Hochschule RheinMain
Studiengang Angewandte Informatik B.Sc.
Prof. Dr. Robert Kaiser

Probeklausur Betriebssysteme ohne Rechnerarchitektur!(LV????)

Wintersemester 2016/2017

Nachname:	Vorname:
Matrikelnummer:	
Datum: 13.02.2017	**Unterschrift:**

Sie erhalten eine geheftete Klausur. **Bitte lösen Sie die Heftung nicht**. Bitte tragen Sie zu Beginn der Bearbeitungszeit Ihren Namen und Ihre Matrikelnummer an den dafür vorgesehenen Stellen ein und unterschreiben Sie die Klausur. Die Klausur ist **nur mit Unterschrift** gültig. Die Klausur muss mit dem Verlassen des Raumes abgegeben werden.

Zum Bestehen der Klausur sind 43 Punkte (50%) notwendig

Im Falle nicht ausreichenden Platzes benutzen Sie bitte zusätzliche Blätter, die Sie mit Name und Matrikelnummer versehen. Machen Sie bitte eindeutig kenntlich, auf welche Aufgabe sich Ihre Antwort bezieht.

Dauer: 90 min

Hilfsmittel: Taschenrechner für arithmetische Operationen, eigene Formelsammlung von maximal einer doppelseitig handbeschriebenen DIN A4 Seite.

Punkte:

Aufgabe	**Soll-Punkte**	**Ist-Punkte**
1	8	
2	4	
3	8	
4	6	
5	8	
6	6	
7	6	
8	8	
9	7	
10	8	
11	4	
12	4	
13	4	
14	6	
Gesamt	87	

Note:

Aufgabe 1: (8 Punkte)

Beantworten Sie bitte folgende Fragen:

Frage	**Antwort**
Warum sind Textdateien zwischen verschiedenen Betriebssystemen (UNIX / MacOS / Windows) nicht ohne Weiteres übertragbar?	
Wie nennt man die spezielle Form eines Semaphors, mit dem sich wechselseitiger Ausschluss (*mutex*) ralisieren läßt?	
Was kennzeichnet eine „race condition“?	
Werden mit `semget()` erzeugte Semaphore in C automatisch zerstört, wenn der erzeugende Prozess terminiert?	
Wie bezeichnet man die Technik der Argumentübergabe von C? (Call by ...)	
Wo wird die Rückkehradresse einer aufgerufenen Routine bzw. Methode zwischengespeichert?	
Was kann bei preemptivem Multithreading vorkommen, bei kooperativem Multithreading hingegen nicht?	
Wie viele verschiedene Werte kann eine Variable vom Typ `unsigned char` annehmen?	
Wie viele verschiedene Werte kann eine Variable vom Typ `char` annehmen?	
Welche Prozess-ID hat bei einem UNIX-System der init-Prozess?	
Was kennzeichnet den Kern eines Betriebssystems?	
Wodurch wird bei einem Betriebssystemaufruf der Wechsel in den privilegierten Modus bewerkstelligt?	
Mit dem „Peterson-Algorithmus“ kann wechselseitiger Ausschluss zwischen maximal .... konkurrierenden Prozessen oder Threads sichergestellt werden.	
Wie sind unter UNIX Dateiverweise („Links“) über Dateisystemgrenzen realisierbar	
Geben Sie ein Beispiel für einen Sekundärspeicher, der **nicht** auf magnetischen Speicherverfahren basiert	
Warum sind SSD-Festplatten i.d.R. schneller als herkömmliche magnetisch aufzeichnende Festplatten?	

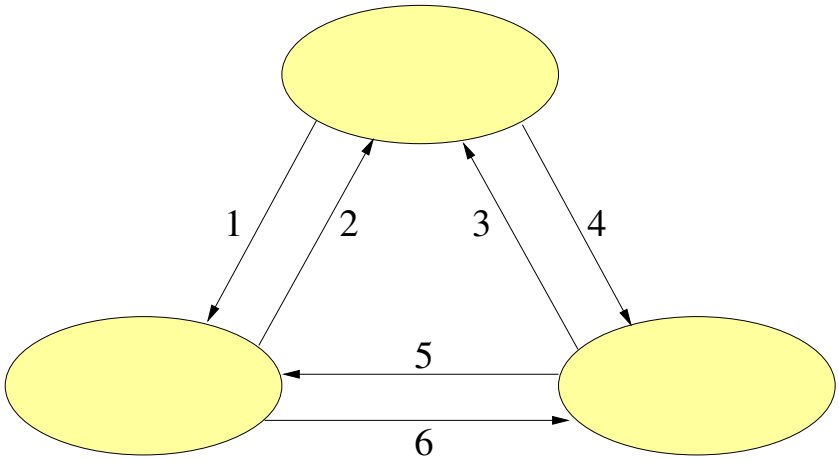
Aufgabe 2: (4 Punkte)

a) Beschreiben Sie kurz, wie ein Buffer-Overflow zu einem Exploit genutzt werden kann. **(2P)**

b) Nennen Sie zwei mögliche Gegenmaßnahmen, um dieser Bedrohung zu begegnen. **(2P)**

Aufgabe 3: (8 Punkte)

a) Tragen Sie bitte im folgenden Zustandsdiagramm die Namen der drei möglichen Zustände ein, die ein Prozess im Laufe seiner Existenz annehmen kann **(1P)**



b) Beschreiben Sie jeweils mit einigen Worten die im Diagramm als Pfeile dargestellten Zustandsübergänge **(3P)**
Hinweis: Einige Übergänge kommen in der Realität nicht vor, geben Sie dies jeweils mit an.

No.	Beschreibung
1	
2	
3	
4	
5	
6	

c) Einer der Übergänge kann auf zwei verschiedene Weisen ausgelöst werden. Welcher ist es, wie werden diese beiden möglichen Übergangsweisen bezeichnet, und worin liegt der Unterschied?**(2P)**

d) Grenzen Sie die Begriffe „Prozess" und „Thread" gegeneinander ab**(2P)**

Aufgabe 4: (6 Punkte)

a) Nennen Sie die Forderungen an einen „guten“ Algorithmus zum wechselseitigen Ausschluss! **(2P)**

b) Grenzen Sie die Begriffe Semaphore, Mutex und Spinlock gegeneinander ab. **(2P)**

c) Welche Unix System Calls existieren für den Umgang mit Semaphoren? **(1P)**

d) Nennen Sie Verfahren zur Betriebsmittelzuteilung, für die das Auftreten eines Deadlocks prinzipiell ausgeschlossen ist. **(1P)**

Aufgabe 5: (8 Punkte)

Die folgenden Aufträge treffen in einem 2-Prozessorsystem zu den angegebenen Zeitpunkten ein. Eine höhere Zahl für die Priorität drücke eine höhere Wichtigkeit aus. Die Aufträge seien reine Rechenaufträge ohne I/O. Die Prozesswechselzeiten werden vernachlässigt.

Auftrag	Ankunftszeit	Bedienzeit	Priorität			
1	0	9	2			
2	2	6	4			
3	3	3	5			
4	4	1	6			
5	5	2	1			
6	7	4	3			
7	10	1	4			
8	11	5	2			

a) Geben Sie für das unterbrechende Prioritäts-Scheduling-Verfahren ein Belegungs-Diagramm für die beiden CPUs für die Abarbeitung der Prozesse an. **(3P)**

b) Bestimmen Sie die mittlere Verweilzeit. (Sie können die freien Spalten in der Tabelle verwenden). **(1P)**

c) Diskutieren Sie das Verhalten eines Round-Robin-Schedulers, wenn das Quantum Q entweder gegen 0 oder gegen unendlich konvergiert und die Zeitdauer für einen Kontextwechsel entweder als Null oder als Konstante c angenommen wird. **(2P)**

Aufgabe 6: (6 Punkte)

Betrachten Sie das Erzeuger-Verbraucher-Problem: Ein Erzeuger-Prozess erzeugt laufend Daten, die über einen Puffer mit `MAX` Elementen an einen Verbraucher-Prozess übermittelt werden sollen. Geben Sie eine Lösung für die Synchronisation an, basierend auf Semaphoren. Dazu seien die folgende Funktionen verfügbar:

`semaphore C(int anfangswert)`	Erzeugen und Initialisieren eines Semaphors mit vorgegebenem Anfangswert für den Zähler
`P(semaphore sem)`	P-Operation nach Dijkstra: Zähler dekrementieren, ggf. Blockieren
`V(semaphore sem)`	V-Operation nach Dijkstra: Zähler inkrementieren, ggf. De-Blockieren

Ergänzen Sie das im folgenden angegebene Pseudocode-Skelett um die Aufrufe zur Initialisierung der Semaphoren sowie zur korrekten Synchronisation im Erzeuger-Programm (producer) und im Verbraucher-Programm (consumer).

```
/**
 * Deklaration der Semaphoren
 */


semaphore




/**
 * Initialisierung der Semaphoren
 */
```

```
/**
 * Erzeuger
 */
function producer()
{
    int datum;
    while(TRUE)
    {



        datum = produce_datum()   /* erzeuge Eintrag */




        insert_datum(datum)       /* fuege Eintrag in den Puffer ein */



    }
}
```

```
/**
 * Verbraucher
 */
function consumer()
{
    int datum
    while(TRUE)
    {




        remove_datum(datum)     /* Eintrag aus Puffer entnehmen */




        consume(datum);         /* verarbeite Eintrag */



    }
}
```

Aufgabe 7: (6 Punkte)

Das folgende C-Programm:

```
#include <stdio.h>

void main()
{
    printf("Zeigergroesse: %d\n", sizeof(void*));
    printf("Zahlengroesse: %d\n", sizeof(int));
    printf("Seitengroesse: %d\n", getpagesize());
    printf("Info: 0x%x\n", *((int*)"1234"));
}
```

liefert auf einem realen Rechner folgende Ausgabe:

```
Zeigergroesse: 4
Zahlengroesse: 4
Seitengroesse: 4096
Info: 0x34333231
```

Hinweis: Der ASCII-Code der Ziffer `"1"` ist `0x31`, der von `"2"` ist `0x32`, usw.

a) Handelt es sich um einen Little- oder Big-Endian Rechner?**(0.5P)**

b) Wieviele Adressbits besitzt dieser Prozessor? **(0.5P)**

c) Wieviele virtuelle Speicherseiten gibt es demnach? **(1P)**
Hinweis: Geben Sie bei dieser und den weiteren Fragen größere Zahlenwerte soweit das möglich ist als Zweierpotenzen an (also z.B. 2^{12} statt 4096).

d) Die Seitentabelle der Memory Management Unit (MMU) des Prozessors ordnet jeder virtuellen Speicherseite einen Eintrag von der Größe einer Integer-Zahl zu. Wieviele Bytes Speicherplatz wären demnach zum Halten einer Seitentabelle erforderlich, die jeder virtuellen Speicherseite einen individuellen Eintrag zuordnet? Wieviele Megabyte[1] wären das?(**1.5P**)

e) Das Betriebssystem des Rechners läßt maximal $2^{10} = 1024$ Prozesse zu. Jeder dieser Prozesse hat seinen eigenen Adressraum und damit seine eigene Seitentabelle. Wieviel Speicherplatz müsste damit das System für das gleichzeitige Halten aller Seitentabellen bereitstellen? **(1P)**

f) Der reale Rechner, auf dem das Programm ausgeführt wurde, hat insgesamt 2^{32} Byte = 4 GB physikalischen Speicher. Mit welchen „Trick" schafft es dieses System, den Platzbedarf für die Seitentabellen so weit zu reduzieren, dass es mit dieser Speichermenge auskommt und dass dabei noch effizientes Arbeiten möglich ist?**(1P)**

g) Würde es Sinn machen, den Hauptspeicher (d.h. das RAM) dieses Rechers über seine aktuelle Größe von 4GB hinaus zu vergrößern? (Begründung)(**0.5P**)

[1] **Hinweis:** 1 Megabyte = 2^{20} Byte

Aufgabe 8: (8 Punkte)

Gegeben sei ein Rechner mit 64 Byte physischem Speicher. Die Seitengröße des Rechners betrage 16 Byte. Die Seitentabelle habe folgenden Inhalt:

Index	**Inhalt**
0	3
1	1
2	2
3	2

Tragen Sie in der unten angegebenen, tabellarischen Darstellung des physischen Speichers ein, welche Inhalte durch die Ausführung des folgenden Programms in welche Speicherzellen geschrieben werden. (**8 P**)

```
main()
{
    char *string = (char*)10;
    char *s      = (char*)48;

    strcpy(string, "Hallo Welt");
    strcpy(s, string);
}
```

Adresse	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0																
16																
32																
48																

Aufgabe 9: (7 Punkte)

Eine Speicherverwaltung arbeitet mit einer Allokations-Bitmap: Ein belegter Speicherblock wird durch eine „1“ in der Bitmap repräsentiert, d.h. Bit Nummer N der Bitmap ist gesetzt, wenn Block Nummer N belegt ist. Der gesamte von der Speicherverwaltung verwaltete Speicher ist 160.000 Bytes groß. Die Zuteilung geschieht nach dem „first fit“-Verfahren, d.h. bei einer Anfrage wird stets ausgehend vom Anfang des Speichers das erste hinreichend große Stück zugeteilt.

a) Wie groß ist bei dieser Speicherverwaltung der Verwaltungs-Overhead[2] (in Prozent), wenn die gewählte Blockgröße

   a1) 1000 Byte oder

   a2) 256 Byte

   beträgt? **(1P)**

b) Die Speicherverwaltung beantwortet nun eine Folge von sechs Anfragen:

Anfrage Nr.	**Angeforderte Anzahl Bytes**
1	999
2	255
3	158.000
4	200
5	256
6	1

Ab welchem Punkt ist der gesamte Speicher erschöpft, wenn die Blockgröße

[2] Verhältnis zwischen für die Verwaltung benötigtem Speicher zu Nutz-Speicher

b1) 1000 Byte oder

b2) 256 Byte

beträgt? **(4P)**

Hinweise:

- Verwenden Sie die beiden nachfolgenden Tabellen
- Tragen Sie zunächst jeweils die Anzahl der belegten Blöcke ein
- Müssen mehr Blöcke belegt werden als vorhanden sind, so ist der Speicher erschöpft

b1) Tabelle für Blockgröße 1000 Byte:

Nr.	Bytes	Kummuliert	Bel.Blöcke	erschöpft?	Bel.Speicher	Verlust	Fragm.
1	999	999					
2	255	1254					
3	158.000	159.254					
4	200	159.454					
5	256	159.710					
6	1	159.711					

b2) Tabelle für Blockgröße 256 Byte:

Nr.	Bytes	Kummuliert	Bel.Blöcke	erschöpft?	Bel.Speicher	Verlust	Fragm.
1	999	999					
2	255	1254					
3	158.000	159.254					
4	200	159.454					
5	256	159.710					
6	1	159.711					

c) Berechnen Sie die interne Fragmentierung[3] (in %), die sich jeweils nach den oben angegebenen Speicheranfragen ergibt. Betrachten Sie auch hier wieder die beiden Blockgrößen 256 Byte und 1000 Byte. **(4P)**
Hinweise:

- Verwenden Sie auch hier die beiden Tabellen
- Der belegte Speicher ergibt sich aus den belegten Blöcken durch Multiplikation mit der Blockgröße
- Der Verlust ist die Differenz aus angefordertem Speicher und belegtem Speicher
- Die Fragmentierung ist das Verhältnis aus Verlust und angefordertem Speicher

[3] Anteil an nicht genutztem Speicher innerhalb der zugeteilten Speicherblöcke

Aufgabe 10: (8 Punkte)

Die unten stehende Tabelle stelle einen Aussschnitt aus dem 32-bit breit organisierten RAM-Speicher eines „Little Endian" Rechners ohne Memory Management Unit dar. Tragen Sie in die einzelnen Felder bitte jeweils in Form zweistelliger Hexadezimalzahlen ein, wie der Inhalt dieses Speichers nach der Ausführung des angegebenen C-Programms aussieht.

Hinweis: Es gilt: `sizeof(char) = 1`, `sizeof(short) = 2` und `sizeof(int) = 4`

```
main()
{
    char  *P = (char*)0x1000;
    short *S = (short*)0x1004;
    int   *L = (int*)0x1008;
    char  *Str = { 0x01, 0x02, 0x03, 0x04, '\0'};

    *P = 0x5a;
    P[1] = 0xa5;
    *S = 0x1234;
    S[1] = 0x5678;
    *L = 0x1234;
    strcpy((char*)&L[1], Str);
    if(S[0] == S[2])
        L[2] = -1;
    else
        L[2] = 0;
}
```

Adresse/Byte	**0**	**1**	**2**	**3**
0x1000				
0x1004				
0x1008				
0x100C				
0x1010				
0x1014				

Aufgabe 11: (4 Punkte)

In der UNIX-Prozessverwaltung spielen die Systemfunktion `fork()` und die `exec()`-Familie eine zentrale Rolle.

a) Erläutern Sie kurz, was beim Aufruf von `fork()` geschieht. (**1P**)

b) Welche Ausgabe erzeugt das folgende Programm? (**1P**) Hinweis: Nehmen Sie dabei an, dass die nächste vom System vergebene Prozess-ID die 1234 ist.

```
1  #include <unistd.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdio.h>
5  int main(void) {
6          int pid, n = 0;
7          printf("Ich habe pid %d\n",getpid());
8          pid = fork();
9          if(pid == 1) {
10                 perror("Fehler bei fork()");
11         } else if (pid == 0) {
12                 printf("Ich bin der Sohn!\n");
13         } else {
14                 printf("Ich bin Vater von pid %d\n",pid);
15         }
16         n = n + 1;
17         printf("%d Tschuess von %d\n", n, getpid());
18         return 0;
19 }
```

Ausgabe:

c) Erläutern Sie kurz, was beim Aufruf von `exec()` geschieht.(**1P**)

d) Welche Ausgabe erzeugt das folgende Programm? (**1P**)
Hinweis: Dabei ist davon auszugehen, dass `/bin/echo` ein ausführbares Programm ist, das seine Argumente auf der Standardausgabe ausgibt.

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      char *argv1[] = {"echo", "Hallo Otto!!", NULL};
7      int i;
8          for(i = 0; i < 15; i++)
9          {
10                 printf("%d\n", i);
11                 execv("/bin/echo", argv1);
12         }
13     printf("Ende\n");
```

```
14      return 0;
15  }
```

Ausgabe:

Aufgabe 12: (4 Punkte)

Das folgende C-Programm besitzt Sektionen für den Programmcode („.text"), initialisierte Daten („.data"), nicht-initialisierte Daten („.bss") nur-lese-Daten („.rodata"), Heap und Stack:

```
char *string1 = "Hallo";
char string2[] = "Welt\n";
int len1 = 5;
int len2;
main()
{
    int len3 = len1 + len2;
    len2 = strlen(string2);

    printf("%s %s", string1, string2);
#ifdef VARIANTE1
    string1 = "Machs gut schnoede";
#else
    strcpy(string1, "Machs gut schnoede");
#endif
    printf("%s %s...und danke fuer den Fisch\n",
     string1, string2);
}
```

a) Ordnen Sie die in der folgenden Tabelle aufgelisteten Symbole und Objekte jeweils einer der genannten Sektionen zu **(2P)**:

Symbol/Objekt	**Sektion**
`string1`	
`string2`	
`len1`	
`len2`	
`len3`	
`main`	
String `"Hallo"`	
String `"Welt\n"`	
String `"Machs gut schnoede"`	
String `"%s %s...und danke fuer den Fisch\n"`	

b) Welcher der folgenden beiden Compileraufrufe erzeugt aus dem angegebenen Quellcode ein Programm, das unter Linux mit einem Segmentation Fault abbricht? **(2P)**

b1) cc -DVARIANTE1 programm.c

b2) cc programm.c

Aufgabe 13: (4 Punkte)

Besonders im Zusammenhang mit Serversystemen hört man häufig den Begriff „RAID".

a) Was versteht man unter einem „RAID-System"? (**1P**)

b) Erläutern Sie bitte die Eigenschaften und je einen Vorteil von RAID 0 und RAID 1 (**5P**).
RAID 0 bedeutet:

Ein Vorteil:

RAID 1 bedeutet:

Ein Vorteil:

c) Ein RAID 5 System werde mit vier Festplatten betrieben. Platten 1 bis 3 dienen der Nutzdatenhaltung, Platte 4 enthält Paritätsdaten. Plötzlich knirscht es vernehmlich in Platte 2 ihre Daten sind nicht mehr lesbar. Die ersten Bits der einzelnen Platten sehen nach diesem Unfall wie folgt aus:

Platte 1: 1 0 1 1 0 1 0 0 1 0 1 1

Platte 2: ? ? ? ? ? ? ? ? ? ? ? ? ?

Platte 3: 0 1 0 1 1 1 0 0 1 1 1 0

Platte 4: 1 1 1 0 0 1 1 0 0 0 1 1

Kann der Inhalt von Platte 2 automatisch wiederhergestellt werden? (bitte ankreuzen) (**1P**)

- ☐ Ja
- ☐ Nein

Falls ja: Erläutern Sie bitte den Wiederherstellungsvorgang und geben Sie die ersten 12 verlorenen Bits für Platte 2 an.(**2P**)

Falls nein: Erläutern Sie bitte Funktionsweise von RAID 5 und insbesondere die Bedeutung der Daten auf Platte 4.(**2P**)

Aufgabe 14: (6 Punkte)

a) Was versteht man unter einer Deadlock-Situation? (**2P**)

b) Es gibt vier Bedingungen,die gleichzeitig erfüllt sein müssen, damit es zu einem Deadlock kommen kann. „Spooling“ und „Preclaiming“ sind zwei Vorgehensweisen, welche jeweils das Eintreten einer dieser Bedingungen (und damit eines Deadlocks) vermeiden. Erläutern Sie bitte, was man unter „Spooling“ und „Preclaiming“ versteht und gegen welche Deadlock-Vorbedingung sie sich jeweils richten. (**4P**)

b1) Spooling wirkt gegen die Deadlock-Voraussetzung _____ und funktioniert wie folgt:

b2) Preclaiming wirkt gegen die Deadlock-Voraussetzung _____ und funktioniert wie folgt: