Tutorium

Grundlagenpraktikum: Rechenarchitektur (IN0005)

Lukas Ketzer

Lehrstuhl für Rechnerarchitektur & Parallele Systeme (Prof. Schulz)
TUM School of Computation, Information and Technology
Technische Universität München

30. April 2024

Allgemeines

- System-Design-Zweig
- Meine Tutorien:
  - Gruppe 18: Do. 10:00 - 12:00 (00.08.059)
  - Gruppe 25: Do. 16:00 - 18:00 (01.06.011)
- Kommunikation
  - Zulip: öffentliche GRA-Streams
  - Zulip DMs: Lukas Ketzer (nur in dringenden Fällen)
  - lukas.a.ketzer@gmail.com (nur in dringenden Fällen)

Tutoriums-Webseite

Abbildung 1 home.in.tum.de/ketz/

Zulip

Abbildung 2 Zulip Gruppe 18 (Do. 10:00 - 12:00

Abbildung 3 Zulip Gruppe 25 (Do. 16:00 - 18:00

Quiz

Was sind Command Line Optionen?

```
> gcc -o main main.c
```

```
> ls -la
```

```
> python3 main.py
```

Was sind Command Line Optionen?

```
> gcc -o main main.c
```

```
> ls -la
```

```
> python3 main.py
```

- Parameter / Argumente, die man an ein Programm übergibt
- Alles was nach dem **Programmnamen** kommt

Was sind Command Line Optionen?

```
> gcc -o main main.c
```

```
> ls -la
```

```
> python3 main.py
```

- Parameter / Argumente, die man an ein Programm übergibt
- Alles was nach dem **Programmnamen** kommt
- Format:
  - `-o value`
  - `--option value`
  - `-ovalue`

Was sind Command Line Optionen?

```
> gcc -o main main.c
```

```
> ls -la
```

```
> python3 main.py
```

- Parameter / Argumente, die man an ein Programm übergibt
- Alles was nach dem **Programmnamen** kommt
- Format:
  - `-o value`
  - `--option value`
  - `-ovalue`
- `getopt()` oder `getopt_long()` in C

Woher kommen die `Command Line Optionen`?

Woher kommen die Command Line Optionen?

```
1    int main(int argc, char** argv) {
2        /* code */
3        return 0;
4    }
```

Woher kommen die Command Line Optionen?

```
1  int main(int argc, char** argv) {
2      /* code */
3      return 0;
4  }
```

- `char** argv`:
  Argument vector, `string`-Array mit den Parametern
- `int argc`:
  Anzahl elemente in `char** argc`-Array

Beispiel

```
> gcc -o main main.c
```

Beispiel

```
> gcc -o main main.c
```

```
char** argv = {"gcc", "-o", "main", "main.c"}

int argc = 4;
```

Aufgabe 1: numquad

- numquad ist die Hausaufgabe
- Wir wollen das Rahmenprogramm dafür schreiben.
- Integrale
- Funktionsweise:
  - `./numquad -a 10.0 -b 10.0 -n 10 fn_1`
  - `./numquad -h` : Help-Message
  - `./numquad -t` : Tests ausführen

Aufgabe 1: Nutzereingabe

Auf welcher `man`-Page kann man Informationen zu `getopt()` nachlesen? (Tipp: `whatis getopt`

Aufgabe 1: Nutzereingabe

Auf welcher `man`-Page kann man Informationen zu `getopt()` nachlesen? (Tipp: `whatis getopt`

```
man 3 getopt
```

Aufgabe 1: optstring

```
int getopt(int argc, char *argv[], const char *optstring);
```

Aufgabe 1: `optstring`

```
int getopt(int argc, char *argv[], const char *optstring);
```

- `const char *optstring`: Enthält alle valide Optionsargumente
  - `:` | Argument
  - `::` | **Optionales** Argument
  - `<nichts>` | **Kein** Argument
- Beispiel:
  - optstring: `"ha:b::c"`

Aufgabe 1: `optstring`

```
int getopt(int argc, char *argv[], const char *optstring);
```

- `const char *optstring`: Enthält alle valide Optionsargumente
  - `:` | Argument
  - `::` | **Optionales** Argument
  - <nichts> | **Kein** Argument
- Beispiel:
  - optstring: "`ha:b::c`"
  - `-h` und `-c`
  - `-a5`
  - `-b6` oder `-b`
- `optstring` für diese Aufgabe:

Aufgabe 1: `optstring`

```
int getopt(int argc, char *argv[], const char *optstring);
```

- `const char *optstring`: Enthält alle valide Optionsargumente
  - `:` | Argument
  - `::` | **Optionales** Argument
  - `<nichts>` | **Kein** Argument
- Beispiel:
  - optstring: "`ha:b::c`"
  - `-h` und `-c`
  - `-a5`
  - `-b6` oder `-b`
- `optstring` für diese Aufgabe: "**`a:b:n:th`**"

Aufgabe 1: Wie funktioniert `getopt()`

- `int getopt(int argc, char *argv[], const char *optstring);`
- `getopt()` erkennt automatisch die Optionselemente (`-x`)
- **Rückgabewert**:
  - Optionselement (`char`)
  - -1: Erfolgreiches aberbeiten von `argv`
  - `?`: Falls **Optionselement** nicht in `optstring`

Aufgabe 1: Wie funktioniert `getopt()`

- `int getopt(int argc, char *argv[], const char *optstring);`
- `getopt()` erkennt automatisch die Optionselemente (`-x`)
- **Rückgabewert**:
  - Optionselement (`char`)
  - -1: Erfolgreiches aberbeiten von `argv`
  - ?: Falls **Optionselement** nicht in `optstring`

```
1         char opt;
2         while((opt = getopt(argc, argv, "ha:b::c")) != -1) {
3             switch(opt) {
4                 case 'h':
5                     /* code */
6             }
7         }
8
```

Aufgabe 1: `-h` und `-t` implementieren

- `-t`: Run `tests()`
- `-h`: `print_help()`

Aufgabe 1: -h und -t implementieren

- -t: Run `tests()`
- -h: `print_help()`

```
1   char opt;
2   while((opt = getopt(argc, argv, "a:b:n:th")) != -1) {
3       switch(opt) {
4           case 'h':
5               print_help(progname);
6               return EXIT_SUCCESS;
7           case 't':
8               tests();
9               return EXIT_SUCCESS;
10          default:
11              print_usage(progname);
12              return EXIT_FAILURE;
13      }
14
15  }
```

Aufgabe 1: Argumente extrahieren

`-n`, `-a` und `-b` implementieren
Wir wollen die 2 von `-n 2` bekommen.

Aufgabe 1: Argumente extrahieren

`-n`, `-a` und `-b` implementieren
Wir wollen die 2 von `-n 2` bekommen.

`optarg`-Variable
Globale Variable, die automatische von `getopt()` gesetzt wird

Wenn **`opt = 'n'`** ist, dann wird **`optarg = "2"`** gesetzt
Falls kein Argument, ist **`optarg = NULL`**

Aufgabe 1: `-a`, `-b` und `-n` implementieren

Aufgabe 1: -a, -b und -n implementieren

```
1   while((opt = getopt(argc, argv, "a:b:n:th")) != -1) {
2       switch(opt) {
3           case 'a':
4               if (!optarg) {
5                   print_help(progname);
6                   return EXIT_FAILURE;}
7               a = atof(optarg);
8               break;
9           case 'b':
10              /* code */
11          case 'n':
12              if (!optarg) {
13                  print_help(progname);
14                  return EXIT_FAILURE;}
15              n = atoi(optarg);
16              break;
17          /* code */
18      }
19  }
```

Aufgabe 1: Positionale Argumente

Wir wollen jetzt noch das letzte Argument, die verwendete Funktion, parsen.

Aufgabe 1: Positionale Argumente

Wir wollen jetzt noch das letzte Argument, die verwendete Funktion, parsen.

Während `getopt()`, wird parallel die Variable `optind` hochgezählt.
`optind` immer auf das aktuell verarbeitet Element in `argv[]`

Wenn alle Optionselemente von `getopt()` verarbeitet wurden,
zeigt `optind` auf die restlichen Argumente in `argv[]`.

```
1     char* fn_name = argv[optind];
```

Aufgabe 1: Positionale Argumente implementieren

```
1     char* fn_name;
2     if(optind < argc) {
3         fn_name = argv[optind];
4     } else {
5         fprintf(stderr, "No function supplied\n");
6         print_usage(progname);
7         return EXIT_FAILURE;
8     }
```

Aufgabe 2: `strtol, strtod, strtof...`

Bis jetzt haben wir `atol` und `atof` verwendete.

Problem: Die Funktion kann nicht unterscheiden zwischen **Fehlerhafter Eingabe** und **0.0 als Eingabe**.

Lösung:

- `strtol`
- `strtoul`
- `strtoull`
- `strtof`
- `strtod`

Aufgabe 2: `strtol, strtod, strtof...`

```
long strtol(const char *ptr, char **endptr, int base);
float strtof(const char *ptr, char **endptr);
```

Funktionsweise

- `const char* ptr` : Der String mit der Zahl
- `char** endptr` : String, in den der fehlerhafte Teil des Strings geschrieben wird.

strtol: Erfolgreich

```
1     const char* ptr;
2     char* endptr;
3
4     // Erfolgreich
5     ptr = "1234"
6     long out = strtol(ptr, &endptr, 10);
7     /*
8     out = 1234
9     entptr = "\0"
10    */
```

strtol: Fehlerhaft

```
1     const char* ptr;
2     char* endptr;
3
4     // Fehlerhaft
5     ptr = "12345aaa3442";
6     long out = strtol(ptr, &endptr, 10);
7     /*
8     out = 12345
9     char* endptr = "aaa3442";
10    */
```

strtol in numquad.c

```
1     while((opt = getopt(argc, argv, "a:b:n:th")) != -1) {
2         switch(opt) {
3             case 'a':
4                 /* code */
5
6                 a = strtod(optarg, &entptr);
7                 if ((*entptr) != '\0' || errno) {
8                     fprintf(stderr, "Invalid number: %lf\n", a);
9                     return EXIT_FAILURE;
10                }
11                break;
12            case 'b':
13            /* code */
14        }
15    }
```

Vielen Dank für euer Aufmerksamkeit!