Freie Universität Berlin

Prof. Dr. Margarita Esponda

Nichtsequentielle Programmierung, SoeSe 2017

Übungsblatt 5

TutorIn: Lilli Walter
Tutorium 6

Boyan Hristov, Sergelen Gongor

20. Juni 2017

Link zum Git Repository: https://github.com/BoyanH/FU-Berlin-ALP4/tree/master/Solutions/Homework5

Aufgabe 1

Zu zeigen: Await-Bedingung für $P_i \equiv B \equiv inD \leq afterF$

$$\begin{aligned} wp(inD \leftarrow inD + 1, PCI) =& wp(inD \leftarrow inD + 1, (inD \leq afterF + 1) \wedge (inF \leq afterD)) = \\ =& (inD + 1 \leq afterF + 1) \wedge (inF \leq afterD) \qquad \text{(Zuweisungsregel)} \\ =& (inD \leq afterF) \wedge (inF \leq afterD) \end{aligned}$$

Da es $P \wedge INV \wedge B \Rightarrow wp(S, Q \wedge INV)$ gelten muss ist hier $B = inD \leq afterF$

□

Zu zeigen: Await-Bedingung für $C_i \equiv B \equiv inF < afterD$

$$\begin{aligned} wp(inF \leftarrow inF + 1, PCI) =& wp(inF \leftarrow inF + 1, (inD \leq afterF + 1) \wedge (inF \leq afterD)) = \\ =& (inD \leq afterF + 1) \wedge (inF + 1 \leq afterD) = \qquad \text{(Zuweisungsregel)} \\ =& (inD \leq afterF + 1) \wedge (inF < afterD) = \end{aligned}$$

Da es $P \wedge INV \wedge B \Rightarrow wp(S, Q \wedge INV)$ gelten muss ist hier $B = inF < afterD$

□

Aufgabe 2

Leider war das Pseudo-Code, das in der Vorlesungsfolien stand, nicht richtig. Wir haben es repariert und danach die Bedingung für die Lese verändert -> diese können frei reinkommen ohne warten nur wenn es keine wartende Schreiber gibt und auch nur dann können sie weitere Leser reinlassen.

```java
package fu.alp4;

public class Writer extends IDataUser {

    public void run() {
        while (true) {
            try {
                // take a random rest to simulate different scenarios
                randomNap(5000, 10000);

                E.acquire();

                /**
                 * If there are currently some other writers or readers, wait
                 * for an available writer spot to be freed from a reader
                 * In te given time, release E, but remember to acquire it before
                 * incrementing nw++ for synchronization
                 */
                if (nw > 0 || nr > 0) {
                    dw++;
                    System.out.println("Waits to start writing! Stop letting further readers!");
                    E.release();
                    W.acquire();
                    E.acquire();
                }

                nw++;
                System.out.printf("Started writing; nr: %s; nw: %s; dr: %s; dw: %s\n", nr, nw, dr, dw);
                E.release();

                randomNap(2000, 4000);

                E.acquire();
                nw--;

                if (dr > 0 && dw == 0) {
                    dr--;
                    R.release();
                } else if (dw > 0) {
                    /**
                     * Deferred writers have higher priority, because we thought it's important to
                     * let writers as soon as possible so readers get the latest and greatest ^^
                     *
                     * E.g if both writers and readers are waiting, let the writer in
                     */

                    dw--;
                    W.release();
                }
                System.out.printf("Finished writing; nr: %s; nw: %s; dr: %s; dw: %s\n", nr, nw, dr, dw);
                E.release();


            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
package fu.alp4;
```

```
public class Reader extends IDataUser {

    public void run() {
        while (true) {
            // take a random rest to simulate different scenarios
            randomNap(500, 2000);
            try {
                E.acquire();
                // skip waiting only if there are no deferred or non-deferred writers!
                // if a writer is waiting, he has the priority
                if (nw > 0 || dw > 0) {
                    dr++;
                    E.release();
                    R.acquire();
                    E.acquire();
                }

                nr++;
                // again, waiting writers have priority, don't let further readers in this
    case
                if (dr > 0 && dw == 0) {
                    dr--;
                    R.release();
                }

                System.out.printf("Started reading; nr: %s; nw: %s; dr: %s; dw: %s\n", nr,
    nw, dr, dw);
                E.release();

                // read
                randomNap(500, 2000);


                E.acquire();
                nr--;
                if (nr == 0 && dw > 0) {
                    dw--;
                    W.release();
                }

                System.out.printf("Finished reading; nr: %s; nw: %s; dr: %s; dw: %s\n", nr,
     nw, dr, dw);
                E.release();

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
package fu.alp4;

public class Main {

    public static void main(String[] args) {

        for (int i = 0; i < 5; i++) {
            if (i < 4) {
                new Reader().start();
            } else {
                new Writer().start();
            }
        }
    }
}
```

Aufgabe 3

a) $\sum_{n=1}^{n} C_{ij} + A_j = E_j$ wobei C ist die Belegungsmatrix, A der Ressourcenrestvektor, E der Ressourcenvektor (Aus Vorlesungsfolien). Also an dem Beispiel $\sum_{n=1}^{n} C_{ij} + R_j = E_j$ wobei R der Ressourcenrestvektor ist.

$$\Rightarrow R_j = E_j - \sum_{n=1}^{n} C_{ij}$$
$$\Rightarrow R = [(3-(1+1)), (15-(1+3+6)), (12-(1+5+3+1)), (11-(1+4+2+4))] =$$
$$= [1, 5, 2, 0]$$

Der Ressourcenrestvektor bzw. die noch vorhande Ressourcen R = [1, 5, 2, 0]

b) Das System befindet sich in einem sicheren Zustand, weil es eine Scheduling-Reihenfolge gibt, die nicht zu Deadlock führt. Solche Reihenfolge ist z.B:

$$T_4 \Rightarrow R = [1, 11, 5, 2]$$
$$\rightarrow T_5 \Rightarrow R = [1, 11, 6, 6]$$
$$\rightarrow T_1 \Rightarrow R = [1, 11, 7, 7]$$
$$\rightarrow T_2 \Rightarrow R = [2, 12, 6, 6]$$
$$\rightarrow T_3 \Rightarrow R = [3, 15, 12, 11]$$

$R = E \Rightarrow$ sicheren Zustand

c) Da R = [1,5,2,0] wird nach dem Teilanforderung von Thread T_2 R = [1,2,0,0]. Das ist kein sicheren Zustand, da alle Threads danach (inklusive T_2 mit seinem neuen Restanforderung von [0,3,3,0]) mindestens eins von den letzten zwei Ressourcen brauchen, es gibt aber keine mehr vorhanden. Deswegen gibt es auch keine Scheduling-Reihenfolge, die nicht zu einem Deadlock führt.

$\Rightarrow$ soll nicht bedient werden.

1 Aufgabe 4

a) Done

b) Der Algorithmus bearbeitet alle Requests nach einander, wobei er immer Ressourcen gibt, wenn solche vorhanden sind. Dafür speichert diese Lösung initial die freie Ressourcen, die von der Methode getFreeRes() in der Klasse Resources zurückgegeben werden und verändert immer die freie Ressources entsprechend. Da es ein Array ist werden die entsprechend auch in dem Objekt resources gesetzt.

Der Algorithmus funktioniert nur unter der Annahme, dass die gegebene Reihenfolge von Requests von links nach rechts abgearbeitet werden kann, ohne dass das System in einem Deadlock geht. Es ist praktisch kein Algorithmus, egal was die Threads wollen, wird es zu denen gleich gegeben, falls es genug Resourcen gibt, sonst nicht.

c)

1. ExampleAlgo2 benutzt eine Warteschlange, wo neue Prozesse immer eingefügt werden, als diese kommen, und dann wieder entfernt werden, als diese fertig sind. So gibt der Algorithmus Ressources nur zu den Prozessen, die auf den ersten zwei Positionen in der Warteschlange sind und zwar nur wenn es genug freie Ressource gibt. Weiter wenn der 1 Prozess in der Warteschlange nie fertig wird, wird er auch nicht entfernt.

2. Der Algorithmus funktioniert nur unter der Annahme richtig, dass zwei (oder bzw. auch 1) Prozess nicht so Ressourcen erfordern kann, dass es zu einem Deadlock kommt. Falls aber z.B es nicht genug Ressourcen für die erste zwei Prozessen in der Warteschlange gibt, dann kriegt man ein schönes Deadlock.

3. Man kann immer schauen, ob die maximal erforderten Ressourcen von dem konkreten Prozess und dem letzten in der Warteschlange weniger als den freien Ressourcen sind. Da immer nur der erste Prozess in der Warteschlange entfernt wird ist man sicher, dass Prozesse, die nicht nebeneinander in der Warteschlange stehen, zusammen die Ressourcen sich teilen werden.

   Eine konkrete Implementierung die in allen Fällen richtig funktioniert (nur die IF-Anweisung):

```
if(!q.contains(id) &&
                    (
                            (q.isEmpty() && CUtil.greaterEqual(free, req.
    getProcess().getPendingRes())) ||
                            (q.size() >= 1 && CUtil.greaterEqual(free, CUtil.add(
    processes[q.get(q.size() - 1)].getPendingRes(), req.getProcess().
    getPendingRes())))
                    )
            ) {

               q.add(id);
           }
```

4. Erstmall die SafeChecker Klasse, das gehört auch zum Teils zur nächsten Aufgabe.

```
package allocationAlgorithm;

import calculationUtil.CUtil;
import process.Process;
import resources.Resources;

public class SafeChecker {

    public static boolean checkIfSafeState(Resources resources, Process[]
    processes) {
        return checkIfSafeState(processes, getAvailableResources(resources,
    processes));
    }

    public static boolean checkIfSafeState(Process[] processes, int[]
    availableResources) {
        int[][] pendingResources = getPendingResources(processes);
        int[][] acquiredResources = getAcquiredResources(processes);

        return checkIfSafeState(pendingResources, acquiredResources,
    availableResources);
    }

    public static boolean checkIfSafeState(int[][] pendingResources, int[][]
    acquiredRes, int[] availableResources) {
        assert pendingResources.length == acquiredRes.length;

        boolean[] ready = new boolean[acquiredRes.length];
        boolean changed = true;
        int counter = 0;
        int[] localAvailableRes = availableResources.clone();
```

```
28          // until there are changes and not all processes are servived
29          while (changed && counter < pendingResources.length) {
30              changed = false; // a new cycle began, see if there are changes this
    time

32              for (int i = 0; i < pendingResources.length; i++) {
33                  // check if the current process can be serviced, if so mark it as
     ready and acquire resources
34                  if (CUtil.greaterEqual(localAvailableRes, pendingResources[i]) &&
     !ready[i]) {
35                      localAvailableRes = CUtil.add(localAvailableRes, acquiredRes[
    i]);
36                      ready[i] = true;
37                      changed = true;
38                      counter++;
39                  }
40              }
41          }

43          return counter == pendingResources.length; // are all processes marked as
     serviced?
44      }

46      public static int[] getAvailableResources(Resources resources, Process[]
    processes) {
47          int[] totalResources = resources.getTotalRes();
48          int[] availableResources = totalResources.clone();

50          for (int i = 0; i < processes.length; i++) {
51              availableResources = CUtil.sub(availableResources, processes[i].
    getAcquiredRes());
52          }

54          return availableResources;
55      }

57      public static int[][] getPendingResources(Process[] processes) {
58          int[][] pendingResources = new int[processes.length][];

60          for (int i = 0; i < processes.length; i++) {
61              pendingResources[i] = processes[i].getPendingRes().clone();
62          }

64          return pendingResources;
65      }

67      public static int[][] getAcquiredResources(Process[] processes) {
68          int[][] acquiredResources = new int[processes.length][];

70          for (int i = 0; i < processes.length; i++) {
71              acquiredResources[i] = processes[i].getAcquiredRes().clone();
72          }

74          return acquiredResources;
75      }

77      public static void updateAcquired(int[][] acquiredResources, Process[]
    processes) {
78          for (int i = 0; i < acquiredResources.length; i++) {
79              if (processes[i].isFinished()) {
80                  acquiredResources[i] = CUtil.sub(acquiredResources[i],
    acquiredResources[i]);
81              }
82          }
83      }
84  }
```

So kann man es danach verwenden (die ganze Klasse ist hier nicht relevant, das ganze Code

ist im GitHub):

```
return String.format("Safe state: %s", SafeChecker.checkIfSafeState(processes,
    resources.getFreeRes()));
```

e) Klasse für Bankieralgorithmus

```
package allocationAlgorithm;

import calculationUtil.CUtil;
import process.Process;
import process.Process.AllocationRequest;
import resources.Resources;

public class BankersAlgo implements Algorithm {

    Resources resources;
    Process[] processes;

    int[][] acquiredRes;
    int[][] pendingRes;

    @Override
    public void init(Resources resources, Process[] processes) {
        this.resources = resources;
        this.processes = processes;
        this.pendingRes = SafeChecker.getPendingResources(processes);
        this.acquiredRes = SafeChecker.getAcquiredResources(processes);
    }

    @Override
    public String nextStep(AllocationRequest[] requests) {

        int[] freeRes = resources.getFreeRes();
        SafeChecker.updateAcquired(acquiredRes, processes);

        for (AllocationRequest req : requests) {

            int[] freeResSimulation = freeRes.clone();
            int[][] pendingResSimulation = pendingRes.clone();
            int[][] acquiredResSimulation = acquiredRes.clone();
            int processIdx = this.getProcessIndex(req);

            // simulate resource allocation
            pendingResSimulation[processIdx] = CUtil.sub(pendingResSimulation[
    processIdx], req.getRequestedRes());
            acquiredResSimulation[processIdx] = CUtil.add(acquiredResSimulation[
    processIdx], req.getRequestedRes());
            freeResSimulation = CUtil.sub(freeResSimulation, req.getRequestedRes());

            // if state is safe after allocation, grant permission
            if (SafeChecker.checkIfSafeState(pendingResSimulation,
    acquiredResSimulation, freeResSimulation)) {
                freeRes = CUtil.sub(freeRes, req.getRequestedRes());
                this.pendingRes[processIdx] = CUtil.sub(this.pendingRes[processIdx],
    req.getRequestedRes());
                this.acquiredRes[processIdx] = CUtil.add(this.acquiredRes[processIdx],
     req.getRequestedRes());
                req.grant();
            }
        }

        return String.format("safe: %s", SafeChecker.checkIfSafeState(resources,
    processes));
    }

    public int getProcessIndex(AllocationRequest req) throws IllegalArgumentException
    {
```

```
55          for (int i = 0; i < processes.length; i++) {
56              if (req.getProcess().getId() == processes[i].getId()) {
57                  return i;
58              }
59          }

61          throw new IllegalArgumentException("Invalid request!");
62      }

64  }
```