Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2020	**Aufgabenblatt 5** zu – Systemprogrammierung – Prof. Dr. Odej Kao
Abgabetermin:	**12.07.2020 23:55 Uhr**

Aufgabe 5.1: Betriebsmittelverwaltung (Tafelübung)

a) Nennen Sie Beispiele für Betriebsmittel. Nach welchen Kriterien lassen sie sich klassifizieren?

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: $f = \begin{pmatrix} 1 & 1 \end{pmatrix}$

b) Wie sieht die Restanforderungs-Matrix aus?

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

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

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

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

Aufgabe 5.2: 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:

```
// Ressourcen
// von jeder Ressource gibt es genau 1 Instanz, die nicht verbraucht wird
single_unit_R peanutbutter, jelly, cheese, toast, bread;
// Uli - Prozess 1:
int main() {
    allocate_r(&cheese);
    allocate_r(&jelly);
    allocate_r(&bread);
    eat(); // Brot mit Jelly und Cheese
    release_r(&bread);
    release_r(&jelly);
    release_r(&cheese);
    return 0;
}

// Shanti - Prozess 2:
```

```
int main() {
    allocate_r(&peanutbutter);
    allocate_r(&bread);
    eat(); // Peanutbutter & Brot
    release_r(&bread);
    release_r(&peanutbutter);
    return 0;
}

//Lara - Prozess 3:
int main() {
    allocate_r(&toast);
    allocate_r(&cheese);
    eat(); // Käse-Toast (Lara ist auf Diät)
    release_r(&cheese);
    release_r(&toast);
    return 0;
}

// Gwendolin - Prozess 4:
int main() {
    allocate_r(&jelly);
    allocate_r(&peanutbutter);
    allocate_r(&toast);
    eat(); // it's Peanut Butter Jelly Time!!
    release_r(&toast);
    release_r(&peanutbutter);
    release_r(&jelly);
    return 0;
}
```

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) In der Vorlesung wurden im Kontext der Betriebsmittelverwaltung zwei verschiedene Typen von Graphen vorgestellt, der Wartegraph und der Betriebsmittelgraph. Wie unterscheiden sich diese?

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

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

Aufgabe 5.3: Betriebsmittelverwaltung mit Fremdbelegung (2 Punkte) (Theorie)

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:

$A_1 = 10$ Einheiten, $A_2 = 6$ Einheiten, $A_3 = 7$ Einheiten, $A_4 = 1$ 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.	$F_2 = 12$ Einh.	$F_3 = 8$ Einh.	$F_4 = 2$ Einh.	$F_5 = 4$ Einh.

In welcher Reihenfolge werden die gegebenen Anforderungen bei Abarbeitung nach

a) FCFS, (0,5 Punkte)

b) First Fit, (0,5 Punkte)

c) Best Fit und (0,5 Punkte)

d) Best Fit mit dynamischem Fenster von $L_{max} = 3$ (0,5 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	...	

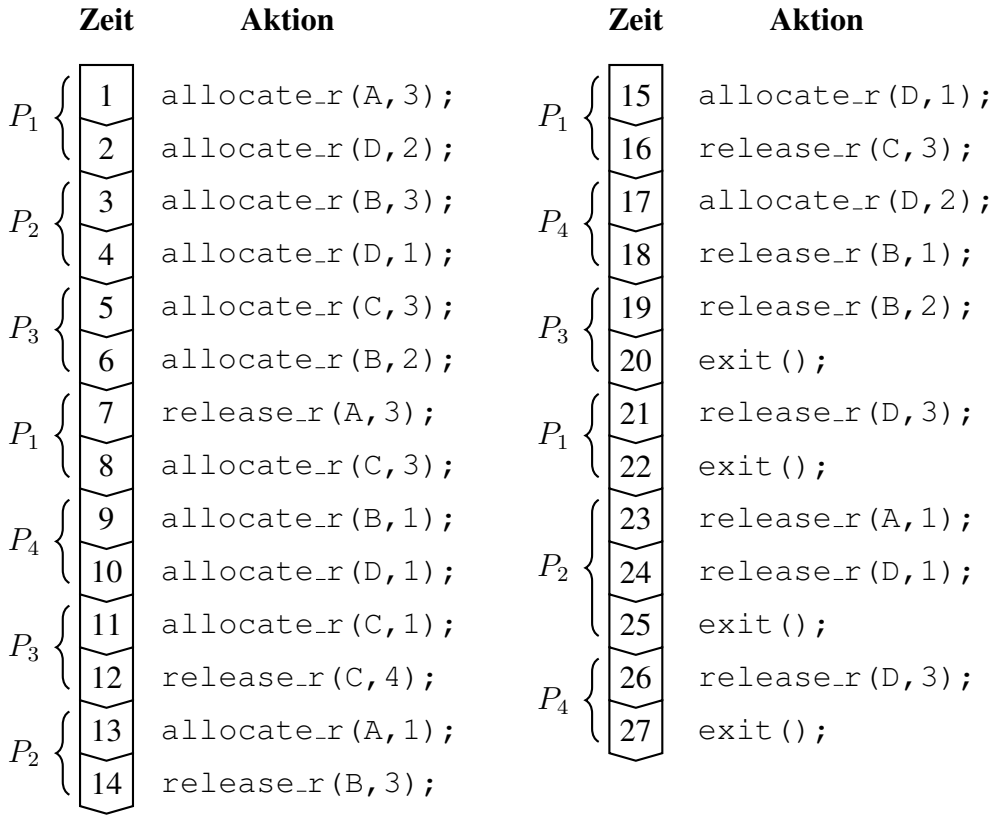
Geben Sie für Best Fit mit dynamischem Fenster außerdem in jeder Spalte die aktuelle Fenstergröße mit an.

Aufgabe 5.4: Handsimulation des Bankieralgorithmus (2 Punkte) (Theorie)

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:

[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)`



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 zu jedem Zeitpunkt die aktuellen Belegungen, die Restanforderungen und die freien Betriebsmittel in der aus der Vorlesung bekannten Matrixschreibweise an, prüfen Sie mit dem Bankieralgorithmus, ob die Allokation zu einem unsicheren Zustand führt oder nicht. 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 5.5: Bankier-Alorithmus (5 Punkte) (Praxis)

Um dieses Aufgabenblatt geschmackvoll abzurunden, ist es nun Ihre Aufgabe, den Bankier-Algorithmus in C zu implementieren. Stellen sie sich dazu folgendes Szenario vor:
Sie arbeiten bei der Bau- und Sanierungsfirma *ASKYY*™ und sind dort als Fachinformatiker im Bereich Planung & Organisation angestellt. *ASKYY*™ 's Kernkompetenz ist (unter Anderem) die Renovierung von alten Großbauten, wozu im Allgemeinen eine große Menge verschiedenster Gerüstmodule und Werkzeuge vonnöten ist. Da eben diese Gegenstände sperrig und teuer sind, hat die Firma nur ein begrenztes Reservoir an Gerüsten und Werkzeugen, die sorgfältig auf die aktuell laufenden Projekte (Baustellen) der Firma verteilt werden müssen.

Dabei ist es häufig unsinnig, alle für ein neues Projekt benötigten Güter von Anfang an neben der Baustelle zu lagern, da der Lagerplatz knapp ist und das Gerüst nur Stück für Stück gebaut werden kann. So kommt es, dass die Firmenleitung jeder Baustelle nur genau die Ressourcen zuweist, die tatsächlich gerade benötigt werden.

An dieser Stelle kommen Sie ins Spiel: Während die Firmenleitung einen Plan darüber zusammenstellt, welche Ressourcen wann und in welchen Mengen zu welcher Baustelle gelangen, sollen Sie ein Programm schreiben, welches diesen Plan auf seine Ausführbarkeit (bzw. darauf, ob Verklemmungssituationen entstehen) überprüft.

Um dieses Programm wartbarer zu gestalten, hat Ihr Vorgesetzter bereits eine Vorlage verfasst, welche Sie vollenden sollen. Diese beinhaltet unter Anderem eine Matrix-Bibliothek, welche Sie fertigstellen und mit deren Hilfe Sie danach den Bankier-Algorithmus implementieren sollen. Außerdem sind bereits zwei Funktionen zur übersichtlichen Ausgabe der Matrizen vorgegeben, die Sie zum Debuggen Ihres Programms nach Belieben einsetzen können (siehe *print_matrix()* und *print_BRf())*.

- Implementieren Sie alle gekennzeichneten Funktionen in der Datei matrix.c (**2,7 Punkte**)
- Implementieren Sie anschließend alle gekennzeichneten Funktionen in der Datei bankier.c (**2,3 Punkt**)

Hinweise:

- *main.c* und das Makefile sind schon vorgegeben und müssen nicht verändert werden. Die *main.c* darf aber zu Testzwecken nach Belieben angepasst werden..
- Das Abgabearchiv soll nur *bankier.c/.h* und *matrix.c/.h* beinhalten.
- Das Einbinden weiterer Bibliotheken ist unzulässig (Absprachen möglich).
- Verändern Sie die Vorgaben **nur** an den Stellen, die mittels */TODO: /** gekennzeichnet sind.
- Sie finden sowohl in den .c-, als auch in den .h-Dateien eine detaillierte Anleitung für jede zu implementierende Funktion.
- Die Funktionen *memset()* und *memcpy()* dienen zum Auffüllen bzw. Kopieren von Speicherblöcken und können Ihnen viel Arbeit ersparen.
- Nutzen Sie das struct *matrix* so, wie in der Vorgabe beschrieben. Andernfalls schlagen die automatischen Tests fehl.
- **Vorgaben:** Bitte halten Sie sich bei der Programmierung immer an die Vorgaben. Eine Missachtung kann zu Punktabzug führen.
- **Makefile:** Bitte verwenden Sie für diese Aufgabe das Makefile aus der Vorgabe.
- **Dynamischer Speicher:** Um zu evaluieren ob der gesamte allokierte Speicher wieder freigegeben wurde, empfiehlt sich das Kommandozeilen Werkzeug **valgrind**[2] (unter linux über apt-get install valgrind). **Memory leaks führen zu Punktabzug.**
- Bitte testen Sie Ihren Bankier-Alorithmus sorgfältig! Das erfolgreiche Ausführen der vorgegebenen main Funktion bedeutet nicht zwingend, dass Ihr Algorithmus fehlerfrei ist!

[2] http://valgrind.org/

main.c: Die vorgegebene *main.c* stellt den Service zur Verfügung, Ihre Implementierung an einem Beispiel zu testen. Dazu müssen Sie einen Anfangszustand der Ressourcen vorgeben und diesen in der Datei *data.txt* (per default) abspeichern. Diese Datei muss beim Aufrufen des Programmes übergeben werden (siehe *main.c*). Sie können die *data.txt* nach Belieben verändern, um verschiedene Situationen durchzuspielen.

Wichtig: Die *main.c* benutzt die von Ihnen zu implementierenden Funktionen und funktioniert daher nur, wenn Sie alle benötigten Funktionen bereits korrekt implementiert haben.