Abgabefrist Übungsaufgabe 4

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – Übung U4 in KW24 (10.06. - 13.06.): Abgabe bis **Mittwoch, den 19.06.2024 23:59**
- **W2** – Übung U4 in KW25 (17.06. - 20.06.): Abgabe bis **Montag, den 24.06.2024 10:00**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Es ist **nicht** mehr möglich, Einzelabgaben im AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt dies bitte **vorher** mit eurem Übungsleiter! Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.
- Die Aufgaben sind über AsSESS (`https://sys-sideshow.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt **ein** Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder. Matrikelnummer, Vor- und Nachname sind bei dem Abgabevorgang in die entsprechenden Eingabefelder des AsSESS und nicht in die Abgabedateien einzutragen. Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sowohl für die Programmieraufgabe als auch für die Theorieaufgaben stellen wir Ihnen Vorlagen bereit, die Sie zur Bearbeitung der Aufgaben verwenden sollen. Diese werden im Moodle als ZIP-Ordner (`vorgaben-A4.zip`) zum herunterladen zur Verfügung gestellt.
- Sobald eine Abgabe korrigiert wurde, kann das Ergebnis ebenfalls hier eingesehen werden.
- Ihre Programme müssen von `gcc` (also kein C++) mit der Option `-Wall` kompilierbar sein (als Referenzumgebung dienen die Poolrechner, wie empfehlen zusätzlich `-Werror`). Sollten Warnungen entstehen können ihnen dafür Punkte abgezogen werden. Programm die nicht übersetzt werden können werden nicht als Lösung akzeptiert und mit maximal einem Punkt bewertet.
- Nutzen sie in ihrem Quelltext eine sinnvolle Einrückung für Blöcke und vermeiden sie mehrere Anweisungen in einer Zeile. In Fällen von unleserlicher Formatierung können ihnen Punkte abgezogen werden.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8

Aufgabe 4: Speicherverwaltung (10 Punkte)

4.1 Theorieaufgaben (5 Punkte)

a) Buddy-Verfahren (1.5 Punkte)

Ein Computersystem verfügt über **32 MBytes Hauptspeicher**, die **kleinst mögliche Blockgröße** beträgt **1 MByte** und es kommt das Buddy-Verfahren zum Einsatz.

Zu Beginn ist der Hauptspeicher leer. Anschließend treffen die folgenden Speicheranforderungen bzw. -freigaben ein:

(1) Anforderung A von 2 MBytes

(2) Anforderung B von 42 KBytes

(3) Anforderung C von 12 MBytes

(4) Anforderung D von 1,6 MBytes

(5) Anforderung E von 2,5 MBytes

(6) Freigabe von Anforderung D

(7) Anforderung F von 4,2 MBytes

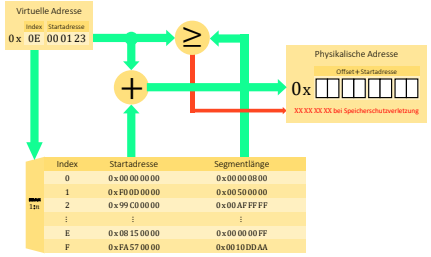
Wenn für eine Anforderung nicht genügend Speicher zur Verfügung steht, wird diese abgelehnt und es werden keine Änderungen an der Speicherbelegung vorgenommen.

Stellen Sie die Speicherbelegung nach jedem Anforderungs- bzw. Freigabeschritt dar. Das folgende Format mit einer Zeile je Anforderung bzw. Freigabe bietet sich dafür an. Nutzen Sie zur Darstellung Ihrer Lösung die Tabelle aus der Vorlage (⇒ `antworten.txt`). In ihr repräsentiert jedes Zeichen einer Zeile einen Block des Speichers. Die Stellen an denen der Speicher geteilt wird sind nicht zu markieren. Es genügt für jeden Block den entsprechenden Buchstaben (`A`, `B`, `C`, `D`, `E`) einzutragen. Ein noch nicht belegter Block wird durch einen Bindestrich '`-`' dargestellt.

Die Anforderungen (1) und (2) wurde bereits in der Vorlage gelöst. Orientieren Sie sich bei der Erstellung Ihrer Lösung an diesem Format!

b) Segmentierung (0.5 Punkte)

In dieser Aufgabe soll die Speichersegmentierung näher betrachtet werden. Übersetzen Sie mit Hilfe der Speichersegmentierung die virtuellen Adresse `0x0E000123` in eine physikalische Adresse. Die höchstwertigen 8 Bit der logischen Adresse geben die Position innerhalb der Segmenttabelle an. Tragen Sie Ihre Ergebnisse an der entsprechend markierten Stelle in der Vorlage (⇒ `antworten.txt`) als Hexadezimalzahl im Format `0x????????` ein. Löst eine Speicheranfrage eine Speicherschutzverletzung aus, machen Sie dies bitte durch `0xXXXXXXXX` kenntlich. Schreiben Sie in diesem Fall in die Vorlage (⇒ `antworten.txt`) den Grund für die Speicherschutzverletzung. Unter dem Platzhalter für die physikalische Adresse findet sich ggf. ein weiterer Platzhalter für Ihre Begründung.



c) LRU-Algorithmus (1.5 Punkte)

Sowohl im Zusammenhang mit Übersetzungspuffern (TLB) von voll- bzw. mengenassoziativen Speichern als auch bei der Arbeit mit Caches werden Ersetzungsalgorithmen eingesetzt. Mit einem solchen Algorithmus lässt sich entscheiden, welche Kachel optimalerweise zu entfernen ist, wenn kein weiterer Eintrag bzw. keine weitere Kachel mehr aufgenommen werden kann.

In der Vorlesung wurde unter anderem der LRU-Algorithmus (Least Recently Used) eingeführt, der direkt in der Hardware implementiert wird. Naiv lässt sich der LRU-Algorithmus folgendermaßen implementieren: Die CPU besitzt einen Zähler, der bei jedem Speicherzugriff inkrementiert wird. bei jedem Zugriff wird der aktuelle Zählerwert in den jeweiligen Seitendeskriptor geschrieben. Wird eine nicht eingelagerte Kachel aufgerufen, so wird die am längsten nicht mehr aufgerufene Kachel mit dieser ersetzt.

Führen Sie den LRU-Algorithmus für die in der Tabelle gegebene Referenzfolge aus. Tragen Sie die in jedem Zeitschritt eingelagerten Seiten in die vorbereitete Registertabelle der Vorlage (⇒ `antworten.txt`) ein und verwenden Sie `X` für einen noch unbelegten Rahmen. Trennen Sie die Einträge mit einem Leerzeichen. Die Tabelle mit den Kontrollzuständen ist nicht mit abzugeben!

Registertabelle mit den eingelagerten Seiten

Referenzfolge:	1	5	3	3	5	4	4	2	7	4	9	1	4	6
Kachel 0:	1	1	1	1										
Kachel 2:	X	5	5	5										
Kachel 2:	X	X	3	3										
Kachel 3:	X	X	X	X										

Zugriffstabelle mit den Kontrollzuständen (Zahl der Zyklen seit des letzten Aufrufs)

Kachel 0:	0	1	2	3										
Kachel 2:	INT_MAX	0	1	2										
Kachel 2:	INT_MAX	INT_MAX	0	0										
Kachel 3:	INT_MAX	INT_MAX	INT_MAX	INT_MAX										

d) LRU-Algorithmus mit Adjazenzmatrix (1.5 Punkte)

Eine weitere Möglichkeit zur Implementierung des LRU-Algorithmus besteht darin, die Einträge durch eine 'Ist älter als oder gleich alt wie'-Relation (bezeichnet mit $\preceq$) miteinander in Beziehung zu setzen. Ein Eintrag ist genau dann 'älter' als ein anderer Eintrag, wenn er länger nicht mehr aufgerufen wurde als der andere. Wird ein Eintrag neu hinzugefügt oder aufgerufen, wird dieser automatisch zum jüngsten Eintrag. Der älteste Eintrag wird bei bedarf entfernt. Die Relation ändert sich also bei jedem Zugriff. Im Rechner wird die Relation $\preceq$ in Form einer Adjazenzmatrix $Adj_{\preceq}$ festgehalten.

$Adj_{\preceq}[i,j]$ bezeichnen den Eintrag in der i-ten Zeile und der j-ten Spalte der Adjazenzmatrix. $Adj_{\preceq}[i,j] = 1$ gilt also immer genau dann, wenn i älter als oder gleich alt wie j ist ($j \preceq i$). Andernfalls gilt $Adj_{\preceq}[i,j] = 0$. Da die Relation $\preceq$ reflexiv und antisymmetrisch ist, reicht es aus, den Teil oberhalb der Hauptdiagonalen zu speichern.

Der Vorteil bei dieser Implementierung besteht darin, dass sich eine solche Adjazenzmatrix leicht analysieren lässt. Der älteste Eintrag erfüllt immer die beiden folgenden Eigenschaften:

$\forall i < x.\ \mathrm{Adj}_{\preccurlyeq}[i,x] = 0$ bzw. $\forall i < x.\ i \prec x$: Spalte ist Null
$\forall j > x.\ \mathrm{Adj}_{\preccurlyeq}[x,j] = 1$ bzw. $\forall j > x.\ j \preccurlyeq x$: Zeile ist Eins

Welcher Eintrag ist gemäß der nachfolgenden Adjazenzmatrix nun der älteste?

i)

i \ j	0	1	2	3	4	5	6	7
0	X	**1**	**1**	**1**	**0**	**1**	**0**	**1**
1	X	X	**0**	**1**	**0**	**0**	**0**	**0**
2	X	X	X	**1**	**0**	**0**	**0**	**1**
3	X	X	X	X	**0**	**0**	**0**	**0**
4	X	X	X	X	X	**1**	**0**	**1**
5	X	X	X	X	X	X	**0**	**1**
6	X	X	X	X	X	X	X	**1**
7	X	X	X	X	X	X	X	X

Tragen Sie Ihre Antwort als einzelne Ziffer an der entsprechend markierten Stelle in der Vorlage (⇒ `antworten.txt`) ein.

Beim Aufruf oder dem Hinzufügen wird die betroffene Kachel jeweils zum jüngsten Eintrag. Ein Eintrag ist genau dann der jüngste, wenn er die folgenden beiden Eigenschaften erfüllt:

$\forall j > k.\ \mathrm{Adj}_{\preccurlyeq}[k,j] = 0$ bzw. $\forall j > k.\ k \prec j$: Zeile auf Null setzen
$\forall i < k.\ \mathrm{Adj}_{\preccurlyeq}[i,k] = 1$ bzw. $\forall i < k.\ k \preccurlyeq i$: Spalte auf Eins setzen

Bestimmen Sie jeweils die Adjazenzmatrizen für den Fall, dass erst die Kachel $k = 3$ und darauf folgend die Kachel $k = 4$ aufgerufen wird. Notieren Sie die Adjazenzmatrizen im vorgegebenen Format! (⇒ `antworten.txt`)

i \ j	0	1	2	3	4	5	6	7
0	X	**0**	**0**	**0**	**0**	**0**	**1**	**0**
1	X	X	**0**	**0**	**0**	**0**	**1**	**0**
2	X	X	X	**0**	**1**	**1**	**1**	**0**
3	X	X	X	X	**1**	**1**	**1**	**0**
4	X	X	X	X	X	**0**	**1**	**0**
5	X	X	X	X	X	X	**1**	**0**
6	X	X	X	X	X	X	X	**0**
7	X	X	X	X	X	X	X	X

$k = 3$

ii)

i \ j	0	1	2	3	4	5	6	7
0	X							
1	X	X						
2	X	X	X					
3	X	X	X	X				
4	X	X	X	X	X			
5	X	X	X	X	X	X		
6	X	X	X	X	X	X	X	
7	X	X	X	X	X	X	X	X

$k = 4$

iii)

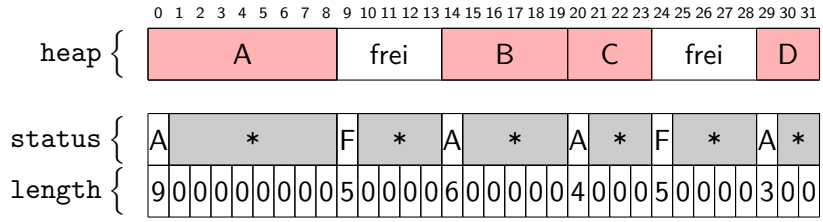
i \ j	0	1	2	3	4	5	6	7
0	X							
1	X	X						
2	X	X	X					
3	X	X	X	X				
4	X	X	X	X	X			
5	X	X	X	X	X	X		
6	X	X	X	X	X	X	X	
7	X	X	X	X	X	X	X	X

4.2 Programmierung: Speicherverwaltung mit dem Next-Fit-Verfahren (5 Punkte)

In dieser Aufgabe implementieren Sie eine kleine Speicherverwaltung (angelehnt an die Funktionalität von `malloc(3)` und `free(3)`). Dazu bekommen Sie einen **256 KiB großen Speicherbereich** vorgegeben, der in **Blöcke (*Chunks*) von 4 KiB** aufgeteilt ist und verwaltet werden soll. **Zur Verwaltung** verwenden Sie zunächst **ein Feld**, welches für jeden Block **eine Metadatenstruktur vom Typ `mem_info`** enthält. Diese gibt an, ob am jeweiligen Block ein zusammenhängender Speicherbereich einer bestimmten Länge beginnt (`length`) und wenn ja, ob dieser frei oder belegt ist (`status`).

Beispiel:

Für vier belegte Bereiche in einem 128 KiB großen Speicher können die Verwaltungsinformationen folgendermaßen aussehen. Hier steht **A** für belegt (*allocated*), **F** für frei (*free*) und **Sternchen** dafür, dass der status ignoriert werden kann, da `length` $== 0$.



In der Vorgabe im Moodle findet sich ein Makefile und eine vorgegebene Modulstruktur, in die Sie ihre Lösungen integrieren sollt.

Den simulierten Hauptspeicher geben wir bereits als Array `heap` in der Datei `nextfit.c` vor. Diese enthält ebenfalls die Liste `allocation_list` vom Typ `mem_info[]`, in welche Sie den Belegungsstatus für jeden Speicherblock ablegen sollen. Im Detail stehen Ihnen die folgenden Hilfsmittel zur Verfügung:

- Die `mem_info`-Struktur enthält die Member `status` und `length`. Ist `length` $== 0$, so beginnt am zugehörigen Speicherblock kein zusammenhängender Speicherbereich und die `status`-Angaben werden nicht berücksichtigt, der Status ist hier also nicht definiert. Andernfalls beginnt an dieser Stelle ein `length` Blöcke umfassender Speicherbereich, der entweder frei (`status` $==$ `CHUNK_FREE`) oder belegt (`status` $==$ `CHUNK_ALLOCATED`) ist.
- Auf die Blockgröße in Bytes könnt ihr über das Makro `CHUNK_SIZE` zugreifen, die Speichergröße (in Chunks) ist über `NUM_CHUNKS` verfügbar.
- `size_to_chunks(`*`bytes`*`)` gibt die zur Speichergröße *bytes* korrespondierende Anzahl an Speicherblöcken zurück. Falls nötig, wird aufgerundet.
- Für Größenangaben und Array-Indizes sei der plattformunabhängige Typ `size_t` zu verwenden. Im Regelfall entspricht dieser einem `unsigned long int`.
- `dump_memory()` gibt den aktuell in `allocation_list` vermerkten Belegungsstatus auf der Standardausgabe aus und führt einige einfache Konsistenztests durch. Eine Raute (#) steht für einen belegten Speicherblock, ein - für einen freien.

a) Speicherallokation (2.5 Punkte)

Implementieren Sie die Funktion `nf_alloc(size_t size)` in der Datei `4a.c`. Diese soll nach dem Next-Fit-Verfahren einen Speicherbereich reservieren, der mindestens `size` Bytes groß ist, und die Startadresse des reservierten Speicherbereichs zurückgeben. Dazu muss insbesondere vorgehalten werden, wo der letzte Speicherbereich reserviert wurde.

Falls 0 Bytes angefragt werden oder kein genügend großer freier Speicherbereich mehr vorhanden ist, soll die Funktion `NULL` zurückgeben.

Zum Testen kann die Datei `test_4a.c` und der Befehl `make 4a` verwendet werden. Dieser führt einige in `test_4a.c` definierte Speicherbelegungen durch und gibt dabei jeweils die zurückgegebene Speicheradresse und die aktuelle Speicherbelegung aus. `test_4a.c` deckt möglicherweise nicht alle Randfälle ab – es könnte sich also lohnen, sie um zusätzliche Randfälle zu erweitern. Die Tests sollen nicht mit abgegeben werden und sind somit nicht Teil der bewerteten Aufgabenstellung.

⇒ `4a.c`

b) Speicherfreigabe (2.5 Punkte)

Implementieren Sie die Funktion `nf_free(void *addr)` in der Datei `4b.c`. Diese Funktion wird aufgerufen, um einen zuvor reservierten Speicherbereich wieder freizugeben, und erhält die Startadresse des Speicherbereichs als Argument. Aktualisieren Sie die `allocation_list` entsprechend. Wenn die Speicherbereiche vor und/oder nach dem freigegebenen Speicherbereich ebenfalls frei sind, muss dieser zu einem neuen (großen) Freispeicherbereich vereinigt werden.

Dabei kann es vorkommen, dass der in a) gespeicherte zuletzt reservierte Speicherplatz nach der Vereinigung nicht mehr auf den Beginn eines zusammenhängenden Speicherbereichs zeigt, sondern mitten im neuen Freispeicherbereich liegt. In diesem Fall soll er zum Beginn dieses Bereichs (d.h. „nach links“) verschoben werden.

Falls die Adresse `NULL` übergeben wird, soll nichts passieren. Bei anderweitig ungültigen Adressen, die nicht der Startadresse eines belegten Speicherbereichs entsprechen oder gar nicht erst im verwalteten Arbeitsspeicher liegen, soll eine Fehlermeldung ausgegeben und das gesamte Programm mit dem Exit-Code 255 beendet werden.

Zum Testen kann die Datei `test_4b.c` und der Befehl `make 4b` verwendet werden. Dieser führt die in `test_4b.c` definierten Speicherbelegungen und -freigaben durch und gibt dabei jeweils die aktuelle Speicherbelegung aus.

⇒ `4b.c`

Zusatzaufgabe 4: Allokation nach dem Best-Fit-Verfahren (2 Punkte)

Nun soll die Funktion `bf_alloc(size_t size)` in der Datei `4extended.c` implementiert werden. Diese soll nach dem Best-Fit-Verfahren einen Speicherbereich belegen, der mindestens `size` Bytes groß ist und wie bei a) die Startadresse des belegten Speicherbereichs zurückgeben.

Falls 0 Bytes angefragt werden oder kein genügend großer freier Speicherbereich mehr vorhanden ist, soll die Funktion `NULL` zurückgeben.

Damit Speicher auch wieder freigegeben werden kann, soll zusätzlich die Funktion `bf_free(void *addr)` in der selben Datei implementiert werden. Die Funktion entspricht hierbei der aus b), es muss sich aber nicht um die Behandlung des zuletzt reservierten Speicherplatzes gekümmert werden.

Verwendet zum Testen den Befehl `make 4extended`. Dieser führt einige in `test_4extended.c` definierte Speicherbelegungen durch und gibt dabei jeweils die zurückgegebene Speicheradresse und die aktuelle Speicherbelegung aus.

⇒ `4extended.c`

Tipps zu den Programmieraufgaben:

- Kommentieren Sie den Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Es ist unbedingt daran zu denken, dass viele Systemaufrufe fehlschlagen können! Diese Fehlerfälle sind abzufangen (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages) und durch entsprechende Fehlermeldungen sichtbar zu machen (z.B. unter Zuhilfenahme von **perror(3)**). Das Programm ist danach ordnungsgemäß zu beenden.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Es ist das mitgelieferte Makefile (Kommando `make`) zu verwenden, oder der Compiler mit den folgenden Parametern aufzurufen::
  `gcc -std=c11 -Wall -o ziel datei.c`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-Wpedantic -Werror -D_POSIX_SOURCE`
- Achten Sie darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt; z.B. durch Nutzung von `-Werror`.