Übungsblatt 5

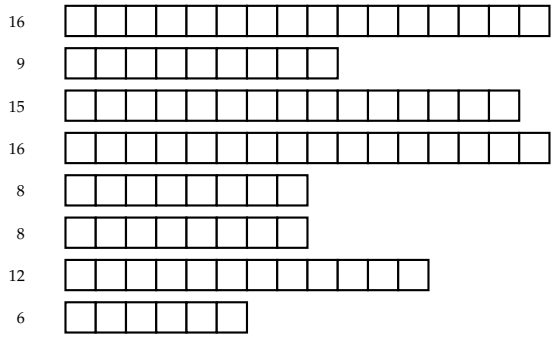
Abgabe bis spätestens 7.12.2023 in Stud.IP

Allgemeiner Hinweis

Das in StudIP zur Verfügung gestellte Archiv `ueb5-vorgabe.zip` enthält TikZ-Quellen, die Ihr bei Bedarf als Gerüst für die in Aufgabe 1 und 3 geforderten Grafiken nutzen könnt.

Aufgabe 1 (5 Punkte)

Die folgende Grafik visualisiert eine prozess-interne Freispeicherliste mit den freien Chunks der Größen 16,9,15,16,8,8,12,6 (in dieser Reihenfolge ab dem Beginn der Freispeicherliste).



a) Zeigt anhand dieser Grafik für jeden der drei Algorithmen *First-Fit*, *Next-Fit* und *Best-Fit*, wie die folgenden Speicheranforderungen (*alloc*), bzw. Freigaben (*free*), behandelt würden und tragt das jeweilige Ergebnis analog zum Beispiel im Tutorium in die Grafik ein (jeweils eine Grafik pro Algorithmus). *First-Fit*, *Next-Fit* und *Best-Fit* seien so implementiert, dass ein freier Chunk immer so geteilt wird, dass die linke Hälfte zuerst belegt wird. Für *Next-Fit* stellt die rechte Hälfte den neuen Beginn der zyklischen Freispeicherliste dar.

A)	`a = alloc(5)`	F)	`f = alloc(8)`	K)	`free(g)`
B)	`b = alloc(9)`	G)	`g = alloc(6)`	L)	`l = alloc(6)`
C)	`c = alloc(12)`	H)	`free(d)`	M)	`m = alloc(10)`
D)	`d = alloc(3)`	I)	`i = alloc(10)`	N)	`n = alloc(4)`
E)	`free(a)`	J)	`j = alloc(16)`	O)	`o = alloc(5)`

b) Diese soeben belegten Speicherbereiche werden wieder freigegeben (dabei erfolgen keine Verschmelzungen). Würde eine Anfrage von 14 danach erfüllbar sein? Wo?

c) Wie sähe die finale Speicheraufteilung für die Anforderungen aus Aufgabenteil a) beim *Buddy-Algorithmus* aus, wenn initial ein 128 Einheiten großer Speicherbereich zur Verfügung steht und die minimale Blockgröße 4 Einheiten beträgt? Visualisiert die Speicherverteilung auf geeignete Weise.

Aufgabe 2 (3 Punkte)

Unter `/home/ti2/ueb/05/aufgabe2` im Fachbereichsnetz findet ihr ein Programm `zaehlen`, das mit dem ebenfalls dort befindlichen `Makefile` auf dem Rechner x03 erzeugt worden ist.

a) Ermittelt (z. B. mit gdb[1]) die Bereiche im virtuellen Adressraum des Prozesses, den die Funktion `main` einnimmt. Dazu könnt ihr beispielsweise das gdb-Kommando `disassemble` verwenden. (Mit der Option `/r` zeigt `disassemble` zusätzlich die Maschineninstruktionen als Folge von Hexadezimalzahlen an.)

b) In Zeile 36 des Programms wird eine Zeichenkette auf der Konsole ausgegeben. Ermittelt den Adressbereich im Datensegment, in dem sich diese Zeichenkette befindet. Neben dem gdb-Kommando `disassemble` eignet sich dazu das Kommando `x` , gefolgt von der Formatangabe `/s` (also `x/s ADDRESSE`). Am einfachsten ist es, dazu einen Breakpoint auf die Adresse zu setzen, das Programm mittels `run <data.txt` zu starten und nach dem Anhalten die Argumente für den folgenden Funktionsaufruf genauer zu untersuchen (freundlicherweise zeigt `gdb` im laufenden Betrieb die gesuchten Adressen als Kommentar an).

c) Zeigt exemplarisch anhand der in Aufgabenteil b ermittelten Anfangs- und Endadresse der Zeichenkette die Umrechnung der virtuellen Adressen in reale Speicheradressen. Nehmt dazu an, dass die betreffende Page auf den Pageframe mit der (hexadezimalen) Adresse 12E1 abgebildet wird und eine Page 4 KiB groß ist.

Aufgabe 3 (2 Punkte)

Gegeben sei ein Prozess mit einem virtuellen Adressraum von fünf Pages. Dieser Prozess soll auf einem Rechner mit vier Page Frames (A, B, C, D) ausgeführt werden. (Reale Systeme sind natürlich viel „größer“, aber hier reicht ein „kleines“ System.) Auf die Pages wird bei der Abarbeitung wie folgt lesend zugegriffen:

`1-2-3-4-4-2-1-3-2-1-4-5-5-5-6-2-2-5-1-2-1-4-4-5-1-1-4-4-2-6-6-2-3-3-2-3`

Simuliert die Seitenverdrängung mittels folgender in der Vorlesung vorgestellter Algorithmen. Die Darstellung soll in einer Tabelle analog zum Beispiel aus der Vorlesung erfolgen. **Gebt für jeden der Algorithmen an, wie oft eine Page von der Festplatte gelesen werden muss. Für den Clock-Hand-Algorithmus sind außerdem die Reclaims deutlich zu markieren.**

a) First In, First Out (FIFO),

FIFO	1	2	3	4	4	2	1	3	2	1	4	5	5	5	6	2	2	5	1	2	1	4	4	5	1	1	4	4	2	6	6	2	3	3	2	3
D	1																																			
C		2				2																															
B			3																																		
A				4	4																																

b) Least Frequently Used (LFU),

[1] GNU Debugger, siehe auch die offizielle Dokumentation. Als Kurzreferenz ganz nützlich ist das Cheatsheet. (Beide URIs zuletzt besucht am 15.11.2023.)

LFU	1	2	3	4	4	2	1	3	2	1	4	5	5	5	6	2	2	5	1	2	1	4	4	5	1	1	4	4	2	6	6	2	3	3	2	3
D	1																																			
C		2				2																														
B			3																																	
A				4	4																															

c) Clock-Hand.

CH	1	2	3	4	4	2	1	3	2	1	4	5	5	5	6	2	2	5	1	2	1	4	4	5	1	1	4	4	2	6	6	2	3	3	2	3
D	1^0				F																															
C		2^0				2^0																															
B			3^0																																	
A				4^0	4^1																															

Hinweis: Der Clock-Hand-Algorithmus verdrängt bei Bedarf. Er soll vereinfachend so umgesetzt werden, dass pro Speicherzugriff ein Page Frame (egal, ob belegt oder nicht) untersucht und ggf. freigegeben wird, d. h. der Clock-Hand ist nur schwach von den Speicherzugriffen entkoppelt. Anders wäre es bei einer Aufgabe „mit Papier und Bleistift“ zu komplex. Die Überprüfung durch den Clock-Hand findet jeweils **nach dem Speicherzugriff** statt; als erstes wird nach dem ersten Speicherzugriff der Page Frame A untersucht, nach dem zweiten B usw. Potentiell baut sich hierdurch eine gewisse „Freispeicherreserve“ auf. In der Tabelle wird die Position des Clock-Hand mit einem kleinen Punkt in der rechten unteren Ecke visualisiert. Der jeweilige Wert des R-Bits wird in der rechten oberen Ecke gezeigt. Hinzuzufügen sind außerdem Informationen darüber, ob ein Page-Frame zu einem bestimmten Zeitpunkt als frei markiert wird („f“) und ob eine Page geladen wurde.

Wenn beim Clock-Hand-Algorithmus verdrängt werden muss, aber kein Frame als frei markiert ist, erfolgt *Paging on Demand*, d. h die Überprüfung der Page-Frames wird solange zyklisch fortgesetzt, bis ein als frei markierter Frame angetroffen wird. Sind mehrere Page-Frames frei, wird analog zu LRU belegt.

Abgabe

Bis 23:59 Uhr am 7.12.2023 digital in StudIP. **Es gelten die vereinbarten Scheinbedingungen (siehe StudIP).** Bitte beachtet unsere ergänzenden Hinweise ebenda.

Ladet die abgaberelevanten Dateien in DoIT hoch.

Eure Ansätze und der gewählte Lösungsweg müssen nachvollziehbar sein. Achtet insofern auf eine saubere Dokumentation im Quelltext und im abgegebenen PDF-Dokument. Benennt alle von Euch verwendeten Quellen, auch Zusammenarbeit mit anderen Gruppen und verwendete Unterlagen aus früheren Jahrgängen.

Programmieraufgaben sind im Zweifel in C++20 zu entwickeln. Die Korrektheit der Lösung bzw. deren Grenzen sind grundsätzlich in der Abgabe nachzuweisen. Dies geschieht neben der Dokumentation des Programmcodes in den Quelldateien und zusätzlich in dem abgegebenen PDF-Dokument mit der Lösungsbeschreibung durch geeignete Tests, deren Auswahl und Eignung begründet werden müssen. Als **Referenzplattform** gelten die Linux-Rechner im Rechnerpool in MZH E0.