Übung zur Vorlesung Betriebssysteme und Systemprogrammierung im WS 2010/11

1.1 C-Programm

Aufgabe 1: Parameterübergabe an Kommandos

```
#include <stdio.h>

int     main(int argc, char *argv[])
{
        int     i;

        printf("Programm: %s, argc: %d\n", argv[0], argc);
        for (i = 1; i < argc; ++i)
        {
                printf("Argument %3d: <%s>\n", i, argv[i]);
        }
        return 0;
}
```

Übersetzen Sie das Beispielprogramm `args.c` und starten Sie es mit jeweils den nachfolgenden Parametern:[1]

1. (ohne Parameter)
2. „`eins zwei drei`“
3. „`eins"zwei" drei`“
4. „`'eins' "zwei drei"`“
5. „`""`“ (leere Zeichenkette zwischen doppelten Gänsefüßchen)
6. „`"'eins' zwei" drei`“
7. „`'"eins" zwei' drei`“
8. „`*.\*.* ~ \\\\`“
9. „`* "$HOME" * '$HOME'`“
10. „`"'$nix'" '"$nix"'`“

Frage 1.1: Welche Parameter erhält jeweils das Programm? [4]

Frage 1.2: Können Sie feststellen, wie viele Argumente maximal übergeben werden können, bzw. wie lang die Kommandozeile höchstens sein darf? [2]

Aufgabe 2: Ein- und Ausgabe mit Bibliotheksfunktionen

```
 1:#include     <stdio.h>
 2:#include     <string.h>
 3:#include     <errno.h>
 4:
 5:int main(int argc, char *argv[])
 6:{
 7:    FILE     *ifp, *ofp;
 8:    int      result = 0;
 9:
```

[1] Da dieses Programm in Standard-C geschrieben wurde, sollte es auch unter MS-DOS oder Microsoft Windows funktionieren. Eine lehrreiche Fleißaufgabe bestünde darin, die jeweiligen Ergebnisse zu vergleichen.

```
10:     if (--argc != 2) {
11:             fprintf(stderr, "%s: Two arguments are required\n", argv[0]);
12:             return 1;
13:     }
14:     if ((ifp = fopen(argv[1], "r")) != NULL) {
15:             if ((ofp = fopen(argv[2], "w")) != NULL) {
16:                     char    buffer[BUFSIZ];
17:                     size_t  size;
18:
19:                     while ((size = fread(buffer, 1, sizeof(buffer), ifp))
20:                            > 0) {
21:                             if (fwrite(buffer, 1, size, ofp) != size) {
22:                                     fprintf(stderr,
23:                                             "%s: fwrite() failed: %s\n",
24:                                             argv[0], strerror(errno));
25:                                     result = 1;
26:                                     break;
27:                             }
28:                     }
29:                     if (fclose(ofp) != 0) {
30:                             fprintf(stderr,
31:                                     "%s: failed to close output file: %s\n",
32:                                     argv[0], strerror(errno));
33:                             result = 1;
34:                     }
35:             } else {
36:                     fprintf(stderr,
37:                             "%s: failed to open %s for writing: %s\n",
38:                             argv[0], argv[1], strerror(errno));
39:                     result = 1;
40:             }
41:             if (fclose(ifp) != 0) {
42:                     fprintf(stderr,
43:                             "%s: failed to close input file: %s\n",
44:                             argv[0], strerror(errno));
45:                     result = 1;
46:             }
47:     } else {
48:             fprintf(stderr, "%s: failed to open %s for reading: %s\n",
49:                     argv[0], argv[1], strerror(errno));
50:             result = 1;
51:     }
52:     return result;
53:}
```

Das C-Programm „copy1.c“ demonstriert die Verwendung von Ein- und Ausgaberoutinen aus der C-Bibliothek, sowie die Fehlerbehandlung. Dieses Programm sollte auf jedem System übersetzbar und lauffähig sein, auf dem ein Standard C-Compiler verfügbar ist.

Verfolgen die den Programmablauf, und untersuchen Sie, welche Betriebssystemroutinen des Systemkerns aufgerufen werden. Vergleichen Sie diese mit den Routinen der C-Bibliothek. [5]

Aufgabe 3: Dateioperationen mit Systemaufrufen

```
 1:#include     <stdio.h>
 2:#include     <unistd.h>
 3:#include     <string.h>
 4:#include     <errno.h>
 5:#include     <sys/types.h>
 6:#include     <sys/stat.h>
 7:#include     <fcntl.h>
 8:
 9:int main(int argc, char *argv[])
10:{
```

```
11:     int      ifd, ofd;
12:     int      result = 0;
13:
14:     if (--argc != 2) {
15:              fprintf(stderr, "%s: Two arguments are required\n", argv[0]);
16:              return 1;
17:     }
18:     if ((ifd = open(argv[1], O_RDONLY)) != -1) {
19:              if ((ofd = open(argv[2], O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR))
20:                  != -1) {
21:                       char     buffer[BUFSIZ];
22:                       size_t   size;
23:
24:                       while ((size = read(ifd, buffer, sizeof(buffer))) > 0) {
25:                                if (write(ofd, buffer, size) != size) {
26:                                         fprintf(stderr,
27:                                                  "%s: write() failed: %s\n",
28:                                                  argv[0], strerror(errno));
29:                                         result = 1;
30:                                         break;
31:                                }
32:                       }
33:                       if (close(ofd) != 0) {
34:                                fprintf(stderr,
35:                                         "%s: failed to close output file: %s\n",
36:                                         argv[0], strerror(errno));
37:                                result = 1;
38:                       }
39:              } else {
40:                       fprintf(stderr,
41:                                "%s: failed to open %s for writing: %s\n",
42:                                argv[0], argv[1], strerror(errno));
43:                       result = 1;
44:              }
45:              if (close(ifd) != 0) {
46:                       fprintf(stderr,
47:                                "%s: failed to close output file: %s\n",
48:                                argv[0], strerror(errno));
49:                       result = 1;
50:              }
51:     } else {
52:              fprintf(stderr, "%s: failed to open %s for reading: %s\n",
53:                       argv[0], argv[1], strerror(errno));
54:              result = 1;
55:     }
56:     return result;
57:}
```

Das C-Programm „`copy2.c`“ implementiert die gleiche Funktion wie „`copy1.c`“, verwendet aber stattdessen Systemaufrufe. Verfolgen Sie auch hier den Programmablauf und vergleichen Sie die Systemaufrufe mit dem vorangegangenen Beispiel.

Frage 3.1: Welches der beiden Programme ist vermutlich effizienter? Stellen Sie vor allem theoretische Betrachtungen an. [2]

Aufgabe 4: Gemischte Verwendung

```
1:#include      <stdio.h>
2:#include      <unistd.h>
3:#include      <string.h>
4:
5:int  main(int argc, char *argv[])
6:{
7:      const char         hello[] = "Hello, ";
```

```
 8:     const char        world[] = "world!\n";
 9:
10:     fwrite(hello, 1, strlen(hello), stdout);
11:     write(fileno(stdout), world, strlen(world));
12:     return 0;
13:}
```

Schließlich verwendet das Programm „`mix1.c`“ eine Kombination von Bibliotheks- und Systemaufrufen. Versuchen Sie (ohne das Programm zu starten) herauszufinden, was das Programm ausgeben wird. Übersetzen Sie anschließend das Programm und überprüfen Sie ihre Erwartung. Wenn Sie sich geirrt haben, versuchen zu erklären, was Sie nicht bedacht haben. **[3]**