Exercise sheet 12 – I/O

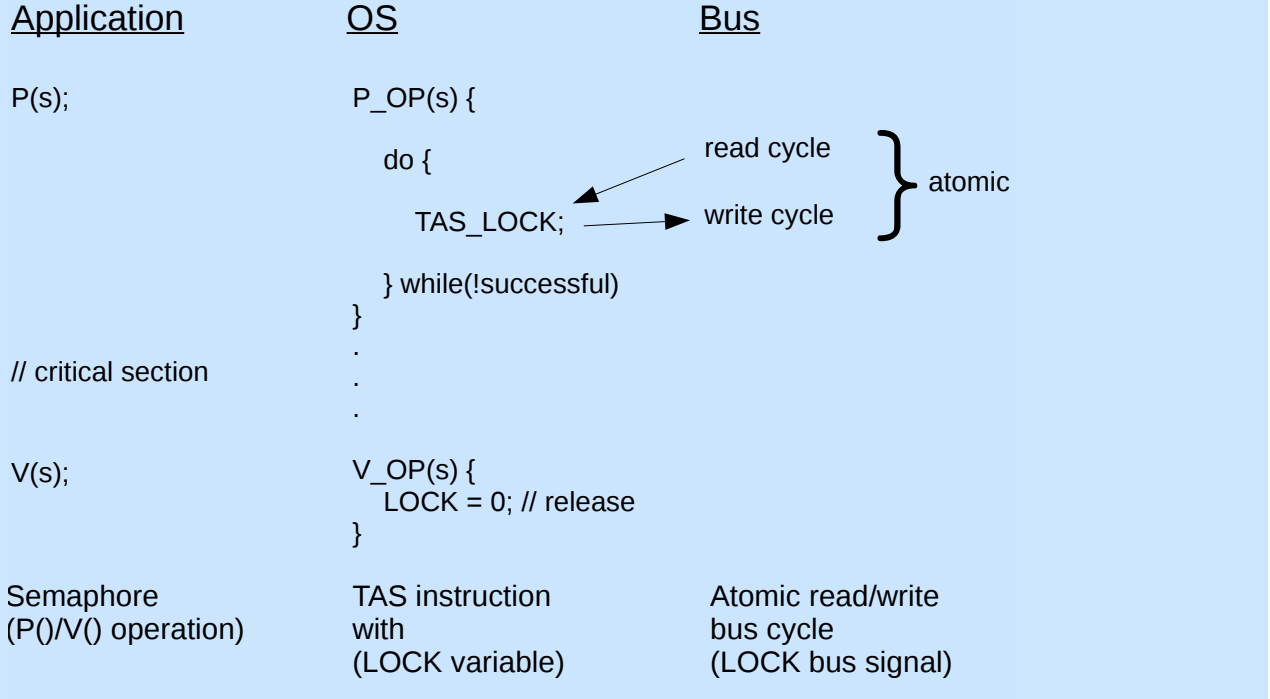
Goals:

- Programmed I/O
- Interrupt driven I/O
- DMA

Exercise 12.1: Synchronisation commands

(a) Describe a machine instruction for synchronisation and its interaction with the application, OS (operating system), and bus level. *Hint: You may consider the TAS command and semaphores, learned in the Operating Systems (Betriebssysteme) lecture as well as in this lecture. Use some pseudo-code to describe your ideas.*

Proposal for solution: Remember the *TAS* (test and set) command from lecture the Operating Systems? We learned that the OS is using it to provide semaphores for synchronization.



Exercise 12.2: Programmed I/O (single transfer) with busy wait (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the busy wait approach. Compare the lecture for that.

(a) Update the `RA_exercises` repository with `git pull`.

(b) In `RA_exercises/sheet_12/io_pc_prog_io_busy_wait/io_pc_prog_io_busy_wait.c` you will find a skeleton file.

(c) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```c
#include <stdlib.h>
#include <inttypes.h>

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
    uint16_t RBUF; //receive buffer register
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

#define BUFFER_SIZE (16)
uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

int main(void) {
    for(int i = 0; i < BUFFER_SIZE; ++i) {
        while((bool)(FSI.CSR & RRDY) == false) { //wait until FSI has provided next ch
            //busy wait (do nothing)
        }
        buffer[i] = (uint8_t)FSI.RBUF;
    }

    return EXIT_SUCCESS;
}
```

Exercise 12.3: Programmed I/O (single transfer) with polling (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the polling approach. Compare the lecture for that.

(a) In
`RA_exercises/sheet_12/io_pc_prog_io_polling/io_pc_prog_io_polling.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```c
#include <stdlib.h>
#include <inttypes.h>
#include <stdbool.h> //bool

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
    uint16_t RBUF; //receive buffer register
```

```
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

#define BUFFER_SIZE (16)
uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

int main(void) {
    for(int i = 0; i < BUFFER_SIZE; ++i) {

        while(true) {
            if((bool)(FSI.CSR & RRDY) == true) { //if the FSI has provided next charac
                buffer[i] = (uint8_t)FSI.RBUF;   //copy the received character
                break;                           //proceed with next character
            } else {
                //do something else...
            }
        }
    }

    return EXIT_SUCCESS;
}
```

Exercise 12.4: Interrupt driven I/O (single transfer) (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the interrupt control approach. Compare the lecture for that.

(a) In
`RA_exercises/sheet_12/io_pc_interrupt_io/io_pc_interrupt_io.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```
#include <stdlib.h>
#include <inttypes.h>

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
    uint16_t RBUF; //receive buffer register
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
```

```c
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

typedef void (*ISR_t)(void); //Function pointer for an ISR
//INTVECTOR: ISR_t is mapped to the memory position 0x000000C8
#define INTVECTOR (*((ISR_t*)(0x000000C8)))

#define BUFFER_SIZE (16)
volatile uint8_t counter = 0;
volatile uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

void ISR_serial_read(); //prototype

int main(void) {
    //Start transfer
    INTVECTOR = &ISR_serial_read; //set ISR for receive character
    FSI.CSR |= RIE; //enable RIE flag

    //do something else while transfer is in progress...

    return EXIT_SUCCESS;
}

//an interrupt is triggered if
//the hardware has provided a new character
void ISR_serial_read() {
    buffer[counter] = (uint8_t)FSI.RBUF;
    counter++;
    if (counter == BUFFER_SIZE){
        FSI.CSR &= ~RIE; //delete the RIE flag to stop the transfer
                         //~ is the bitwise inversion.

        //buffer full -> inform outer world (somehow)
    }
}
```

Exercise 12.5: DMA programming (pseudo C code)

Consider a system with the *F-Bus DMA disk* (FDD). You want to write data (some words) from the memory with the DMA approach to the disk. Compare the lecture for that.

(a) In
`RA_exercises/sheet_12/io_pc_dma/io_pc_dma.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to write 16 words (4 bytes per word) from the memory to the *F-Bus DMA disk* (FDD).

- Source (memory) starting address: 0x400000
- Target (disk) starting address: 0x4711

Hint: Source, destination, how much, GO!

Proposal for solution:

```c
#include <stdlib.h>
#include <stdbool.h>
#include <inttypes.h>

```

```c
typedef volatile struct { //FDD interface
    uint32_t CSR;  //control and status register
    uint32_t DARH; //disk address register HI
    uint32_t DARL; //disk address register LO
    uint32_t BAR;  //bus address register
    uint32_t BCR;  //byte count register
} FddStruct;

//FDD: FddStruct is mapped to the memory position 0xFF000010
#define FDD  (*((FddStruct*)(0xFF000010)))

#define GO    (0x01) //Mask for GO
#define IE    (0x40) //Mask for IE
#define WRITE (0x02) //Mask for WRITE
//more defines...

typedef void (*ISR_t)(void); //Function pointer for an ISR
//INTVECTOR: ISR_t is mapped to the memory position 0x00000108
#define INTVECTOR (*((ISR_t*)(0x00000108)))

void ISR() { //interrupt service routine for the end of the transfer
    //notify application that everything is transferred
}

int main(void) {
    //In principle: {source, destination, how much, GO}
    INTVECTOR = &ISR; //ISR address is set to INTVECTOR (address 0x00000108)

    //Configure DMA interface
    FDD.BAR  = 0x400000; //source memory address
    FDD.DARH = 0x0;      //destination LBA address (bit 32 to 47)
    FDD.DARL = 0x4711;   //destination LBA address (bit 0 to 31)
    FDD.BCR  = 0x40;     //how much: number of bytes
    FDD.CSR  = IE | WRITE | GO; //0x43

    //DMA transmits data now without CPU.
    //At the end there is an interrupt!

    return EXIT_SUCCESS;
}
```