Abgabefrist Übungsaufgabe 3

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

- **W1** – eigene Übung U3 am 01./02.06.: Abgabe bis **Donnerstag, den 10.06.2021 08:00**
- **W2** – eigene Übung U3 am 08./09.06.: Abgabe bis **Dienstag, den 15.06.2021 08:00**

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://ess.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 herunterzuladen, zu entpacken und zu vervollständigen. Die Datei `vorgabe-A3.tar.gz` lässt sich mittels `tar -xzvf vorgabe-A3.tar.gz` entpacken.

> Die Archäologen des Kronos Institut führen Ausgrabungen an verschiedenen Ausgrabungsstätten durch. Jedem Archäologen wird eine Ausgrabungsstätte zugeteilt. Leider sind die Fördermittel für die Ausgrabung begrenzt, daher sind manche Messgeräte und Werkzeuge nur in geringen Stückzahlen verfügbar und lagern zentral im Institut. Welche Geräte für die jeweilige Ausgrabungsstätte benötigt werden, ergibt sich erst nach und nach im Laufe der Ausgrabung. Wird ein neues Gerät benötigt, so holt der Archäologe dieses vom Institut und fährt mit der Ausgrabung fort. Ist das Gerät nicht im Institut, dann wartet er darauf, dass es zurückgebracht wird. Solange die Ausgrabung nicht abgeschlossen ist, befinden sich noch alle Geräte in Benutzung und können daher nicht zurück zum Institut gebracht werden. Ist eine Ausgrabung beendet, dann bringt der Archäologe alle Geräte zum Institut zurück, und ihm wird eine neue Ausgrabungsstätte zugeteilt.

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 (5 Punkte)

Bei den Ausgrabungen können leider Verklemmungen entstehen. Diese wollen wir uns hier näher anschauen.

1. Benennt in dem oben beschriebenen Szenario das Betriebsmittel. Ist es konsumierbar oder wiederverwendbar?
2. Beschreibt in eigenen Worten, wie im oben beschriebenen Szenario eine Verklemmung auftreten kann. Wodurch werden dabei die Bedingungen `mutual exclusion`, `hold and wait`, `no preemption` und `circular wait` erfüllt?
3. In den Programmieraufgen 3 b) und 3 c) werden unterschiedliche Methoden verwendet um mit Deadlocks zu verfahren. Erklärt, in eigenen Worten, die Vor- und Nachteile der beiden Methoden.

⇒`antworten.txt`

Programmieraufgaben (5 Punkte)

Wir wollen nun die Ausgrabungen des Archäologischen Instituts durch ein C-Programm darstellen. Dazu stellt die Vorgabe folgende Hilfsmittel bereit (Schnittstellen, Konstanten und globale Variablen siehe `vorgabe.h`).

- Ein Gerät wird durch die Struktur `ExcavationTool` dargestellt. Sie enthält den Namen des Geräts (`name`) und eines Semaphors, welcher die aktuell im Institut verfügbare Anzahl angibt (`sem`).

- Die Geräte befinden sich im globalen Array `tools`; ihre Anzahl ist durch die Konstante `TOOL_COUNT` gegeben.
- Zu Beginn sind alle Geräte im Institut. Das Holen eines Geräts wird durch Belegen des Semaphors simuliert, das Zurückbringen durch Freigabe.
- Die Fahrzeiten zum Institut können ignoriert werden.
- Durch den Aufruf von `get_next_step()` wird eine Struktur `ExcavationStep` zurückgegeben, welche den nächsten Ausgrabungsschritt beschreibt.
- Jeder Archäologe wird durch einen Thread dargestellt. Die Konstante `ARCHAEOLOGIST_COUNT` gibt die Anzahl an Archäologen an.

a) Fäden implementieren (3 Punkte)

Implementiert die Funktion `work()` in der Datei `a3_a.c`, die von den Archäologen-Threads ausgeführt wird. Der Ablauf soll folgender sein:

- Planen des nächsten Arbeitsschrittes: `get_next_step()`
  - Das Gerät aus dem Institut holen, falls nicht vorhanden (Semaphor `tool[i].sem` belegen).
  - Ausgrabungsschritt durchführen (Warten der entsprechenden Zeit `ExcavationStep.time`).
  - Falls in einem Arbeitsschritt das Feld `finished == true` ist, dann soll die Ausgrabung nach diesem beendet werden (Semaphor freigeben).
- Dieser Ablauf soll `EXCAVATIONS_PER_ARCHAEOLOGIST` mal **erfolgreich** durchgeführt werden, bevor der Archäologe in Rente geht (Thread beenden).

⇒ `a3_a.c`

Ihr könnt das Programm mit `make a` kompilieren und anschließend mit `./a3_a` ausführen.

b) Verklemmungsauflösung (2 Punkte)

Geht zur Verklemmungsauflösung wie folgt vor:

- Begrenze die Wartezeit für Geräte auf 10 Sekunden.
- Wenn das Gerät danach nicht verfügbar ist, bringe alle Geräte zurück und warte 1 Sekunde.
- Starte dann mit einer neuen Ausgrabung.
- Die erfolglose Ausgrabung soll nicht zu den erledigten Ausgrabungen zählen.

Nehmt dazu ggf. die Funktionen **sem_timedwait(3)** und **clock_gettime(3)** zu Hilfe.

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

⇒ `a3_b.c`

Ihr könnt das Programm mit `make b` kompilieren. Die ausführbare Datei wird als `a3_b` abgelegt.

c) Zusatzaufgabe: Verklemmungsvorbeugung (2 Sonderpunkte)

Durch neuste Bodenradartechnik ist es nun möglich, im Vorfeld alle Ausgrabungsschritte zu planen.

- Zunächst werden alle Ausgrabungsschritte bestimmt, legt hierfür ein Array der Länge 100 an.
- Dann wird der Geräteraum des Instituts reserviert (Mutex `lock` aus `vorgabe.h`).

- Dann holt sich der Archäologe *alle benötigten Geräte* direkt nacheinander. Wenn ein Gerät nicht verfügbar ist, wartet er ganz normal.
- Dann gibt er das Inventar wieder frei und fängt mit der Ausgrabung an.
- Schließlich gibt er alle Geräte wieder zurück. Hierfür muss der Geräteraum nicht reserviert werden.

Benutzt die globale Mutex-Variable `lock`, um das Reservieren des Geräteraums zu simulieren.

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

⇒ `a3_c.c`

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