Lösung von Übungsblatt 8

Aufgabe 1 (Schedulingverfahren)

1. Warum existiert in einigen Betriebssystemen ein Leerlaufprozess?

   Ist kein Prozess im Zustand `bereit`*, kommt der Leerlaufprozess zum Zug. Der Leerlaufprozess ist immer aktiv und hat die niedrigste Priorität. Durch den Leerlaufprozesses muss der Scheduler nie den Fall berücksichtigen, dass kein aktiver Prozess existiert.*

2. Erklären Sie den Unterschied zwischen präemptivem und nicht-präemptivem Scheduling.

   Bei präemptivem Scheduling (verdrängendem Scheduling) kann einem Prozess die CPU vor seiner Fertigstellung entzogen werden.

   Bei nicht-präemptivem Scheduling (nicht-verdrängendem Scheduling) kann ein Prozess die CPU so lange belegen wie er will.

3. Nennen Sie einen Nachteil von präemptivem Scheduling.

   Höherer Overhead als nicht-präemptives Scheduling wegen der häufigeren Prozesswechsel.

4. Nennen Sie einen Nachteil von nicht-präemptivem Scheduling.

   Belegt ein Prozess die CPU, ist es häufig so, dass andere, vielleicht dringendere Prozesse für lange Zeit nicht zum Zuge kommen.

5. Beschreiben Sie, wie das Multilevel-Feedback-Scheduling funktioniert.

   Es arbeitet mit mehreren Warteschlangen. Jede Warteschlange hat eine andere Priorität oder Zeitmultiplex. Jeder neue Prozess kommt in die oberste Warteschlange und hat damit die höchste Priorität. Innerhalb jeder Warteschlange wird Round Robin eingesetzt. Gibt ein Prozess die CPU freiwillig wieder ab, wird er wieder in die selbe Warteschlange eingereiht. Hat ein Prozess seine volle Zeitscheibe genutzt, kommt er in die nächst tiefere Warteschlange mit einer niedrigeren Priorität.

6. Welche Schedulingverfahren sind fair?

   Ein Schedulingverfahren ist „fair“, wenn jeder Prozess irgendwann Zugriff auf die CPU erhält.

   ☐ Prioritätengesteuertes Scheduling
   ☒ First Come First Served
   ☒ Round Robin mit Zeitquantum
   ☒ Earliest Deadline First
   ☒ Fair-Share

7. Welche Schedulingverfahren arbeiten präemptiv (= *unterbrechend*)?

☐ First Come First Served
☒ Round Robin mit Zeitquantum
☒ Fair-Share
☒ Multilevel-Feedback-Scheduling

Aufgabe 2 (Scheduling)

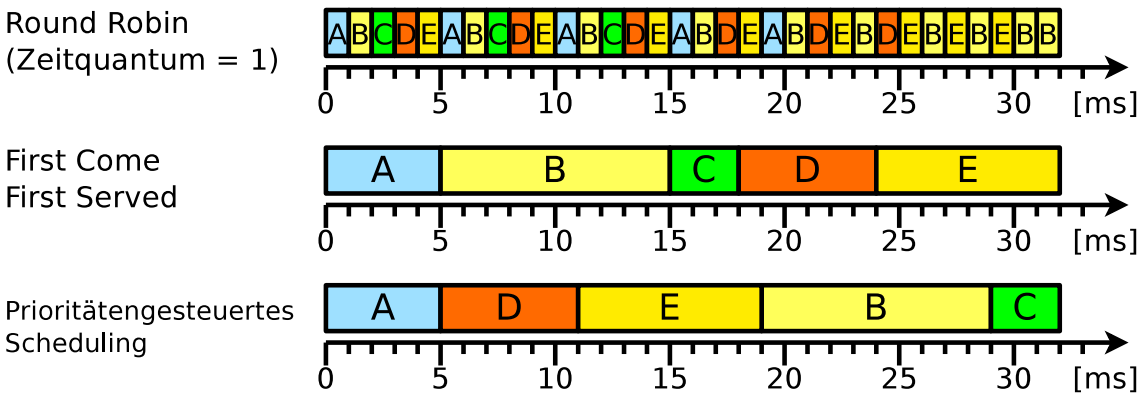
Auf einem Einprozessorrechner (mit nur einem CPU-Kern) sollen fünf Prozesse verarbeitet werden. Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`. Hohe Prioritäten sind durch hohe Zahlen gekennzeichnet.

Prozess	Rechenzeit	Priorität
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

Skizzieren Sie die Ausführungsreihenfolge der Prozesse mit einem Gantt-Diagramm (Zeitleiste) für **Round Robin** (Zeitquantum $q = 1\,\text{ms}$), **FCFS** und **Prioritätengesteuertes Scheduling**.

Die Spalte Priorität in der Tabelle ist nur für das Prioritätengesteuerte Scheduling relevant und nicht für Round Robin or FCFS.

Berechnen Sie die mittleren Laufzeiten und mittleren Wartezeiten der Prozesse.



Die Rechenzeit ist die Zeit, die der Prozess Zugriff auf die CPU benötigt, um komplett abgearbeitet zu werden.

Laufzeit = „Lebensdauer“ = Zeitspanne zwischen dem Anlegen und Beenden eines Prozesses = (Rechenzeit + Wartezeit).

Laufzeit	**A**	**B**	**C**	**D**	**E**
RR	20	32	13	25	30
FCFS	5	15	18	24	32
Prioritätengesteuertes Scheduling	5	29	32	11	19

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

Die Wartezeit ist die Zeit in der `bereit`-Liste.

Wartezeit = Laufzeit - Rechenzeit.

Wartezeit	**A**	**B**	**C**	**D**	**E**
RR	15	22	10	19	22
FCFS	0	5	15	18	24
Prioritätengesteuertes Scheduling	0	19	29	5	11

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

Aufgabe 3 (Shell-Skripte)

1. Schreiben Sie ein Shell-Skript, das den Benutzer bittet, eine der vier Grundrechenarten auszuwählen. Nach der Auswahl einer Grundrechenart wird der Benutzer gebeten, zwei Operanden einzugeben. Die beiden Operanden werden mit der zuvor ausgewählten Grundrechenart verrechnet und das Ergebnis in der folgenden Form ausgegeben:

   `<Operand1> <Operator> <Operand2> = <Ergebnis>`

```
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. Ändern Sie das Shell-Skript aus Teilaufgabe 1 dahingehend, dass für jede Grundrechenart eine eigene Funktion existiert. Die Funktionen sollen in eine externe Funktionsbibliothek ausgelagert und für die Berechnungen verwendet werden.

```
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. Schreiben Sie ein Shell-Skript, das eine bestimmte Anzahl an Zufallszahlen bis zu einem bestimmten Maximalwert ausgibt. Nach dem Start des Shell-Skripts, soll dieses vom Benutzer folgende Parameter interaktiv abfragen:

   - Maximalwert, der im Zahlenraum zwischen 10 und 32767 liegen muss.
   - Gewünschte Anzahl an Zufallszahlen.

```
1  #!/bin/bash
2  #
3  # Skript: random.bat
4  #
5  echo "Geben Sie den Maximalwert ein: "
6  read MAX
7  echo "Geben Sie an, wie viele Zufallszahlen Sie wünschen: "
8  read ANZAHL
9
10 for ((i=1; i<=${ANZAHL}; i+=1))
11 do
12   echo "Zufallszahl Nr. $i hat den Wert `expr $RANDOM % $MAX`"
13 done
```

4. Schreiben Sie ein Shell-Skript, das die folgenden leeren Dateien erzeugt:

   `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. Schreiben Sie ein Shell-Skript, das die Dateien aus Teilaufgabe 4 nach folgendem Schema umbenennt:

   `BTS_Übung_<JAHR>_<MONAT>_<TAG>_0000.jpg`
   `BTS_Übung_<JAHR>_<MONAT>_<TAG>_0001.jpg`
   `BTS_Übung_<JAHR>_<MONAT>_<TAG>_0002.jpg`
   ...
   `BTS_Übung_<JAHR>_<MONAT>_<TAG>_9999.jpg`