# Exercise Sheet 8

## Exercise 1 (Scheduling Strategies)

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

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

3. Name one drawback of preemptive scheduling.

4. Name one drawback of non-preemptive scheduling.

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

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

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!)

8. Explain how multilevel feedback scheduling works.

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

10. Mark the fair scheduling methods.

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

11. Mark the preemptive scheduling methods.

    ☐ First Come First Served          ☐ Fair-share
    ☐ Round Robin with time quantum          ☐ Multilevel feedback scheduling
    ☐ Completely Fair Scheduler          ☐ 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 $q = 1$ 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.
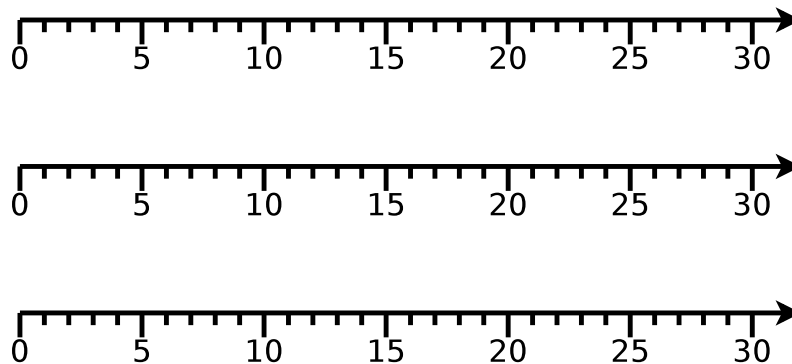






The CPU time is the time that the process needs to access the CPU to complete its execution.

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

| Runtime | A | B | C | D | E |
|---------|---|---|---|---|---|
| RR | | | | | |
| FCFS | | | | | |
| Priority-driven scheduling | | | | | |

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

Waiting time = runtime - CPU time.

| Waiting time | A | B | C | D | E |
|---|---|---|---|---|---|
| RR | | | | | |
| FCFS | | | | | |
| Priority-driven scheduling | | | | | |

# 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>`

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.

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.

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

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

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
   ```