

Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2018/19

Übung 6

Rebecca Felsheim
Benedict Herzog

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Dateien & Dateikanäle

- Ein- und Ausgaben erfolgen über gepufferte Dateikanäle
- `FILE *fopen(const char *path, const char *mode);`
 - öffnet eine Datei zum Lesen oder Schreiben (je nach mode)
 - liefert einen Zeiger auf den erzeugten Dateikanal

`r` Lesen
`r+` Lesen & Schreiben
`w` Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
`w+` Lesen & Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
`a` Schreiben am Ende der Datei; Datei wird ggf. erstellt
`a+` Schreiben am Ende der Datei; Lesen am Anfang; Datei wird ggf. erstellt
- `int fclose(FILE *fp);`
 - schreibt ggf. gepufferte Ausgabedaten des Dateikanals
 - schließt anschließend die Datei

1

Ein-/Ausgaben



- standardmäßig geöffnete Dateikanäle
 - `stdin` Eingaben
 - `stdout` Ausgaben
 - `stderr` Fehlermeldungen
- `int fgetc(FILE *stream);`
 - liest ein einzelnes Zeichen aus der Datei
- `char *fgets(char *s, int size, FILE *stream);`
 - liest max. size Zeichen in einen Buffer ein
 - stoppt bei Zeilenumbruch und EOF
- `int fputc(int c, FILE *stream);`
 - schreibt ein einzelnes Zeichen in die Datei
- `int fputs(const char *s, FILE *stream);`
 - schreibt einen null-terminierten String (ohne das Null-Zeichen)

2

- make: Build-Management-Tool
- baut automatisiert ein Programm aus den Quelldateien
- baut nur die Teile des Programms neu, die geändert wurden

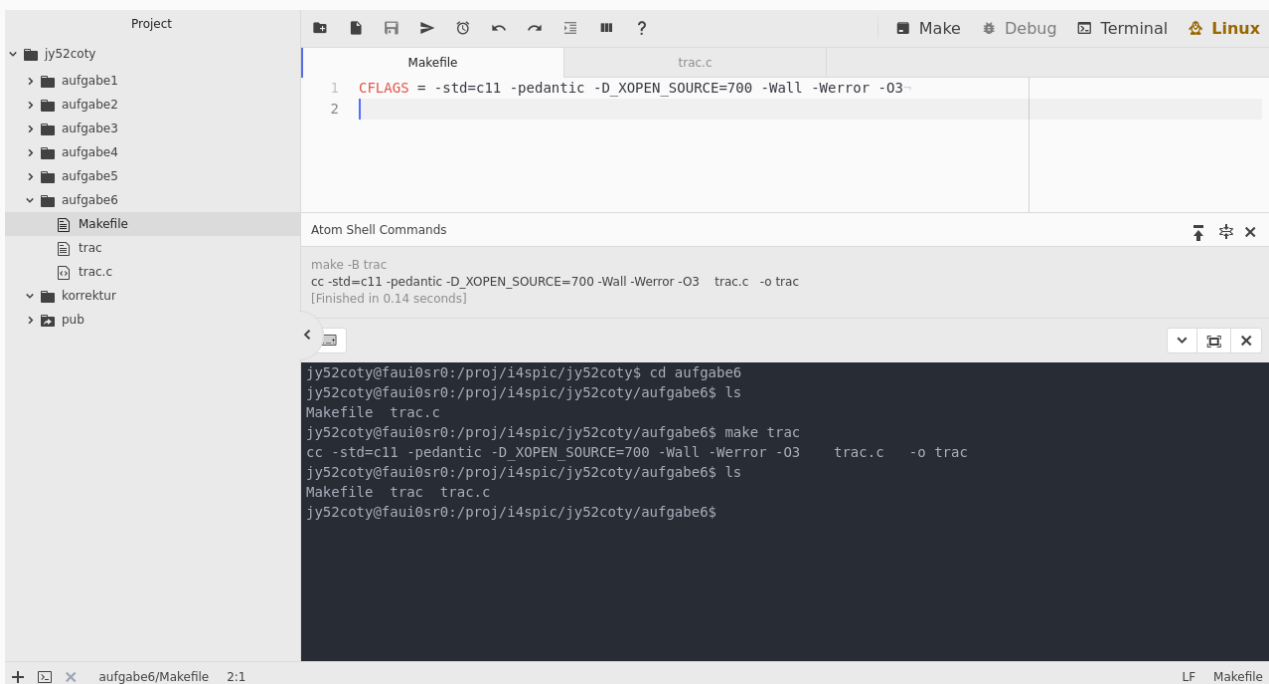
```

01 CFLAGS = -pedantic -Wall -Werror -O3 -std=c11 -D_XOPEN_SOURCE=700
02
03 trac.o: trac.c
04     gcc $(CFLAGS) -c -o trac.o trac.c
05
06 trac: trac.o
07     gcc $(CFLAGS) -o trac trac.o

```

- Objektdati trac.o wird aus Quelldati trac.c gebaut (Compiler)
- Binary trac wird aus Objektdati trac.o gebaut (Linker)

3



The screenshot shows the SPIc-IDE interface. On the left, a project tree shows a folder named 'jy52coty' containing subfolders 'aufgabe1' through 'aufgabe6', and files 'Makefile', 'trac.c', 'korrektur', and 'pub'. The 'Makefile' is selected. The main editor shows the content of the 'Makefile' with two rules:


```

1 CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3
2
trac.o: trac.c
    cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -c trac.c -o trac.o

trac: trac.o
    cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o trac trac.o
    
```

 Below the editor, the 'Atom Shell Commands' panel shows the command 'make -B trac' and its output:


```

make -B trac
cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -c trac.c -o trac.o
[Finished in 0.14 seconds]
    
```

 At the bottom, a terminal window shows the user's commands and the system's responses:


```

jy52coty@fau10sr0:/proj/i4spic/jy52coty$ cd aufgabe6
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ ls
Makefile trac.c
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ make trac
cc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -c trac.c -o trac.o
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$ ls
Makefile trac trac.o
jy52coty@fau10sr0:/proj/i4spic/jy52coty/aufgabe6$
    
```

- SPIc-IDE erkennt Makefiles (Make Button)
 - ⇒ alternativ: make <binary>
- make hat eingebaute Regeln (ausreichend für SPIc)
 - ⇒ nur Angabe der Compilerflags (CFLAGS) nötig

4

Aufgabe: trac

Aufgabe: trac



■ TRAC: TRAnslate Arbitrary Characters

```
01 $ ./trac <FIND> <REPLACE>
02
03 $ ./trac aie bei      # a -> b; i -> e; e -> i
04 dies ist ein Test    # Eingabe
05 deis est ien Tist    # Ausgabe
```

- Ähnlich wie das Kommando `tr(1)` in Unix-artigen Betriebssystemen
- Programmablauf
 - Kommandozeilenparameter prüfen
 - ggf. Fehlermeldung ausgeben und Programm beenden
 - Zeichen einlesen von `stdin` bis EOF oder Fehler
 - Zeichen ggf. entsprechend Mapping ersetzen
 - Zeichen ausgeben
 - evtl. Fehler erkennen und Fehlermeldung ausgeben
 - Programm beenden (mit entsprechendem `exit`-Status)



- `size_t strlen(const char *s);`
 - Länge einer Zeichenfolge bestimmen
→ `strlen(3)`
- `int getchar(void);`
 - Einzelnes Zeichen von `stdin` einlesen
→ `getchar(3)`
- `int putchar(int c);`
 - Einzelnes Zeichen auf `stdout` ausgeben
→ `putchar(3)`
- `int fprintf(FILE *stream, const char *format, ...);`
 - Formatierte Ausgaben auf einem Dateikanal (z.B. `stderr`)
→ `fprintf(3)`



- `int ferror(FILE *stream);`
 - Fehlerzustand eines Dateikanals abfragen
→ `ferror(3)`
- `void perror(const char *s);`
 - gibt die übergebene Zeichenfolge gefolgt von einer Beschreibung des letzten Fehlers auf `stderr` aus
 - nutzt die globale `errno` Variable (⇒ nur für Bibliotheksfehler)
→ `perror(3)`
- `void exit(int status);`
 - Beenden des Programms mit übergebenem exit-Status
→ `exit(3)`

```
01 $ ./trac <FIND> <REPLACE>
```

■ Bedienungsfehler

- falsche Anzahl an Argumenten
- unterschiedliche Länge von FIND und REPLACE
- selbes Zeichen kommt mehrfach in FIND vor

⇒ Erklärung der Verwendung auf `stderr` ausgeben und Programm beenden

■ Ein-/Ausgabefehler

- Zugriff auf eine Datei ist nicht möglich (umgeleiteter Dateikanal)

⇒ Fehler auf `stderr` ausgeben und Programm beenden

8

Fehlerbehandlung `getchar()`



```
01 int c;  
02 while ((c=getchar()) != EOF) {  
03     ...  
04 }  
05  
06 /* EOF oder Fehler? */  
07 if(ferror(stdin)) {  
08     /* Fehler */  
09     ...  
10 }
```

- „`fgetc()`, `getc()` and `getchar()` return the character read as an unsigned char cast to an int **or EOF on end of file or error.**“
- Wie kann man den Fehlerfall von EOF unterscheiden?
⇒ `ferror(3)`

9

```
01 ...  
02 int main(int argc, char *argv[]){  
03     strcmp(argv[argc - 1], ... )  
04     ...  
05     return EXIT_SUCCESS;  
06 }
```

■ Übergabeparameter:

- `main()` bekommt vom Betriebssystem Argumente
- `argc`: Anzahl der Argumente
- `argv`: Vektor aus Strings der Argumente (Indices von 0 bis `argc-1`)

■ Rückgabeparameter:

- Rückgabe eines Wertes an das Betriebssystem
- Zum Beispiel Fehler des Programms: `return EXIT_FAILURE;`

Hands-on: file-concat

- Konkateniert den Inhalt mehrerer Dateien
⇒ Übergebene Dateien öffnen & nacheinander auf stdout ausgeben
- Hilfreiche Funktionen:
 - `fopen(3)` ⇒ Öffnen der Datei
 - `fgetc(3)` ⇒ Einlesen einzelner Zeichen
 - `fputc(3)` ⇒ Ausgeben einzelner Zeichen
 - `fclose(3)` ⇒ Schließen der Datei
- Sinnvolle Fehlerbehandlung beachten
 - Fehlende Dateien melden und überspringen
 - Fehlermeldungen auf stderr ausgeben
- Erweiterung
 - Behandlung von „-“ Zeichen als speziellen Parameter - vgl. `cat(1)`
⇒ Zeichen von stdin einlesen (und auf stdout ausgeben) bis EOF
 - Verwendung: `./file-concat test1.txt - test2.txt`