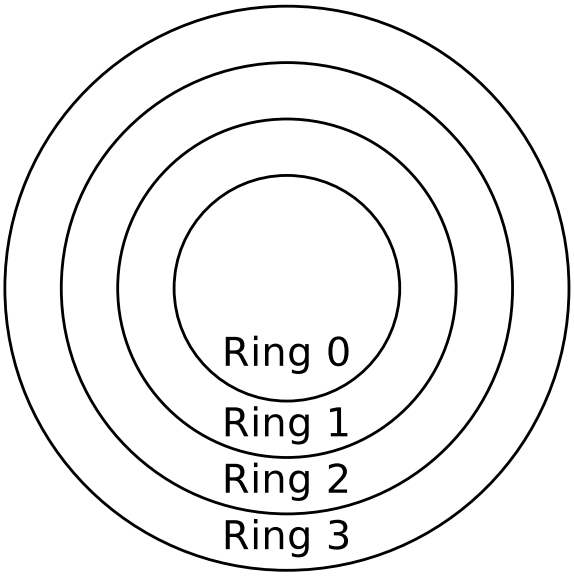
Exercise Sheet 4

Exercise 1 (System Calls)

1. x86-CPUs contain 4 privilege levels ("rings") for processes. Mark in the diagram (*clearly visible!*) the kernel mode and the user mode.



2. Which ring contains the kernel of the operating system?

3. Which ring contains the applications of the users?

4. Processes of which ring have full access to the hardware?

5. Name a reason for the differentiation between user mode and kernel mode.

6. What is a system call?

7. What is a context switch?

8. Name two reasons why user mode processes should not call system calls directly.

9. What alternatives exist, if user mode processes should not call system calls directly?

Exercise 2 (Processes)

1. Which three sorts of process context information stores the operating system?

2. Which process context information are not stored in the process control block?

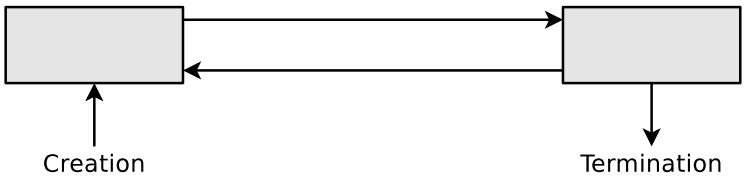
3. Why does the process control block not store all process context information?

4. List all information stored in the process control block of a RIOT process (thread)? (Check at `https://doc.riot-os.org`))

5. What is the task of the dispatcher?

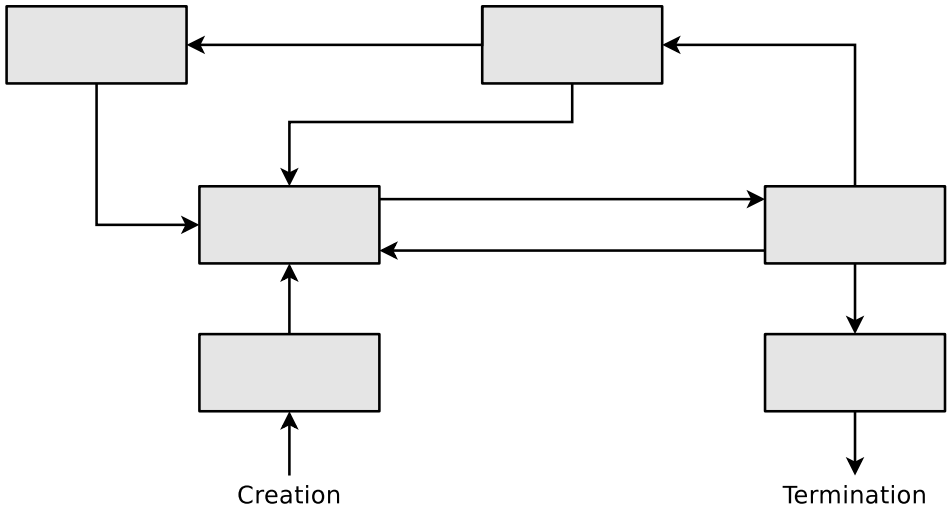
6. What is the task of the scheduler?

7. The process state model with 2 states is the smallest possible process model. Enter the names of the states in the diagram of the process state model with 2 states.



8. Does the process state model with 2 states make sense? Explain your answer shortly.

9. Enter the names of the states in the diagram of the process state model with 6 states.



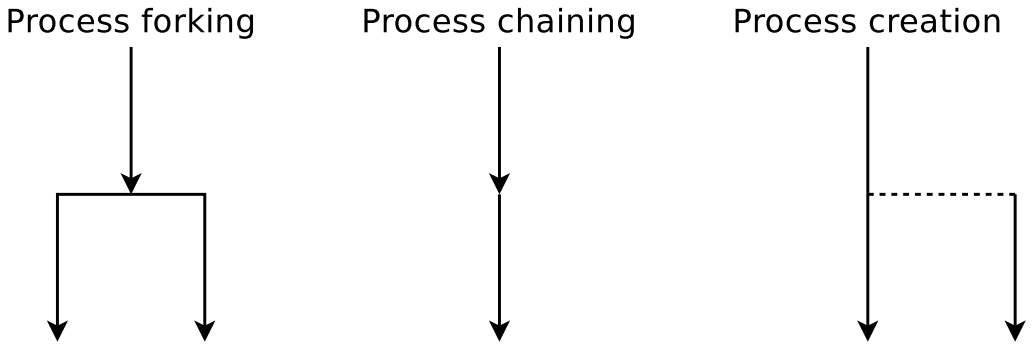
10. What is the task of the process table?

11. How many status lists for processes in *blocked* state manages the operating system?

12. Describe the effect of calling the system call `fork()`.

13. Describe the effect of calling the system call `exec()`.

14. The three diagrams below show all existing ways of creating a new process. Specify for each diagram, which system call(s) are required to implement the illustrated way of process creation.



15. A parent process (PID = 75) with the characteristics, described in the table below, creates a child process (PID = 198) by using the system call `fork()`. Enter the four missing values into the table.

	Parent Process	**Child Process**
PPID	72	
PID	75	198
UID	18	
Return value of `fork()`		

16. Describe what `init` is and what its task is.

17. Name the differences of a child process from the parent process shortly after its creation.

18. Describe the effect, when a parent process is terminated before the child process.

19. Describe what data the Text Segment contains.

20. Describe what data the Heap contains.

21. Describe what data the Stack contains.

Exercise 3 (Process States)

Implement a program that create new processes and turn them into zombies.

Check the information about the processes in the *proc* filesystem (⟶ `man 5 proc`).

Exercise 4 (Forking Processes)

In this programming exercise you have to work with processes including forking, executing other programs, and waiting for child processes.

In this assignment, you will write a little application to launch another program and measure its CPU runtime.

Basic functionality when user executes your program **mytime**:

1. If no command line arguments are given, the program should print information on how to call the program correctly.
2. When executed with at least one command line argument the first argument should be interpreted as a program name. Your program should then execute this program – as a separate process – and pass all remaining command line arguments as arguments to the executed program.
3. For any executed program its return value and the CPU time of the process in milliseconds should be printed.
4. You must create a *Makefile* such that when someone types `make` in your working directory it will compile the program with an output of `mytime`.

You will need the system commands `fork`, a version of `exec`, `waitpid`, and `clock_gettime` to complete this task. For details on how to use these, you can use UNIX's man pages. There is also an online version at `https://www.kernel.org/doc/man-pages/`.