Abgabefrist Übungsaufgabe 3

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – eigene Übung U3 in KW 21 (24.05.-20.05.): Abgabe bis **Donnerstag, den 02.06.2022 12:00 Uhr**
- **W2** – eigene Übung U3 in KW 22 (31.05.-02.06.): Abgabe bis **Montag, den 06.06.2022 12:00 Uhr**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Ab jetzt ist es *nicht* mehr möglich, Einzelabgaben im AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt dies bitte *vorher* mit eurem Übungsleiter! Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- 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.
- Die Aufgaben sind über AsSESS (`https://sys-sideshow.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt **ein** Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder (Matrikelnummer, Vor- und Nachname erforderlich!). Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sobald eine Abgabe korrigiert wurde, kann das Ergebnis ebenfalls im AsSESS eingesehen werden.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8

Aufgabe 3: Deadlock (10 Punkte)

Ziel der Aufgabe ist es, das Entstehen, Erkennen und Auflösen von Deadlocks praktisch umzusetzen.

ACHTUNG: Zur Programmieraufgabe existiert eine Vorgabe in Form von C-Dateien mit vorimplementierten Code-Rümpfen, die ihr erweitern sollt. Diese Vorgabe ist von der Veranstaltungswebseite[4] herunterzuladen, zu entpacken und zu vervollständigen. Die Datei `vorgabe-A3.tar.gz` lässt sich mittels `tar -xzvf vorgabe-A3.tar.gz` entpacken.

In der Bar Hau-Wech arbeiten einige Bartender, die durch Kombination von verschiedenen Spirituosen versuchen, die Kunden mit neuen Rezeptkreationen glücklich zu machen. Jeder Bartender kann eigenständig für sich selbst Drinks anmischen, muss sich jedoch die einzelnen Spirituosen aus dem gemeinsam geteilten Vorratsregal nehmen.
Ein Bartender kann nur eine Spirituosenflasche gleichzeitig holen. Die gerade geholte Flasche wird erst von ihm zum Anmischen des aktuellen Drinks verwendet, bevor er die nächste Flasche holen kann. Nachdem er seine Rezeptkreation fertig gestellt hat, stellt der Bartender alle seine Spirituosenflaschen wieder zurück ins Regal. Bevor der Drink nicht fertig zubereitet ist, werden von keinem der Bartender Spirituosenflaschen zurück ins Regal gestellt.
Alle Bartender müssen sich aus dem gemeinsamen Regal bedienen. Für jede Spirituosensorte existiert nur eine begrenzte Anzahl an Flaschen, sodass eine Spirituose nicht zu jedem Zeitpunkt sofort verfügbar ist. Ein Bartender warten dann solange, bis ein anderer Bartender eine Flasche der benötigen Spirituosensorte zurückstellt.
Wir gehen davon aus, dass in den Drinks nur geringe Mengen der einzelnen Spirituosen benötigt werden. Leere Flaschen gibt es in diesem Szenario daher nicht.

Das oben beschriebene Szenario soll in den folgenden Aufgaben mithilfe von Semaphoren und POSIX-Threads implementiert werden. Die Implementierung soll in mehreren Schritten erfolgen, die in separaten Dateien (`a3_X.c`) abgegeben werden. Auf der Veranstaltungswebseite findet ihr zu dieser Aufgabe eine Vorgabe (`vorgabe-A3.tar.gz`). Sie enthält eine Reihe von C-Dateien, die bereits einen Rahmen für euer Programm zur Verfügung stellen und bildet somit das Grundgerüst für das beschriebene Szenario. Zudem findet ihr ein Makefile in der Vorgabe, sodass ihr das gesamte Projekt schnell mit den Befehlen `make` und `make clean` verwalten könnt. Beachtet, dass das Projekt erst nach Implementierung aller Teilaufgaben vollständig kompilierbar ist.

Theoriefragen (4 Punkte)

In der Bar Hau-Wech können leider Verklemmungen entstehen. Diese wollen wir uns hier näher anschauen.

1. Lest euch die obige Beschreibung der Ausgangssituation durch. Benennt die im Szenario beschriebenen Betriebsmittel. Sind sie konsumierbar oder wiederverwendbar?
2. Damit Verklemmungen entstehen können, müssen bestimmte Bedingungen erfüllt sein: **mutual exclusion**, **hold and wait** und **no preemption**. Beschreibt in eigenen Worten, wie im obigen Szenario eine *Verklemmung* auftreten kann und wodurch diese Bedingungen erfüllt werden. Damit wirklich eine *Verklemmung* entsteht, muss zur Laufzeit noch **circular wait** eintreten. Beschreibt kurz eine Situation, wie in der Bar eine *Verklemmung* auftritt.
3. Zur Verklemmungsvorbeugung wird in Teil c) der Programmieraufgabe eine der Bedingungen für Verklemmungen entkräftet. Um welche Bedingung handelt es sich? Erklärt zudem anhand des Szenarios, warum nun keine Verklemmungen mehr auftreten können.

⇒`antworten.txt`

[4] `https://sys-sideshow.cs.tu-dortmund.de/Teaching/SS2022/BS/Uebung/vorgabe-A3.tar.gz`

Programmieraufgaben (6 Punkte)

Wir wollen nun die Bar durch ein C-Programm darstellen. Dazu stellt die Vorgabe folgende Hilfsmittel bereit.

- Eine Spirituose wird durch die Struktur `struct ingredient` mit dem Typalias `ingredient_t` dargestellt. Sie enthält den Namen der Spirituose (`name`), die Verarbeitungsdauer in Sekunden (`time_needed`), die Anzahl der insgesamt vorhandenen Flaschen mit dieser Spirituose (`bottles`) und ein Semaphor, das die aktuell im Regal verfügbare Flaschenzahl angibt (`sem`).
- Die Spirituosen befinden sich im globalen Array `ingredients`; ihre Anzahl ist durch das Makro `INGREDIENTS_NUM` gegeben.
- Zu Beginn sind alle Flaschen im Regal. Das Holen einer Spirituose wird durch Belegung des Semaphors simuliert, das Zurückbringen durch Freigabe.
- Rezeptkreationen werden durch die Funktion `get_recipe` generiert und als Array von Zeigern auf die enthaltenen Spirituosen dargestellt. Rezepte haben die Länge `RECIPE_SIZE` und enthalten jede Spirituose höchstens einmal.
- Jeder Bartender wird durch einen Thread dargestellt. Das Makro `BARTENDER_NUM` gibt die Anzahl an Bartendern an.

ZUR ERINNERUNG: Fehlende und falsche Fehlerbehandlungen geben Punktabzüge!

Wir erachten den Hinweis als notwendig, da ein Großteil der bisherigen Abgaben, keine oder unzureichende Fehlerbehandlungen enthielten! Die **man-Pages**, speziell die Sektion **RETURN VALUE**, gibt Aufschluss darüber, welche Rückgabewerte einen Fehlerzustand anzeigen. Achtet dabei besonders darauf, dass z. B. bei **fork**, nur der Rückgabewert **-1** einen Fehler anzeigt, während der Rückgabewert **-1234** einen Erfolg darstellt. (Das heißt, dass im Beispiel von **fork** eine Prüfung **„== -1“** durchgeführt werden muss und nicht **„< 0“**)

Eine akzeptable Fehlerbehandlung kann dann bspw. daraus bestehen, dass einfach eine Ausgabe des aufgetretenen Fehlers getätigt (`perror("fork fehlgeschlagen");`) sowie eine *geeigneten Aktion zum Zurückkehren in einen sicheren Systemzustand* durchgeführt wird. Diese kann beispielsweise das Beenden des Programms mittels `exit(2)` sein.

a) Semaphoren initialisieren (1 Punkt)

Implementiert die Funktion `init_semaphores()`, welche die Semaphoren der Spirituosen mit passenden Werten initialisiert, sowie die Funktion `destroy_semaphores()`, die alle Semaphoren wieder sauber entfernt.

⇒ `a3_a.c`

b) Fäden implementieren (3 Punkte)

Implementiert nun die Funktion `work()`, die von den Bartender-Threads ausgeführt wird. Der Ablauf soll folgender sein:

- Rezept generieren
- Für jede der enthaltenen Spirituosen:
  - Eine Spirituosenflasche aus dem Regal holen
  - Spirituose verarbeiten (Warten der entsprechenden Zeit mit `sleep()`)

- Sobald **alle** Spirituosen aus dem Rezept **verarbeitet** wurden: Alle Zutaten wieder zurückbringen
- Dieser Ablauf soll `RECIPES_PER_BARTENDER` mal durchgeführt werden, bevor der Bartender nach Hause geht (Thread beenden).

⇒ `a3_b.c`

mit `make b` kompilieren und anschließend mit `./a3_b` ausführen.

c) Verklemmungsvorbeugung (2 Punkte)

Um Deadlocks vorzubeugen, ändern die Bartender nun ihre Vorgehensweise:

- Zunächst wird das gesammte Regal reserviert.
- Dann holt der Bartender sich alle benötigten Flaschen direkt nacheinander. Wenn eine Flasche nicht verfügbar ist, wartet er ganz normal.
- Dann gibt er das Regal wieder frei und fängt erst dann an, die Spirituosen zu verarbeiten.
- Schließlich bringt er die Flaschen zurück. Hierfür muss das Regal nicht reserviert werden.

Benutzt das globale Mutex `lock`, um das Reservieren des Regals zu simulieren.

Kopiert für diese Aufgabe eure Lösung für **Aufgabe b)** und passt sie an.

⇒ `a3_c.c`

Ihr könnt das Programm mit `make c` kompilieren. Die ausführbare Datei wird als `a3_c` abgelegt.

d) Zusatzaufgabe: Verklemmungsauflösung (2 Sonderpunkte)

Die Bartender probieren diesmal eine andere Methode, um das Problem zu beheben: Sie warten immer nur eine bestimmte Zeit auf eine Spirituose. Wenn diese nicht rechtzeitig verfügbar wird, wird die gesamte Bar zurückgesetzt.

Kopiert dazu eure Lösung für **Aufgabe b)** erneut und passt sie so an, dass die Bartender nur so lange warten, wie ein maximal zeitintensives Rezept (Rezeptlänge mal längste Verarbeitungsdauer der Spirituosen) brauchen würde.

Nach Ablauf der Zeit soll die globale Variable `restart = 1` gesetzt werden (dadurch startet die main-Funktion die Threads neu, nachdem alle beendet sind), alle anderen Threads abgebrochen und schließlich auch der aktuelle Bartender-Thread beendet werden.

Schaut euch die Funktionen `sem_timedwait`, `clock_gettime`, `pthread_cancel, pthread_self` und `pthread_equal` an.

⇒ `a3_d.c`

Ihr könnt das Programm mit `make d` kompilieren. Die ausführbare Datei wird als `a3_d` abgelegt.

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.
- Unter *MacOS* stehen die hier verwendeten (unnamed) POSIX Semaporen nicht zur Verfügung. Nutzt daher ein Linux-Systeme wie z.B. auf den Pool-Rechner oder die bereitgestellte VM. Bei Fragen oder Problemen könnt ihr euch gerne an unseren **HelpDesk** wenden.