
Technische Informatik III:
Betriebssysteme und Rechnernetze WS 2007/08
Musterlösung zum Übungsblatt Nr. 1

Aufgabe 1: Begriffe

8 Punkte

Beschreiben Sie jeden der folgenden Begriffe durch maximal 2 Sätze: Benutzerapplikation, Betriebssystem, Systemaufruf, Time Sharing Systeme, Microkernel, monolithischer Kernel, Instruction Set, Interrupt.

- Benutzerapplikation: Software auf der obersten Abstraktionsschicht. Sie hat für den Benutzer eine nützliche Funktion, die nichts direkt mit dem Betrieb des Computers zu tun hat. Diese Software steht im Gegensatz zum Betriebssystem, System- und Hilfsprogrammen.
- Betriebssystem: Software, die die Ressourcen der Hardware den Benutzerapplikationen zuteilt.
- Systemaufruf: Schnittstelle, die es den Benutzerapplikationen erlaubt mit dem Betriebssystem zu kommunizieren, bzw. dessen Funktionen zu nutzen.
- Time Sharing Systeme: Konzept in der mehrere Benutzer, meistens durch Terminals, sich einen Computer teilen. Jedem Benutzer erscheint es so, als ob er das ganze System für sich alleine hat.
- Microkernel: Der Microkernel verwaltet nur Speicher, Prozesse, Kommunikation und grundlegende Systeme
- monolithischer Kernel: Ein monolithischer Kernel verwaltet im Gegensatz zum Microkernel noch zusätzlich Hardwaretreiber und weitere Funktionen.
- Instruction Set: Befehlssatz des Mikroprozessors.
- Interrupt: Unterbrechungsroutine, die das aktuelle Programm, das der Prozessor bearbeitet unterbricht.

Aufgabe 2: Entwicklung von Betriebssystemen

6 Punkte

Fassen Sie kurz in Stichpunkten die historische Entwicklung des Unix-Betriebssystems zusammen. Wann fanden Weiterentwicklungen in den Bereichen Speicherverwaltung, Mehrbenutzerbetrieb sowie der Anbindung an Computernetze statt.

- Loader (1950er) - Erster Ansatz eines Betriebssystems, zum Laden von Lochkarten.
- Batch System (1960er) - Prozesskontrolle.
- Multi-User / Time Sharing Systems (1970er) - Interaktivität, Multibenutzer, Multitasking.
- 1969: Erste Implementierung von Unix auf einem PDP-7 als Alternative zu Multic. Konnte nur von zwei Benutzern benutzt werden (Multiuser Implementierung erst später). Speicherverwaltung: Bis zu 16 KB Speicher für das System, bis zu 8 KB für die Benutzerprogramme, Verwaltung einer 512-KB-Festplatte und Dateien bis 64 KB Größe.
- 1973: Portierung auf C, genannt Unix V4
- 1977: Unix Derivat: Berkeley Software Distribution (BSD)
- 1979: Letzte quelloffene Version Unix V7 von AT&T
- 1982: SunOS (später Solaris) - Unix Version von Sun
- 1983: Unix System V - Implementierung von TCP/IP
- 1987: Andrew S. Tanenbaum entwickelt Lehrsystem Minix
- 1988: POSIX-Standard wird verabschiedet (Portable Operating System Interface).
- 1991: Linus Torvalds entwickelt Linux

Aufgabe 3: Protection Rings

4 Punkte

Beschreiben Sie in kurzen Sätzen die Idee hinter den Protection Rings. Wieviel Ringe werden von gängigen Betriebssystemen zu welchem Zweck verwendet?

Die Idee hinter den Protection Rings ist Sicherheit. Jedes Programm soll nur soviel Rechte haben wie es braucht. Unix hat 4 Ringe. Ring 0 ist für Kernel reserviert. Die Ringe 1 und 2 sind für Bibliotheken und Treiber reserviert. Der Rest, wie zum Beispiel Benutzerapplikation, wird im Ring 3 ausgeführt.

Aufgabe 4: Fifty/Fifty

8 Punkte

Entscheiden Sie, ob folgenden Aussagen zutreffend oder nicht zutreffend sind, und begründen Sie Ihre Entscheidung:

- **Ein Microkernel beschränkt sich auf grundlegende Speicherverwaltung und Kommunikation zwischen Prozessen.**

Richtig. Genau dies sind die beiden Kernaufgaben eines Microkernels.

- **Im Kernelmodus muss das Betriebssystem darauf achten, dass die Speicherbereiche verschiedener Prozesse von einander isoliert sind.**

Richtig. Unabhängig vom Modus hat jeder Prozess seinen eigenen Speicherbereich.

- **Wenn Interrupt Service Routinen nebenläufig abgearbeitet werden, muss das Betriebssystem darauf achten, dass der Prozessor beim Timer-Interrupt nicht in den Benutzermodus wechselt.**

Falsch. Ein Prozess kann in den Benutzermodus wechseln und nicht der Prozessor. Die ISR läuft nicht nebenläufig ab und sie läuft immer im Kernelmodus.

- **Bei Microkernels kommt es häufiger zu einem Kontextwechsel als bei monolithischen Kernen.**

Richtig. Da die meisten üblichen Betriebssystemfeatures nicht im Microkernel selbst zu Verfügung gestellt werden.

Aufgabe 5: Syscalls

8 Punkte

Beschreiben Sie in wenigen Sätzen die vier in der Vorlesung vorgestellten Implementierungen von Systemaufrufen bei monolithischen Kernel und Microkernel in Hinsicht auf den Ablauf der Wechsel zwischen User- und Kernelspace.

- **Unterprogrammaufruf im Betriebssystem (z.B. MS-DOS):**

Hier erfolgt der Systemaufruf durch einen Sprung an eine feste Adresse im Speicher, wo der Systemaufruf implementiert ist. Bei der Übersetzung des Programms wird die Adresse in das Programm geschrieben.

Hier gibt es keine Trennung der Speicherbereiche von Prozessen.

- **Befehl „Systemaufruf“ auf Maschinenebene im monolithischen Kernel (z.B. traditionelles Unix):**

Das Programm erzeugt einen Softwareinterrupt, woraufhin die Interrupt Service Routine (ISR) einen Systemprogramm aufruft (Unterprogramm Aufruf innerhalb des Kernels).

Die Abarbeitung des Interrupt, so wie die des Systemprogramms findet im vollständig im Kernelspace statt.

- **Aufruf eines Systemobjektes/Systemmodul im Microkernel:**

Der Microkernel leitet den Aufruf des Systemobjektes an das zuständige Programm weiter, welches

sein Ergebnis über den Microkernel wieder an das aufrufende Programm übergibt.

Das aufrufende Programm und das Programm für das Systemobjekt laufen im Userspace, und nur der Programmwechsel findet im Kernelspace statt.

- **Versenden einer Aufgabe an einen Systemdienst im Microkernel (z.B. Minix):**

Hier existiert eine Interprozesskommunikationssystem im Microkernel, so dass Prozesse Nachrichten an andere Prozesse senden können. Hier erfolgt der Systemaufruf über dieses Kommunikationssystem durch das Senden einer Nachricht an den zuständigen Systemdienst. Dieser teilt seinerseits die Abarbeitung seiner Aufgabe dem aufrufendem Programm über eine Nachricht im Kommunikationssystem mit.

Hier laufen das aufrufende Programm und der Systemdienst im Userspace und das Versenden und Empfangen der Nachrichten finden jeweils im Kernelspace statt.

Aufgabe 6: C Syntax

4 Punkte

Kommentieren Sie das folgende Programm, ein Kommentar pro Zeile. Welches Spiel wird hier gespielt?

Zahlen raten. Dieses Programm generiert zufällig eine Zahl zwischen 1 und 100 und der Benutzer muss raten, welche es ist. Bei einem Fehlversuch sagt das Programm, ob die geratene Zahl zu groß oder zu klein im Vergleich zur generierten Zahl ist. Dann hat der Benutzer noch einen Versuch. Bei einem Treffer kann sich der Benutzer entscheiden, ob er noch eine Runde spielt.

aufgabel.6.c

```
1 #include <stdlib.h> // Standardbibliothek enthält srand(), rand()
2 #include <string.h> // Bibliothek für string-Manipulation, hier nicht benötigt
3 #include <stdio.h> // Bibliothek für Ein-, Ausgabe
4 #include <time.h> // Bibliothek für die Zeit
5 #define BUFFER_SIZE 100 // Makro für die Buffergröße
6
7 int main(int argc, char *argv[]) { // main-Funktion Einstiegspunkt für das Programm
8     int i, guess, target, goes, max = 100; // Deklaration von Ganzzahlvariablen
9     char buffer[BUFFER_SIZE]; // Deklaration des Lesepuffers
10    srand((unsigned) time(NULL)); // Initialisierung des Zufallszahlengenerators
11    printf("Welcome!\n"); // Ausgabe auf die Kommandozeile
12    for(i = 0; i < 8; i++) { // Beginn for-Schleife
13        printf("="); // Acht '='-Zeichen auf die Kommandozeile
14    } // Ende der for-Schleife
15    printf("\n\n"); // Zweimal Zeilenumbruch
16    do { // Beginn do-while-Schleife, Hauptschleife des Spiels
17        goes = 0; // Zähler für die Anzahl der Versuche
18        target = rand() % max + 1; // Erzeuge Zufallszahl für die zu erratende Zahl
19        guess = 0; // Initialisierung der Variable für den Rateversuch
20        while(guess != target) { // Beginn while-Schleife - Vergleich Rateversuch <-> Ergebnis
21            printf("Enter your guess: "); // Ausgabe auf die Kommandozeile
22            fgets(buffer, BUFFER_SIZE - 1, stdin); // Lese Kommandozeileingabe in den Puffer ein
23            while(!sscanf(buffer, "%d", &guess)) { // Formatierte Ausgabe vom Puffer in
                // Ganzzahlvariabel, wenn erfolglos -> while-Schleife
24                printf("Enter a number! "); // Ausgabe auf die Kommandozeile
25                fgets(buffer, BUFFER_SIZE - 1, stdin); // Ausgabe auf die Kommandozeile
26            } // Ende while-Schleife
27            if(guess < target) // Vergleich Rateversuch < Ergebnis
28                printf("Too low! "); // Ausgabe auf die Kommandozeile
29            else if(guess > target) // Vergleich Rateversuch > Ergebnis
30                printf("Too high! "); // Ausgabe auf die Kommandozeile
31            ++goes; // Inkrementierung des Versuchs Zählers
32        } // Ende der while-Schleife -> richtig geraten
33        printf("Well done, it was %d!\n", target); // Ausgabe auf die Kommandozeile
34        printf("You took %d goes.\n\n", goes); // Ausgabe auf die Kommandozeile
35        printf("Another go? (y/n) "); // Ausgabe auf die Kommandozeile
36        fgets(buffer, BUFFER_SIZE - 1, stdin); // Ausgabe auf die Kommandozeile
37    } while (buffer[0] == 'y' || buffer[0] == 'Y'); // Überprüfe, ob erstes Zeichen 'y' ->
        // Wiederholung des Spiels
38    printf("Goodbye!\n"); // Ausgabe auf die Kommandozeile
39 }
```

Aufgabe 7: C Syntax

8 Punkte

Verwenden Sie einen Text-Editor Ihrer Wahl, um das „Hello, world!“ Programm wie folgt zu modifizieren:

Testlauf:

```
nawab:~/ti3> ./aufgabe1.7
```

```
1. Part:
```

```
Hello, world  
Hello, world  
Hello, world  
Hello, world  
Hello, world
```

```
2. Part:
```

```
Hello, world  
Hello, world  
Hello, world  
Hello, world  
Hello, world
```

```
3. Part:
```

```
Hello, world  
Hello, world  
Hello, world  
Hello, world  
Hello, world
```

```
4. Part:
```

```
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world
```

```
5. Part:
```

```
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world
```

```
6. Part:
```

```
Hello, world  
Goodbye, world  
Hello, world  
Goodbye, world  
Goodbye, world
```

```
7. Part:
```

```
Hello, world?  
yes  
Hello  
Hello, world?
```

no
Goodbye, world
Hello, world?
no
Goodbye, world
Hello, world?
yes
Hello
Hello, world?
yes
Hello

8. Part:
Hello, world?
yes
Hello
Hello, world?
hmm
Hello
Hello, world?
don't know
Hello
Hello, world?
no
Goodbye, world

aufgabe1.7.c

```
1  /**
2  * @file aufgabe.1.7.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 1,
7  * Aufgabe 7: Kommandozeilenparameter
8  *
9  * Dieses Programm gibt auf verschiedene Arten "Hello, world" aus.
10 *
11 * @author Christian Grümme
12 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
13 * @date 2007-10-21
14 */
15 #include <stdlib.h>
16 #include <string.h>
17 #include <stdio.h>
18 #include <time.h>
19 #define BUFFER_SIZE 100
20
21 /** \fn void part_1(unsigned int count)
22 * Gibt count -mal "Hello, world" in einer for-Schleife aus.
23 *
24 * @param count Wie oft "Hello, world" ausgegeben wird
25 */
26 void part_1(unsigned int count)
27 {
28     unsigned int i;          /* Zähler */
29
30     for(i = 0; i < count; ++i) /* for-Schleife */
31     {
32         printf("Hello, world\n"); /* Ausgabe auf die Kommandozeile */
33     }
34 }
35
36 /** \fn void part_2(unsigned int count)
37 * Gibt count -mal "Hello, world" in einer while-Schleife aus.
38 *
39 * @param count Wie oft "Hello, world" ausgegeben wird
40 */
41 void part_2(unsigned int count)
42 {
43     unsigned int i = 0;      /* Zähler */
44 }
```

```

45 | while(i < count)           /* while-Schleife */
46 | {
47 |     printf("Hello, world\n"); /* Ausgabe auf die Kommandozeile */
48 |     ++i;                    /* Inkrementierung des Zählers*/
49 | }
50 |
51 |
52 | /** \fn void part_3(unsigned int count)
53 | * Gibt count -mal "Hello, world" in einer do-while-Schleife aus.
54 | *
55 | * @param count Wie oft "Hello, world" ausgegeben wird
56 | */
57 | void part_3(unsigned int count)
58 | {
59 |     unsigned int i = 0;      /* Zähler */
60 |
61 |     do                       /* do-while-Schleife */
62 |     {
63 |         printf("Hello, world\n"); /* Ausgabe auf die Kommandozeile */
64 |         ++i;                  /* Inkrementierung des Zählers*/
65 |     } while( i < count);
66 | }
67 |
68 | /** \fn void part_4(unsigned int count)
69 | * Gibt count -mal abwechselnd "Hello, world" und "Goodbye, world" in einer for-Schleife aus.
70 | *
71 | * @param count Wie oft "Hello, world" und "Goodbye, world" ausgegeben wird
72 | */
73 | void part_4(unsigned int count)
74 | {
75 |     unsigned int i = 0;      /* Zähler */
76 |
77 |     for(i = 0; i < count; ++i) /* Äußere for-Schleife */
78 |     {
79 |         if( (i % 2) == 0)
80 |         {
81 |             printf("Hello, world\n"); /* Ausgabe auf die Kommandozeile */
82 |         }
83 |         else
84 |         {
85 |             printf("Goodbye, world\n"); /* Ausgabe auf die Kommandozeile */
86 |         }
87 |     }
88 | }
89 |
90 | /** \fn void part_5(unsigned int count)
91 | * Gibt count -mal abwechselnd "Hello, world" und "Goodbye, world" in einer for-Schleife aus.
92 | *
93 | * @param count Wie oft "Hello, world" und "Goodbye, world" ausgegeben wird
94 | */
95 | void part_5(unsigned int count)
96 | {
97 |     unsigned int j,i = 0;    /* Zähler */
98 |
99 |     for(i = 0; i < count; ++i) /* Äußere for-Schleife */
100 |    {
101 |        j = i % 2;           /* Bestimme, ob i gerade oder ungrade ist */
102 |
103 |        switch(j)            /* switch-Anweisung */
104 |        {
105 |            case 0:
106 |                printf("Hello, world\n"); /* Ausgabe auf die Kommandozeile */
107 |                break;         /* Ende dieses Falles */
108 |            case 1:
109 |                printf("Goodbye, world\n"); /* Ausgabe auf die Kommandozeile */
110 |                break;         /* Ende dieses Falles */
111 |        }
112 |    }
113 | }
114 |
115 | /** \fn void part_6(unsigned int count)
116 | * Gibt count -mal zufällig "Hello, world" oder "Goodbye, world".
117 | *
118 | * @param count Wie oft "Hello, world" oder "Goodbye, world" ausgegeben wird
119 | */
120 | void part_6(unsigned int count)
121 | {
122 |     unsigned int target, i = 0; /* Zufallszahlenspeicher, Zähler */
123 |     unsigned int max = 2;       /* Maximum der Zufallszahlen */
124 |     srand((unsigned) time(NULL)); /* Initialisierung des Generators */
125 |
126 |     for(i = 0; i < count; ++i) /* Äußere for-Schleife */

```

```

127 {
128     target = rand() % max;    /* generiere Zufallszahl */
129     if(target == 0)
130     {
131         printf("Hello, world\n");    /* Ausgabe auf die Kommandozeile */
132     }
133     else
134     {
135         printf("Goodbye, world\n"); /* Ausgabe auf die Kommandozeile */
136     }
137 }
138 }
139
140 /** \fn void part_7(unsigned int count)
141  * Fragt count -mal "Hello, world?", jeweils liebt die Antwort ein und antwortet.
142  *
143  * @param count Wie oft "Hello, world?" gefragt wird
144  */
145 void part_7(unsigned int count)
146 {
147     unsigned int i = 0;    /* Zufallszahlenspeicher, Zähler */
148     char buffer[BUFFER_SIZE]; /* Puffer für die Eingabe */
149
150     for(i = 0; i < count; ++i) /* Äußere for-Schleife */
151     {
152         printf("Hello, world?\n");    /* Ausgabe auf die Kommandozeile */
153         fgets(buffer, BUFFER_SIZE - 1, stdin);
154         if(buffer[0] == 'y' || buffer[0] == 'Y')
155         {
156             printf("Hello\n");    /* Ausgabe auf die Kommandozeile */
157         }
158         else if(buffer[0] == 'n' || buffer[0] == 'N')
159         {
160             printf("Goodbye, world\n"); /* Ausgabe auf die Kommandozeile */
161         } else {
162             printf("?\n");    /* Ausgabe auf die Kommandozeile */
163         }
164     }
165 }
166
167 /** \fn void part_8(void)
168  * Fragt "Hello, world?", liebt die Antwort ein solange bis 'n' geantwortet wird.
169  */
170 void part_8(void)
171 {
172     unsigned int end = 1;    /* Wahrheitswert */
173     char buffer[BUFFER_SIZE]; /* Puffer für die Eingabe */
174
175     while(end)    /* Unendliche while-Schleife */
176     {
177         printf("Hello, world?\n");    /* Ausgabe auf die Kommandozeile */
178         fgets(buffer, BUFFER_SIZE - 1, stdin);
179         if(buffer[0] == 'n' || buffer[0] == 'N')
180         {
181             end = 0;    /* Ende der Schleife */
182             printf("Goodbye, world\n"); /* Ausgabe auf die Kommandozeile */
183         }
184         else
185         {
186             printf("Hello\n");    /* Ausgabe auf die Kommandozeile */
187         }
188     }
189 }
190
191 /** \fn int main(int argc, char *argv[])
192  * Hauptfunktion zum Aufrufen der obernen Teilaufgaben
193  * @param argc Anzahl der Argumente
194  * @param argv Argumente, werden ignoriert
195  * @return Gibt bei Erfolg 0 aus.
196  */
197 int main(int argc, char *argv[])
198 {
199     printf("1. Part:\n");
200     part_1(5);
201     printf("\n2. Part:\n");
202     part_2(5);
203     printf("\n3. Part:\n");
204     part_3(5);
205     printf("\n4. Part:\n");
206     part_4(10);
207     printf("\n5. Part:\n");
208     part_5(10);

```

```

209     printf("\n6. Part:\n");
210     part_6(5);
211     printf("\n7. Part:\n");
212     part_7(5);
213     printf("\n8. Part:\n");
214     part_8();
215     return EXIT_SUCCESS; /* Ende des Programms */
216 }

```

Aufgabe 8: Kommandozeilenparameter

4 Punkte

Schreiben Sie ein Programm, das seine Kommandozeilenparameter in umgekehrter Reihenfolge ausgibt.

Testlauf:

```

nawab:~/ti3> ./aufgabe1.8 I'm going the 8th street down
down
street
8th
the
going
I'm

```

aufgabe1.8.c

```

1  /**
2  * @file aufgabe.1.8.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 1,
7  * Aufgabe 8: Kommandozeilenparameter
8  *
9  * Dieses Programm gibt alle seine Kommandoparameter umgekehrt aus.
10 *
11 * @author Christian Grümme
12 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
13 * @date 2007-10-21
14 */
15 #include <stdlib.h> /* Standardbibliothek */
16 #include <stdio.h> /* Bibliothek für Ein- und Ausgabe */
17
18 /** \fn int main(int argc, char *argv[])
19 * Hauptfunktion gibt die Argumente umgekehrt aus.
20 * @param argc Anzahl der Argumente
21 * @param argv Argumente
22 * @return Gibt bei Erfolg 0 aus.
23 */
24 int main(int argc, char *argv[])
25 {
26     if(argc == 1) /* Überprüfung, ob ein Argument existiert */
27     {
28         printf("No input!\n"); /* Ausgabe auf die Kommandozeile */
29     }
30     else
31     {
32         while(argc >= 2) /* Schleife durch alle Argumente */
33         {
34             printf("%s\n", argv[argc - 1]); /* Ausgabe des letzten Arguments */
35             --argc; /* Dekrementiere Argumentenzähler */
36         }
37     }
38     return EXIT_SUCCESS;
39 }

```