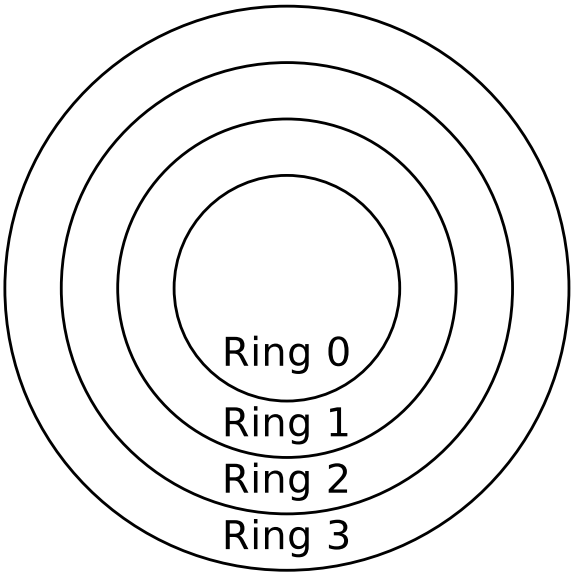
Exercise Sheet 7

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. Name the ring to which the operating system is assigned. Name the ring to which the user applications are assigned.
3. Name the ring to which processes are assigned that have full access to the hardware.
4. Name a reason for the differentiation between user mode and kernel mode.
5. Explain what a system call is.
6. Explain what a context switch is.
7. Name two reasons why user mode processes should not call system calls directly.
8. Name an alternative if user mode processes shall not call system calls directly.

Exercise 2 (Processes)

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

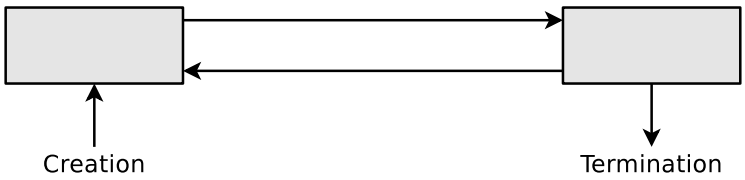
2. Name the process context information that are not stored in the process control block.

3. Explain why the process control block does not store all process context information.

4. Explain the task of the dispatcher.

5. Explain the task of the scheduler.

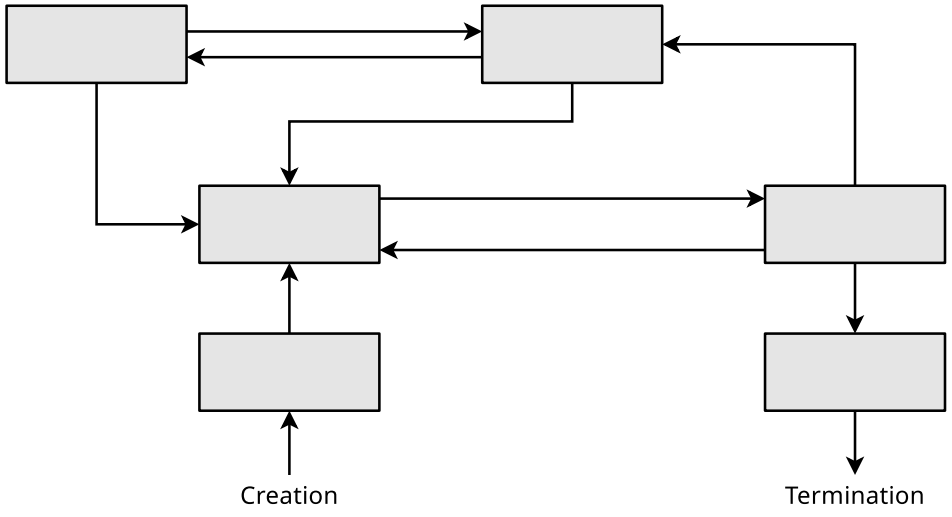
6. The process state model with 2 states is the smallest possible process model. Enter the names of the states in the diagram of the process state model with 2 states.



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

8. Explain why we have extended the 3-state process model in class by the states `new` and `exit`.

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



10. Explain what a zombie process is.

11. Explain the task of the process table.

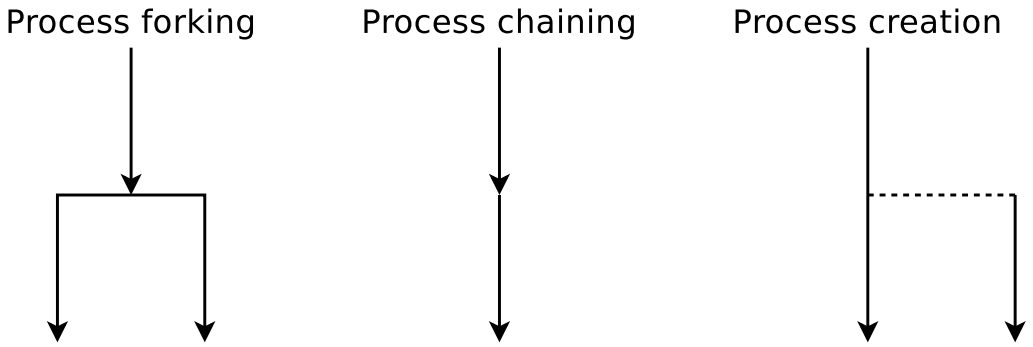
12. Give the number of waiting queues, the operating system manages for processes in „blocked“ state.

13. Explain what happens if a new process is to be created, but the operating system has no more free process IDs (PID) left.

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

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

16. 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.



17. 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()`		

18. The following C source code creates a child process. Give the value of the `returnvalue` variable for the child process and for the parent process. In your answer, explain the importance of the return value in the parent process.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4
5  void main() {
6    int returnvalue = fork();
7
8    if (returnvalue < 0) {
9      printf("Error.\n");
10     exit(1);
11   }
12   if (returnvalue > 0) {
13     printf("Parent Process.\n");
14     exit(0);
15   }
16   else {
17     printf("Child Process.\n");
```

```
18      exit(0);
19    }
20 }
```

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

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

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

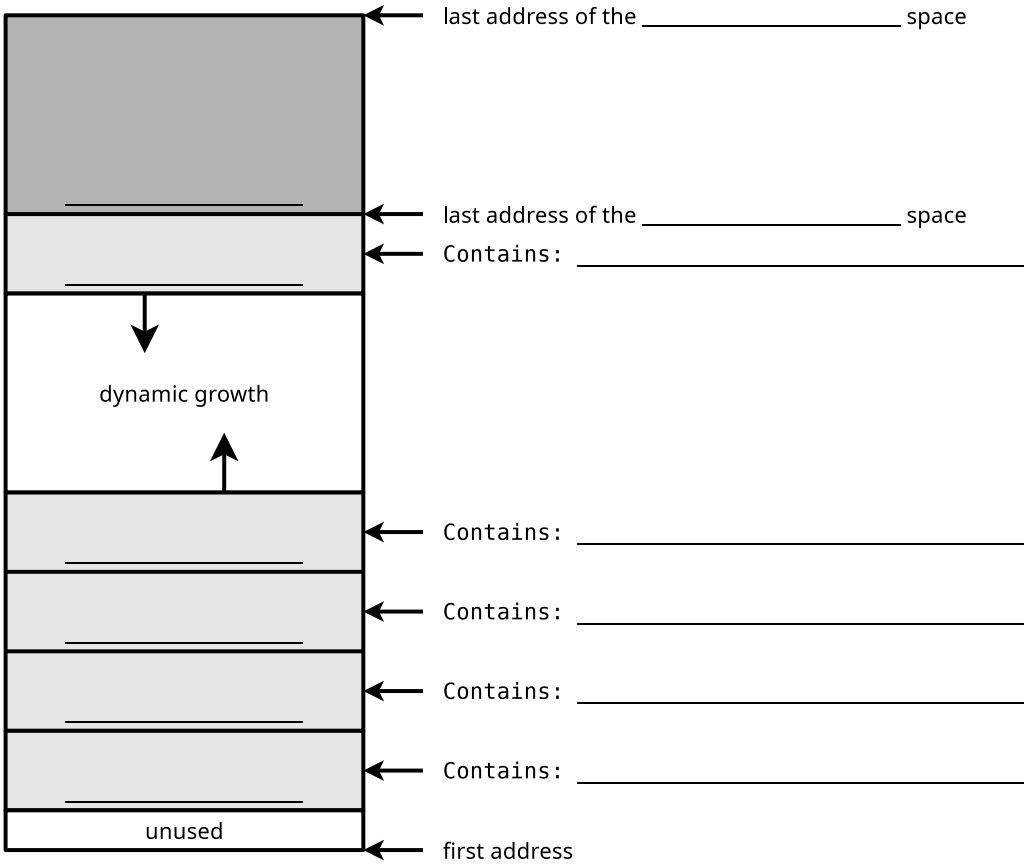
22. Describe what data the Text Segment contains.

23. Describe what data the Data Segment contains.

24. Describe what data the BSS contains.

25. Describe what data the Stack contains.

26. The figure shows the structure of a UNIX process in memory. Fill in the missing labels (technical terms) of the process-related data and the missing information about the content of this data.



Exercise 3 (Information about Processes in the Operating System)

The output of the `ps` command contains helpful information about the processes in the operating system.

```
$ ps -eFw
UID      PID   PPID  C      SZ    RSS PSR STIME TTY        TIME CMD
root       1      0  0   42090  12820   0 Aug29 ?      00:00:03 /sbin/initroot
root       2      0  0       0      0   4 Aug29 ?      00:00:00 [kthreadd]
...
bnc     2149   1782  1  258958 133484   7 Aug29 ?      00:11:20 xfwm4 --display :0.0  ...
bnc     2474   1782  0  137013  54512   8 Aug29 ?      00:03:28 xfce4-panel --display :0.0 ...
bnc     2478   1782  0  166034 138652  15 Aug29 ?      00:00:20 xfdesktop --display :0.0 ...
bnc     3252   2474  3 8590107 577484   9 Aug29 ?      00:51:07 /opt/google/chrome/chrome
bnc     3530   1721  0  157125  62824   0 Aug29 ?      00:00:44 /usr/libexec/gnome-terminal-server
bnc     3568   3530  0    3271   9556  15 Aug29 pts/0  00:00:01 bash
root    6706      1  0    7087  10556   3 Aug29 ?      00:00:00 /usr/sbin/cupsd -l
root    6737      1  0   44549  18680  12 Aug30 ?      00:00:00 /usr/sbin/cups-browsed
bnc    72577  72539  0    2773   7224   4 Aug31 pts/1  00:00:00 /bin/bash
bnc    90775  72577  1  279130 187352   9 09:39 pts/1  00:00:04 okular thesis.pdf
bnc    94414   3568  0    2861   4952   6 11:19 pts/0  00:00:00 ps -eFw
```

1. Explain the content of the column `UID`.
2. Explain the content of the column `PID`.
3. Explain the content of the column `PPID`.
4. Explain the content of the column `C`.
5. Explain the content of the column `SZ`.
6. Explain the content of the column `RSS`.
7. Explain the content of the column `PSR`.
8. Explain the content of the column `STIME`.
9. Explain the content of the column `TTY`.
10. Explain the content of the column `TIME`.
11. Name the parent process of the process that has printed this overview of the processes in the command-line interface.

Exercise 4 (Time-based Command Execution, Control Structures, Archiving)

1. Program a shell script, which reads two numbers as command line arguments. The script should check whether the numbers are identical, and print out the result of the check.

2. Extend the shell script in a way that if the numbers are not identical, it is checked, which one of the two numbers is the larger one. The result of the check should be printed out.

3. Program a shell script, which creates a backup of a directory of your choice. The script should create an archive file with the file extension `.tar.bz2` from the directory. The archive file should be stored in the directory `/tmp`. The name of the archive file should correspond to the following naming scheme:

   `Backup_<USERNAME>_<YEAR>_<MONTH>_<DAY>.tar.bz2`

   The fields `<USERNAME>`, `<YEAR>`, `<MONTH>` and `<DAY>` should be replaced by the current values.

4. Program a shell script, which checks if already today an archive file was created according to the naming scheme of subtask 3. The result of the check should be printed out in the shell.

5. Write two `cron` jobs. She first `cron` job should execute at 6:15 am on every day (except on weekends) the shell script from subtask 3, which creates the archive file with the backup.

   She second `cron` job should execute at 11:45 am on every day (except on weekends) the shell script from subtask 4, which checks, whether already today an archive file was created.

   The output from the shell scripts should be appended to a file `/tmp/Backup-Log.txt`. If the archive file `Backup...tar.bz2` has been created successfully, this should be noted in the log file `/tmp/Backup-Log.txt`.

   Before each new entry in the file, lines according to the following pattern (with current values) should be inserted into the log file `/tmp/Backup-Log.txt`.

   ```
   ****************************
   20.11.2013 --- Time: 21:39:51
   ```

Exercise 5 (Shell Scripts)

1. Program a shell script, which checks for a file, which is specified as an argument, whether it exists and if it is a file, a directory, a symbolic link, a socket or a named pipe.

   - The script should print out the result of the check.

2. Extend the shell script from subtask 1 in a way that if the file, which is specified as an argument, exists, it is checked, if the file could be executed and if write access would be possible.

3. Program a shell script, which reads so long text on the command line, until it is terminated by typing `END`.
   - The script should convert the text, which is read in from the command line, to uppercase.
4. Program a shell script, which prints out the number of running processes for all logged in users.
5. Extend the shell script from subtask 4 in a way that that the output is sorted.
   - The user with most processes should stand at the beginning.
6. Program a shell script, which checks after start every 10 seconds, if a file `/tmp/lock.txt` exists.
   - Each time after the script has checked the existence of the file, it should output an appropriate message on the shell.
   - Once the file `/tmp/lock.txt` exists, the script should terminate itself.