Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

Gegeben sei das in Abbildung 2 veranschaulichte Szenario:

- Der Zugang zur Mensa erfolgt über die zentrale *Türe*. In der *Essensausgabe* der Mensa wählt man sein Essen am *Menü* aus, holt sein Essen und geht zur *Kasse*, hinter der sich die Tische befinden.
- Einigen Studenten fällt beim Lesen des Menüs ein, dass sie eine wichtige Vorlesung vergessen haben, die sie nicht versäumen wollen und verlassen daher sofort wieder die Mensa.
- Die anderen essen, geben ihr Tablett ab und verlassen dann die Mensa über den Ausgang, der wieder zur zentralen Türe führt.

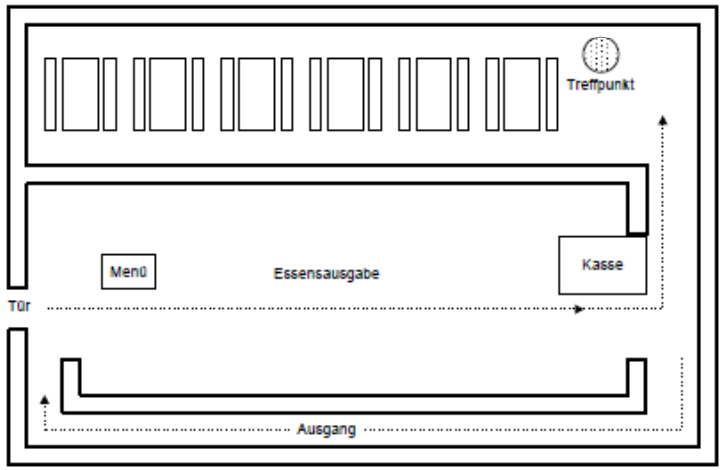

Abbildung 2: Übersicht über Mensa.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

- Achten Sie darauf, ob in der jeweiligen Teilaufgabe der Studentenprozess (Alg. 3) und/oder der Kassierer-Prozess (Alg. 2) erweitert werden sollen. Globale Variablen (Alg. 1) können in jeder der Teilaufgaben verwendet werden. Andere denkbare Prozesse sind **nicht** Bestandteil dieser Aufgabe.
- Kennzeichen Sie immer deutlich, welches Codegerüst Sie jeweils erweitern.
- Achten Sie auf eine effiziente Implementierung (insbesondere kein *busy waiting*).
- Schreiben Sie Ihre Lösungen **nicht** auf diese Angabe.
- Aus Gründen der Übersichtlichkeit ist in dieser Aufgabe nur die männliche Form (Student, Kassierer) angegeben. Selbstverständlich können damit auch Studentinnen bzw. eine Kassiererin gemeint sein.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

Verwenden Sie die folgende Notation:

- `int i=0;`, wenn Sie eine ganzzahlige Variable `i` mit Initialisierungswert `0` benutzen wollen.
- `Semaphore semBeispiel(1);`, um einen mit `1` initialisierten Semaphor `semBeispiel` zu deklarieren.
  - `down(semBeispiel);`, um den Semaphor `semBeispiel` anzufordern.
  - `up(semBeispiel);`, um den Semaphor `semBeispiel` freizugeben.
- `Barrier barBeispiel(5);`, um eine mit `5` initialisierte Barriere zu deklarieren.
  - `wait(barBeispiel);`, um den aufrufenden Prozess an der Barriere `barBeispiel` zu synchronisieren.
- `MessageQueue qBeispiel;`, um eine Nachrichten-Warteschlange (*message queue*) `qBeispiel` zu deklarieren.
  - `sendMsg(q, m)`, um die Nachricht `m` über die Warteschlange `q` zu verschicken.
  - `m = recvMsg(q)`, um eine Nachricht von der Warteschlange `q` zu empfangen und diese in `m` zu speichern. Falls `q` leer ist, blockiert der Aufruf von `recvMsg` so lange, bis wieder ein Element in die Warteschlange geschrieben wurde.
- `Signal sigBeispiel;`, um ein pures (nicht-wertbehaftetes) Signal `sigBeispiel` zu deklarieren.
  - `sendSignal(sigBeispiel)`, um das Signal `sigBeispiel` zu verschicken. Die Funktion kehrt nach dem Aufruf sofort zurück, es gehen keine versendeten Signale verloren.
  - `waitForSignal(sigBeispiel)`, um auf das Auftreten des Signals `sigBeispiel` zu warten. Falls das Signal noch nicht angelegen hat, blockiert der Aufruf von `waitForSignal` bis zu dessen Auftreten.
- Kennung, persönlicher Semaphor
  - Jeder Student besitzt eine eindeutige Kennung vom Typ `int`, die in einer **lokalen** Variable (`ID`) gespeichert ist, d.h. diese ist nur innerhalb des Studentenprozesses zugänglich.
  - Jedem Student ist zudem ein Semaphor zugeordnet, auf den über die **global** zugängliche Funktion `Semaphore getSemaphoreById(int)` zugegriffen werden kann. Gehen Sie davon aus, dass die Semaphoren jeweils mit `0` initialisiert wurden.

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

Algorithmus 1 Codegerüst für Deklaration globaler Variablen

```
1: // Deklaration von globalen Variablen, Semaphoren, etc.
2:
3: Tür tür;
```

Algorithmus 2 Codegerüst für Kassiererprozess

```
1: // Deklaration von lokalen Variablen, Semaphoren, etc. für Kassiererprozess
2:
3: // Code für Kassiererprozess
4:
5: while (!feierabend()) do
6:
7:     kassiere();
8:
9:     schaueAufDieUhr();
10:
11: end while
12:
```

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

Algorithmus 3 Codegerüst für Studentenprozess

```
1:  // Deklaration von lokalen Variablen, Semaphoren, etc. für Studentenprozess
2:  Essen essen;
3:  int ID; // Eindeutige Kennung für jeden Studentenprozess
4:
5:  // Codegerüst für Studentenprozess
6:
7:  betreteMensa(tür);
8:
9:  essen = liesMenü();
10:
11: if (essen == KeinEssen) then
12:
13:     verlasseMensa(tür);
14:
15:     return
16:
17: end if
18:
19: holeEssen();
20:
21: bezahle();
22:
23: essen();
24:
25: gibTablettZurück();
26:
27: verlasseMensa(tür);
28:
```

Klausur WS 10/11 – Nebenläufigkeit (30 P = 30 min)

Programmieraufgaben:

a) Ergänzen Sie den Studentenprozess (Alg. 3) so, dass sich zu jedem Zeitpunkt maximal ein Student in der zentralen Türe befindet.

b) In der Essensausgabe haben maximal 50 Studenten Platz. Ergänzen Sie den Studentenprozess (Alg. 3) so, dass diese Einschränkung sichergestellt wird, und die Studenten vor der Mensa warten, falls kein Platz in der Essensausgabe ist.

c) Stellen Sie sicher, dass Studenten in der gleichen Reihenfolge ihr Essen bezahlen, in der sie die Auswahl des Essens abgeschlossen haben. Ergänzen Sie hierzu den Studentenprozess (Alg. 3) und/oder den Kassiererprozess (Alg. 2).

d) Die Studenten Tina, Tim und Thomas haben vereinbart, sich hinter der Kasse zu treffen, um gemeinsam einen freien Tisch zu suchen. Erstellen Sie eine effiziente Implementierung im Studentenprozess (Alg. 3), mit der die Verabredung der drei umgesetzt werden kann. Gehen Sie davon aus, dass die Funktion `is3T()` den Wert `true` zurückliefert, falls der Aufrufer einer der drei Studenten ist.

Allgemeine Fragen (**keine** Implementierung nötig!):

e) Dort entdecken die drei eine Gruppe von Philosophen, die jeweils mit einer Gabel in ihrer linken Hand hungrig vor ihren Tabletts sitzen. Erläutern Sie *kurz* vier Bedingungen, die erfüllt sein müssen, damit es zu einer solchen Verklemmung (*deadlock*) kommen kann.

f) Den Studenten wird klar, dass sie soeben ein Beispiel für das Problem der Aussperrung (*starvation*) erlebt haben. Nennen Sie zwei weitere Probleme der nebenläufigen Programmierung und erläutern Sie *kurz* mögliche Lösungen.