Übung Nr. 3:

Das Problem der dinierenden Philosophen:

Um einen Tisch sitzt eine Gruppe von (meist: 5) Philosophen. Die Tätigkeit jedes von ihnen besteht darin, nachzudenken, hungrig zu werden und zu essen, um wieder nachzudenken etc.. Auf dem Tisch steht vor jedem der Philosophen ein Spaghetti-Teller, zwischen zwei Tellern liegt jeweils eine Gabel (bzw.: ein Eßstäbchen). Da jeder Philosoph zwei Gabeln (bzw. Stäbchen) zum Essen benötigt, können Tischnachbarn nicht gleichzeitig essen; ebensowenig kann ein Philosoph allein essen. Gesucht ist die Implementierung einer algorithmischen Lösung, die unter Berücksichtigung der Gegebenheiten allen Mitspeisenden regelmäßiges Essen sichert.

Unter http://homepages.thm.de/christ/ sind als „Funktionsmuster“ drei Implementierungsstufen zu finden:

- **1SophStarv** enthält die „Infrastruktur“ des Diners und hält die „Eß-Regel“ ein: Ein bis fünf Starts können erfolgen, solange keiner der Prozesse erstmalig den Zustand des Essens erreicht hat. Der 6. und alle weiteren sowie verspätete Starts werden abgewiesen. Nach Beendigung eines Prozesses (<Esc>) beenden auch die übrigen. Alle Zustandswechsel werden durch (beliebigen) Tastendruck eingeleitet. Die Funktionsweise läßt sich gut bei drei Starts erkennen; dabei kann es (zufällig) zum Verhungern eines der Philosophen kommen. Verhungern kann man provozieren und vorführen am leichtesten bei vier Starts, wenn diagonal gegenübersitzende Philosophen (z.B. Indizes 1 und 3) abwechselnd essen, während die anderen beiden versuchen, auf die Ressourcen (Gabeln) zuzugreifen.
- **2SophNoStarv** enthält zusätzlich einen Schutz gegen Verhungern, weil dort vorausgesetzt wird, daß ein Philosoph nur dann den Zustand des Hungers erlangen darf, wenn keiner seiner Nachbarn hungrig ist. Am besten macht sich der Unterschied zur ersten Realisierung auch hier bei vier Starts bemerkbar.
- **3SophNoStarvTimAtom** ist die am weitesten ausgebaute Stufe: Nach Erreichen des Eß-Zustands wird ein etwas abgewandelter Timer aktiv: Der jeweilige Zustand wird entweder für die eingestellte Zeit eingehalten oder aber vorzeitig verlassen, wenn <CR> gedrückt wird. Die aktuelle (im Code fest installierte) Einstellung ist: Essen 5, Nachdenken 3 Sekunden lang – bewußt so gehalten, um „Staus“ unter hungernden Philosophen zu provozieren (Empfehlung: Ausführung mit 5 Philosophen). Zudem enthält diese Realisierung eine zusätzliche („hedonistische“, s.u.) „Atomisierung“: Bevor ein Philosoph seine Daten in die Datei überträgt (`Sema.txt` – zum Zeitpunkt der Übertragung bereits blockiert), prüft er nochmal, ob inzwischen (d.h. seit der Überprüfung des Datei-Inhalts) ein anderer Prozeß ihm zuvorgekommen ist, so daß der eigene Zustandswechsel einen Regelbruch bedeuten würde.

Der hierfür entwickelte Code ist enthalten in der Header-Datei `Sophos.h`, der Hilfsdatei `SophAux.c`, der Hauptdatei `Sophos.c` und der zusätzlichen Datei `Sophie.c`.

Die Datei `Sophie.c` enthält drei Funktionen, die alle unvollständig sind:

- `common()` zur Unterbringung der global benötigten Struktur
- `take_forks()` zur Philosoph-Anmeldung als hungrig oder essend
- `freserve ()` zur Sperrung der Datei `Sema.txt`

Für eine erste erfolgreiche Compilierung dieses Code-Fragments muß zunächst `common()` ergänzt werden.

Meldungen:

Neben den bereits geschilderten werden bei der Ausführung der o.a. Muster auch weitere Meldungen ausgegeben. Sie werden im folgenden erläutert; für die eigene Implementierung sind sie freiwillig oder bereits im Code enthalten.

Meldung	Anlaß
`"Ich habe den Index %d!"`	Erfolgreiche Anmeldung zum Diner
`"Zugriffsversuch"`	Versuch, CS (Gabeln, `Sema.txt`) für sich zu reservieren
`"Ich will nicht mehr nachdenken!“`	Philosoph versucht erfolglos, sich `HUNGRY` zu melden (Tischnachbarn sind auch hungrig).
`"(nicht-atomare Zone...)“`	Zwischen Prüfung des Zustands der Tischnachbarn und Anmeldung als `HUNGRY` oder `EATING` hat sich einer der Nachbarn erfolgreich angemeldet, so daß die z.Z. laufende eigene Meldung zurückgenommen werden mußte.
`"Ich bin hungrig!“`	Erfolgreiche Anmeldung als `HUNGRY`
`"Ich koennte doch essen! ...“`	Keiner der beiden Tischnachbarn ißt, aber der Philosoph kann dennoch nicht essen: Er kann sich nicht hungrig melden, weil mindestens einer der beiden Nachbarn hungrig gemeldet ist (Forderung nach „Fortschreiten“, s. Vorlesung).

Aufgabe:

Der Code in der Datei `Sophie.c` ist zu drei Implementierungen nach der o.a. Beschreibung zu ergänzen.

Vorgaben für die Implementierung:

In Anlehnung an die o.a. Muster sind vor allem folgende Merkmale gefordert:

- Eine beliebige Anzahl von Philosophen (2 bis `MAXPHIL`, mit den Indizes 0 bis `MAXPHIL-1`) kann mitwirken. Erreicht ein Philosoph den Zustand `EATING`, ohne daß mindestens einem weiteren ein Index zugewiesen wurde, so terminiert das Programm mit der Ausgabe des Makros `MINTWO`.
- Beim Versuch, sich einem eröffneten Diner anzuschließen (d.h.: nachdem der erste Philosoph angefangen hat zu essen), terminiert das Programm mit der Ausgabe des Makros `TOOLATE`. Bei regulärer Terminierung (`<Esc>` bei einem der laufenden Prozesse) melden sich nacheinander alle laufenden Prozesse mit `THATSIT` ab.
- Die Kommunikation unter den Prozessen erfolgt über die formatierte Datei `Sema.txt`. Bei Programm-Start kann sie vollkommen fehlen (Löschung der Datei), als leere Textdatei existieren (Löschung des Inhalts) oder von einer früheren Ausführung mit ordnungsgemäßer Beendigung stammen. Es finden aber keine Plausibilitätsüberprüfungen statt – das ist z.B. dann problematisch, wenn Fenster laufender Prozesse geschlossen werden (z.B. per Mausklick), ohne daß das Programm ordnungsgemäß beendet wird: Einträge über nicht mehr existente Mitdinierende finden sich danach in `Sema.txt`. Ihr Inhalt kann wie üblich während der Ausführung der o.a. Muster eingesehen werden (z.B. mit Editor).
- Es sollen keine globalen Variablen verwendet werden.

Die (relativ schwer zu unterscheidende) Programm-Version mit zusätzlicher Überprüfung ist nur für Hedonist/inn/en vorgesehen (d.h.: freiwillig).

Vorgaben für die Programm-Tests:

Bei Einhaltung der obigen Vorgaben sollte folgende Testreihe erfolgreich verlaufen (Vorführung vor dem Einreichen):

- Erste Programm-Version (nur `MIT_VERHUNGERN` ist aktiviert, vgl. 1SophStarv) mit einem Philosophen laufen lassen. Vorführung der „Weigerung“ des Philosophen, allein zu essen (Makro `MINTWO`). Mit `<Esc>` beenden.
- Datei `Sema.txt` löschen.
- Dieselbe (erste) Programm-Version mit 3 Philosophen laufen lassen; Abweisung eines verspäteten 4. Philosophen vorführen. Mit `<Esc>` beenden.
- Inhalt von `Sema.txt` löschen, Neustart mit 4 dinierenden Philosophen; Aushungern zweier Philosophen durch die anderen zwei vorführen. Mit `<Esc>` beenden.
- Neu compilieren (beide `#define`'s deaktiviert, entsprechend 2SophNoStarv) und mit 4 Philosophen laufen lassen (`Sema.txt` von zuvor übernehmen); vorführen der Verhinderung eines Aushungerns. Mit `<Esc>` beenden.
- `Sema.txt` übernehmen, für die 3. Version (nur `MIT_TIMER` ist aktiviert, entsprechend 3SophNoStarvTimAtom) neu compilieren; mit 5 Philosophen einen „kurzen Dauertest“ inkl. Eingriff in den Timer (`<CR>`) vorführen.