| Technische Universität Berlin<br>Fachgebiet<br>Komplexe und Verteilte IT-Systeme<br><br>Sommersemester 2015 | **Aufgabenblatt 4**<br>zu – Systemprogrammierung –<br>Prof. Dr. Odej Kao,<br>Dr. Peter Janacik, Tutoren |
|---|---|
| Abgabetermin: | [1] – 08.06.2015<br>[2] – 15.06.2015 |

# Aufgabe 4.1: Koordination (0,5 Punkte) (Theorie[1])

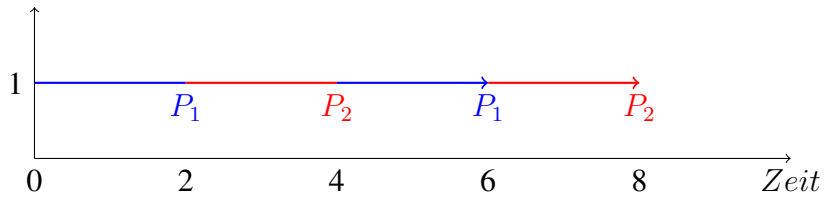


Abbildung 1: Konzept A

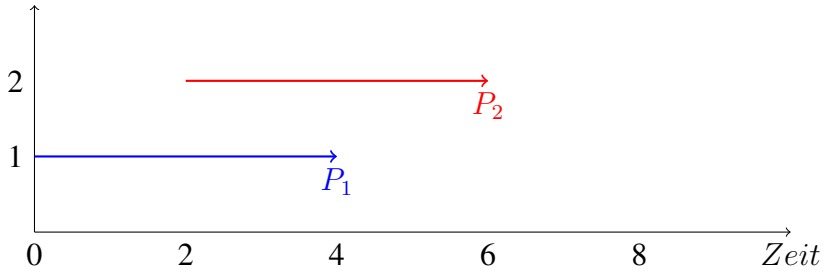


Abbildung 2: Konzept B

Beantworten Sie folgende Fragen:

a) Welche in der Vorlesung vorgestellten Konzepte werden in den Abbildungen 1 und 2 beschrieben?

b) In welchem Verhältnis stehen sie zueinander und welche Vorraussetzungen müssen für Konzept B gegeben sein?

c) Wann ist eine Koordination zwischen P1 und P2 notwendig? Spielt es hierbei eine Rolle, ob Konzept A oder Konzept B Anwendung findet?

## Aufgabe 4.2: Synchronisation und Kooperation (1 Punkt) (Theorie[1])

Die Threads T1 und T2 sollen abwechselnd „JA“ und „NEIN“ in eine Datei schreiben.

```
1 T1:
2 while(1) {
3   fprintf(f,"J");
4   fprintf(f,"A");
5
6
7 }
```

Listing 1: Thread 1

```
1 T2:
2 while(1) {
3   fprintf(f,"N");
4   fprintf(f,"E");
5   fprintf(f,"I");
6   fprintf(f,"N");
7 }
```

Listing 2: Thread 2

a) Begründen Sie, ob es sich um eine sogenannte „Race-Condition“ handelt. Geben Sie ein geeignetes Beispiel an, wie die Ausgabe aussehen könnte, wenn beide Threads unverändert ausgeführt werden.

b) Das Schreiben in die Datei soll als kritischer Abschnitt geschützt werden. Stellen Sie mit Hilfe der in der Vorlesung vorgestellten Verfahren zur Prozess- oder Threadsynchronisation sicher, dass sich die Ausgaben beider Threads nicht überschneiden. Begründen Sie wieder, ob es sich um eine „Race-Condition“ handelt.

c) Die beiden Threads sollen jetzt kooperativ das „Problem“ lösen, indem T2 seine Ausgabe erst startet, nachdem T1 seinen Text ausgegeben hat und umgekehrt. Stellen Sie sicher, dass abwechselnd „JA“ und „NEIN“ geschrieben wird.

d) Vergleichen Sie die Lösung von b) und c). Wie unterscheiden sich die Ausgaben?

## Aufgabe 4.3: Recherche: Spurious Wakeup (0,5 Punkte) (Theorie[1])

Was versteht man unter dem Begriff *Spurious Wakeup*?
Beachtet ihre Lösung von Aufgabe 4.2 c) die Problematik von Spurious Wakeups?
Ändern Sie gegebenenfalls Ihren Code so, dass Spurious Wakeups berücksichtigt werden.

## Aufgabe 4.4: Synchronisation: Produzent – Konsument (1. Tutorium) (Tafelübung)

Es soll ein Dönerladen synchronisiert werden. In diesem Dönerladen gibt es einen Spieß und mindestens zwei Verkäufer. Der Spieß kann nur von einem Verkäufer gleichzeitig genutzt werden. Auf den Salat kann gleichzeitig von beiden zugegriffen werden.

a) Wir machen uns ersteinmal keine Sorgen um zu viel produzierte Döner.
Im Folgenden ist der Verkäufer-Prozess in Pseudocode beschrieben. Ergänzen Sie die nötige Synchronisation.

```
Variablen:
int döner = 0;

Verkäufer:
while(true){
        fleischSchneiden();
        salatUndSo ß e();
        döner ++;
}
```

b) Kunden betreten in unvorhersagbaren Abständen den Laden, um einen Döner zu kaufen (startende Kunden-Prozesse). Kunden können nur Döner essen, wenn auch Döner fertig sind, andernfalls müssen sie warten. Ergänzen Sie Ihre Lösung aus der letzten Aufgabe und die folgende Beschreibung eines Kunden-Prozesses um die notwendige Synchronisation.

```
Kunde:
        döner --;
        dönerEssen();
```

c) Verkäufer sollen nur dann etwas produzieren, wenn auch ein Kunde auf den Döner wartet. Ergänzen Sie Ihre Lösung aus der letzten Aufgabe um die notwendige Synchronisation.

## Aufgabe 4.5: POSIX Threads (2. Tutorium) (Tafelübung)

Implementieren Sie die Lösung des Aufgabenteils c) der Aufgabe 4.2 unter Verwendung von POSIX Threads.

## Aufgabe 4.6: Aliens (1,5 + 1,5 Punkte) (Praxis[2])

Wie uns die NASA kürzlich mitteilte, existiert außerirdisches Leben. Entdeckt wurden verschiedene Spezies, die sich anscheinend im Anflug auf die Erde befinden. Es ist der NASA gelungen, den Chat mit dem sich die Aliens koordinieren, abzufangen. Leider ist dieser verschlüsselt, jedoch ist es weiterhin gelungen, die Hashwerte der Passwörter, mit denen der Chat verschlüsselt wurde, abzufangen. Ihre Aufgabe ist es nun einen „Passwort-Cracker“ zu implementieren, der versucht, den Chat zu entschlüsseln. Vielleicht können wir so etwas über die Pläne der Aliens erfahren und uns vorbereiten.

Da ein bruteforce Passwort-Cracker sehr rechenintensiv ist, soll das Programm mittels Erzeuger- und Verbraucherthreads parallelisiert werden.

a) Implementieren Sie einen Ringpuffer mit einem festen Fassungsvermögen von 20 Elementen (*RINGBUFFER_SIZE*). Dieser soll die folgenden, in der Datei *buffer.h* vorgegebenen Funktionen bereitstellen:

- `ringbuffer* make_buffer(void)`

  Alloziert und initialisiert eine neue Ringpufferstruktur und alle eventuell darin enthaltenen Strukturen. Wenn dabei kein Fehler auftritt, soll der neue Puffer zurückgegeben werden.

- `void deposit(ringbuffer* buf, char* str)`

  Fügt den übergebenen Zeiger auf eine Zeichenkette `str` in den angegebenen Ringpuffer `buf` ein. Bei einem vollen Puffer soll der Funktionsaufruf blockieren, bis wieder Platz verfügbar ist.

- `char* fetch(ringbuffer* buf)`

  Entnimmt das vorderste Element aus dem angegebenen Ringpuffer `buf` und gibt dieses zurück. Ist der Puffer leer, soll der Funktionsaufruf blockieren, bis wieder ein Element verfügbar ist.

- `void destroy_buffer(ringbuffer* buf)`

  Gibt den Ringpuffer `buf` und alle darin enthaltenen Strukturen frei.

Beachten Sie, dass der Puffer von mehreren Threads gleichzeitig verwendet werden soll und daher geeignet abgesichert werden muss. Außerdem sollte verhindert werden, dass Werte aus leeren Puffern entnommen oder in volle Puffer eingefügt werden. In diesen Fällen muss der Puffer blockieren.
Es ist sinnvoll, den Puffer zuerst in einem eigenen kleinen Testprogramm auf Fehlerfreiheit zu prüfen.

b) Nun soll das Analyseprogramm durch Umsetzung des Erzeuger-Verbraucher-Schemas vervollständigt werden. Implementieren Sie dazu jeweils einen Thread-Typ für die folgenden Arbeitsschritte:

- `void* generate_passwords_thread(void* args)`

  Dieser soll alle Permutationen von 1 bis 5 stelligen Wörtern erzeugen und über den Puffer weiterleiten. Es werden nur **kleine Buchstaben** benötigt. Große Buchstaben und Sonderzeichen/-Zahlen können vernachlässigt werden.

- `void* password_cracker_thread(void* args)`

  Dieser entimmt ein Passwort aus dem Eingabepuffer, erzeugt einen MD5 Hashwert des Passworts und vergleicht diesen mit den abgefangen Hashwerten. Wurde eine Übereinstimmung gefunden, wird jenes Passwort in den Ausgabepuffer weiter gegeben und somit an den dritten Threadtyp übergeben.

- `void* decrypter_messages_thread(void* args)`

  Dieser Thread entnimmt die gefunden Passwörter aus dem Aufgabepuffer und versucht mit jedem Passwort jede Zeile des Chats zu entschlüsseln, da wir nicht wissen welches Passwort für welche Chatzeile verwendet wurde.

## Hinweise:

- Um einem Thread-Typ der nachfolgenden Arbeitsstufe zu signalisieren, dass keine weiteren Daten mehr verfügbar sind und er sich beenden kann, soll als letzter Eintrag `NULL` in den entsprechenden Puffern deponiert werden.

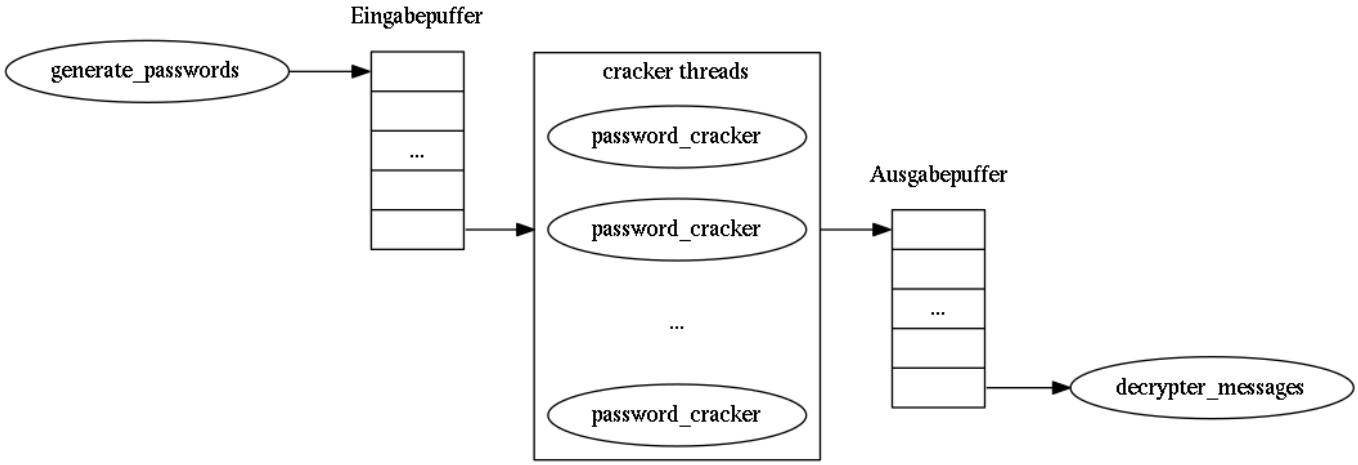


Abbildung 3: Die Threads sollen unter Verwendung des zuvor implementierten Ringpuffers kooperieren

- In den Dateien *crypto.c* und *utils.c* liegt der Code mit dem die Daten ver- und entschlüsselt werden. Wichtig für die Bearbeitung der Aufgabe sind dabei nur die folgenden Funktionen:
  - `void compute_hash(const char* tohash, md5hash* hash);`

    Diese Funktion nimmt die Zeichenkette `tohash` und berechnet daraus den Hashwert, der dann in die per Adresse übergebene Variable `hash` geschrieben wird.
  - `int compare_hash(const md5hash* hash1, const md5hash* hash2 );`

    Diese Funktion vergleicht die beiden übergebenen Hashes. Der Rückgabewert ist $0$, wenn die Hashes gleich sind und $-1$ wenn sie sich unterscheiden. In *passwd_cracker.c* liegt eine Hilfsfunktion, die einen Hash innerhalb eines Arrays von Hashes finden kann.
  - `void decrypt(const byte *chiffre, const uint size, const byte *key, byte *clear);`

    Diese Funktion entschlüsselt den verschlüsselten Text `chiffre` mithilfe des Schlüssels `key` und speichert den entschlüsselten Text in `clear`. `chiffre` und `clear` sind Arrays der Länge `size`, key ist nullterminiert. Diese Funktion wird aus der Hilfsfunktion `print_solutions` aufgerufen, die in *passwd_decrypter.h* implementiert ist.
- Die Datenstrukturen, die den Threads als Argumente übergeben werden, sind in den jeweiligen Headerdateien *passwd_generator.h*, *passwd_cracker.h* und *passwd_decrypter.h* definiert.