Weitere Hinweise zu den BS-Übungen

- Ab jetzt ist es **nicht** mehr möglich, Einzelabgaben in AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt das bitte **vorher** mit eurem Übungsleiter!
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.

Aufgabe 2: Thread-Synchronisation (10 Punkte)

Theoriefragen: Scheduling / Synchronisation (4 Punkte)

⇒ antworten.txt

Betrachtet die folgenden drei Prozesse, die gleichzeitig in die *Ready*-Liste aufgenommen werden (Ankunftszeit 0). Zur Vereinfachung sei angenommen, dass die CPU- und E/A-Stöße pro Prozess immer gleich lang und dem Scheduler bekannt sind. Die Prozesse führen periodisch erst einen CPU-Stoß und dann einen E/A-Stoß durch (Zeiteinheit: 1 ms).

Prozess-ID	CPU-Stoßlänge	E/A-Stoßlänge
A	6	2
B	3	3
C	2	4

Ihr könnt AnimOS (https://ess.cs.tu-dortmund.de/Software/AnimOS/ → CPU-Scheduling) verwenden, um euch einen ersten Überblick über die verschiedenen Schedulingverfahren zu machen.

Achtung: Während der Klausur habt ihr AnimOS nicht zur Verfügung. Ihr solltet die Lösungen daher auch ohne AnimOS erarbeiten können.

1. Wendet auf die Prozesse A, B und C die Scheduling-Verfahren *Virtual Round Robin* und *First-Come First-Served* aus der Vorlesung an. Der Scheduler gibt bei VRR jedem Prozess eine Zeitscheibe von 3 ms. Notiert die CPU- und E/A-Verteilung für die ersten 30 ms. (Annahmen: Prozesswechsel dauern 0 ms und sind damit zu vernachlässigen; es können mehrere E/A-Vorgänge parallel ausgeführt werden.)
   Stellt eure Ergebnisse wie in der folgenden ASCII-Zeichnung (Beispiel) dar, eine Spalte entspricht dabei 1 ms. Nutzt dazu in eurem Editor eine Monospace-Schriftart und keine Tabulatoren.
   ```
   +---+----------     +
   | A |CCCEE---CCC     | C: Prozess nutzt die CPU
   | B |---CCEEE--- ... | E: Prozess führt E/A-Operationen durch
   | C |-----CCC---     | -: Prozess ist laufbereit
   +---+----------     +
   ```
2. Was ist der Vorteil des Verfahrens *Virtual Round Robin* gegenüber *Round Robin*?
3. Beschreibt in eigenen Worten, wozu man Semaphoren verwendet.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.

Programmierung: Supermarktbesuch (6 Punkte)

In einem Supermarkt dürfen sich nicht mehr als 42 Kunden gleichzeitig aufhalten. Es gibt 5 Kassen, an denen gleichzeitig bezahlt werden kann. Die Kassen sollen durch leichtgewichtige Prozesse (POSIX-Threads) simuliert werden. Ein Bezahlvorgang dauert in der Simulation jeweils 3 Sekunden; nach dem Bezahlvorgang verlässt der Kunde den Supermarkt sofort. Zu Beginn der Simulation ist der Supermarkt voll.

a) Threads erzeugen, starten und beenden (2 Punkte)

⇒ aufgabe2_a.c

- Legt eine Variable an, die die Anzahl der Kunden im Supermarkt enthält.
- Startet mit **pthread_create(3)** einen Thread für jede Kasse, an der Kunden bezahlen können.
- Die Threads aller Kassen sollen terminieren, wenn der Supermarkt leer ist.
- Dabei sollen die Threads zunächst unsynchronisiert auf die Kundenanzahl-Variable zugreifen und nach jedem Bezahlvorgang die Kundenanzahl dekrementieren.
- Gebt vor jedem Verkauf und am Ende des Programms die Anzahl der Kunden im Supermarkt aus.
- Stellt dabei mit **pthread_join(3)** sicher, dass das Programm erst beendet wird, wenn alle Threads ihre Aufgabe erledigt haben.

Die Threads sollen in dieser Teilaufgabe noch nicht synchronisiert werden! Die Implementierung soll in einer separaten Datei `aufgabe2_a.c` abgegeben werden. Ihr müsst euer Programm mit dem Flag `-pthread` übersetzen, damit die richtigen Bibliotheken eingebunden werden.

b) Analyse (2 Punkte)

⇒ antworten.txt

Wahrscheinlich wird euch aufgefallen sein, dass Kunden den Supermarkt mehrfach verlassen, bzw. dass, wenn das Programm endet, weniger als 0 Kunden im Supermarkt sind.

1. Wie nennt man eine solche Situation?
2. Beschreibt schrittweise anhand von zwei parallel ausgeführten Threads, wie eines der beobachteten Probleme entstehen kann.

c) Synchronisation (2 Punkte)

⇒ aufgabe_2c.c

Löst dieses Problem, indem ihr mit einem Mutex den Zugriff auf die Kundenanzahl synchronisiert. Nutzt dazu **pthread_mutex_lock(3)** und **pthread_mutex_unlock(3)**, initialisiert die Mutexvariable zuvor mit **pthread_mutex_init(3)** und entfernt sie mit **pthread_mutex_destroy(3)**, wenn sie nicht mehr gebraucht wird. Das Dekrementieren der Kundenanzahl soll nun vor dem Bezahlvorgang stattfinden.

d) Zusatzaufgabe: (2 Punkte)

⇒ aufgabe_2d.c

Legt zwei weitere Threads an, die jeweils einen Eingang darstellen. Wenn der Supermarkt nicht voll ist, können sie jeweils genau drei neue Kunden einlassen, bevor sie 2 Sekunden warten. Die Eingänge sollen ansonsten mithilfe von Condition-Variablen darauf warten, dass Platz im Supermarkt frei wird. Verwendet hierzu die Funktionen **pthread_cond_signal(3)** und **pthread_cond_wait(3)**. Die Kassen-Threads sollen nicht mehr terminieren, wenn sich keine weiteren Kunden im Supermarkt befinden. Das Programm soll dann mit Strg+C beendet werden.

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denkt daran, dass viele Systemaufrufe fehlschlagen können! Fangt diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), gebt geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beendet euer Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist mit den folgenden Parametern aufzurufen:
  `gcc -Wall -D_GNU_SOURCE -pthread`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-ansi -Wpedantic -Werror -D_POSIX_SOURCE`
- Achtet darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.

Abgabe bis Donnerstag, 4. Juni 8:00 Uhr (Übungsgruppen in geraden Kalenderwochen) bzw. Dienstag, 9. Juni 8:00 Uhr (Übungsgruppen in ungeraden Kalenderwochen)