Solution of Exercise Sheet 8

Exercise 1 (Scheduling Strategies)

1. Explain why some operating systems have one or more system idle processes.

   If no process is in the state `ready`, the system idle process gets the CPU assigned. The system idle process is always active and has the lowest priority. Due to the system idle process, the scheduler must never consider the case that no active process exists. Modern operating systems create one system idle process for every CPU core in the system.

2. Explain the difference between preemptive and non-preemptive scheduling.

   When using preemptive scheduling, the CPU may be removed from a process before its execution is completed.

   When using non-preemptive scheduling, a process, which gets the CPU assigned by the scheduler, remains in control over the CPU until its execution is finished or it voluntarily gives the control back.

3. Name one drawback of preemptive scheduling.

   Higher overhead compared with non-preemptive scheduling because of the frequent process switches.

4. Name one drawback of non-preemptive scheduling.

   A process may occupy the CPU for as long as it wants and other (maybe more important) processes need to wait.

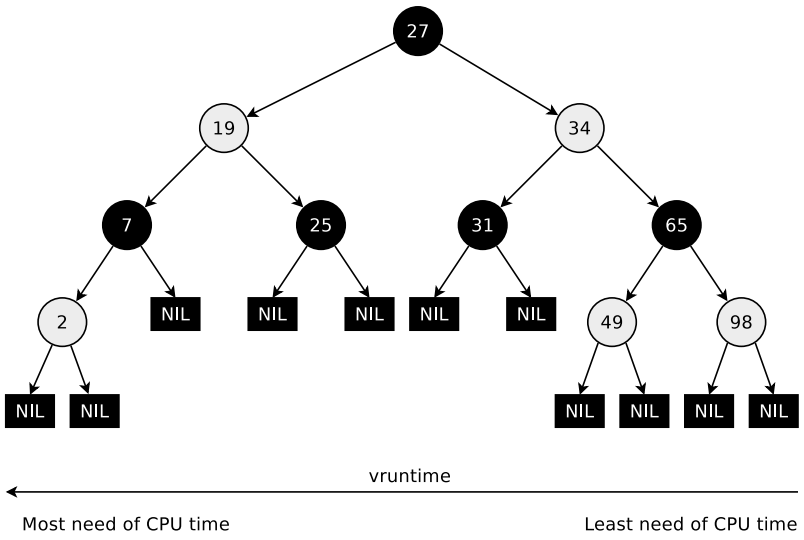
5. Name the scheduling method that Windows operating systems implement.

   Multilevel feedback scheduling.

6. Name one scheduling method that modern Linux operating systems implement.

   Multilevel feedback scheduling (outdated)
   O(1)-Scheduler (outdated)
   Completely Fair Scheduling (until Kernel version 6.5)
   Earliest Eligible Virtual Deadline First (since Kernel version 6.6)

7. Explain how the Completely Fair Scheduler of the Linux kernel (Kernel 2.6.23 until Kernel 6.5.13) works.
   (Hint: A schematic diagram may help here!)



The kernel implements a CFS scheduler for every CPU core and maintains a variable `vruntime` *(virtual runtime) for every* `SCHED_OTHER` *process. The value represents a virtual processor runtime in nanoseconds.* `vruntime` *indicates how long the particular process has already used the CPU core. The process with the lowest* `vruntime` *gets access to the CPU core next. The management of the processes is done using a red-black tree (self-balancing binary search tree). The processes are sorted in the tree by their* `vruntime` *values.*

Aim: All processes should get a similar (fair) share of computing time of the CPU core they are assigned to. For n processes, each process should get $1/n$ of the CPU time. If a process got the CPU core assigned, it can run until its `vruntime` *value has reached the targeted portion of $1/n$ of the available CPU time. The scheduler aims for an equal* `vruntime` *value for all processes.*

The values are the keys of the inner nodes. leaf nodes (NIL nodes) have no keys and contain no data. NIL stands for none, nothing, null, which means it is a null value or null pointer. For fairness reasons, the scheduler assigns the CPU core next to the leftmost process in the tree. If a process gets replaced from the CPU core, the `vruntime` *value is increased by the time the process did run on the CPU core.*

The nodes (processes) in the tree move continuously from right to left $\Longrightarrow$ fair distribution of CPU resources.

The scheduler takes into account the static process priorities (`nice` *values) of the processes. The* `vruntime` *values are weighted differently depending on the* `nice` *value. In other words: The virtual clock can run at different speeds.*

8. Explain how multilevel feedback scheduling works.

   It works with multiple queues. Each queue has a different priority or time multiplex. Each new process is inserted in the top queue and this way it has the highest priority. For each queue, Round Robin is used. If a process resigns the CPU on voluntary basis, it is inserted in the same queue again. If a process utilized its complete time slice, it is inserted in the next lower queue, with has a lower priority.

9. Describe what it means for a scheduling procedure to be fair.

   A scheduling method is fair when every process gets the CPU assigned at some point.

10. Mark the fair scheduling methods.

    ☐ Priority-driven scheduling
    ☒ First Come First Served
    ☒ Round Robin with time quantum
    ☒ Multilevel feedback scheduling
    ☒ Earliest Deadline First
    ☒ Fair-share
    ☒ Completely Fair Scheduler
    ☒ Earliest Eligible Virtual Deadline First

11. Mark the preemptive scheduling methods.

    ☐ First Come First Served
    ☒ Round Robin with time quantum
    ☒ Completely Fair Scheduler
    ☒ Fair-share
    ☒ Multilevel feedback scheduling
    ☒ Earliest Eligible Virtual Deadline First

Exercise 2 (Scheduling)

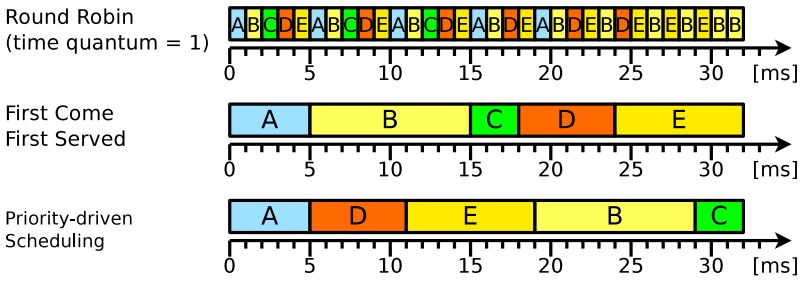
Process	CPU time	Priority
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

Five processes shall be processed on a single CPU/core system. All processes are at time point 0 in state `ready`. High priorities are characterized by high values.

Draw the execution order of the processes with a Gantt chart (timeline) for **Round Robin** (time quantum $q = 1\,\text{ms}$), **FCFS** and **priority-driven scheduling**.

The Priority column in the table is only relevant for the priority-driven scheduling and not for Round Robin or FCFS.

Calculate the average runtimes and average waiting times of the processes.



The CPU time is the time that the process needs to access the CPU to complete its execution.

Runtime = „lifetime“ = time period between the creation and the termination of a process = (CPU time + waiting time).

Runtime	**A**	**B**	**C**	**D**	**E**
RR	20	32	13	25	30
FCFS	5	15	18	24	32
Priority-driven scheduling	5	29	32	11	19

$$\begin{aligned} \text{RR} \quad & (20+32+13+25+30)\ /\ 5 = \quad 24 \text{ ms} \\ \text{FCFS} \quad & (5+15+18+24+32)\ /\ 5 = 18,8 \text{ ms} \\ \text{PS} \quad & (5+29+32+11+19)\ /\ 5 = 19,2 \text{ ms} \end{aligned}$$

Waiting time = time of a process being in state `ready`.

Waiting time = runtime - CPU time.

Waiting time	**A**	**B**	**C**	**D**	**E**
RR	15	22	10	19	22
FCFS	0	5	15	18	24
Priority-driven scheduling	0	19	29	5	11

$$\begin{aligned} \text{RR} \quad & (15+22+10+19+22)\ /\ 5 = 17,6 \text{ ms} \\ \text{FCFS} \quad & (0+5+15+18+24)\ /\ 5 = 12,4 \text{ ms} \\ \text{PS} \quad & (0+19+29+5+11)\ /\ 5 = 12,8 \text{ ms} \end{aligned}$$

Exercise 3 (Shell Scripts)

1. Program a shell script, which requests the user to select one of the four basic arithmetic operations. After selecting a basic arithmetic operation, the user is

requested to enter two operands. Both operands are combined with each other via the previously selected basic arithmetic operation and the result is printed out in the following form:

```
<Operand1> <Operator> <Operand2> = <Result>
```

```
#!/bin/bash
#
# Skript: operanden1.bat
#
echo "Bitte geben Sie den gewünschten Operator ein."
echo "Mögliche Eingaben sind: + - * /"
read OPERATOR
echo "Bitte geben Sie den ersten Operanden ein:"
read OPERAND1
echo "Bitte geben Sie den zweiten Operanden ein:"
read OPERAND2

# Eingabe verarbeiten
case $OPERATOR in
  +)  ERGEBNIS=`expr $OPERAND1 + $OPERAND2` ;;
  -)  ERGEBNIS=`expr $OPERAND1 - $OPERAND2` ;;
  \*) ERGEBNIS=`expr $OPERAND1 \* $OPERAND2` ;;
  /)  ERGEBNIS=`expr $OPERAND1 / $OPERAND2` ;;
  *)  echo "Falsche Eingabe: $OPERATOR" >&2
      exit 1
      ;;
esac

# Ergebnis ausgeben
echo "$OPERAND1 $OPERATOR $OPERAND2 = $ERGEBNIS"
```

2. Modify the shell script from subtask 1 in a way that for each basic arithmetic operation a separate function exists. These functions should be relocated into an external function library and used for the calculations.

```
#!/bin/bash
#
# Skript: operanden2.bat
#
# Funktionsbibliothek einbinden
. funktionen.bib

echo "Bitte geben Sie den gewünschten Operator ein."
echo "Mögliche Eingaben sind: + - * /"
read OPERATOR
echo "Bitte geben Sie den ersten Operanden ein:"
read OPERAND1
echo "Bitte geben Sie den zweiten Operanden ein:"
read OPERAND2

# Eingabe verarbeiten
case $OPERATOR in
  +)  add $OPERAND1 $OPERAND2 ;;
  -)  sub $OPERAND1 $OPERAND2 ;;
  \*) mul $OPERAND1 $OPERAND2 ;;
```

```
  /)  div $OPERAND1 $OPERAND2 ;;
  *)  echo "Falsche Eingabe: $OPERATOR" >&2
      exit 1
      ;;
esac

# Ergebnis ausgeben
echo "$OPERAND1 $OPERATOR $OPERAND2 = $ERGEBNIS"
```

```
# Funktionsbibliothek funktionen.bib

add() {
  ERGEBNIS=`expr $OPERAND1 + $OPERAND2`
}

sub() {
  ERGEBNIS=`expr $OPERAND1 - $OPERAND2`
}

mul() {
  ERGEBNIS=`expr $OPERAND1 \* $OPERAND2`
}

div() {
  ERGEBNIS=`expr $OPERAND1 / $OPERAND2`
}
```

3. Program a shell script, which prints out a certain number of random numbers up to a certain maximum value. After starting the shell script, it should interactively query the values of these parameters:

   - Maximum value, which must be in the number range from 10 to 32767.
   - Desired number of random numbers.

```
#!/bin/bash
#
# Skript: random.bat
#
echo "Geben Sie den Maximalwert ein: "
read MAX

if [[ $MAX -ge 10 && $MAX -le 32767 ]]; then
  echo "Der eingegebene Wert ist innerhalb des Wertebereichs."
else
  echo "Der eingegebene Wert ist außerhalb des Wertebereichs."
  exit 1
fi

echo "Geben Sie an, wie viele Zufallszahlen Sie wünschen: "
read ANZAHL

for ((i=1; i<=${ANZAHL}; i+=1))
do
  echo "Zufallszahl Nr. $i hat den Wert `expr $RANDOM % $MAX`"
```

```
21 done
```

4. Program a shell script, which creates the following empty files:

   `image0000.jpg`, `image0001.jpg`, `image0002.jpg`, ..., `image9999.jpg`

```
1 #!/bin/bash
2 #
3 # Skript: dateien_anlegen.bat
4 #
5 for i in {0..9999}
6 do
7   filename="image"$(printf "%04u" $i)".jpg"
8   touch $filename
9 done
```

5. Program a shell script, which renames the files from subtask 4 according to this scheme:

   `BTS_Exercise_<YEAR>_<MONTH>_<DAY>_0000.jpg`
   `BTS_Exercise_<YEAR>_<MONTH>_<DAY>_0001.jpg`
   `BTS_Exercise_<YEAR>_<MONTH>_<DAY>_0002.jpg`
   `...`
   `BTS_Exercise_<YEAR>_<MONTH>_<DAY>_9999.jpg`

```
1 #!/bin/bash
2
3 # Get the current date components
4 YEAR=$(date +%Y)
5 MONTH=$(date +%m)
6 DAY=$(date +%d)
7
8 # Iterate over the files that match the pattern imageXXXX.jpg
9 for file in image[0-9][0-9][0-9][0-9].jpg; do
10   # Extract the number from the file name
11   num=$(echo $file | sed -e 's/^image\([0-9]\{4\}\)\.jpg$/\1/')
12
13   # Create the new file name
14   new_file=$(printf "BTS_Übung_%s_%s_%s_%s.jpg" "$YEAR" "$MONTH
       " "$DAY" "$num")
15
16   # Rename the file
17   mv "$file" "$new_file"
18 done
```