Blatt 03

Besprechung: 10. Mai 2012

1. filewatch.c (10P)

Schreibt das Programm `filewatch`, das den Zustand von Dateien ueberwacht und bei Aenderungen ein externes Programm aufruft. Die Aufrufsyntax soll folgendermassen sein:

```
filewatch [-i INTERVAL] -e EVENT [-e EVENT]... PRO-
GRAM FILE...
```

Mit `-i` kann man optional ein Interval in Sekunden angeben, wie oft der Zustand ueberprueft werden soll. Waehlt einen sinnvollen Standardwert.

Dem `-e` Schalter folgt der Name eines Ereignisses. Das kann beispielsweise ‘‘appeared’’ oder ‘‘grown’’ sein, ist aber von eurer Implementierung abhaengig. Euer Programm soll mehrere Ereignisse gleichzeitig beobachten koennen.

`PROGRAM` ist der Name eines ausfuehrbaren Programms, das aufgerufen wird wenn Ereignisse stattgefunden haben. Das Programm soll pro Intervall nur maximal einmal aufgerufen werden und alle (Ereignis, Datei)-Paare hintereinander als Argumente uebergeben bekommen. Die Reihenfolge dieser Paare ist egal. Das Programm soll anhand von `$PATH` gefunden werden.

Alle weiteren Argumente (beliebig viele) sind Dateien. Fuer jede dieser Dateien sollen alle angegebenen Ereignisse beobachtet werden.

Das Programm *echo*(1) eignet sich hervorragend zum Testen.

Achtet darauf, dass euer Programm keine Zombiefabrik wird. Die `WNOHANG` Option von *waitpid*(2) ist geeignet, um auf Verdacht evtl. vorhandene Zombies zu beseitigen. Lest in der Manpage nach.

Hinweis: Ein fehlgeschlagenes *stat*(2) und Co. muss nicht zwangslaeufig bedeuten, dass die Datei nicht existiert. Mit Hilfe von `errno` koennt ihr dies unterscheiden. Lest dazu in der Manpage nach.

Achtet darauf, dass euer Programm im Hinblick auf unterstuetzte Ereignisse erweiterbar ist. Ihr duerft dabei annehmen, dass alle Ereignisse nur auf Daten aus einem Aufruf von *lstat*(2) basieren, also auf `stuct stat` und die ggf. gesetzte `errno`. **Eure Implementierung sollte jedes unterstuetzte Ereigniss in einer separaten Funktion kapseln und diese in einer Tabelle per Funktionszeiger mit dem Ereignisnamen verknuepfen.** Dadurch wird eure Implementierung wartbarer. Ueberprueft das selbst, indem ihr weitere Ereignisse hinzufuegt, zumindest aber die folgenden sechs:

- appeared
- vanished
- grown (st_size)
- shrunken (st_size)
- modified (st_mtime)
- accessed (st_atime)

Findet eine Moeglichkeit, die Namen der unterstuetzten Ereignisse auflisten zu lassen.

Um dynamisch Argumentlisten aufzubauen, koennt ihr Herrn Borcherts *strlist*-Implementierung aus der Vorlesung verwenden. Ihr findet sie auf der Vorlesungswebsite.

Beispielaufruf:

```
$ touch bar

$ ( sleep 5 ; touch foo ; rm bar ) &
[1] 13397

$ ./filewatch -e appeared echo foo bar
appeared foo
^C

$ ( sleep 5 ; touch quux ; rm foo ) &
[1] 13402

$ ./filewatch -e appeared -e vanished echo foo quux bla
appeared quux vanished foo
^C

$ ./filewatch -l
all     (watch for all supported events)
appeared
vanished
grown
shrunken
modified
accessed

$ ( sleep 5 ; date >quux ; sleep 1 ; cat quux >/dev/null ) &
[1] 20605

$ ./filewatch -e all echo quux
grown quux modified quux
accessed quux
```

Reicht eure Loesung folgendermassen ein:

```
submit ss2 3 [team] [notes] filewatch.c
```

Fragen zur Selbstkontrolle

Die folgenden Fragen muesst ihr nicht beantworten, ihr solltet sie aber beantworten koennen ... oder sagen wir besser, die einfachen davon. Manche sind auch Einstiegspunkte fuer technische und philosophische Diskussionen. ;-)

- Wie arbeitet eine klassische Unix-Shell schematisch?
- Was wird fuer PIDs zugesichert?
- Wie erzeugt man einen Zombie? Und wie ueberlebt man einen Zombieangriff?
- Was passiert mit Waisen?
- Warum rufen zuweilen Programme zweimal direkt hintereinander `fork()` auf, wenn sie doch nur ein Programm ausfuehren wollen?
- Wozu mehrere exec-Varianten und nicht nur eine? Und warum gibt es `execvpe()` nicht?
- Was ist der Unterschied zwischen einer Bibliotheksfunktion und einem Systemcall? Was von beidem ist `fork()`? Warum?