

**Exercises and assignments for the course DIT632 Development of Embedded systems Part 3**

**Work package nr 6 : Processes and threads.**

**(12 p in total)**

**Exercise 6\_1 Test of threads (exerc 6\_1.c)**

**(1p)**

On the homepage in Canvas you can find a file *sortandfind\_20.c* . The code performs the following operations:

1. It creates an array of integer numbers with the given number of elements.
2. Fills such array with random integer numbers whose max value is given by the user.  
There is a dummy delay in this random generator part just to view the problem in question 5.
3. Sorts the array.
4. Looks for a specific value given by the user using binary search.

**Answer the following questions:**

1. How many threads (in total) are created during the execution of the program?
2. What are lines 16, 17, 20, 25 and 26 intended for?
3. How many parameters are passed to function runner?
4. How can function runner know the values for parameters that, such as max value, not are passed to it?
5. Test how the behavior of the program changes if line 26 is omitted. Explain why.

**To do :** Study the program and write your answer in a simple text document *exerc6\_1.txt* and view it for the TA and they will give you a pass code if ok. Hand it in together with the other files.

**Exercise 6\_2 More about threads**

**(exerc 6\_2.c)**

**(2p)**

Rewrite the application in *sortandfind\_20.c* so that each of the three main steps (filling the array / sorting the array and finally finding the value given by the user) are executed by 3 different threads and work as expected. Clarify the function of each thread with a `printf( )` stating which of the 3 steps the thread is running. Possible output could be as below.

```
C:\Embedded C-program\Win_thread>demo
Enter max value
200
Executing runner 1
Executing runner 2
Enter value to find
35
Executing runner 3
35 found at location 32.
```

**Exercise 6\_3 (exerc 6\_3.c)**

**Time based control program**

**(2p)**

In this exercise you should develop a draft for a program, possible to compile but not possible to run or test. **You should just show and explain the solution for the TA.**

You shall develop a time-based implementation of an automatic door. The door is a sliding door which when open signal activates opens slowly to full open (opened) and when close signal activates

closes slowly to closed. The open-door signal activates when someone is near the door. The close signal is activated by the control program when time to close the door.

The door will operate in the following manner:

- ✓ If someone approaches the door space the door will open.
- ✓ After the door had been open (in several seconds, controlled by a simple loop) the door should start to close.
- ✓ The door is closing very slowly, takes several seconds to close. If someone approaches the door space while the door is closing, it shall immediately reopen.
- ✓ The main structure of the program is described roughly by the following flowchart:

The implementation will be done entirely in the programming language C and in one single module (file).

### Control of the door unit.

The door is connected to the computer system and controlled via four different 8 bits registers all possible to read or write to depending on the register.

### About the registers for this task:

**Control register:** Address 0x0B00 (write operation).

By bit 0 you can start opening the door, it will open slowly so it will take some time before it is fully open. By bit 1 you can start closing the door, it will close slowly. Closing and opening of the door could be done anytime even if the door is in its opening or closing phase.

Bit 7-2, Not used.

Bit 1, CLOSE write 1 => Actuate "Close the Door"

Bit 0, OPEN write 1 => Actuate "Open the Door".

Bits will be cleared (set to 0) by the system after the operation started.

**Status register :** Address 0x0B10 ( read operation). Each bit is represented the status of the door or the sensor. For example, door is closed, door is opened. You can by reading the bits in this register check the actual status of the door.

Bit 7, CLOSING. 1 = Door is closing slowly.

Bit 6, OPENING. 1 = Door is opening slowly.

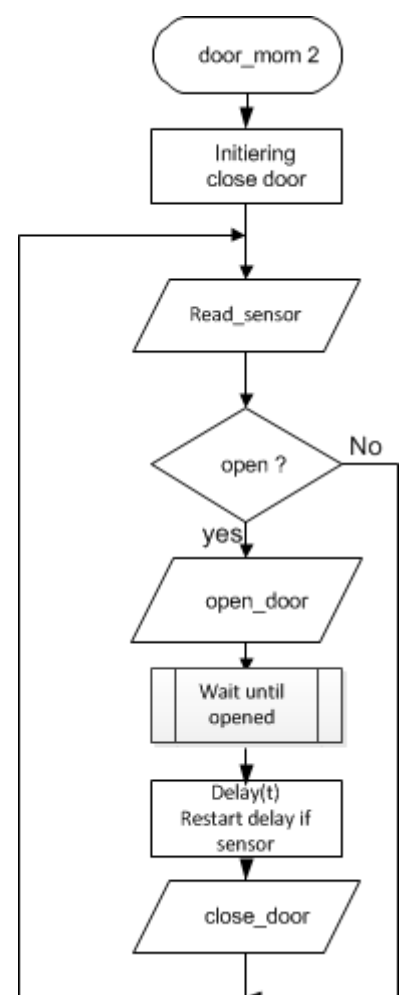
Bit 5, S2: Sense 2. 1 = Door is closed

Bit 4, S1: Sense 1. 1 = Door is wide open.

Bit 3, Sensor ; 1 = Sensor is activated, someone close to the door

Bit 2, Not of interest

Bit 1, Not of interest



**Note:** When reading a specific bit, you can't be sure that the other bits are 0.

**To do:**

Your task is to write a draft for a control program. Below you have a program head with some declarations. Try to make a draft for a control program that solve the problem with a structure as flowchart above views. The Delay() function can be implemented as a simple for-loop with a counter.

Compile the program to ensure the syntax. **You can't test the program.** The only thing you can do is to explain the program to the TA and they will respond if it seems to be a working solution or not. If rather good he/she will give you a pass code and you can hand in the solution.

```
#define ML13_Status    0x0B10
#define ML13_Control  0x0B00

void main () {
...

}
```

**Exercise 6\_4 (exerc 6\_4.c)**

**(3p)**

**Timer-controlled I/O management**

To control events in a program, you can use a "process" (in this case a thread) that manage a global system time variable tick. For example, the time could tick with a resolution of 0,1 seconds.

With help of this function you can read the system-time in ms :

```
#include <sys/time.h>

double get_time_ms(){
    struct timeval t;
    gettimeofday(&t, NULL);
    return (t.tv_sec + (t.tv_usec / 1000000.0)) * 1000.0;
}
```

If you let a thread function call this get\_time\_ms() and by help of this count a global program time counter variable *program\_time* with a resolution of 1 s.

Then it is possible to control all other events with help of the *program\_time*. For example, you can read a specific in-port every x seconds.

**To do:** Write a program with the following structure:

```
int program_time; // The global time, start value 0

int main(){

    // Start up the thread time_count.
    // Start up the thread read_inport.

    While ( program_time < 50){

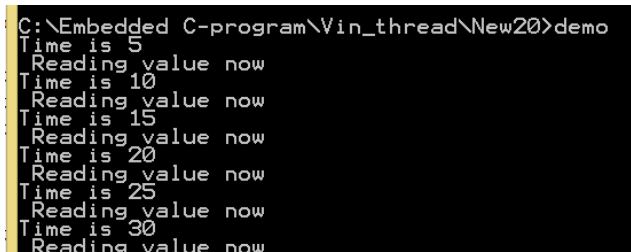
        //Print out system time every second.

    }
}
```

```
}  
// --- End of main thread -----  
  
// ----- Tread functions --  
//-----  
void *time_count( ....) {  
  
while ( program_time < 50){  
    // Check system-time ( get_time_ms())  
    // Increase program_time by one every second.  
}  
// exit thread;  
} //End-----  
  
void *read_inport(...){  
    while (program_time<50){  
        // Read Inport every 5 second.  
        ( Simulate this just by print out a text : Reading Inport now)  
    }  
    Exit thread  
} //End-----  
  
// ----- Function get_time_ms -----  
double get_time_ms(){  
..  
..  
} // ----- End -----
```

Tip: Try to do the printing of program time every second on same row until a printout of reading inport is done.

Example of console text:



```
C:\Embedded C-program\Vin_thread\New20>demo  
Time is 5  
Reading value now  
Time is 10  
Reading value now  
Time is 15  
Reading value now  
Time is 20  
Reading value now  
Time is 25  
Reading value now  
Time is 30  
Reading value now
```

## Exercise 6\_5

(4p)

### Testing synchronization of threads in C-programming

By using a library pthread.h you can create several theads (light weight processes) from your program. This task for you in these exercises is to test threads and some synchronization primitives such as mutex (simple semaphore) and condition variables signals.

You can read related information about this at:

<http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>

Read and study about : Basics, Creation and termination, Synchronizations.

The main task for you is to try to develop a program demonstrating a global circular buffer in which it should be possible for threads to write (put in) and read (fetch out) characters in the buffer. All writing and reading from buffer should be without losing data or taking it out in wrong order.

The Buffer and its two index number is global

Global / Shared variables

```
char buffer[MAX];          // circular buffer . Test for MAX 5 and 10.
int inpos;                 // index for next character to be put in buffer.
int outpos;                // index for next character to be read ( fetch )
int count;                 // the number of characters in buffer not fetched.
```

For synchronization between threads there are some function in the pthread library you shall use:

pthread\_mutex\_init(); pthread\_mutex\_lock(); pthread\_mutex\_unlock() and  
pthread\_cond\_signal(), pthread\_cond\_wait.

To compile the program you have to add the library pthread when compiled as follows:

```
gcc filename.c -o program -lpthread
```

**The program should have the following structure and function.**

A main program for setting up and start running two threads as described below. The main program should after started up the threads continue in a loop as you can see in the program skeleton below. (You find the code at the homepage in file **skeleton\_exerc\_6\_5.c**)

The first thread, **put()**, should create characters a,b,c...z ; a,b,c.... and store them in the buffer in an infinitive loop. For every character stored it should call a conditional signal (cond\_signal) telling other threads that buffer not empty. If buffer full (MAX number of characters in buffer) the **put()** thread should wait for a *cond\_signal* 'not full' from the other consuming thread telling that the buffer is not full. The second thread, **fetch()**, should fetch out the character in first position to be fetched from the buffer and print it out in the console window. If no character in buffer it should wait for a *cond\_signal* 'not empty' from the producer thread that signal every time it stores a new character in buffer. The result should be that the program prints out 'abcdef...z' 'abcd...z' in correct alphabetical order. Between the letters the main() thread in a loop prints out a textstring when it is running as the picture in figure below shows as one possible example of output. Do not need to be exactly same output but the characters should be in the expected ordnung.

You can find additional information regarding circular buffer, pthread and so on internet

// **File skeleton\_exerc\_6\_5.c** Skeleton for exercise nr 5 in WP 6 course DIT632.

```
#include <stdio.h>
```

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

```
#include <pthread.h>
```

```
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_cond_t not_empty = PTHREAD_COND_INITIALIZER;
```

```
pthread_cond_t not_full = PTHREAD_COND_INITIALIZER;
```

// Global buffer and corresponding counters.

```
char letter = 'a'; //the starting letter
```

```
#define MAX 10//buffer size
```

```
unsigned short count = 0;
```

```
char buffer[MAX]; // circular buffer
```

```
int inpos = 0; // index for next character to be put in buffer (put)
```

```
int outpos = 0; // index for next character to be fetched out from buffer ( fetch ).
```

```
void *put();
```

```
void *fetch();
```

```
int main(){
```

```
int i;
```

```
while(1){
```

```
....
```

```
....
```

```
}
```

```
}
```

```
void *put(){
```

```
while(1){
```

```
....
```

```
....
```

```
}
```

```
}
```

```
void *fetch(){
```

```
while(1){
```

```
....
```

```
....
```

```
}
```

```
}
```

----- End of WP 6 assignments exercises -----

```

Kommandotolken
Buffer store
Buffer output : a
Main is executing
Buffer store
Buffer output : b
Main is executing
Buffer store
Buffer output : c
Main is executing
Buffer full
Buffer output : d
Main is executing
Buffer output : e
Buffer store
Buffer store
Main is executing
Buffer store
Buffer output : f
Main is executing
Buffer store
Buffer output : g
Main is executing
Buffer store
Buffer output : h
Main is executing
Buffer store
Main is executing
Buffer output : i

```

## Some theory exercises regarding real-time systems for course week no 6.

Below you find some theory exercises regarding Real-Time systems suitable to solved or answered before the final exam. Solution of this should not be handed in or demonstrated for any TA.

After all exercises you find proposals for answers or solutions.

### RT exercise 1

Suppose we have a system consisting of several periodic processes that are executed with support of a real-time operating system. As usual, the processes can be described with values according to  $P_n$  ( $p_n, c_n, d_n$ ).

Assume that the processes are executed with the support of a dynamic scheduler and that we have at least three processes P1 to P3 where the processes are prioritized according to P1 highest priority and P3 lowest of the three. The scheduling uses pre-emptive scheduling.

P1 and P3 shares a semaphore to ensure mutual exclusion of their critical regions.

In connection with the use of semaphores and different priorities, something called priority inversion may arise. Try to briefly explain how the problem can arise and indicate how one can handle and partly solve the problem in a real-time operating system.

### RT exercise 2

Suppose we have a set of processes with data according to the table below.

The processes should be scheduled with a static schedule. The processes are of the pre-emptive type.

a) Calculate the LCM (Least Common Multiple) number for the process time periods. Then adjust the period times in a suitable manner so we get a lower LCM number for the period times suitable for the scheduling and then calculate the new LCM number.

b) Draw a possible static schedule for the processes based on the adjusted periodic times.

Process	p	c	d
P1	6	2	4
P2	10	2	5
P3	16	5	10

### RT exercise 3

Suppose we have a set of processes with data according to the table to the right. The processes P1 and P3 do include critical regions for which mutual exclusion is managed by a semaphore S1.

Decide if the system is possible to schedule according to a RMSA analyses.

Proc.	Pri.	p (ms)	d(ms)	c(ms)	Semafor S1
P1	0	6	6	2	2 ms
P2	1	13	8	3	
P3	2	25	15	5	3 ms

#### RT exercise 4

Suppose we have a system with three processes with program code as figure to the right. The processes are executed in a system supported by a Real-time operating system with support for semaphores. Process two and three share a semaphore to ensure mutual exclusion for the respective critical regions. The processes have fix priority according to P3, P1 and P2, where P3 has the highest priority.

Execution times for the functions are: compute1 (): 5 sec, compute2 (): 4 sec and compute3 () 2 sec.

The functions: parkForEvent\_x () means that the process awaits a certain event no x to occur and is not assigned execution time during this wait. When an event occurs, the process changes to Ready and is given execution time based on the fixed priority it has, which may mean that it might either wait until a higher priority processes is completed or given execution time immediately if it has a higher priority than the one currently executed.

Assume that events # 1 - 3 occur in the following order, event #2 , event #3 and last event #1, with one second between each event. We must assume that no other processes affect the execution of the three processes.

We also assume that we start with all processes in the parkForEvent () mode.

```
Process 1() {
    while(1) {
        parkForEvent_1();
        compute1();
    }
}
//-----

process2(){
    while(1) {
        parkForEvent_2();

        waitsem(S1);
        comput2();
        signalsem(S1)
    }
}
//-----

Process3(){
    while(1) {
        parkForEvent_3();
        waitsem(S1);
        compute3();
        signalsem(S1)
    }
}
//-----
--
```

#### To do :

a) Draw a time diagram for the execution of all processes from the first event until all three processes have performed one iteration of their executions, ie the function computex() has been executed for one period.

Draw the time diagram so that the state of the processes (**waiting and running**) is shown.

b) Due to the use of the semaphore, one will have an undesirable effect in the execution of the processes. Describe this effect and specify what it is called.



c) There is a method to minimize the impact of this effect. Briefly describe this method and what it usually is called.

---

## Proposals for answers or solutions for the theory exercises regarding Real-Time systems.

### Answer to RT exercise 1

Suppose the following scenario together with the use of semaphore S1:

P3 executes, calls, and blocks S1, executes a part of it's critical region.

Process change, P1 starts executing, calls S1, S1 is already blocked and P1 moves to the wait queue for S1.

Process change, P2 executes its entire time period.

Process change to P3, continue executes the critical region, releases semaphore S1.

Process switch to P1 which now can block the semaphore and execute further.

As a result of P3:s blocking of the semaphore P1 will be delayed in it's execution by this lower process P3. In this case P3 is working as it had a higher or equal priority than P1. This effect is called (Priority inversion).

The problem is solved by minimizing the delay (blocking) by temporarily assigning P3 to the same or higher priority than P1 when P1 calls S1. P1 can then execute further and then release the temporary priority at same time as it releases the semaphore. The method is called Priority Heritage

### Answer to RT exercise 2

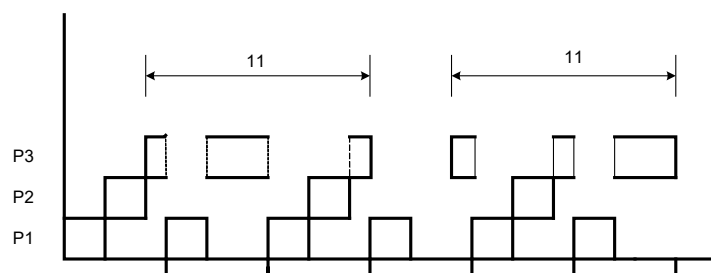
$$\text{LCM}(6,10,16) = \text{LCM}(\text{LCM}(6,10),16) = \text{LCM}((6*10)/2,16) = 30*16/2 = 240$$

[ Based on the fact that  $\text{LCM}(a,b,c) = \text{LCM}(\text{LCM}(a,b),c)$  and

$\text{LCM}(a,b) = (a*b)/\text{lcd}$ ; lcd : least common denominator (minsta gemensamma nämnaren )

Adjust the periodic times to 5,10,15 witch gives  $\text{LCM} = 30$  (Alt. 6,8,16 with  $\text{LCM} = 48$ )

b) Possible static schedule for the period times 5,10,15



Note: If we assume that the system is scheduled by a dynamic scheduler under run-time you can with a RMSA analyses calculate the response time for P3 to 15 ms. This result because RMSA gives the

worst-case response-time. Worst case is when all three processes are ready at the same time (here  $t=0$ ). P1 will run first then P2 and last P3 starts and it takes 11 ms to execute finished as the schedule views.

### Answer RT exercise 3

Blocking of the semaphore by P1 and P3 gives the blocking factors:  $b_1 = 3$  ms and  $b_2 = 3$  ms.

R3 is then calculated to 12 ms with RMSA analyses which is less than deadline  $d_3 = 15$  ms

R2 is calculated to 10 ms which is greater than the deadline  $d_2 = 8$  ms.

The system is by this not possible to schedule with a dynamic scheduler with the given fix priority (RMSA) due to the fact that one process don't meet the requirement for the deadline.

### Answer RT exercise 4

P2 starts running and blocks the semaphore S1, at event 3 P3 starts running due to its higher priority. P3 asks for blocking the semaphore but it's already blocked by P2 and P3 sets to waiting for semaphore. Only P2 is in ready mode now so it continues running until event 1 occur which make system switch over to run P1. P2 still holds the semaphore so P3 stays in waiting mode. P1 ends and P2 gets into run mode and release the semaphore and ends. P3 is now getting running time and can block the semaphore and execute until its end.

As the figure shows, P3, despite its high priority, will have to wait for both P1 and P2. P1 and P2 operate in this position as if they had a higher priority than P3. The effect is called Priority inversion.

