Fakultät für Informatik und Mathematik
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Betriebssysteme, SoSe 2019

Praktische Übung 8: Scheduling & Modularbeit

Ziele des Labortermins

- Scheduling-Verfahren
- Linux Scheduling

Aufgabe 8.1: Verständnisfragen

a) Was ist der Unterschied zwischen CPU-lastigen und I/O-lastigen Threads?

b) Welche Kriterien sind bei der Stapelverarbeitung, interaktiven Systemen und Echtzeitsystemen jeweils am wichtigsten?

c) Für welche der traditionellen Scheduling-Verfahren FCFS, SJF und SRT ist eine Schätzung der verbleibenden Laufzeit notwendig?

d) Welche der traditionellen Scheduling-Verfahren FCFS, SJF und SRT minimieren die durchschnittle Ausführungsdauer?

e) Welche Rolle spielen die Zeitscheiben bei Round Robin-Scheduling?

f) Was ist Priority Inversion?

g) Was bedeutet Prozessor-Affinität?

Aufgabe 8.2: Traditionelle Scheduling-Verfahren

Im folgenden sollen einige traditionelle Scheduling-Verfahren miteinander verglichen werden. Als Basis für das zeitliche Verhalten gelte eine abstrakte Zeiteinheit. Gegeben seien die folgenden Threads:

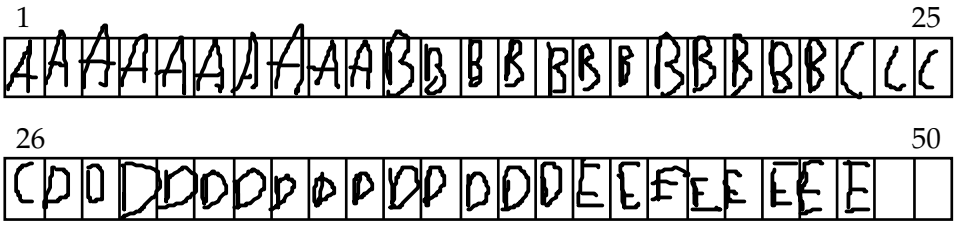
- Thread A mit einer Laufzeit von 10 Zeiteinheiten wird zum Zeitpunkt 1 erstellt
- Thread B mit einer Laufzeit von 12 Zeiteinheiten wird zum Zeitpunkt 2 erstellt
- Thread C mit einer Laufzeit von 4 Zeiteinheiten wird zum Zeitpunkt 2 erstellt
- Thread D mit einer Laufzeit von 14 Zeiteinheiten wird zum Zeitpunkt 4 erstellt
- Thread E mit einer Laufzeit von 8 Zeiteinheiten wird zum Zeitpunkt 6 erstellt

Die Zeitpunkt zu dem ein Thread erstellt wird, wird auch Ankunftszeit genannt. Die "Startzeit" im folgenden sei der Zeitpunkt (bzw. die Zeitpunkte) an denen ein Thread die Resource CPU zugewiesen bekommt (also ausgeführt wird).

Im folgenden sollen verschiedene Scheduling-Verfahren verglichen werden. Vervollständigen Sie jeweils den Zeitstrahl der Zuweisung der CPU und vervollständigen Sie die Tabelle.

.

a) **First Come First Served (FCFS)**

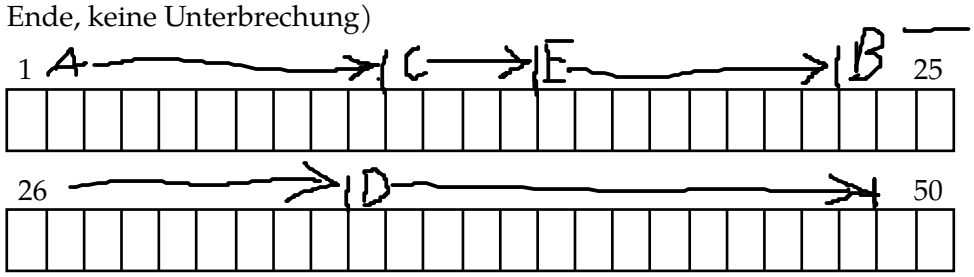


Thread	Ankunftszeit	Startzeit	Endzeit	Turnaround	Turnaround/ Rechenzeit
A	1	1	10	9	9/10
B	2	11	22	11	11/12
C	2	23	26	3	3/4
D	4	27	40	13	13/14
E	6	41	48	7	7/8

Durchschnittliche Turnaround-Zeit (Verweildauer): 8,6

Durchschnittliches Turnaround/Rechenzeit: 0,87

b) **Shortest Job First** (**SJF**) mit "run-to-completion" (jeder Thread läuft bis zum Ende, keine Unterbrechung)

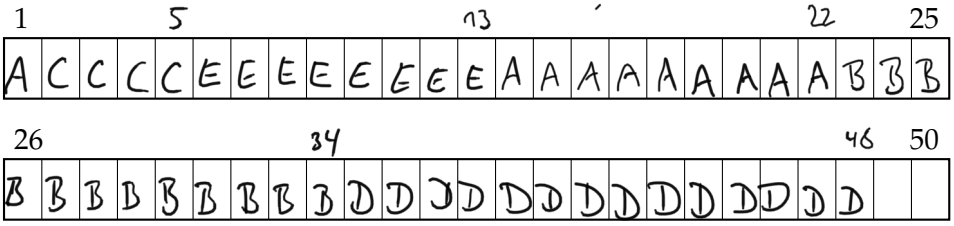


Thread	Ankunftszeit	Startzeit	Endzeit	Turnaround	Turnaround/ Rechenzeit
A	1	1	10	9	9/10
B	2	23	34	11	11/12
C	2	11	14	3	3/4
D	4	35	48	13	13/14
E	6	15	22	7	7/8

Durchschnittliche Turnaround-Zeit (Verweildauer): 8,6

Durchschnittliches Turnaround/Rechenzeit: 0,87

c) **Shortest Remaining Time (SRT)** wobei bei jedem Ereignis (neuer Thread erstellt zum Beispiel) eine Scheduling-Entscheidung getroffen wird.

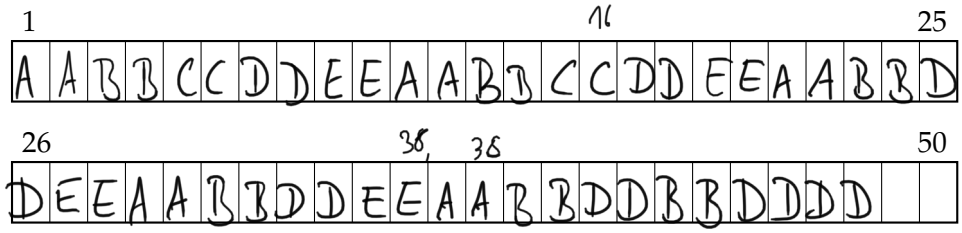


Thread	Ankunftszeit	Startzeit	Endzeit	Turnaround	Turnaround/ Rechenzeit
A	1	1	22	21	2,10
B	2	23	34	32	2,67
C	2	2	5	3	0,75
D	4	35	49	45	3,21
E	6	6	13	7	0,88

Durchschnittliche Turnaround-Zeit (Verweildauer): 21,6

Durchschnittliches Turnaround/Rechenzeit: 1,92

d) **Round-Robin** mit 2 Zeiteinheiten Quantum

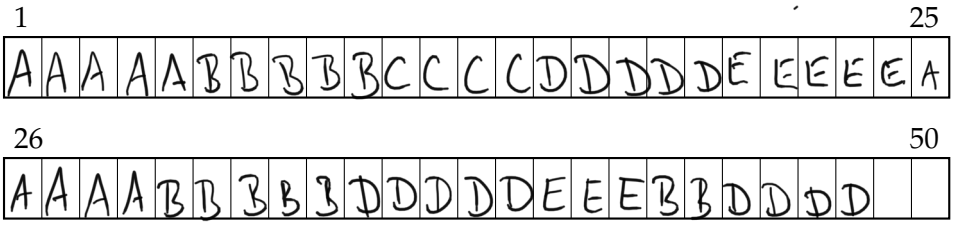


Thread	Ankunftszeit	Startzeit	Endzeit	Turnaround	Turnaround/ Rechenzeit
A	1	1	38	37	3,70
B	2	3	44	42	3,50
C	2	5	16	14	3,50
D	4	7	48	44	3,14
E	6	9	36	30	3,75

Durchschnittliche Turnaround-Zeit (Verweildauer): 33,4

Durchschnittliches Turnaround/Rechenzeit: 3,52

e) **Round-Robin** mit 5 Zeiteinheiten Quantum



Thread	Ankunftszeit	Startzeit	Endzeit	Turnaround	Turnaround/ Rechenzeit
A	1	1	29	28	2,80
B	2	6	44	42	3,5
C	2	11	14	12	3,0
D	4	15	48	44	3,14
E	6	20	42	36	4,5

Durchschnittliche Turnaround-Zeit (Verweildauer): 32,4

Durchschnittliches Turnaround/Rechenzeit: 3,39

Modularbeit

Aufgabe 8.3: Linux Realzeit-Scheduling (praktisch)

Sie haben bereits früher die Möglichkeit kennengelernt, den Linux Kernel zu tracen. Im Folgenden sollen die Scheduling-Entscheidungen angeschaut werden und die Prioritäten verändert werden.

Eine Tracing session schaut in etwa wie folgt aus:

```
lttng create --output <tracename>
lttng enable-event --kernel sched_switch
lttng start; <program>; lttng destroy
```

In der Datei `main.c` finden Sie ein einfaches Programm, das in einer Schleife immer wieder `usleep()` aufruft und 10 Mikrosekunden schläft dadurch.

Für die Durchführung empfiehlt es sich die Anzahl der Kerne der Virtuellen Maschine auf einen Kern zu reduzieren. Außerdem finden Sie das Programm `load.c` das einfach eine Endlosschleife ausführt und damit eine konstante Last mit der Standard-Priorität darstellt.

Übersetzen Sie zuerst die Programme und führen Sie dann qualitative Messungen mit verschiedenen Prioritäten und Scheduling-Verfahren wie folgt durch. Starten Sie das Programm `load`, so dass es für die Messungen im Hintergrund läuft: `./load &` (eventuell mehrfach)

a) Starten Sie das Programm ganz normal im Rahmen einer Tracing-Session, sichern Sie den Trace in einem Ordner `trace-normal` und untersuchen Sie den Trace qualitativ mit `lttng-scope`.

b) Starten Sie das Programm mit einem hohen Nice-Wert 19 (mithilfe des `nice`-Programms), wieder im Rahmen einer Tracing-Session, sichern Sie den Trace in einem Ordner `trace-nice` und untersuchen Sie den Trace qualitativ mit `lttng-scope`.

c) Starten Sie das Programm als Realzeit-Programm mit der `SCHED_RR`-Policy und der Priorität 50 (mithilfe des `chrt`-Programms, mit `sudo`), wieder im Rahmen einer Tracing-Session, sichern Sie den Trace in einem Ordner `trace-rr` und untersuchen Sie den Trace qualitativ mit `lttng-scope`.

d) Welche Unterschiede stellen Sie fest?

Aufgabe 8.4: Zusammenfassung

a) Beschreiben Sie stichpunktartig den grundlegenden Ablauf eines Kontextwechsels zwischen zwei Threads bei der Präemption durch den Scheduling-Timer.

b) Ihr Mitbewohner hat die Vorlesung Betriebssysteme I nur rudimentär besucht. Er möchte für sein altes Notebook mit nur einem Prozessorkern ein Programm schreiben und sagt: ”Ich brauche keine Threads weil ich eh nur eine CPU habe”.

Wieso ist diese Aussage im Allgemeinen nicht sinnvoll? Nennen Sie ein Beispiel für die sinnvolle Verwendung mehrerer Threads mit einer CPU.

c) Zwei Stunden später. Nehmen Sie zu folgender Aussage ihres Mitbewohners überzeugend Stellung: "Ich habe ein Programm unter Linux mit zwei Threads. Aber ich brauche keine Synchronisierung mit Mutexen oder so, weil ich eh nur einen Prozessorkern verwende". Beziehen Sie sich nur auf die Thematik der Synchronisierung.