Name ______________________ Matriculation no. ______________________ Tutorial no. ________

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marius Hillenbrand
Dipl.-Inform. Marc Rittinghaus

Operating Systems 2013/14
0x4D4153 Assignment 4

Submission Deadline: Monday, January 13th, 2014 – 9:30 a.m.

In this assignment you will study sychronization, deadlocks, and memory management. All the organizational remarks of the first assignment are still valid!

Please print out the pages containing **T-Questions** and answer them on your printout. Clearly mark every page with your name, matriculation number and **tutorial number**. Simply put it in the mailbox in the basement of building 50.34 (Info-Neubau).

P-Questions are programming assignments. Download the provided tarball from the VAB and make sure to use the included templates and Makefiles. Do not fiddle with the compiler flags. Submission instructions can be found in the first assignment.

Any assignment handed in after its deadline will be ignored!

T-Question 4.1: Deadlocks

a. Analyze the code fragment below. Can a deadlock occur? Why, or why not? **2 T-pt**

```
Mutex m1, m2, m3 = 1;

Thread1()
{
    wait(m1);
    // update some data
    signal(m1);
    wait(m2);
    wait(m3);
    // update some more data
    signal(m2);
    signal(m3);
}

Thread2()
{
    wait(m2);
    // update some data
    wait(m3);
    // update some more data
    signal(m2);
    wait(m1);
    // update even more data
    signal(m3);
    signal(m1);
}
```

b. Explain why an unsafe state does not always lead to a deadlock! **1 T-pt**

Name ______________________ Matriculation no. ______________________ Tutorial no. ______________________

c. Given a system with 3 processes, P_1 to P_3, and three resource types R_1 to R_3. Assume that there is only one instance of each type. Depict the resource-allocation graph for the following situation: P_1 has requested R_1 and R_2. P_2 is holding R_1 and is waiting for R_3. P_3 has acquired R_3 and is waiting for R_2. **1 T-pt**

d. Has a deadlock occured in the above situation? Why, or why not? (Assume that resources can neither be shared nor preempted.) **1 T-pt**

e. The processes P_1, P_2, and P_3 described above only need the resources they have already requested or acquired already. Is the system in a safe state? Prove your claim! **2 T-pt**

T-Question 4.2: Memory Allocation

a. What HW support is required for a simple partition-based memory management? **1 T-pt**

b. What is the difference between internal and external fragmentation? **1 T-pt**

c. Why would you prefer the allocation policy first fit over best fit? **1 T-pt**

d. Why might best fit produce more fragmentation than first fit in some scenarios? **1 T-pt**

T-Question 4.3: Segmentation

a. What parts make up a logical address, when segmentation is used? **1 T-pt**

b. Consider a system with 16-bit logical addresses that supports 4 different segments. What is the maximum size a segment can have? **1 T-pt**

Name ____________________ Matriculation no. ____________________ Tutorial no. __________

c. Assume a system with 16-bit logical addresses that supports four different segments, which uses the following segment table: **4 T-pt**

segment no.	base	limit
0	0xdead	0xbeef
1	0xf154	0x013a
2	0x0000	0x0000
3	0x0000	0x4711

Complete the following table and explain briefly how you derived your solution for each row in the table.

logical address	segment number	offset	valid?	physical address
	3	0x3999		
0x2020				
0x9859				
			yes	0xf15f

T-Question 4.4: Bonus Questions

a. A spinlock that is frozen in place must be nuked from orbit. Explain why. **0 T-pt**

b. How does an OS handle the diversity in the gap? (Note: There is no orbit around the gap.) **0 T-pt**

P-Question 4.1: Contiguous Allocation

You are in a team that implements an OS that uses contiguous allocation for process memory. Your task is to write the allocator that allocates and frees memory partitions of arbitrary sizes (with a granularity of 1 kB). You decide to develop and test your allocator as a user-level process before integrating it into your new OS.

Your allocator will manage memory of size `MEM_SIZE` kB that you first allocate on the heap using `malloc`. Your allocator should label each memory partition with the identifier of the process running in that partition. For easier testing, you chose to identify processes with letters (A, B and so on). Make sure that your code works independently of the actual value of the macro `MEM_SIZE`.

Add your functions to the file `allocator.c`, but keep the function signatures in `allocator.h` unchanged. You may add test code to `main.c`.

a. First, design and implement the data structures you want to use in your memory allocator. Then, add a function that prints all current allocations in a *memory map* to help you debug your code: For each kB in the managed memory, print the partition it is assigned to, ordered from address 0 to `MEM_SIZE-1` kB. Print unallocated space as '0'. For example, three partitions 'A', 'B', and 'C', with sizes of 3, 1, and 2, might occur in the memory map as `AAA0CCB000...0`. **2 P-pt**

b. Implement the *first fit* allocation policy and place it in `allocate_partition_first_fit`. Test your solution using the memory map function. Try to provoke fragmentation. **2 P-pt**

c. Your project management decides to use the *best fit* policy instead. Implement the new policy in the function `allocate_partition_best_fit`. **2 P-pt**

d. Finally, also implement the *worst fit* allocation policy as a third alternative in function `allocate_partition_worst_fit`. **2 P-pt**

P-Question 4.2: Segmentation

Implement the address translation mechanism of segmentation. Follow the scheme in T-Question 4.3: Use 16-bit logical addresses and four different segments.

Put your solution into `segment.c`. Do not change `segment.h`. The segment table is filled in `main.c` where you may add arbitrary test code (try the values from the theory question). However, your solution must work with any values in the segment table.

a. Write code that calculates the following values from a logical address and a segmentation table:

- the segment number,
- the offset,
- the validity of the logical address, and
- the physical address (or 0 for invalid logical addresses)

Fill out the function stubs in `segment.c`. **2 P-pt**

P-Question 4.3: Shared Memory

Have a look at `asst4-sharedmem/sharedmem.c`. This code template forks a child process that is intended to act as a client, while the parent process represents the server. Your job is to implement both client and server and the communication between the two, using POSIX shared memory.
Use the lecture slides on process coordination as a starting point to find out how to use shared memory. Also, have a look at the man-pages of `shmget`, `shmat`, and `shmdt`.

a. Establish a shared memory segment before forking the client so that both server and client know the id of the shared segment. For every shared memory call you do, write a comment, that explains what the passed arguments do and why you pass them! You will not be accredited points for the calls that do not have this mandatory comment. **2 P-pt**

b. Implement the `client()` function to read two unsigned integer values from standard input and pass them to the server via the shared memory segment.
Wrap all values you pass through the shared memory segment in one single struct! The shared memory segment may only contain a single variable (the struct) for a legal solution! **1 P-pt**

c. Implement the `server()` function to read the two values written by the client, calculate the product of the two values, and print the result followed by a newline to standard output. Limit the output of your solution to the mentioned elements. Don't print more! **1 P-pt**

d. Use POSIX semaphores for signaling between the client and the server process: The server must wait until the client entered two values before processing the input, the client must wait until the server is done processing the input and printing the output before prompting for new input values. Hint: Use two semaphores. **2 P-pt**

e. Entering -1 for both values shall terminate both the client and the server process. Take care of cleaning up the shared memory segment. **1 P-pt**

Total:
17 T-pt
17 P-pt