Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2015	**Aufgabenblatt 5** zu – Systemprogrammierung – Prof. Dr. Odej Kao, Dr. Peter Janacik, Tutoren
Abgabetermin:	[1] – 22.06.2015 [2] – 29.06.2015

Aufgabe 5.1: (0,8 Punkte)
Betriebsmittelverwaltung mit Fremdbelegung (Theorie[1])

Ein teilbares Betriebsmittel soll mit Fremdbelegung verwaltet werden. Wir nehmen an, dass zum Zeitpunkt $t_0 = 0$s alle Einheiten belegt sind und die folgenden Anforderungen in der Warteschlange stehen:

$A_1 = 2$ Einheiten, $A_2 = 13$ Einheiten, $A_3 = 4$ Einheiten, $A_4 = 7$ Einheiten, $A_5 = 5$ Einheiten
(Anmerkung: A_1 steht ganz vorn und A_5 ganz hinten in der Warteschlange)

Die Anforderungen müssen dabei jeweils im Ganzen erfüllt werden, d.h. eine Anforderung kann nur erfüllt werden, wenn zum jeweiligen Zeitpunkt mindestens so viele Einheiten frei sind, wie angefordert werden. Angenommen zu den gegebenen Zeitpunkten werden nun folgende Belegungen freigegeben:

Zeitpunkt	$t_1 = 1$s	$t_2 = 2$s	$t_3 = 3$s	$t_4 = 4$s	$t_5 = 5$s
Freigabe	$F_1 = 3$ Einh.	$F_2 = 9$ Einh.	$F_3 = 1$ Einh.	$F_4 = 9$ Einh.	$F_5 = 10$ Einh.

In welcher Reihenfolge werden die gegebenen Anforderungen bei Abarbeitung nach

a) FCFS, (0,2 Punkte)

b) First Fit, (0,2 Punkte)

c) Best Fit und (0,2 Punkte)

d) Best Fit mit dynamischem Fenster von $L_{max} = 3$ (0,2 Punkte)

erfüllt? Geben Sie für jedes Verfahren außerdem die durchschnittliche Wartezeit an.

Reichen Sie für jedes Verfahren eine Lösung in Form der unten dargestellten Tabelle ein. Falls mehrere Aktionen zum selben Zeitpunkt stattfinden, notieren Sie bitte jede in einer eigenen Spalte.

Zeitpunkt	0	1	...	...
Freigaben	-	F_1	...	
Anforderungen	-	-	...	
Aktuell freie Einheiten	0	3	...	

Geben Sie für Best Fit mit dynamischem Fenster außerdem in jeder Spalte die aktuelle Fenstergröße mit an.

Aufgabe 5.2: (1,2 Punkte)
Handsimulation des Bankieralgorithmus (Theorie[1])

Der Bankieralgorithmus kontrolliert Ressourcenallokationen, damit keine unsicheren Zustände auftreten.[1]
Gegeben ist die folgende verzahnte Ausführung der vier Prozesse P_1, P_2, P_3 und P_4:

	Zeit	Aktion
P_1	1	`allocate_r(C,3);`
	2	`allocate_r(B,4);`
P_3	3	`allocate_r(C,1);`
	4	`allocate_r(A,3);`
P_1	5	`release_r(C,3);`
	6	`allocate_r(A,1);`
P_3	7	`release_r(A,3);`
	8	`allocate_r(C,3);`
P_2	9	`allocate_r(C,2);`
	10	`allocate_r(A,3);`
P_1	11	`release_r(B,4);`
	12	`allocate_r(A,1);`

	Zeit	Aktion
P_4	13	`allocate_r(B,3);`
	14	`release_r(B,3);`
	15	`exit();`
P_2	16	`allocate_r(B,1);`
	17	`release_r(B,1);`
P_3	18	`release_r(C,4);`
	19	`exit();`
P_1	20	`release_r(A,2);`
	21	`exit();`
P_2	22	`release_r(A,3);`
	23	`release_r(C,2);`
	24	`exit();`

Die Funktionen `allocate_r` und `release_r` erhalten hierbei jeweils die Kennung für eines der Betriebsmittel A, B, C oder D und die angeforderte bzw. freizugebende Anzahl als Parameter.
Von jedem Betriebsmittel sind zu Beginn 4 Einheiten vorhanden und nicht belegt.

Führen Sie eine Handsimulation für den gegebenen Ablauf durch. Geben Sie dabei für jede Allokation die Belegungen, die Restanforderungen und die freien Betriebsmittel in der aus der Vorlesung bekannten Matrixschreibweise an, prüfen Sie mit dem Bankieralgorithmus, ob die Belegung zu einem unsicheren Zustand führt oder nicht und geben Sie ggf. eine mögliche sichere Ausführungsreihenfolge an. Tritt ein unsicherer Zustand auf oder sind zum Zeitpunkt der Anfrage nicht genug Betriebsmittel vorhanden, wird der dazugehörige Prozess bis zum Ende der Handsimulation blockiert. Blockierte Prozesse geben ihre Betriebsmittel *nicht* frei.

Aufgabe 5.3: (Woche 1)
Betriebsmittelverwaltung (Tafelübung)

Untersuchen Sie die folgende Belegungssituation:

Belegung: $B = \begin{pmatrix} 5 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}$ Gesamtanforderung: $G = \begin{pmatrix} 9 & 3 \\ 2 & 5 \\ 3 & 3 \end{pmatrix}$ Freie Ressourcen: $f = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

a) Wie sieht die Restanforderungs-Matrix aus?

[1] Dieser Ansatz ist bei Banken inzwischen scheinbar in Vergessenheit geraten, kommt dort heute doch bevorzugt eine andere Strategie zum Einsatz: `http://de.wikipedia.org/wiki/Bail-out_(Wirtschaft)`

b) Sind im System genügend Ressourcen vorhanden, um theoretisch alle Anforderungen zu erfüllen?

c) Erläutern Sie, was ein sicherer Zustand ist und prüfen Sie, ob es sich hier um einen solchen handelt!

d) Wie sieht es aus, wenn $f = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ ist?

e) Was kann man üblicherweise tun, wenn sich ein System in einem Deadlock befindet?

Aufgabe 5.4: (Woche 1)
Scheduling mit Prioritäten (Tafelübung)

Recherchieren Sie, welches Problem beim Mars Rover Pathfinder wiederholt zu Systemresets geführt hat. Welches grundlegende Problem liegt dem zugrunde und wie lässt es sich beheben?

Aufgabe 5.5: (Woche 2)
Verklemmung (Tafelübung)

An einem Frühstückstisch in einer WG (Uli, Shanti, Gwen und Lara) gibt es Käse, Toast, Marmelade, Brot und Peanutbutter. Jeden Morgen entsteht ein erbitterter Kampf um die Ressourcen. Die Auseinandersetzung am Frühstückstisch kann näherungsweise durch folgendes C-Programm beschrieben werden:

```
1  // Ressourcen
2  // von jeder Ressource gibt es genau 1 Instanz, die nicht verbraucht wird
3  single_unit_R peanutbutter, jelly, cheese, toast, bread;
4  // Uli - Prozess 1:
5  int main() {
6      allocate_r(&cheese);
7      allocate_r(&jelly);
8      allocate_r(&bread);
9      eat(); // Brot mit Jelly und Cheese
10     release_r(&bread);
11     release_r(&jelly);
12     release_r(&cheese);
13     return 0;
14 }
15
16 // Shanti - Prozess 2:
17 int main() {
18     allocate_r(&peanutbutter);
19     allocate_r(&bread);
20     eat(); // Peanutbutter & Brot
21     release_r(&bread);
22     release_r(&peanutbutter);
23     return 0;
24 }
25
26 //Lara - Prozess 3:
27 int main() {
28     allocate_r(&toast);
29     allocate_r(&cheese);
```

```
30      eat(); // Käse-Toast (Lara ist auf Diät)
31      release_r(&cheese);
32      release_r(&toast);
33      return 0;
34  }
35
36  // Gwendolin - Prozess 4:
37  int main() {
38      allocate_r(&jelly);
39      allocate_r(&peanutbutter);
40      allocate_r(&toast);
41      eat(); // it's Peanut Butter Jelly Time!!
42      release_r(&toast);
43      release_r(&peanutbutter);
44      release_r(&jelly);
45      return 0;
46  }
```

Es sei ein System gegeben, bei welchem jeder Prozess 2 Befehle ausführen kann, bevor zum nächsten Prozess gewechselt wird. Die Reihenfolge der Ausführung soll wie oben Uli→Shanti→Lara→Gwen→Uli. . . sein.

a) Simulieren sie den Frühstückstisch. Geben Sie in jedem Schritt an, wer welche Ressource hat oder freigibt.

b) Wie sieht der Ablauf aus, wenn jeder Prozess nur einen Befehl ausführen kann, bevor zum nächsten Prozess gewechselt wird?

c) Zeichnen Sie den Betriebsmittelgraph wie er am Ende des Ablaufs aus Teil b. aussieht.

d) Welche Bedingungen sind notwendig und welche hinreichend, damit eine Verklemmung auftreten kann?

Aufgabe 5.6: (3 Punkte)
Implementierung des Bankieralgorithmus (Praxis[2])

In Anlehnung an die Theorie Aufgabe 5.2 soll in dieser Aufgabe zunächst der Bankieralgorithmus so modifiziert werden, dass man ihn zum Erkennen einer Verklemmungssituation nutzen kann. Anschließend soll der Bankieralgorithmus implementiert werden. Es soll also sowohl Deadlock Detection als auch Deadlock Avoidance implementiert werden.

Die Vorgabe hat das Szenario aus aus Aufgabe 5.2 bereits implementiert, sprich 4 Threads (T1-T4) die mit den Funktionen `allocate_r` und `release_r` jeweils Betriebsmittel (A bis D) anfordern bzw. freigeben.

a) Die aktuelle Belegungssituation wird in `g_state` gespeichert. Ergänzen Sie zunächst die Funktionen `allocate_r` und `release_r` um das Updaten der Belegungssituation. (0,1 Punkte)

b) Ergänzen Sie nun die Funktion `isSafe`. Zunächst soll nur überprüft werden, ob aktuell genügend Betriebsmittel vorhanden sind. Falls nicht soll `UNSAFE` zurückgegeben werden. (0,1 Punkte)

c) Es kann nun zu einer Verklemmung[2] der Threads kommen. Die Vorgabe stellt einen Deadlock-Watchdog Thread bereit, welcher im Sekundentakt die aktuelle Belegungssituation auf eine Verklemmung überprüft. Implementieren Sie die Funktion `isDeadlocked`. Lassen Sie sich im Fall einer Verklemmung auch die beteiligten Threads in der Variable `out` mit ausgeben. (1,4 Punkte)

d) Erweitern Sie nun die Funktion `isSafe` um eine Verklemmungsvermeidung mit Hilfe des Bankieralgorithmus. Lassen Sie sich ggf. auch die Reihenfolge in der T1-T4 laufen können in der Variable `out` mit ausgeben. (0,7 Punkte)

e) Wie kann man einer Verklemmung vorbeugen, wenn Sie die Vergabe der Betriebsmittel innerhalb der Threads in der Funktion `thread_work` ändern könnten? Geben Sie ihre neue `thread_work` Funktion (nur T1-T4) und eine begründete Antwort in einer extra Datei mit ab. (0,7 Punkte)

Hinweise:

- Bei einer Verklemmung lässt sich das Programm nur durch `ctrl+c` beenden.
- Mit `make debug` lassen sich zusätzliche `printf` Ausgaben einschalten.
- Die Terminalausgabe enthält Farben, um die einzelnen Threads besser auseinander halten zu können. Die Farbausgabe lässt sich durch Entfernen von `-DCOLOR` aus dem Makefile ausschalten. Dies ist zB. sinnvoll, wenn die Ausgabe in eine Datei geschrieben werden soll, etc.

[2]Gegebenenfalls lässt sich eine Verklemmung erst nach einigen Durchläufen beobachten.