Hochschule Augsburg
Fakultät für Informatik

Wintersemester 2019/2020

Klausur

Prüfungsfach:	Systemnahe Programmierung
Datum/Uhrzeit:	30. Januar 2020 / 10:30 Uhr
Raum:	M1.01/M1.02
Prüfer:	Dr. Hubert Högl
Dauer:	**60** Minuten
Hilfsmittel:	keine

Name, Vorname	Matrikel-Nr.	Hörsaal/Sitzplatz

Hinweise:

1. Füllen Sie oben die drei Felder Name / Matrikel-Nr. / Hörsaal+Sitzplatz aus.
2. Die Angaben bestehen aus 9 Seiten. Überprüfen Sie dies bitte sofort am Anfang.
3. Schreiben Sie Ihre Lösungen in die Angaben. Geben Sie die Angaben am Ende ab.
4. Nebenrechnungen machen Sie bitte auf den separat ausgeteilten Karo-Bögen. Diese dürfen Sie nach Hause nehmen.
5. Schreiben Sie nicht mit Bleistift und nicht in roter Farbe.

Viel Glück!

Aufgabe 1 (6 Punkte)

Kreuzen Sie die zutreffenden Aussagen an:

☐ (A) Eine Programmiersprache ist portabel, wenn Programme ohne Änderungen auf verschiedenen Betriebssytemen und Zentraleinheiten (CPU) laufen.

☐ (B) Wer in Assembler programmiert der muss sein Programm komplett neu schreiben, wenn er es auf einer anderen CPU laufen lassen möchte.

☐ (C) Die Sprache C läuft nur auf UNIX-ähnlichen Betriebssystemen, da sie in diesem Bereich auch entstanden ist. Aus diesem Grund gibt es C nicht für Microsoft Windows.

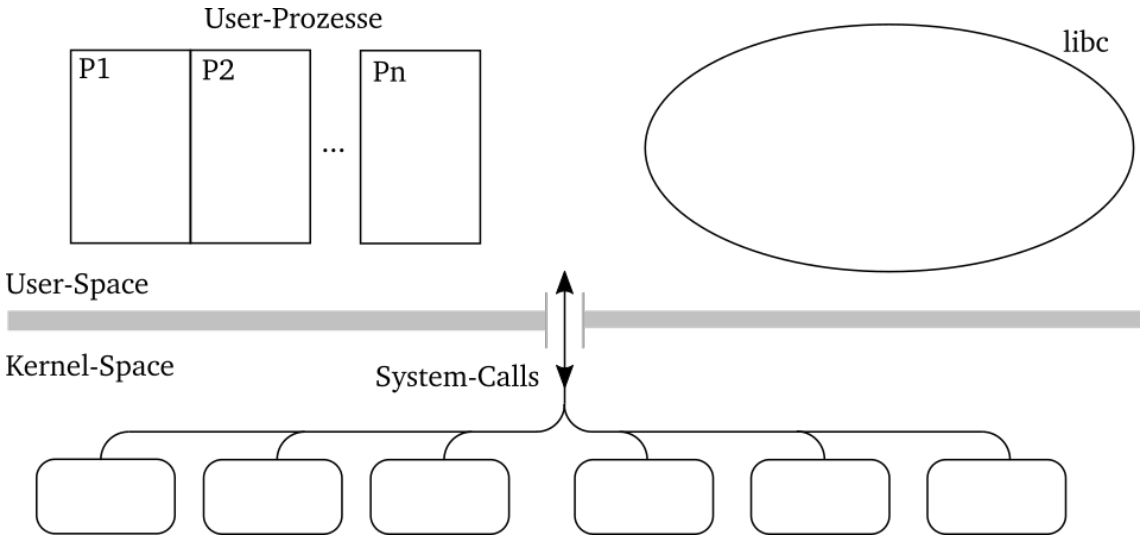
☐ (D) Die Produktivität bei jeder Programmiersprache ist in etwa gleich gross, d.h. unabhängig ob man in Assembler, C oder Python programmiert erzeugt eine Person in etwa die gleiche Anzahl Maschinenbefehle pro Tag.

☐ (E) High-level Sprachen haben alle eine automatische Speicherverwaltung. Bei low-level Sprachen wie C und C++ verwaltet die ProgrammiererIn den Speicher.

☐ (F) Durch den grossen Erfolg der high-level Programmiersprache Python werden low-level Sprachen wie C und C++ überflüssig.

Aufgabe 2 (8 Punkte)

In der folgenden Abbildung sehen Sie den „User-Space“ (mit Prozessen und der C Bibliothek) und den „Kernel-Space“ (System-Calls).



An welchen Orten in der obigen Skizze erfolgt zeitlich nacheinander die Ausführung des folgenden Programmes:

```
int main()
{
    printf("Hello World\n");
    exit(0);
}
```

Tragen Sie die Ablauffolge in die obige Skizze ein.

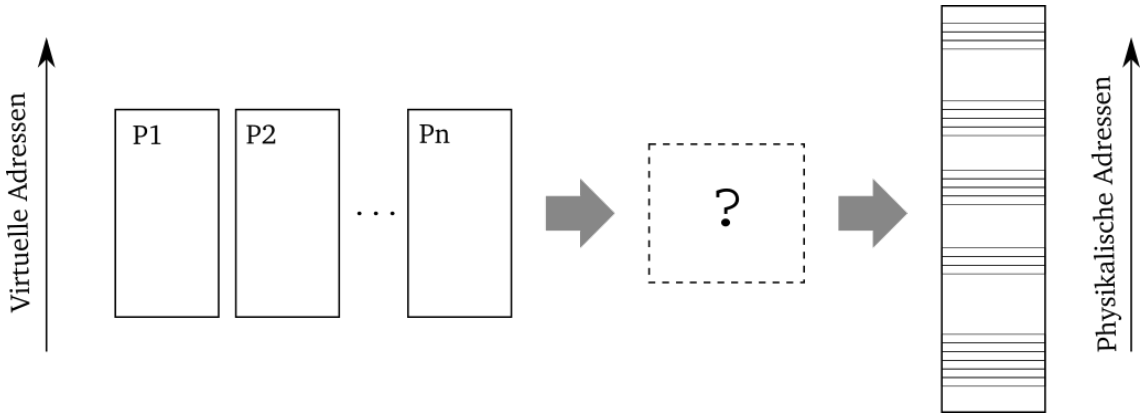
Aufgabe 3 (8 Punkte)

Beschreiben Sie kurz alle Schritte die beim **Starten eines Programmes** durch einen Kommandozeileninterpreter („Shell“) ablaufen. Denken Sie an folgende Begriffe:

- Shell
- Festplatte
- RAM
- Systemaufruf
- Kommandozeilenargumente
- Statisches/dynamisches Linken

Aufgabe 4 (8 Punkte)

Alle Prozesse die auf einem Rechner laufen belegen die gleichen virtuellen Adressen. Wie kann das funktionieren, da beim physikalischen Speicher an einer bestimmten Adresse immer nur ein Prozess sein kann?

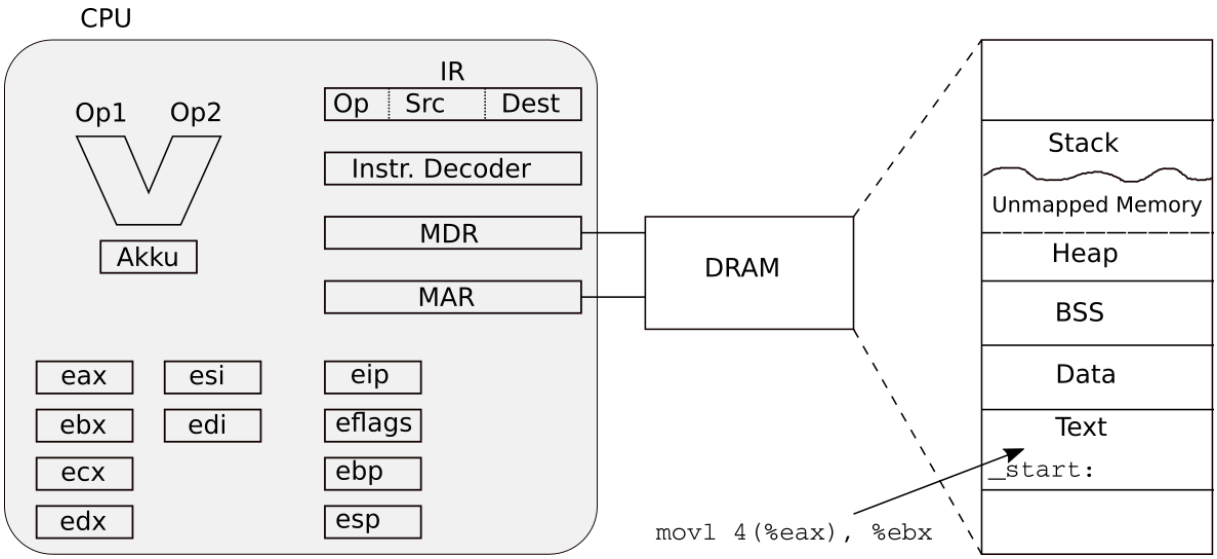


Aufgabe 5 (8 Punkte)

Wie läuft der Befehl

```
movl 4(%eax), %ebx
```

im Fetch-Decode-Execute Diagramm ab?



Aufgabe 6 (6 Punkte)

Bei den folgenden Aufgaben ist immer genau eine der Auswahlmöglichkeiten korrekt:

(a) Welche effektive Adresse wird bei der folgenden Adressierung verwendet:

```
movl 16(%ecx, %edx, 4), %eax     # ecx = 0xf00, edx = 0x10
```

Auswahl: a1 `0xf80` a2 `0xf416` a3 `0xf50`

(b) Wie liegen bei Little Endian die einzelnen Bytes des folgenden Wortes (4 Byte) von niedriger zu hoher Adresse im Speicher: `0x12345678`

Auswahl: b1 `12 34 56 78` b2 `78 56 34 12` b3 `87 65 43 21`

(c) Ab welchem Label startet ein Programm, nachdem es vom Betriebssystem in den Speicher geladen wurde?

Auswahl: [c1] `main` [c2] `_text` [c3] `_start`

(d) Das `esp` Register zeigt auf den Stack. Sie möchten nun mit dem GDB die obersten fünf Werte auf dem Stack in 32-Bit Breite ausgeben.

Auswahl: [d1] `x/5xw $esp` [d2] `x/5xw %esp` [d3] `p/5s $esp`

(e) Bei einem Systemaufruf werden die Argumente übergeben

Auswahl: [e1] auf dem Stack [e2] auf dem Heap [e3] über Register

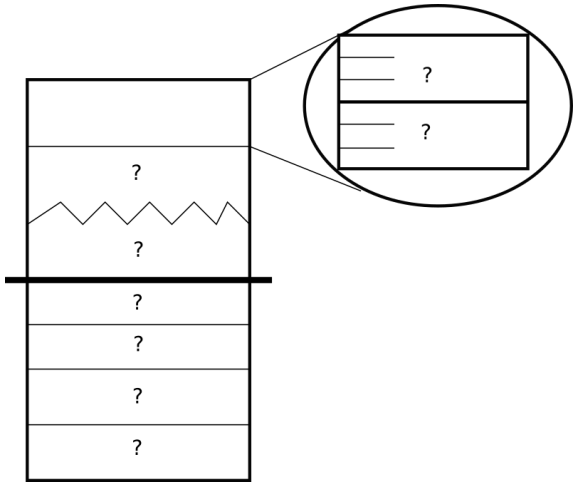
(f) Was dürfen rekursive Funktionen auf keinen Fall haben?

Auswahl: [f1] globale Variablen [f2] lokale Variablen [f3] einen Returnwert

---

Aufgabe 7 (8 Punkte)

Benennen Sie die verschiedenen Abschnitte in dem folgendem Abbild eines Prozesses:



In welchen Abschnitten kommen die folgenden Variablen (1), (2) und (3) zu liegen?

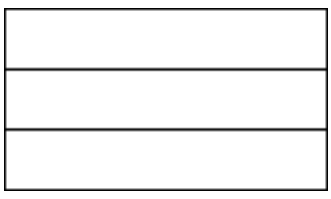
```
int gi = 42;        //  (1)
char buffer[64];    //  (2)

int f()
{
   j = 2020;        //  (3)
   ...
}

int main()
{
   f();
}
```

Aufgabe 8 (8 Punkte)

Schreiben Sie eine Funktion `int strlen(char *cp)` in x86 Assembler. Das Argument `cp` ist ein Zeiger auf einen String, der Rückgabewert ist die Länge des Strings, d.h. die Anzahl der Zeichen im String. Der String ist mit einer Null terminiert.

```
    .section .data

s1: .ascii   "Hallo\0"

    .section .text
    .globl _start

strlen:
    ----------------------   # Prolog
    ----------------------   # Prolog
    ----------------------
    ----------------------
    ----------------------
    ----------------------
    ----------------------
    ----------------------
    ----------------------
    ----------------------
    ----------------------   # Epilog
    ----------------------   # Epilog
    ----------------------

_start:
    pushl   $s1
    call    strlen
    addl    $4, %esp
    movl    %eax, %ebx
    movl    $1, %eax
    int     $0x80
```

Zeichnen Sie das Stack Diagramm direkt nach dem Prolog. Zeichnen Sie auch die Zeiger ESP und EBP ein!



Aufgabe 9 (8 Punkte)

Sehen Sie sich das folgende Assemblerprogramm an und beantworten Sie die unten folgenden Fragen dazu:

```
.section .data

array:
   .byte  1, 2, 3, 4, 5, 6, 7, 8, 0

.section .bss
   .lcomm speicher, 8 * 8

.section .text

.globl _start

_start:
    movl $array, %eax
    movl $speicher, %ebx
loop:
    cmpb $0, (%eax)
    je  exit
    movb (%eax), %cl
    movb %cl, speicher(%ebx)
    addl $1, %eax
    addl $1, %ebx
    jmp  loop
exit:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

1. (4 Punkte) Stellen Sie sich die Variable `speicher` als 2-dimensionale Speichertabelle vor mit 64 Einträgen (siehe die folgende Tabelle). Welche Werte stehen in der Tabelle wenn das Programm am Label `exit` angelangt ist?

2. (4 Punkte) Wie müssen Sie das Programm verändern, so dass die Werte aus `array` in die **Diagonale** der Speichertabelle geschrieben werden? Schreiben Sie die dazu nötige Änderungen rechts neben den obigen Quelltext.

Speichertabelle:

```
<speicher>:    0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+8>:  0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+16>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+24>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+32>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+40>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+48>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
<speicher+56>: 0x__   0x__   0x__   0x__   0x__   0x__   0x__   0x__
```

Aufgabe 10 (6 Punkte)

Statisches und dynamisches Linken von Bibliotheken. Markieren Sie die zutreffenden Aussagen.

☐ Bibliotheken sind eigentlich nicht nötig, da man auch durch die Aufteilung von Programmen in mehrere Quelltextmodule das gleiche erzielen kann.

☐ Statisch gelinkte Programme sind zwar meist gross, können jedoch ohne Probleme auf anderen Rechnern ausgeführt werden, da es keine Abhängigkeiten gibt.

☐ Statisches sowie dynamisches Linken wird beides erst zur Laufzeit des Programmes gemacht.

☐ Dynamisches Linken erzeugt ausführbare Dateien die noch grösser sind als beim statischen Linken, da der dynamische Linker `ld.so` zusätzlich noch im Executable enthalten ist.

☐ Dynamisches Linken erzeugt kleine ausführbare Dateien, jedoch bestehen oft Abhängigkeiten zu vielen dynamischen Bibliotheken, die auch noch dazu in unterschiedlichen Versionen vorhanden sein können.

☐ Wenn man die Funktion `printf()` aus der C Standardbibliothek verwenden möchte, dann muss man unbedingt gegen die C Bibliothek statisch linken.

Ende der Prüfung