

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. 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.

6. 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.

7. Mark the fair scheduling methods.

- ☐ Priority-driven scheduling
 - ☒ First Come First Served
 - ☒ Round Robin with time quantum
 - ☒ Earliest Deadline First
 - ☒ Fair share

8. Mark the preemptive scheduling methods.

- ☐ First Come First Served
 - ☒ Round Robin with time quantum
 - ☒ Fair share
 - ☒ Multilevel feedback scheduling

Exercise 2 (Scheduling)

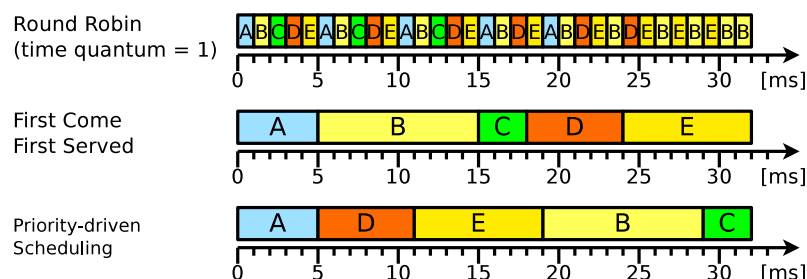
Process	CPU time	Priority
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

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	20	32	13	25	30
FCFS	5	15	18	24	32
Priority-driven scheduling	5	29	32	11	19

$$\text{RR} \quad (20 + 32 + 13 + 25 + 30) / 5 = 24 \text{ ms}$$

$$\text{FCFS} \quad (5 + 15 + 18 + 24 + 32) / 5 = 18,8 \text{ ms}$$

$$\text{PS} \quad (5 + 29 + 32 + 11 + 19) / 5 = 19,2 \text{ ms}$$

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

$$\text{RR} \quad (15 + 22 + 10 + 19 + 22) / 5 = 17,6 \text{ ms}$$

$$\text{FCFS} \quad (0 + 5 + 15 + 18 + 24) / 5 = 12,4 \text{ ms}$$

$$\text{PS} \quad (0 + 19 + 29 + 5 + 11) / 5 = 12,8 \text{ ms}$$

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>

```
1 #!/bin/bash
2 #
3 # Skript: operanden1.bat
4 #
5 echo "Bitte geben Sie den gewünschten Operator ein."
6 echo "Mögliche Eingaben sind: + - * /"
7 read OPERATOR
8 echo "Bitte geben Sie den ersten Operanden ein:"
9 read OPERAND1
10 echo "Bitte geben Sie den zweiten Operanden ein:"
11 read OPERAND2
12
13 # Eingabe verarbeiten
14 case $OPERATOR in
15   +)  ERGEBNIS=`expr $OPERAND1 + $OPERAND2` ;;
```

```
16  -)  ERGEBNIS=`expr $OPERAND1 - $OPERAND2` ;;
17  \*) ERGEBNIS=`expr $OPERAND1 \* $OPERAND2` ;;
18  /)  ERGEBNIS=`expr $OPERAND1 / $OPERAND2` ;;
19  *)  echo "Falsche Eingabe: $OPERATOR" >&2
20      exit 1
21      ;;
22 esac
23
24 # Ergebnis ausgeben
25 echo "$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.

```
1  #!/bin/bash
2  #
3  # Skript: operanden2.bat
4  #
5  # Funktionsbibliothek einbinden
6  . funktionen.bib
7
8  echo "Bitte geben Sie den gewünschten Operator ein."
9  echo "Mögliche Eingaben sind: + - * /"
10 read OPERATOR
11 echo "Bitte geben Sie den ersten Operanden ein:"
12 read OPERAND1
13 echo "Bitte geben Sie den zweiten Operanden ein:"
14 read OPERAND2
15
16 # Eingabe verarbeiten
17 case $OPERATOR in
18 +) add $OPERAND1 $OPERAND2 ;;
19 -) sub $OPERAND1 $OPERAND2 ;;
20 \*) mul $OPERAND1 $OPERAND2 ;;
21 /) div $OPERAND1 $OPERAND2 ;;
22 *) echo "Falsche Eingabe: $OPERATOR" >&2
23     exit 1
24     ;;
25 esac
26
27 # Ergebnis ausgeben
28 echo "$OPERAND1 $OPERATOR $OPERAND2 = $ERGEBNIS"
```

```
1  # Funktionsbibliothek funktionen.bib
2
3  add() {
4      ERGEBNIS=`expr $OPERAND1 + $OPERAND2`
5  }
6
7  sub() {
8      ERGEBNIS=`expr $OPERAND1 - $OPERAND2`
9  }
10
11 mul() {
12     ERGEBNIS=`expr $OPERAND1 \* $OPERAND2`
```

```
13 }
14
15 div() {
16     ERGEBNIS=`expr $OPERAND1 / $OPERAND2`
17 }
```

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.

```
1 #!/bin/bash
2 #
3 # Skript: random.bat
4 #
5 echo "Geben Sie den Maximalwert ein: "
6 read MAX
7
8 if [[ $MAX -ge 10 && $MAX -le 32767 ]]; then
9     echo "Der eingegebene Wert ist innerhalb des Wertebereichs."
10 else
11     echo "Der eingegebene Wert ist außerhalb des Wertebereichs."
12     exit 1
13 fi
14
15 echo "Geben Sie an, wie viele Zufallszahlen Sie wünschen: "
16 read ANZAHL
17
18 for ((i=1; i<=${ANZAHL}; i+=1))
19 do
20     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"
15         " "$DAY" "$num")
16
17     # Rename the file
18     mv "$file" "$new_file"
19 done
```