6 Quest

6.1 Aufgabe

1. First-Come, First-Served (FCFS)
   Die Threads werden in der Reihenfolge abgearbeitet, in der die Abarbeitungsaufträge eingehen. Bei freiwilliger Abgabe des Prozessors wird der Thread an das Ende der Warteschlange gesetzt. In Abbildung 1 ist dies etwa beim zweiten Thread der Fall.

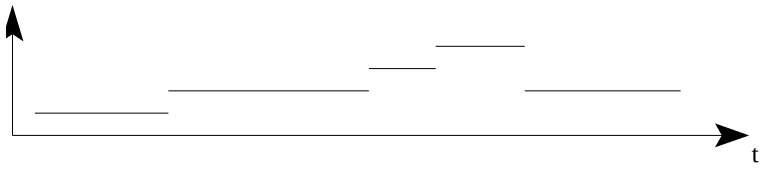

Abbildung 1: Prozess-Zeit-Diagramm zu FCFS

2. Shortest-Job-First (SJF)
   Der Thread mit dem kleinsten nächsten CPU-Burst wird als erstes abgearbeitet. Bei gleichlangem nächsten CPU-Burst wird FCFS angewendet. Dies minimiert die Wartezeit. Problematisch ist dabei, das vor der Abarbeitung die Länge des nächsten CPU-Bursts nicht bekannt ist und somit nur approximiert werden kann.

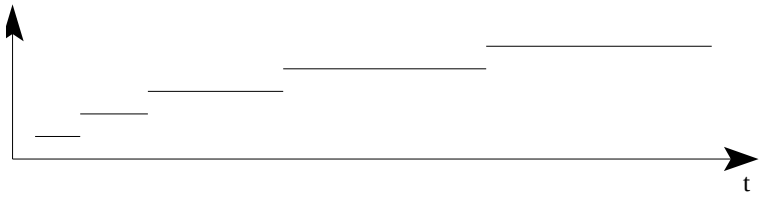

Abbildung 2: Prozess-Zeit-Diagramm zu SJF

3. Round-Robin (RR)
   Alle Prozesse im System werden in einer FIFO-Schlange verwaltet. Dabei wird jedem Thread eine Zeitscheibe zugeteilt. Sobald diese abgelaufen ist, wird der Prozessor abgebeben und der Thread ordnet sich am Ende der FIFO-Schlange ein.

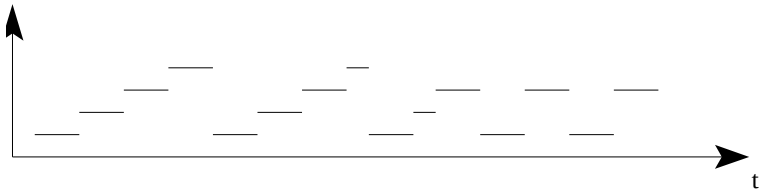

Abbildung 3: Prozess-Zeit-Diagramm zu RR

4. Prioritätsbasiertes Scheduling
   Hier wird jedem Thread eine Priorität zugeordnet. Der Thread mit der höchsten Priorität wird dabei im Prozessor abgearbeitet. Insbesondere wird ein Prozess unterbrochen, sobald ein Thread mit höhrerer

Priorität auftaucht. Bei *statischem* prioritätsbasiertem Scheduling wird diese Priorität zum Erzeugungszeitpunkt festgelegt, bei *dynamischem* kann die Priorität auch nachträglich noch vom Betriebssystem (oder Nutzer) verändert werden.

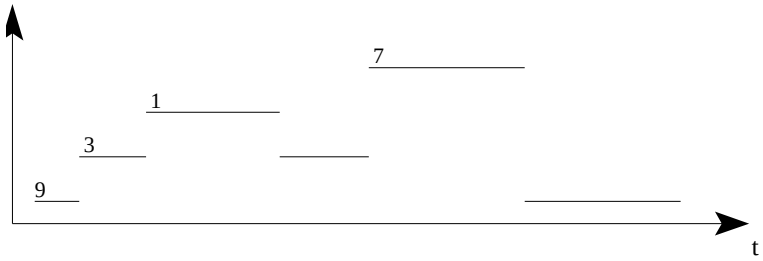

Abbildung 4: Prozess-Zeit-Diagramm zu prioritätsbasiertem Scheduling

5. Echtzeit-Scheduling: Earliest Deadline First (EDF)
   Bei diesem Verfahren wird stets der Thread abgearbeitet, der geringste Zeit bis zum Ablauf seiner Frist aufweist. Ein Prozess mit späterer Deadline wird insbesondere unterbrochen, sobald ein Thread mit früherer Deadline auftaucht.

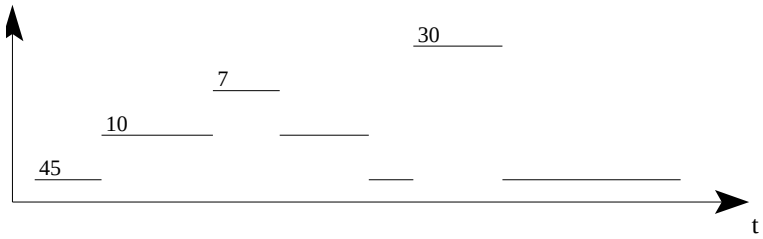

Abbildung 5: Prozess-Zeit-Diagramm zu EDF

6.2 Aufgabe

1. FCFS: $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6$
   $R = \frac{13+19+22+50+55+57+68}{7} = 40\frac{4}{7}$

2. SJF: $P_5 \rightarrow P_2 \rightarrow P_4 \rightarrow P_1 \rightarrow P_6 \rightarrow P_0 \rightarrow P_3$
   $R = \frac{2+5+10+16+27+40+68}{7} = 24$

3. RR (Q=3): Notation: $P^{Restzeit}_{Prozessnummer}$
   $P_0^{10} \rightarrow P_1^3 \rightarrow P_2^0 \rightarrow P_3^{25} \rightarrow P_4^2 \rightarrow P_5^0 \rightarrow P_6^8 \rightarrow P_0^7 \rightarrow P_1^0 \rightarrow P_3^{22} \rightarrow P_4^0 \rightarrow P_6^5 \rightarrow P_0^4 \rightarrow P_3^{19} \rightarrow P_6^2 \rightarrow P_0^1 \rightarrow P_3^{16} \rightarrow P_6^0 \rightarrow P_0^0 \rightarrow P_3^{13} \rightarrow P_3^{10} \rightarrow P_3^7 \rightarrow P_3^4 \rightarrow P_3^1 \rightarrow P_3^0$
   $R = \frac{9+17+26+31+51+52+65}{7} = 35\frac{6}{7}$

4. RR (Q=11): Notation: $P^{Restzeit}_{Prozessnummer}$
   $P_0^2 \rightarrow P_1^0 \rightarrow P_2^0 \rightarrow P_3^{17} \rightarrow P_4^0 \rightarrow P_5^0 \rightarrow P_6^0 \rightarrow P_0^0 \rightarrow P_3^6 \rightarrow P_3^0$
   $R = \frac{17+20+36+38+49+51+62}{7} = 39$

6.3 Aufgabe

1. Sperrvariablen

```
Erzeuger betritt Puffer
Erzeuger sperrt ab
Solange Puffer voll ist:
  Erzeuger sperrt auf
  Erzeuger wartet
  Erzeuger sperrt ab
Erzeuger fuellt Puffer
Erzeuger sperrt auf
Erzeuger verlaesst Puffer

Verbraucher betritt Puffer
Verbraucher sperrt ab
Solange Puffer leer ist:
  Verbraucher sperrt auf
  Verbraucher wartet
  Verbraucher sperrt ab
Verbraucher leert Puffer
Verbraucher sperrt auf
Verbraucher verlaesst Puffer
```

2. Semaphore:
   `leer` initialisiert mit Anzahl der Pufferplätze
   `voll` initialisiert mit 0

```
Erzeuger betritt Puffer: P(leer)
Erzeuger fuellt Puffer
Erzeuger verlaesst Puffer: V(voll)

Verbraucher betritt Puffer: P(voll)
Verbraucher leert Puffer
Verbraucher verlaesst Puffer: V(leer)
```

3. Monitor

```
Erzeuger erzeugt Inhalt
Erzeuger schiebt Inhalt in den Puffer (mittels einer Monitor-Prozedur)

Verbraucher holt Inhalt aus dem Puffer (mittels einer Monitor-Prozedur)
Verbraucher verarbeitet Inhalt
```

4. Nachrichtenaustausch

```
Verbraucher verschickt leere Nachricht an Erzeuger
Erzeuger erzeugt Inhalt
Erzeuger empfaengt Nachrichten
Erzeuger verschickt Nachricht an Verbraucher mit Inhalt
Verbraucher empfaengt Nachricht
Verbraucher extrahiert Inhalt aus der Nachricht
Verbraucher verschickt leere Nachricht an Erzeuger
Verbraucher verarbeitet Inhalt
```

6.4 Aufgabe

Nachrichtenaustausch

6.5 Aufgabe

Semaphor-Variablen: ganzzahlig
Mutex-Variablen: zwei Zustände (offen - abgesperrt)
Anwendungsbeispiel: Siehe Aufgabe 6.3 Semaphore und Sperrvariable

6.6 Aufgabe

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv[])
{
  char in[10];
  int *c;
  printf("Eingabe: ");
  fgets(in, sizeof(in), stdin);

  c = (int*) &in;
  printf("%s\n", c);

  exit(0);
}
```