Blatt 02

Besprechung: 03. Mai 2012

Mining

Beim dezentralen elektronishen Geldsystem *Bitcoin* wird Geld erzeugt indem man ein kryptographisches Problem loest. Diese Arbeit nennt sich dort *Mining*. Ziel dabei ist es, Ausgangsdaten zu finden deren Hashwert kleiner einem vorgegebenen Hashwert, genannt *Target*, ist. Ihr sollt mit eurem Programm ein aehnliches Mining betreiben.

Euer Programm soll einen beliebigen String waehlen, mit Hilfe der Funktion `crypt()` (siehe Manpage) seinen Hashwert generieren und diesen mittels Stringvergleich mit dem Target vergleichen. Ist der getestete Hash kleiner als das Target, dann ist das Problem geloest. Im Falle von Bitcoin, haette man damit Geld generiert.

Um das Mining effizienter zu machen, sollt ihr ein organisiertes Unternehmen auf die Beine stellen. Schliesslich geht es ums Geld generieren.

Ihr seid nun Betreiber einer Mine (`mine.c`). In dieser lasst ihr eine begrenzte Anzahl von Arbeitern (`miner.c`) gleichzeitig arbeiten. Teilt ihnen ein bestimmtes Suchgebiet zu, damit sie sich untereinander nicht in die Quere kommen. Zudem muss jeder Arbeiter wissen wonach er sucht. Mit diesen Informationen schickt ihr eure Arbeiter auf die Suche. Weil euch die Gewerkschaft im Nacken sitzt, solltet ihr eure Arbeiter nicht allzulange am Stueck arbeiten lassen. Erlaubt ihnen auch nach einer festgelegten Anzahl von Suchversuchen erfolglos zurueckzukehren. Schickt dann einfach den naechsten, neuen Arbeiter los. Wiederverwenden muesst ihr keinen eurer Arbeiter, es fehlt euch ja nicht an Nachschub.

Findet ein Arbeiter einen passenden String, dann beendet er seine Arbeit sofort und ruft das gefundene Loesungswort in die Welt. Danach muesst ihr natuerlich keine neuen Arbeiter mehr in die Mine schicken; wartet nur noch bis alle anderen Arbeiter (die unter Tage den Ruf nicht gehoert haben) auch zurueck gekehrt sind und freut euch des Erfolges. (Die Tatsache, dass bei euch Kinder arbeiten, behaltet ihr besser fuer euch.)

1. mine.c und miner.c (10P)

Schreibt die Programme `mine` und `miner`, die jeweils eine *difficulty* uebergeben bekommen. Diese ist eine ganze Zahl zwischen 0 und 12, die angibt an welche Stelle (von links) im String `"............"` der Buchstabe `'z'` gesetzt wird um das Target zu erhalten. Crypt-Hashes beinhalten genau 13 Zeichen aus dem Zeichenvorrat "`./0-9A-Za-z`", von denen der Punkt das (in ASCII) wertmaessig kleinste und das kleine z des wertmaessig groesste Zeichen ist.

Wie ihr das Suchgebiet zwischen euren Arbeitern aufteilt ist euch ueberlassen. Verwendet dafuer weitere Parameter nach Bedarf. Verwendet als *Salt* fuer `crypt()` den String `".."`. Sucht nur nach Woertern aus druckbaren Zeichen um die Darstellung zur vereinfachen. Eure Arbeiterprozesse sollen den Erfolg ihrer Suche in ihrem Exit-Status ausdruecken. Eure Programme sollen `fork()`, `exec()`, und `wait()` verwenden.

Das folgende Codefragment soll als Anhaltspunkt dienen:

```
hash = crypt(guess, "..");
if (strcmp(hash, target) < 0) {
    /* success */
}
```

Auf der Vorlesungswebsite findet ihr ein passendes Makefile.

Reicht eure Loesung folgendermassen ein:

```
submit ss2 2 [team] [notes] mine.c miner.c
```

Fragen zur Selbstkontrolle

Die folgenden Fragen muesst ihr nicht beantworten, ihr solltet sie aber beantworten koennen.

- Weshalb sind bei Unix `fork()` und `exec()` jeweils eigene Systemcalls.
- Was ist der Rueckgabewert von `exec()`? Erklaere!
- Kann `fork()` fehlschlagen? Wann?
- Wie unterscheiden sich die zwei Prozesse nach `fork()`?
- Wenn neue Prozesse nur durch klonen von bestehenden entstehen, woher kommt dann der erste Prozess?
- Warum ist evtl. `fflush()` vor `fork()` noetig? Warum macht `fork()` das nicht von sich aus? (`exit()` kann's doch auch.)