Vorlesung Betriebssysteme WS 2020/21

Aufgabenblatt 3 zu Unit OS4 vom 10. Dezember 2020

(Vorstellung der Lösungen in der Vorlesung am 07.01.2020)

Die Aufgaben auf diesem und den folgenden Übungsblättern sollen Ihnen helfen, Ihr Selbststudium zu strukturieren und Ihnen ein Gefühl dafür geben, welchen Wissensstand Sie bisher in der Veranstaltung erlangt haben sollten. Um die gemeinsame Zeit mit den Tutoren optimal nutzen zu können, sollten Sie versuchen, bis zum jeweiligen Tutorengespräch möglichst viele der Aufgaben zu bearbeiten.

Die Bearbeitung sollte in einer Art und Weise erfolgen, dass a) Sie und Ihre Gruppe gut mit den Ergebnissen arbeiten können und b) sich im Gespräch mit den Tutoren über den Stoff der Aufgaben austauschen können. Es bestehen keine weiteren formalen Anforderungen an die Aufgabenbearbeitung unsererseits.

1. Die Aufgaben des Scheduling in Betriebssystemen verteilen sich auf drei Komponenten. Den *Short-Term Scheduler*, den *Long-Term Scheduler* und den *Dispatcher*. Was ist die Rolle dieser drei Komponenten?

2. Was ist *Echtzeit*? Unterscheiden Sie dabei *harte* und *weiche Echtzeit-Anforderungen*.

3. Was ist ein *Thread-Kontext*? Beschreiben Sie den Ablauf eines *Kontextwechsels*.

4. Unterscheiden Sie *kooperatives* und *preemptives* Scheduling.

5. Was ist ein *Quantum*? Welchen Einfluss hat die Länge des Quantums auf das Verhalten des Betriebssystems?

6. Während ihrer Laufzeit im System nehmen Threads mehrere *Zustände* im Scheduling ein. Zeichnen Sie ein Diagramm und visualisieren Sie diese Zustände und die Übergänge zwischen ihnen. Wodurch werden die Zustandsübergänge jeweils verursacht?

7. Betrachten Sie die folgende Menge an *Tasks* (*Task-Set*):

Thread ID	Startzeit t_s (in ms)	Ausführungszeit e (in ms)	Priorität
1	0	8	9
2	2	6	11
3	2	6	10
4	4	10	11

   (a) Zeichnen Sie für das gegebene Task-Set ein Gantt-Diagramm für einen *FiFo-Scheduler*.

   (b) Zeichnen Sie für das gegebene Task-Set ein Gantt-Diagramm für einen *prioritätenlosen Round-Robin-Scheduler* mit einer Quantenlänge von 3 Zeiteinheiten.

   (c) Zeichnen Sie für das gegebene Task-Set ein Gantt-Diagramm für einen *prioritätenbasierten Round-Robin-Scheduler* mit einer Quantenlänge von 3 Zeiteinheiten.

   Nehmen Sie dabei jeweils an, dass für einen Kontextwechsel keine Zeit verloren geht.

8. Berechnen Sie für die oben gezeichneten Gantt-Diagramme jeweils für jeden Task die *Wartezeit* (waiting-time) und die *Gesamtabarbeitungszeit* (turnaround-time), sowie den *Durchsatz* des Schedulers. Vergleichen Sie die Ergebnisse.

9. In Prioritätenbasierten Schedulern können Threads niederer Priorität *verhungern*. Beschreiben Sie, wie moderne Betriebssysteme mit diesem Problem umgehen und gehen Sie auf ein Beispiel genauer ein. Betrachten Sie dazu auch den Programmrahmen zur Starvation Avoidance:
   `https://gitlab.hpi.de/osm-teaching/operatingsystems2020-labs/osmthread`

   Der gegebene Programmrahmen enthält eine Implementierung für prioritätenbasiertes Scheduling von User-Mode Threads, sowie ein Beispielprogramm welches die Implementung verwendet, und ein Szenario demonstriert, in dem Starvation auftritt. Erweitern Sie den Programmrahmen um einen aus den Materialen bekannten Ansatz zur Starvation Avoidance.

10. Was ist ein *Multilevel-Queue* Scheduler? Beschreiben Sie die verschiedenen *Levels* des Multilevel-Queue Schedules *Completely Fair Scheduler* (CFS) in Linux und wozu sie verwendet werden.

11. Betrachten Sie den Programmrahmen zur *Shell Programmierung*:
`https://gitlab.hpi.de/osm-teaching/opsys19labs/fresh`

    Implementieren Sie die Funktion `do_fork_wait` in `fresh.c`, sodass Ihre shell einfache Programmaufrufe ausführen kann. Verwenden Sie dabei die Funktionen `fork`, und `wait`, sowie eine geeignete Funktion der `exec`-Familie. Die Manual-Seiten zu den Funktionen werden Ihnen weiterhelfen.

    Implementieren Sie die Funktionen `builtin_cd` sowie `builtin_exit` in der Datei `builtin.c`, sodass Ihre shell in der Lage ist, das aktuelle Verzeichnis zu wecheln, sowie sich zu beenden. Wieso müssen beide dieser Funktionen als builtins implementiert werden, und können nicht als Programme mit dem `fork`/`exec` Mechanismus aufgerufen werden?

    Welche weiteren shell builtins kennen Sie? Was fehlt sonst noch zu einer "echten", vollständigen shell? Lesen Sie dazu auch die Manual-Seite ihrer shell, z.B. *bash* oder *zsh*, sowie die entsprechenden Teile der POSIX Spezifikation:
    `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html`

12. Betrachten Sie den Programmrahmen zur Visualisierung von weihnachtlichen Thread Prioritäten und Scheduling-Verhalten:
`https://gitlab.hpi.de/osm-teaching/opsys19labs/lametta`

    Übersetzen Sie den Programmrahmen auf Linux und beobachten Sie das Verhalten. Lesen Sie den Code und beschreiben Sie, welches Systemverhalten des Completely Fair Schedulers (CFS) die Ausgabe auf der Kommandozeile visualisiert.