Dr. Andreas Borchert
Christian Ehrhardt
Abteilung Angewandte Informationsverarbeitung
13. Mai 2003
Blatt 3

Systemnahe Software (SS 2003)

Abgabetermin 20.05.2003

Passworte

Versuche, die in diesem Blatt beschriebenen Verfahren auf reale Passworte von unseren Nutzern anzuwenden, können zu nicht unerheblichem Ärger führen.

Überprüfen von Unix-Passworten

Die Funktion `crypt(3c)` dient dazu, ein eingegebenes Klartextpasswort zu verschlüsseln. Neben dem eigentlichen Passwort bekommt diese Funktion noch einen zweiten Parameter (ein String bestehend aus zwei Zeichen), der die Art wie verschlüsselt wird beeinflußt.
Der Rückgabewert der Funktion `crypt` ist ein String, der aus den beiden Buchstaben des zweiten Parameters und dem verschlüsselten Passwort besteht. Dieser Rückgabewert steht dann in der Passworttabelle.
Das Verschlüsselungsverfahren der Funktion `crypt` ist so gewählt, daß es bisher niemandem gelungen ist, ein effizientes Verfahren anzugeben, wie man zu einem verschlüsselten Passwort wieder das Klartextpasswort findet. Statt dessen wird ein vom Benutzer z.B. beim Anmelden eingegebenes Klartextpasswort ebenfalls mit der Funktion `crypt` verschlüsselt (der Wert für den zweiten Parameter läßt sich ja direkt aus dem in der Passworttabelle gespeicherten String ablesen) und das Ergebnis wird mit dem richtigen verschlüsselten Passwort verglichen. Nur bei einer Übereinstimmung wird das Passwort akzeptiert.

Passworte Raten

Für dieses Blatt soll ein Programm geschrieben werden, das mehrere verschlüsselte Passworte aus einer Datei einliest. Jedes Passwort soll daraufhin überprüft werden, ob es eine der Zahlen zwischen 0 und 9999 ist. Dazu müssen für jedes verschlüsselte Passwort alle diese Zahlen getestet werden.

Parallelisierung

Da der oben beschriebene Prozeß relativ lange dauern kann, soll diese Aufgabe parallelisiert (auf mehrere Prozesse verteilt) werden: Nachdem ein Passwort eingelesen wurde, soll dieses Passwort von einem einzig dafür zu erzeugenden Kindprozeß überprüft werden. Der Vater liest sofort das nächste Passwort ein und versucht für dessen Überprüfung einen weiteren Kindprozeß zu erzeugen. Um die Belastung durch die erzeugten Kindprozesse nicht zu hoch werden zu lassen, dürfen zu jedem Zeitpunkt nicht mehr als 4 Kindprozesse existieren. Ggf. muß vor dem Erzeugen eines Kindprozesses auf die Terminierung eines anderen Kindprozesses gewartet werden.

Hinweise

- Um die Funktion `crypt` verwenden zu können, wird auf manchen Systemen die Headerdatei `crypt.h` benötigt. Außerdem muß unter Umständen die Option “-l crypt” beim Übersetzen angegeben werden.
- Relevante Funktionen für dieses Blatt sind `crypt(3c)`, `fork(2)` und `wait(2)`. Der Systemcall `wait(2)` wird ausführlich erst in der nächsten Vorlesung behandelt. Für dieses Blatt genügt es, einen `NULL`-Zeiger als Parameter zu übergeben. Damit wartet `wait` auf die Terminierung eines Kindprozesses und liefert dessen Prozess-ID (oder -1) zurück.
- Der im Skript empfohlene Aufruf von `fflush (NULL);` vor dem Aufruf von `fork` sollte bei diesem Blatt nicht vergessen werden!
- Da die Testprogramme recht rechenintensiv werden können, sollten sie bitte immer mit nice gestartet werden.
- Testprogramme mit fork sollten nach Möglichkeit zuerst auf dem lokalen Rechner und nicht auf einem der Server getestet werden. Dadurch wird der Betrieb im Falle einer Prozeßlawine für andere nicht unnötig beeinträchtigt.
- Das Kommando `killflood` kann alle (eigenen) Prozesse mit einem bestimmten Namen abschießen. Heißt das Programm zum Beispiel a.out, dann könnte killflood a.out im Falle einer Prozeßlawine recht nützlich sein. Ein ebenfalls nützliches aber etwas brutaleres Kommando ist “kill -9 -1”. Dadurch werden alle (eigenen) Prozesse auf dem fraglichen Rechner abgeschossen.