Übungen Betriebssysteme (BS)

U3 – Deadlocks

https://moodle.tu-dortmund.de/course/view.php?id=34604

Peter Ulbrich

peter.ulbrich@tu-dortmund.de

https://sys.cs.tu-dortmund.de/EN/People/ulbrich/

technische universität
dortmund

arbeitsgruppe
systemsoftware

Theoriefrage 1

Benennt die im Szenario beschriebenen Betriebsmittel.

Theoriefrage 1

Benennt die im Szenario beschriebenen Betriebsmittel.

- Betriebsmittel: Spirituosen & Flaschen

Theoriefrage 1

Benennt die im Szenario beschriebenen Betriebsmittel.

- Betriebsmittel: Spirituosen & Flaschen

Sind sie konsumierbar oder wiederverwendbar?

Theoriefrage 1

Benennt die im Szenario beschriebenen Betriebsmittel.

- Betriebsmittel: Spirituosen & Flaschen

Sind sie konsumierbar oder wiederverwendbar?

- Sie sind wiederverwendbar

Theoriefrage 2

Wodurch werden die Bedingungen für eine Verklemmung erfüllt?

Theoriefrage 2

Wodurch werden die Bedingungen für eine Verklemmung erfüllt?

1) **`Mutual exclusion`**
   - ➔ Ein Bartender behält die Flaschen für sich (keine gemeinsame Nutzung)

Theoriefrage 2

Wodurch werden die Bedingungen für eine Verklemmung erfüllt?

1) **`Mutual exclusion`**
   - ➔ Ein Bartender behält die Flaschen für sich (keine gemeinsame Nutzung)
2) **`Hold and wait`**
   - ➔ Die Bartender holen Flasche für Flasche (Keine gleichzeitige komplette Belegung)

Theoriefrage 2

Wodurch werden die Bedingungen für eine Verklemmung erfüllt?

1) **Mutual exclusion**
   - ➔ Ein Bartender behält die Flaschen für sich (keine gemeinsame Nutzung)
2) **Hold and wait**
   - ➔ Die Bartender holen Flasche für Flasche (Keine gleichzeitige komplette Belegung)
3) **No preemption**
   - ➔ Flaschen werden erst nach Fertigstellung zurückgebracht (Kein Vorzeitiges Zurückbringen)

Theoriefrage 2

Wodurch werden die Bedingungen für eine Verklemmung erfüllt?

1) **Mutual exclusion**
   - ➔ Ein Bartender behält die Flaschen für sich (keine gemeinsame Nutzung)
2) **Hold and wait**
   - ➔ Die Bartender holen Flasche für Flasche (Keine gleichzeitige komplette Belegung)
3) **No preemption**
   - ➔ Flaschen werden erst nach Fertigstellung zurückgebracht (Kein Vorzeitiges Zurückbringen)
4) **Circular wait**
   - ➔ Die Reihenfolge der Spirituosen-Nutzung ist zufällig
   - ➔ 2 Bartender können die 2 gleichen Spirituosen benötigen, aber in unterschiedlicher Reihenfolge
   - ➔ Gegenseitiges Warten

Theoriefrage 3

Welche der Bedingungen wird im Teil c) bei der Verklemmungsvorbeugung entkräftet?

Theoriefrage 3

Welche der Bedingungen wird im Teil c) bei der Verklemmungsvorbeugung entkräftet?

- Hold & Wait wird entkräftet

Theoriefrage 3

Welche der Bedingungen wird im Teil c) bei der Verklemmungsvorbeugung entkräftet?

- Hold & Wait wird entkräftet

Warum treten nun keine Verklemmungen mehr auf?

Theoriefrage 3

Welche der Bedingungen wird im Teil c) bei der Verklemmungsvorbeugung entkräftet?

- Hold & Wait wird entkräftet

Warum treten nun keine Verklemmungen mehr auf?

- Alle Flaschen werden auf einmal angefordert
- Führt zu ineffizienten Verhalten
  - ➔ wegen verlängerten Wartezeiten

Programmieraufgabe (a)

Semaphoren(Spirituosen) **mit passenden Werten initialisieren:**

```
void init_semaphores() {
  /// *** HIER EUER CODE ***
  for(int i = 0; i < INGREDIENTS_NUM; i++) {
    if(sem_init(&(ingredients[i].sem), 0, ingredients[i].bottles)){
      perror("sem_init");
      exit(1);
    }
  }
}
```

Semaphoren(Spirituosen) **sauber entfernen:**

```
void destroy_semaphores() {
  /// *** HIER EUER CODE ***
    for(int i = 0; i < INGREDIENTS_NUM; i++) {
        if(sem_destroy(&(ingredients[i].sem))) {
            perror("sem_destroy");
            exit(1);
        }
    }
}
```

Programmieraufgabe (1) (b)

Bartender(Thread) Rezept generieren:

```
void* work(void* arg) {
  /// *** HIER EUER CODE ***
    ingredient_t *recipe[RECIPE_SIZE];
    get_recipe(recipe);
    printf("[BT %d] Ich habe mir ein Rezept geholt.\n", my_num);
   /** Rezept generieren */
```

Bartender(Thread) Spirituosen holen:

```
void* work(void* arg) {
    /** Rezept generieren */
    for(int j = 0; j < RECIPE_SIZE; j++) {
      printf("[BT %d] Ich hole mir jetzt %s.\n", my_num, recipe[j]->name);
      if(sem_wait(&(recipe[j]->sem))) {
        perror("sem_wait");
        exit(1);
      }
      /** Drink mixen */
    }
```

Bartender(Thread) Drink mixen:

```
void* work(void* arg) {
   /** Spirituosen holen */
      printf("[BT %d] Ich verarbeite jetzt %s.\n", my_num, recipe[j]->name);
      sleep(recipe[j]->time_needed);
  [...]
```

Programmieraufgabe (2) (b)

Bartender(Thread) **Zutaten zurück bringen:**

```
void* work(void* arg) {
    /** Rezept generieren */
    /** Spirituosen holen */
      /** Drink mixen */
    for(int j = 0; j < RECIPE_SIZE; j++) {
      if(sem_post(&(recipe[j]->sem))) {  /** Perror Fehlerbehandlung */  }
    }

    printf("[BT %d] Der Drink ist fertig und alles zurückgebracht.\n", my_num);
  [...]
```

Bartender(Thread) **mixt** RECIPES_PER_BARTENDER **Mal Drinks:**

```
void* work(void* arg) {
/// *** HIER EUER CODE ***
  for(int i = 0; i < RECIPES_PER_BARTENDER; i++) {
    /** Rezept generieren */
    /** Spirituosen holen */
      /** Drink mixen */
    /** Zutaten zurückbringen */
  }

  printf("[BT %d] Fertig für heute!\n", my_num);
  pthread_exit(NULL);
}
```

Programmieraufgabe (c)

Bartender(Thread) **reserviert gesamtes Regal:**

```
void* work(void* arg) {
  /** RECIPES_PER_BARTENDER Mal Drinks mixen */
    /** Rezept generieren */
    if(pthread_mutex_lock(&lock)) {  /** Perror Fehlerbehandlung */  }
    printf("[BT %d] Ich habe das Getränkeregal reserviert.\n", my_num);
    /** Alle Spirituosen holen */
```

Bartender(Thread) **Regal wieder freigeben + Drinks mixen:**

```
void* work(void* arg) {
  /** RECIPES_PER_BARTENDER Mal Drinks mixen */
    /** Rezept generieren */
    /** Regal reservieren */
    /** Alle Spirituosen holen */
    if(pthread_mutex_unlock(&lock)) {  /** Perror Fehlerbehandlung */  }
    printf("[BT %d] Ich habe das Getränkeregal wieder freigegeben.\n", my_num);
    for(int j = 0; j < RECIPE_SIZE; j++) {
     /** Drink mixen */
    /** Zutaten zurückbringen */
  [...]
```

Programmieraufgabe (1) (d)

Berechne maximale Zubereitungszeit:

```
int get_max_wait() {
  int current_max = 0;

  for(int i = 0; i < INGREDIENTS_NUM; i++) {
    if(ingredients[i].time_needed > current_max) {
      current_max = ingredients[i].time_needed;
    }
  }
  return current_max * RECIPE_SIZE;
}
```

Reset der Bar anstoßen:

```
void reset() {
  printf("[ ! ] Reset angestoßen.\n");

  restart = 1;

  for(int i = 0; i < BARTENDER_NUM; i++) {
    if(pthread_equal(bartenders[i], pthread_self()) == 0) {
      pthread_cancel(bartenders[i]);
    }
  }

  pthread_exit(NULL);
}
```

Programmieraufgabe (2) (d)

Bartender(Thread) **Drink mixen mit TimeOut setzen:**

```
void* work(void* arg) {
  /** RECIPES_PER_BARTENDER Mal Drinks mixen */
    /** Rezept generieren */
    for(int j = 0; j < RECIPE_SIZE; j++) {
      printf("[BT %d] Ich hole mir jetzt %s.\n", my_num, recipe[j]->name);
      struct timespec time;
      /// Aktuelle Uhrzeit ermitteln
      if(clock_gettime(CLOCK_REALTIME, &time)){  /** Perror Fehlerbehandlung */  }
      time.tv_sec += get_max_wait();
      if(sem_timedwait(&(recipe[j]->sem), &time)) {
        if(errno == ETIMEDOUT) {
          printf("[BT %d] Ich habe keine Lust mehr zu warten.\n", my_num);
          reset();
        } else {  /** Perror Fehlerbehandlung */  }
      }
      /** Drink mixen */
    /** Zutaten zurückbringen */
  /** END - RECIPES_PER_BARTENDER Mal Drinks mixen */
}
```