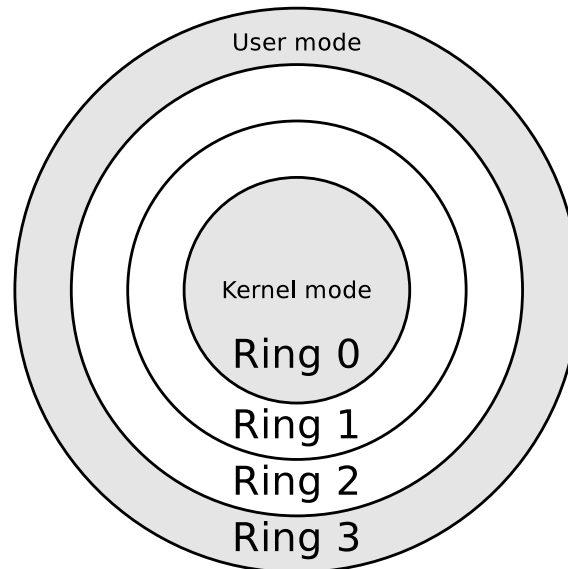# Solution of Exercise Sheet 7

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. Which ring contains the kernel of the operating system?

   In ring 0 (= kernel mode) runs the kernel.

3. Which ring contains the applications of the users?

   In ring 3 (= user mode) run the applications.

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

   Processes in kernel mode (ring 0) have full access to the hardware.

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

   It improves stability and security.

6. What is a system call?

   If a user-mode process must carry out a higher privileged task (e.g. access hardware), it can tell this the kernel via a system call. A system call is a function call in the operating system, which triggers a switch from user mode to kernel mode ($\implies$ context switch).

7. What is a context switch

A process passes the control over the CPU to the kernel and is suspended until the request is completely processed. After the system call, passes the kernel, the control over the CPU back to the user-mode process. The process continues its execution at the location, where the context switch was previously requested.

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

   Working directly with system calls is unsafe and the portability is poor.

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

   Modern operating systems provide a library, which is logically located between the user mode processes and the kernel.

Exercise 2   (Processes)

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

   User context, hardware context and system context.

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

   The user context, which is the allocated address space (virtual memory).

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

   Depending of the architecture, the virtual memory of each process may be several GB in size. Therefore, the user context is just to big in size, to store it twice.
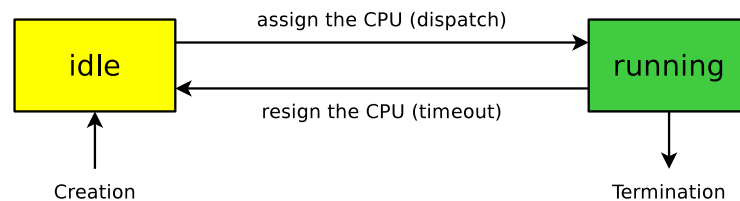
4. What is the task of the dispatcher?

   It carries out the state transitions of the processes.

5. What is the task of the scheduler?

   It specifies the execution order of the processes.

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.

The process state model with two states assumes that all processes are ready to run at any time. This is unrealistic because almost always do processes exist, which are blocked. The idle processes must be categorized into two groups:

- *Processes, which are ready.*
- *Processes, which are blocked.*

8. Enter the names of the states in the diagram of the process state model with 6 states.
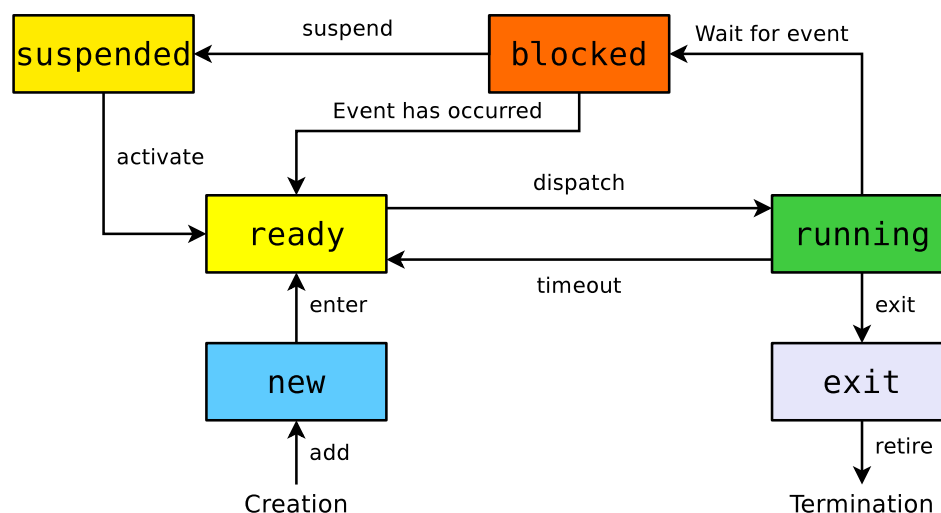
9. What is a zombie process?

A zombie process has completed execution (via the system call `exit`) but its entry in the process table exists until the parent process has fetched (via the system call `wait`) the exit status (return code). Its PID can not yet be assigned to a new process.

10. What is the task of the process table?

For managing the processes, the operating system implements the process table. It contains for each process a record which is called process control block.

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

For each event, there is a separate list of processes waiting for this event.

12. What happens if a new process should be created, but the operating system has no more free process IDs (PID) left?

    In this case, no new process can be created.

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

    *If a process calls **fork**, an identical copy is started as a new process.*

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

    *The system call **exec** replaces a process with another one.*

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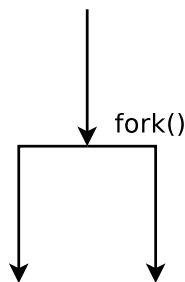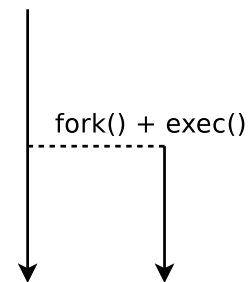
Process forking	Process chaining	Process creation
fork()	exec()	fork() + exec()
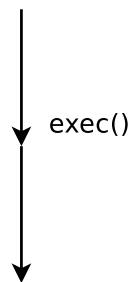
16. A parent process (PID = 75) with the characteristics, described in the table below, creates a child process (PID = 198) by using the system call `fork()`. Enter the four missing values into the table.

	Parent Process	Child Process
PPID	72	*75*
PID	75	198
UID	18	*18*
Return value of `fork()`	*198*	*0*

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

    `init` is the first process in Linux/UNIX. It has PID 1. All running processes originate from `init`. `init` is the father of all processes.

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

    The PID and the memory areas.

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

If a parent process terminates before the child process, it gets `init` as the new parent process assigned. Orphaned processes are always adopted by `init`. The PPID of the cild process then becomes value 1.

20. Describe what data the Text Segment contains.

    It contains the program code (machine code).

21. Describe what data the Heap contains.

    Constants and variables, which get values assigned in global declarations (outside of functions).

22. Describe what data the Stack contains.

    Command line arguments, environment variables of the program call, call parameters and return address of functions, local variables of functions.

Exercise 3 (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.

```
1 #!/bin/bash
2 #
3 # Skript: vergleich.bat
4 #
5 echo "Geben sie zwei Zahlen ein"
6 read -p "Zahl1:" zahl1
7 read -p "Zahl2:" zahl2
8 if [ $zahl1 -eq $zahl2 ] ; then
9   echo "Die beiden Zahlen sind gleich groß."
10 else
11   echo "Die beiden Zahlen sind nicht gleich groß."
12 fi
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: vergleich2.bat
4 #
5 echo "Geben sie zwei Zahlen ein"
6 read -p "Zahl1:" zahl1
7 read -p "Zahl2:" zahl2
8 if [ $zahl1 -eq $zahl2 ] ; then
9   echo "Die beiden Zahlen sind gleich groß."
```

```
10 elif [ $zahl1 -gt $zahl2 ]; then
11   echo "Zahl 1 mit Wert $zahl1 ist größer."
12 else
13   echo "Zahl 2 mit Wert $zahl2 ist größer."
14 fi
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: archiv_erstellen.bat
4 #
5 ARCHIVNAME="Backup_`whoami`_`date +"%Y_%m_%d"`.tar.bz2"
6 VERZEICHNIS="/tmp/testverzeichnis"
7
8 # Archivdatei mit bz2-Kompression erstellen
9 # c => create an archive file.
10 # j => bz2 compression.
11 # v => show detailed output of command.
12 # f => filename of archive file.
13 tar -cjvf $ARCHIVNAME $VERZEICHNIS
14
15 # Archivdatei nach /tmp verschieben
16 mv $ARCHIVNAME /tmp
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: archiv_untersuchen.bat
4 #
5 DATEI="/tmp/Backup_`whoami`_`date +"%Y_%m_%d"`.tar.bz2"
6
7 if [ ! -f $DATEI ] ; then
8   echo "Die Datei $DATEI existiert nicht.";
9 else
10   echo "Die Datei $DATEI existiert.";
11 fi
```

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
```

Um die Lösung zu verstehen, hier im Vorfeld einige wichtige Informationen zur Crontabelle.

Jeder Auftrag / jede Zeile in der crontab besteht aus sechs Feldern. Die ersten fünf Felder werden benutzt, um den Ausführungszeitpunkt des Auftrags zu bestimmen. Im sechsten und letzten Eintrag wird das Programm, Skript oder Kommando festgelegt, das zu dem Ausführungszeitpunkt gestartet werden soll.

- *Spalte 1: Minute (`0-59` oder `*`)*
- *Spalte 2: Stunde (`0-23` oder `*`)*
- *Spalte 3: Tag (`1-31` oder `*`)*
- *Spalte 4: Monat (`1-12`, `Jan-Dec` oder `jan-dec` oder `*`)*
- *Spalte 5: Wochentag (`0-6`, `Sun-Sat` oder `sun-sat` oder `*`)*
- *Spalte 6: Auszuführender Befehl (Programmname und Pfad)*

Ein Eintrag in der Crontabelle darf auf keinen Fall einen Zeilenumbruch enthalten und nicht länger als 1024 Zeichen sein. Kommentare beginnen in der Crontabelle immer mit einer Raute (`#`). Es ist nicht nur möglich, einen Wert pro Zeitspalte anzugeben. Es können auch mehrere Werte pro Spalte angeben werden. Diese werden durch Kommas voneinander getrennt.

Zum Ausgeben und Bearbeiten der eigenen Crontabelle auf der Shell existiert der Befehl `crontab`:

- *`crontab -l`: Die eigene crontab ausgeben.*
- *`crontab -e`: Die eigene crontab bearbeiten.*
- *`crontab -r`: Die eigene crontab löschen.*

Der Editor, den der Befehl crontab aufruft, ist standardmäßig `vi`. Um die Crontabelle mit einem anderen Editor zu bearbeiten, muss die Shellvariable `EDITOR` erzeugt werden. Diese muss den Namen und wenn nötig noch den Pfad des bevorzugten Editors enthalten. Mit dem folgenden Befehl auf der Shell wird in Zukunft die Crontabelle immer mit dem Editor `nano` gestartet:

export EDITOR=/usr/bin/nano

Eine mögliche Lösung:

```
$ export EDITOR=/usr/bin/joe
$ crontab -e
```

Einträge in der Crontabelle passend zur Aufgabenstellung:

```
# 1. Spalte: 15. Minute der Stunde
# 2. Spalte: 6. Stunde des Tages
# 3. Spalte: An jedem Tag des Monates
# 4. Spalte: In jedem Monate des Jahres
# 5. Spalte: An den Wochentage Montag bis Freitag
# 6. Spalte: Kommando
15 6 * * 1-5 echo -e "***********************\n`date
+%d.%m.%Y\ ---\ %X`" >> /tmp/Backup-Log.txt &&
/pfad/zu/archiv_erstellen.bat >> /tmp/Backup-Log.txt
# 1. Spalte: 45. Minute der Stunde
# 2. Spalte: 11. Stunde des Tages
# 3. Spalte: An jedem Tag des Monates
# 4. Spalte: In jedem Monate des Jahres
# 5. Spalte: An den Wochentage Montag bis Freitag
# 6. Spalte: Kommando
45 11 * * 1-5 echo -e "***********************\n`date
+%d.%m.%Y\ ---\ %X`" >> /tmp/Backup-Log.txt &&
/pfad/zu/archiv_untersuchen.bat >> /tmp/Backup-Log.txt
```

Exercise 4   (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.

```
1  #!/bin/bash
2  #
3  # Skript: datei_testen.bat
4  #
5  if test -e $1 ; then
6
7    echo "Die Datei existiert."
8
9    if test -d $1 ; then
10     echo "Die Datei ist ein Verzeichnis."
11   elif test -L $1 ; then
12     echo "Die Datei ist ein symbolischer Link."
13   elif test -S $1 ; then
14     echo "Die Datei ist ein Socket."
15   elif test -p $1 ; then
```

```
16       echo "Die Datei ist eine benannte Pipe (FIFO)."
17    fi
18 fi
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: datei_testen2.bat
4 #
5 if test -e $1 ; then
6
7   echo "Die Datei existiert."
8
9   if test -x $1 ; then
10     echo "Datei ist ausführbar"
11   else
12     echo "Datei ist nicht ausführbar"
13   fi
14
15   if test -w $1 ; then
16     echo "Datei ist schreibbar"
17   else
18     echo "Datei ist nicht schreibbar"
19   fi
20
21   if test -d $1 ; then
22     echo "Die Datei ist ein Verzeichnis."
23   elif test -L $1 ; then
24     echo "Die Datei ist ein symbolischer Link."
25   elif test -S $1 ; then
26     echo "Die Datei ist ein Socket."
27   elif test -p $1 ; then
28     echo "Die Datei ist eine benannte Pipe (FIFO)."
29   fi
30 fi
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: einlesen.bat
4 #
5 while [ true ]
6 do
7   read EINGABE
8   if [ $EINGABE == "ENDE" ] ; then
9     exit
10   else
```

```
11      echo $EINGABE | tr '[:lower:]' '[:upper:]'
12    fi
13 done
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: lock_testen.bat
4 #
5 while [ true ]
6 do
7   if test -f "/tmp/lock.txt" ; then
8     echo "Die Datei lock.txt ist vorhanden."
9   else
10     echo "Die Datei lock.txt ist nicht vorhanden."
11   fi
12   sleep 10
13 done
```