Bearbeitungszeitraum: eine Woche

a) Implementieren Sie das Producer und Consumer Problem aus der Vorlesung in der Programmiersprache C mit Posix Threads (PThreads).

I) Auf den Praktikumsrechnern sollte ein Posix C vorinstalliert sein. Dann können Sie bei Unterpunkt II) weitermachen.
Wenn Sie ihren eigenen Rechner verwenden wollen, installieren Sie eine C Entwicklungsumgebung mit Posix Threads. Hier als Beispiel CLion.

i) Installieren Sie die CLion IDE https://www.jetbrains.com/shop/eform/students. Als StudentIn unserer Hochschule können Sie mit Ihrer Hochschul - E-Mail-Adresse eine einjährige Lizenz kostenlos erhalten.

ii) Legen Sie unter File->New->Project… ein neues Projekt an. In dem Location Textfeld löschen Sie den untiteld1 Teil des Projektpfades und ersetzen Sie ihn durch den Projekt(ordner)namen ProducerConsumer. Wählen Sie in dem linken Menü C executable aus. Drücken Sie den Create Knopf unten rechts.

iii) In dem neuen Projekt wird eine main.c Datei zum Editieren geöffnet. Löschen Sie den Inhalt von main.c. Kopieren Sie das diesem Aufgabenblatt unten angehängte Programm in main.c. Starten Sie das Programm mit dem grünen Pfeil nach rechts ungefähr rechts oben in CLion. Wenn alles funktioniert, sollten jetzt in einem Konsolenfenster unregelmäßig abwechselnd o und x ausgegeben werden. Beenden Sie das Programm mit dem roten Knopf links neben dem Konsolenfenster.

II) Entfernen Sie den obigen Inhalt von main.c und implementieren Sie nun dort das Producer Consumer Problem aus der Vorlesung (mit einem Integer data als Puffer) mit drei Produzenten und zwei Konsumenten.

- Dabei gelten folgende Bedingungen:
  Jeder Producer erzeugt fortlaufend Zahlen (`zahl++`) in einer Endlosschleife und versucht jede neue Zahl in den Puffer `data` einzutragen. Kurz vor dem Eintragen mit `put(int zahl)` gibt der Producer seine Thread ID und die Zahl auf dem Bildschirm aus:
  ```
  printf("[produce]:: Thread %d hat Zahl %d produziert \n ", id, zahl);
  ```
  Die Thread ID kann ein Thread mit
  ```
  int id = pthread_self();
  ```
  bestimmen.
- Jeder Consumer versucht in einer Endlosschleife die Zahl aus dem Puffer `data` auszulesen. Bei erfolgreichem Auslesen gibt der Consumer seine Thread ID und die Zahl auf dem Bildschirm aus:
  ```
  printf("[consume]:: Thread %d hat Zahl %d konsumiert \n ", id, zahl);
  ```
- Jeweils genau vor Aufruf von `pthread_cond_wait` in der `put()` Funktion gibt ein Producer Thread folgendes aus:
  ```
  printf("[put]:: Thread %d geraet in WAITING \n ", id);
  ```
  Direkt nach Verlassen gibt ein Producer Thread dies aus:
  ```
  printf("[put]:: Thread %d macht weiter \n ", id);
  ```

Entsprechendes ist für die `get()` Funktion vorzusehen.

Um diese Aufgabe zu lösen, empfehle ich, das Producer/Consumer Java Programm aus der Vorlesung in C zu übersetzen. Dabei gelten folgende „Übersetzungsregeln“.

Thread Erzeugung

Library: `include <pthread.h>`
Thread ID Variable: `pthread_t derThread;`
Thread Erzeugung und start: `pthread_create(&derThread, NULL, &produce, null);`
Thread Funktion d.h. public void run() in Java: `void* produce(void * unused) {…}`

Synchronisation

zuerst Mutex Variable anlegen:
```
pthread_mutex_t derMutex;
```
dann Mutex initialisieren:
```
pthread_mutex_init(&derMutex, NULL);
```
eintreten in eine synchronized Methode in Java:
```
pthread_mutex_lock(&derMutex);
```
verlassen von synchronized Methode in Java:
```
pthread_mutex_unlock(&derMutex);
```

wait / notify

zuerst Condition Variable anlegen:

```
pthread_cond_t dieCondition;
```

dann Condition Variable initialisieren:

```
pthread_cond_init(&dieCondition, NULL);
```

wait() in Java:

```
pthread_cond_wait(&dieCondition, &derMutex);
```

notify() in Java:

```
pthread_cond_signal(&dieCondition);
```

notifyAll() in Java:

```
pthread_cond_broadcast(&dieCondition);
```

Anhang:

```
#include <pthread.h>
#include <stdio.h>

/* A mutex protecting printf.  */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* Prints x's . The parameter is unused. Does not return.
*/
void* print_xs(void* unused) {
     while (1) {
          pthread_mutex_lock (&mutex);
          printf("x\n");
          pthread_mutex_unlock (&mutex);
     }
     return NULL;
}
/* The main program. */
int main() {
     pthread_t thread_id;
     /* Create a new thread. The new thread will run the
print_xs
      function. */
     pthread_create(&thread_id, NULL, &print_xs, NULL);
     /* Print o's continuously  */
     while (1) {
          pthread_mutex_lock (&mutex);
          printf("o\n");
          pthread_mutex_unlock (&mutex);
     }
     return 0;
}
```