociat întreruperii; în coloanele următoare se afişează numărul de întreruperi care au fost generate pentru fiecare procesor din sistem; ultimele două coloane oferă informaţii despre controller-ul de întreruperi şi numele dispozitivului care a înregistrat un handler pentru acea întrerupere.

Fişierul `/proc/stat` oferă informaţii despre activitatea sistemului, inclusiv numărul de întreruperi generate de la ultima boot-are a sistemului:

```
# cat /proc/stat | grep intr
intr 7765626 7754228 4620 0 0 0 0 2 0 1 0 0 0 2377 0 0 41 3259 1098 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fiecare linie din fişierul `/proc/stat` începe cu un text care specifică semnificaţia informaţiilor de pe linie; pentru informaţii despre întreruperi, acest text este `intr`. Primul număr de pe linie reprezintă numărul total de întreruperi, iar celelalte numere reprezintă numărul de întreruperi pentru fiecare IRQ, începând de la 0. Valoarea contorului reprezintă numărul de întreruperi pentru toate procesoarele din sistem.

Comunicaţia cu hardware-ul în Windows

În Windows, HAL (Hardware Abstraction Layer) pune la dispoziţie funcţii pentru accesarea porturilor I/O, atât pentru platformele care suporta adrese mapate în memorie cât şi pentru cele care suportă spaţiu separat de adrese I/O.

Operaţii de scriere şi citire a porturilor I/O

Întrucât porturile fizice sunt diferenţiate după numărul de biţi (8, 16 sau 32 de biţi), există diferite funcţii de acces a porturilor în funcţie de dimensiunea lor:

- `UCHAR READ_PORT_UCHAR(PUCHAR Port)`, citeşte porturi de dimensiune un octet (8 biţi)
- `VOID WRITE_PORT_UCHAR(PUCHAR Port, UCHAR Value)`, scrie porturi de dimensiune un octet (8 biţi)
- `USHORT READ_PORT_USHORT(PUSHORT Port)`, citeşte porturi de dimensiune doi octeţi (16 biţi)
- `VOID WRITE_PORT_USHORT(PUSHORT Port, USHORT Value)`, scrie porturi de dimensiune doi octeţi (16 biţi)
- `ULONG READ_PORT_ULONG(PULONG Port)`, citeşte porturi de dimensiune patru octeţi (32 biţi)
- `VOID WRITE_PORT_ULONG(PULONG Port, ULONG Value)`, scrie porturi de dimensiune patru octeţi (32 biţi)

Spre exemplu, următoarea secvenţă scrie un octet pe portul serial `COM1` şi apoi îl citeşte:

```
#include <ntddk.h>
#define MY_BASEPORT 0x3F8

UCHAR value = 0xFF;
WRITE_PORT_UCHAR ((PUCHAR) MY_BASEPORT, value);
value = READ_PORT_UCHAR ((PUCHAR) MY_BASEPORT);
```

Întreruperi în Windows

Obţinerea unei întreruperi

La fel ca şi în Linux, device driverele trebuie să facă o cerere de obţinere a întreruperii înainte de folosire. Dacă cererea este acceptată, întreruperea va avea asociată o rutină de tratare. Atunci când device driverul nu mai foloseşte întreruperea, aceasta trebuie eliberată.

Pentru obţinerea întreruperii se foloseşte funcţia `IoConnectInterrupt`:

```
NTSTATUS
  IoConnectInterrupt(
    OUT PKINTERRUPT  *InterruptObject,
    IN PKSERVICE_ROUTINE  ServiceRoutine,
    IN PVOID  ServiceContext,
    IN PKSPIN_LOCK  SpinLock  OPTIONAL,
    IN ULONG  Vector,
    IN KIRQL  Irql,
    IN KIRQL  SynchronizeIrql,
    IN KINTERRUPT_MODE    InterruptMode,
    IN BOOLEAN  ShareVector,
    IN KAFFINITY  ProcessorEnableMask,
    IN BOOLEAN  FloatingSave
  );
```

După cum se observă, obţinerea unei întreruperi în Windows este ceva mai complicată. Pentru a simplifica lucrurile vom discuta parametrii funcţiilor începând cu cei esenţiali.

`InterruptObject` reprezintă întreruperea după ce aceasta a fost obţinută (conectată în terminologia Windows). Device driverul va folosi acest obiect pentru a elibera (deconecta) întreruperea sau pentru a se sincroniza cu rutina de tratare a întreruperii. Acest obiect va fi iniţializat la conectarea întreruperii, în cazul în care conectarea este posibilă.

Un alt parametru important este `ServiceRoutine` şi reprezintă rutina de tratare ce va fi asociată cu întreruperea. Parametrul `ServiceContext` este un parametru ce va fi pasat rutinei de tratare a întreruperii. De obicei, este un pointer către datele private ale driver-ului.

Un alt parametru esenţial este `Vector`. Acest parametru specifică de fapt întreruperea ce se doreşte a fi obţinută, dar nu este numărul întreruperii ca în Linux (sau cel puţin nu putem să ne bazăm pe acest lucru). Vectorul asociat cu o întrerupere poate fi obţinut prin interogarea `DeviceObject`-ului dacă dispozitivul a fost descoperit de `Plug&Play Manager`. Pentru a menţine însă lucrurile cât de cât simple, în cadrul laboratoarelor şi temelor nu vom folosi dispozitive ce pot fi descoperite de `Plug&Play Manager`, prin urmare această opţiune nu este disponibilă. Din fericire există o alternativă: obţinerea vectorului unei întreruperi cu ajutorul funcţiei `HalGetInterruptVector` [1] a cărei signatură este prezentată mai jos:

```
ULONG
  HalGetInterruptVector(
    IN INTERFACE_TYPE  InterfaceType,
    IN ULONG  BusNumber,
    IN ULONG  BusInterruptLevel,
    IN ULONG  BusInterruptVector,
    OUT PKIRQL  Irql,
    OUT PKAFFINITY  Affinity
  );
```

Funcţia primeşte ca parametri tipul magistralei la care este conectat dispozitivul (`Isa`, `Pci`, `Internal`) (`InterfaceType`), numărul magistralei (`BusNumber`), numărul întreruperii folosite de dispozitiv (`BusInterruptLevel`), numărului vectorului pentru magistrala la care este conectat dispozitivul (`BusInterruptVector`). Va întoarce ca valoare vectorul întreruperii, care poate fi folosit la conectarea întreruperii. De asemenea, va întoarce nivelul de întrerupere (`Irql`) la care va rula handler-ul şi setările de afinitate necesare (`Affinity`). Aceşti parametri vor fi apoi pasaţi direct funcţiei de conectare a dispozitivului (parametrii `Vector`, `Irql`, `ProcessorEnableMask`).

Revenind la funcţia de conectare a întreruperii, parametrul `SynchronizeIrql` specifică nivelul de întrerupere la care va rula rutina de tratare a întreruperii. Daca rutina de tratare a întreruperii serveşte mai multe întreruperi, aceasta trebuie să fie valoarea maximă dintre acestea. În caz contrar, `SynchronizeIrql` şi `Irql` vor fi identice.

Parametrul `InterruptMode` poate fi `LevelSensitive` sau `Latched`. Valorile `Latched` şi `LevelSensitive` se refera la intreruperi edge-triggered, respectiv level-triggered.
Parametrul `ShareVector` specifică dacă vectorul de întreruperi poate fi partajat.

Pentru eliberarea întreruperii se foloseşte funcţia `IoDisconnectInterrupt`, care primeşte ca parametru obiectul asociat întreruperii obţinut la conectarea acesteia.

Pentru a sumariza modul în care se conectează o întrerupere, prezentăm mai jos modalitatea de conectare a întreruperii pentru portul serial `COM1`:

```
#define MY_IRQ 4

struct my_device_data {
    PKINTERRUPT pIntObj;
};

KIRQL kIrql;
KAFFINITY kAffinity;
struct my_device_data *my_data;
NTSTATUS status;
ULONG kVector;

kVector = HalGetInterruptVector(Internal, 0, MY_IRQ, 0, &kIrql, &kAffinity);

status =  IoConnectInterrupt(&my_data->pIntObj, myIsr, my_data, NULL, kVector, kIrql,
                             kIrql, Latched, TRUE, kAffinity, FALSE );
if (!NT_SUCCESS(status)) {
    /* cererea de obtinere a intreruperii a fost respinsa */
    return status;
}
```

şi modalitatea de deconectare:

```
IoDisconnectInterrupt(my_data->pIntObj);
```

In funcţia de iniţializare, `DriverEntry` sau în funcţia de deschidere a dispozitivului, `Open`, trebuiesc activate întreruperile pentru a putea fi primite de către dispozitiv. Pentru portul serial trebuiesc realizate două operaţii pentru activarea întreruperilor:

- se activează toate întreruperile prin setarea bitului 3 (`Aux Output 2`) în registrul `MCR`

- se activează întreruperea dorită (RDAI, THREI) prin setarea bitului corespunzător în registrul IER.

În exemplul de mai jos se activează întreruperea RDAI pentru portul COM1:

```
#define MY_BASEPORT 0x3F8

WRITE_PORT_UCHAR ((PUCHAR) (MY_BASEPORT + 4), 0x08);
WRITE_PORT_UCHAR ((PUCHAR) (MY_BASEPORT + 1), 0x01);
```

Implementarea rutinei de tratare a întreruperii

Signatura rutinei de tratare a întreruperii este:

```
BOOLEAN
  InterruptService(
    IN PKINTERRUPT  Interrupt,
    IN PVOID  ServiceContext
  );
```

Funcţia primeşte la apelare obiectul ce descrie întreruperea (Interrupt) şi un parametru ServiceContext, ce a fost pasat kernelului la conectarea întreruperii (de obicei, reprezintă datele private ale dispozitivului).

Rutina de tratare trebuie să întoarcă FALSE în cazul în care a determinat că întreruperea nu aparţine dispozitivului gestionat de device driver sau TRUE în caz contrar. Pentru a determina dacă întreruperea a fost generată de dispozitivul comandat de driver, de obicei se citesc informaţii din registrul de control al dispozitvului (pentru portul serial, acest registru este IIR).

O rutină de tratare a unei întreruperi va avea următoarea structură:

```
BOOLEAN myIsr(PKINTERRUPT pIntObj, PVOID pServiceContext ) {
    /* if interrupt not for this device*/
        /* return FALSE; */
    /* handle interrupt */
    return TRUE;
}
```

Locking

Ca şi în Linux, există situaţii în care device driverele trebuie să se sincronizeze cu rutina de întrerupere. Cum nu se pot folosi spinlock-uri pentru aceasta (altfel există pericolul de deadlock-uri), DDK-ul pune la dispoziţia utilizatorilor funcţia KeSynchronizeExecution:

```
BOOLEAN
 KeSynchronizeExecution(
    IN PKINTERRUPT    Interrupt,
    IN PKSYNCHRONIZE_ROUTINE  SynchronizeRoutine,
    IN PVOID  SynchronizeContext
 );
```

Această funcţie va ridica nivelul de întrerupere la nivelul întreruperii dat de Interrupt, şi va rula funcţia SynchronizeRoutine cu parametrul SynchronizeContext. Se garantează faptul că atât timp cât rulează funcţia, rutina de tratare a întreruperii nu rulează. Valoarea întoarsă de KeSynchronizeExecution este valoarea întoarsă de funcţia SynchronizeRoutine.

Astfel, pentru a accesa o resursă atât din rutina de tratare a întreruperii, cât şi din context proces, trebuie definită următoarea funcţie:

```
BOOLEAN mySynchRoutine(PVOID Context) {
    /* access shared resource */
}
```

care va fi apelată din context proces astfel:

```
    BOOLEAN result;
    my_data->temporary_value = 1;
    result = KeSynchronizeExecution(my_data->pIntObj, mySynchRoutine, my_data);
```

Rutina de tratare a întreruperii poate accesa resursa partajată fără alte mecanisme de sincronizare:

```
BOOLEAN myIsr(PKINTERRUPT pIntObj, PVOID pServiceContext ) {
    /* access shared resource */
    my_data->shared_value = my_data->temporary_value;
}
```

Quiz

Pentru auto-evaluare (de preferat înainte de laborator) răspundeţi la întrebările de aici.

Exerciţii

- Folosiţi arhiva de sarcini a laboratorului.
- Punctaj total: **11 puncte**

Linux

- Folosiţi directorul `lin/` din arhiva de sarcini a laboratorului.
- Punctaj total: **5,5 puncte**

- Creaţi un driver pentru a intercepta tastele apăsate (keylogger). Driver-ul va intercepta IRQ-ul destinat controller-ului de tastatură şi va inspecta codurile tastelor primite, stocându-le într-un buffer.

1. (**1 punct**) Examinaţi codul din scheletul de laborator.
   - Codul care alocă porturile I/O (`STATUS_REG` şi `DATA_REG`) este comentat pentru că ar returna o eroare. Care ar fi cauza erorii?
   - **Hints:**
     - Listaţi conţinutul fişierului `/proc/ioports`. Ce observaţi?
     - Citiţi secţiunea Alocarea porturilor I/O din laborator.
2. (**1 punct**) Scrieţi rutina de tratare a întreruperii şi instalaţi-o pentru IRQ-ul controller-ului de tastatură.
   - **Important:** rutina trebuie să lase întreruperea să fie procesată de handler-ul original; în caz contrar, veţi pierde controlul tastaturii în maşina virtuală.
   - Verificati dacă rutina de tratare a întreruperii a fost înregistrată în `/proc/interrupts`.
   - Afişaţi un mesaj în rutina de tratare a întreruperii pentru a verifica dacă este apelată.
   - **Hints:**
     - Folosiţi macro-ul `I8042_KBD_IRQ`.
     - Rutina de tratare a întreruperii trebuie înregistrată cu `IRQ_SHARED`.
     - Returnaţi întotdeauna `IRQ_NONE` în handler-ul de întrerupere pentru a nu cauza conflicte cu driverul de tastatură.
     - Dacă înregistrarea întreruperii eşuează nu uitaţi să deînregistraţi regiunea, folosiţi label-ul `out2`.
     - Cititi secţiunile Obţinerea unei întreruperi, Implementarea rutinei de tratare a întreruperii şi Statistici despre întreruperi
3. (**1,5 puncte**) Scrieţi tastele apăsate în buffer-ul din cadrul structurii asociate dispozitivului.
   - Implementaţi funcţia pentru citirea registrului `DATA` al controller-ului de tastatură. Ea nu trebuie decât să returneze valoarea registrului (funcţia este deja prezentă în schelet ? `i8042_read_data`).
   - Afişaţi conţinutul registrului `DATA` în cadrul handler-ului de întrerupere.
   - Interpretaţi registrul `DATA` pentru a vedea dacă este o apăsare sau o ridicare de tastă şi care este tasta respectivă.
   - Dacă este o apăsare de tastă, adăugaţi caracterul corespunzător la sfârşitul buffer-ului asociat dispozitivului; când buffer-ul este plin, ignoraţi caracterul.
   - **Hints:**
     - Adresele porturilor sunt definite în macro-urile din `so2_kbd.c`.
     - Folosiţi funcţiile `is_key_press` şi `get_ascii` definite în schelet pentru a interpreta registrul `DATA`. Ambele primesc ca parametru valoarea citită din registrul `DATA`.
     - Utilizaţi câmpul `size` şi macro-ul care defineşte dimensiunea buffer-ului.
     - Cititi sectiunea Operaţii de scriere şi citire a porturilor I/O din laborator.
4. (**1 punct**) Implementaţi funcţia `read` a dispozitivului de tip caracter pentru a citi buffer-ul.
   - Creaţi fişierul de tip caracter `/dev/so2_kbd` folosind utilitarul `mknod`.
   - Testaţi folosind comanda `cat /dev/so2_kbd`
   - **Hints:**
     - Majorul şi minorul dispozitivului sunt definiţi în macro-urile din `so2_kbd.c`.
     - Folosiţi `copy_to_user` pentru scrierea în buffer-ul din userspace.
     - Utilizaţi un spinlock pentru a proteja accesul la bufferul partajat.
     - Întrucât nu puteţi folosi `copy_to_user` în timp ce aveţi un spinlock luat, trebuie să folosiţi un buffer temporar
5. (**1 punct**) Curăţaţi buffer-ul dacă este tastat un cuvânt `MAGIC`.
   - În rutina de tratarea a întreruperii, potriviţi pe parcurs caractere din parolă. Când se detectează întreaga parolă, curătaţi buffer-ul.
   - Setaţi `size` la zero şi umpleţi cu 0 buffer-ul.
   - Folosiţi un spinlock pentru ca întreruperea să nu cureţe buffer-ul în timpul unei citiri pe device.
   - **Hints:**

- Folosiţi o variabilă pentru a contoriza câte caractere din parolă au fost tastate.
- La primirea unui caracter nou, verificaţi dacă acesta corespunde cu caracterul de pe poziţia curentă din parolă. În caz afirmativ, incrementaţi contorul; altfel, resetaţi counterul la 0.
- Folosiţi macroul `MAGIC`.
- Citiţi secţiunea Locking din laborator.
- Iniţializaţi spinlock-urile înainte de înregistrarea întreruperilor.
- Trebuie să folosiţi spinlock şi in `read` şi în rutina de tratare a întreruperilor. În `read` trebuie să folosiţi o variantă de lock/unlock care dezactivează întreruperile.

Windows

- Folosiţi directorul `win/` din arhiva de sarcini a laboratorului.
- **Pentru a putea lucra fără probleme cu porturile seriale, trebuie să faceţi disable pe dispozitivele COM1 şi COM2 în Device Manager.**
- Punctaj total: **5,5 puncte**

- Creaţi un driver simplu pentru portul serial al calculatorului. Driver-ul va fi folosit doar pentru crearea unei rutine de tratare a întreruperii (ISR) şi pentru afişarea unui mesaj în cadrul acesteia.

1. (**2 puncte**) Scrieţi rutina de tratare a întreruperilor şi instalaţi-o pentru COM1 şi COM2.
   - **Hints:**
     - IRQ-urile folosite de porturile seriale sunt definite de macro-urile `COM1_IRQ` şi `COM2_IRQ`
     - Nu uitaţi să deconectaţi întreruperea la descărcarea modulului
     - Cititi sectiunile Obţinerea unei întreruperi si Implementarea rutinei de tratare a întreruperii
     - Afişaţi un mesaj specific în rutina de tratare a întreruperii în care să includeţi adresa de bază a dispozitivului care a primit întreruperea.
2. (**1,5 puncte**) Activaţi întreruperile şi generaţi o întrerupere pentru COM1 şi COM2
   - Activaţi întreruperile pentru COM1 şi COM2
   - **Hints:**
     - Urmăriţi acest link şi aflaţi care sunt registrele de configurare pentru întreruperi.
     - Pentru generarea de întreruperi va trebui să activaţi întreruperile pentru porturile seriale şi să activaţi întreruperea `RDAI`
     - Pentru a activa întreruperile trebuie să setaţi bitul 3 `Aux Output 2` (`0x08`) în registrul `MCR` (`COM_BASE+UART16550_MCR`)
     - Pentru a activa întreruperea `RDAI` trebuie să setaţi bitul 0 `Enable Received Data Available Interrupt` (0x01) în registrul `IER` (`COM_BASE+UART16550_IER`)
   - Parametrii de comunicaţie sunt deja setaţi porturile COM1 şi COM2.
   - **Hints:**
     - Cu ce parametri de comuncaţie sunt setate porturile?
     - Ce face opţiunea `FIFO`?
   - Generaţi o întrerupere pentru porturile COM1 şi COM2
   - Pentru a genera o întrerupere scrieţi în funcţia de iniţializare a modulului caracterul `FIRST_CH` pe portul COM1
   - **Hints:**
     - Veţi primi o întrerupere pe portul COM2 (COM1 şi COM2 sunt legate printr-un pipe în maşina virtuală, astfel încât datele trimise pe un port se primesc pe celălalt)
     - Cititi sectiunea Operaţii de scriere şi citire a porturilor I/O din laborator
3. (**1 punct**) Implementaţi funcţia `write`, în care să scrieţi pe port caracterul primit.
   - Testaţi din `Cygwin` folosind `echo -n 'a' > \\\\.\\so2_com1`
   - Pentru a se genera noi întreruperi pentru portul serial, trebuie să citiţi registrul `IIR` (`COM_BASE+UART16550_IIR`) în intrerupere
   - Trebuie să testaţi că întreruperea este activată la intrarea în rutina de tratare a întreruperii.
   - Afişaţi valoarea citită în întrerupere
   - **Hints:**
     - Mai multe detalii despre registrul `IIR` (Interrupt Status Register) se găsesc aici.
     - Cititi sectiunea Operaţii de scriere şi citire a porturilor I/O din laborator
4. (**1 punct**) Păstraţi în structura asociată dispozitivului ultima valoare scrisa/citita în întrerupere.
   - Va trebui să folosiţi o variabilă temporară în care să salvaţi valoarea la `write`, după să o sincronizaţi cu cea partajată
   - Sincronizaţi accesul la variabilă din structura asociată dispozitivului
   - **Hints:**
     - Citiţi sectiunea Locking din laborator.

Soluţii

- Soluţii exerciţii laborator 6

Resurse utile

Port serial

1. Serial Port
2. Interfacing the Serial / RS232 Port

Port paralel

1. Documentaţie port paralel
2. Program de test pentru portul paralel (Linux)
3. Interfacing the Standard Parallel Port
4. Parallel Port Central
5. Resurse suplimentare port paralel

Controller tastatură

1. Intel 8042
2. Linux and the keyboard
3. drivers/input/serio/i8042.c
4. drivers/input/keyboard/atkbd.c

Linux

1. Linux Device Drivers, 3rd ed., Ch. 9 - Communicating with Hardware
2. Linux Device Drivers, 3rd ed., Ch. 10 - Interrupt Handling
3. Interrupt Handlers

Windows

1. The Windows 2000 Device Driver Book, Second Edition ? Chapter 8. Interrupt-Driven I/O
2. Programming the Microsoft Windows Driver Model, Second Edition - Chapter 7. Reading and Writing Data
3. Microsoft Windows Internals, 4th Ed - Chapter 3. System Mechanisms - Trap Dispatching, Chapter 9. - I/O System,
4. Servicing Interrupts

[1] HalGetInterruptVector ? deşi această funcţie este marcată ca obsolete în DDK, alternativa este mult prea complicată pentru scopurile didactice ale temelor şi laboratoarelor, astfel încât am preferat să prezentăm această abordare