2.Tutorium

Donnerstag, 18. Mai 2017 08:38

blatt2

Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2017	**Aufgabenblatt 2** zu – Systemprogrammierung – Prof. Dr. Odej Kao, Dr. Peter Janacik, TutorInnen
Abgabetermin:	[1] – 21.05.2017 23:55 Uhr [2] – 28.05.2017 23:55 Uhr

Aufgabe 2.1: Prozesse und Threads (1 Punkt) (Theorie[1])

a) Erklären Sie in wenigen Sätzen, warum Betriebssysteme Aufgaben in Prozesse unterteilen. (0,2 Punkte)

b) Was ist ein Process Control Block (PCB)? Wofür wird er benötigt? Nennen Sie fünf Inhalte. (0,4 Punkte)

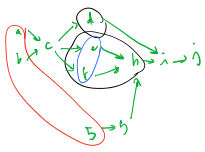
c) Wie wird das automatische Umschalten von Prozessen realisiert? Was wird benötigt? (0,2 Punkte)

d) Was unterscheidet einen User Level Thread von einem Prozess? (0,2 Punkte)

Aufgabe 2.2: Parallelisierung I (1 Punkt) (Theorie[1])

Gegeben ist das folgende nicht-parallele C-Programm. Die Funktionen `jobA ... jobJ` wurden zuvor im Programm implementiert und enthalten längerlaufende Berechnungen.

```
01  int main(void) {
02      int a,b,c,d,e,f,g,h,i,j;
03
04      a = jobA();
05      b = jobB();
06      c = jobC(a,b);
07      d = jobD(a,c);
08      e = jobE(a,b,c);
09      f = jobF(c,b);
10      g = jobG(b);
11      h = jobH(b,e,f,g);
12      i = jobI(c,d,h,f);
13      j = jobJ(a,g,i);
14      return j;
15  }
```



a) Welche Zeilen sind unabhängig voneinander und können in ihrer sequenziellen Reihenfolge verändert werden? Zeichnen Sie hierzu einen Prozessvorgängergraphen. Jede aufgerufene Funktion soll dabei einem Prozess entsprechen. (0,5 Punkte)

b) Schreiben Sie basierend auf dem Prozessvorgängergraphen ein Programm in Pseudocode mit `fork/join` **oder** `parbegin/parend`, das möglichst viele Funktionen parallel ausführt. (0,5 Punkte)

Aufgabe 2.3: Parallelisierung II (Tafelübung)

Abbildung 1: Abhängigkeitsgraph

Wie unterscheidet sich die Herangehensweise von `parbegin/parend` und `fork/join`? Gegeben sei der Abhängigkeitsgraph aus Abbildung 1. Setzen Sie diesen mit Hilfe der aus der Vorlesung bekannten Befehle `fork/join` und `parbegin/parend` in Pseudocode um.

Aufgabe 2.4: Prozessmanagenent (Tafelübung)

Benennen Sie die möglichen Zustände eines Prozesses und skizzieren Sie die Übergänge.

Aufgabe 2.5: Hash Bruteforce (3 Punkte) (Praxis[2])

Aus Sicherheitsgründen werden Passwörter für Websitelogins in der Regel nicht als Klartext gespeichert. Stattdessen werden die Passwörter bei Erzeugung mittels einer Hashfunktion umgerechnet und der sich ergebende Passworthash verschlüsselt in einer Datenbank im Backend abgelegt. Ihnen ist einer dieser Passworthashes in die Hände gefallen. Durch geschickt angewandtes Social Engineering ist Ihnen außerdem bekannt, dass sämtliche Passwörter nur aus Kleinbuchstaben bestehen und fünf Zeichen lang sind. Die Passworthashes wurden mittels der Hashfunktion SHA256 berechnet.

Parallelisieren Sie den gegebenen Bruteforcer nun mittels `fork`, sodass mehrere Hashes gleichzeitig berechnet und verglichen werden können. Die Implementierung des Hashalgorithmus sowie eine grobe Struktur sind vorgegeben und zu ergänzen. Bearbeiten Sie folgende Aufgaben:

a) Teilen Sie den Schlüsselraum der potentiellen Passwörter mithilfe der Funktion `split_work` in sinnvolle Arbeitspakete auf. (0,5 Punkte)

b) Benutzen Sie `fork`, um den aktuellen Prozess in eine variable, über die Kommandozeile übergebene Anzahl von Prozessen zu forken, die `bruteforce` aufrufen. (1 Punkt)

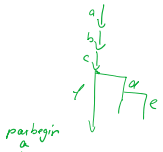
c) Lassen Sie Ihr Programm terminieren, sobald ein Prozess den Klartext gefunden hat. Sie dürfen dazu das Signal `SIGKILL` an alle Prozesse senden. (0,5 Punkte)

d) Ihr Programm soll in der Lage sein den Klartext des gegebenen Hashes zu finden. (0,5 Punkte)

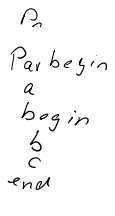
e) Wählen Sie eine möglichst effiziente Anzahl parallel laufender Prozesse und begründen Sie die Wahl. Messen Sie die benötigte Zeit für das parallele und das sequenzielle Programm und geben Sie die Beschleunigung an. (0,5 Punkte)

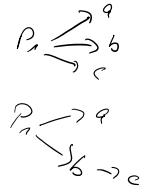
Hinweise

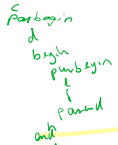
- Nutzen Sie die `manpages`, um sich zu Funktionen wie `fork` zu informieren.
- Sie können die Funktion `split_work` verwenden, um Arbeitspakete aus dem gesamten Schlüsselraum zu schnüren. Diese Funktion nimmt als erstes Argument einen zweidimensionalen `char` Pointer entgegen, der als Array von C Strings benutzt wird. `split_work` legt die C Strings selbst an und prüft dabei nicht, ob Sie den Pointer auf Pointer richtig initialisiert haben. Als zweites Argument wird der Abstand im `charset` angegeben. Beispielsweise generiert ein Abstand von 2 Arbeitspakete mit den Schranken: `a, c, e, ....` Der zweidimensionale Pointer enthält nun Schranken, die `bruteforce` übergeben werden können.
- Sie können die Funktion `bruteforce` verwenden, um einen Teil des Schlüsselraums zu überprüfen. Diese Funktion nimmt den Hash selbst und zwei C Strings entgegen, die Anfang und Ende des zu testenden Teils des Schlüsselraums darstellen.
- Sollten Sie `SIGKILL` benutzen, um Prozesse zu beenden, müssen Sie `join` nicht verwenden.
- Sie finden den Hash in der `main.c`.



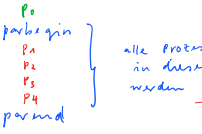
parbegin
a
b
g
parend



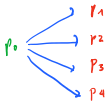


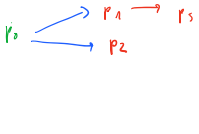


parbegin / parend

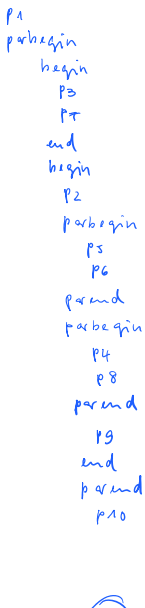


alle Prozesse in diesem Block werden parallel durchgeführt

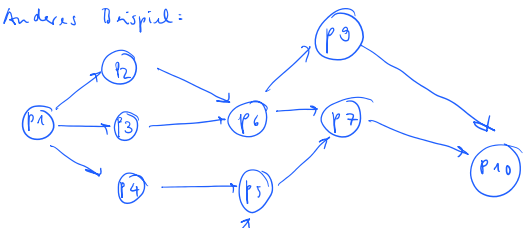




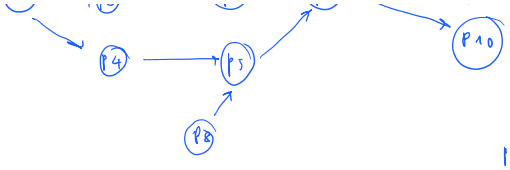
Verschachtelung



∃ Es gibt verschiedene Lösungsmöglichkeiten



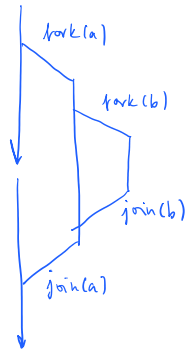
Anderes Beispiel:



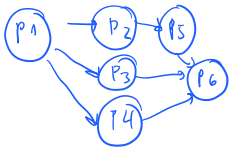

```
parbegin
  P8;
  begin
    P1
    parbegin
      P2
      P3
    parend
      P6
      P9
  end
  P4
  parend
end
parend
  P5
  P7
  P10
```

```
begin
  parbegin
    P2
    P3
    P4
  parend
  parbegin
    P5
    P6
  parend
  parbegin
    P7
    P9
  parend
end
parend
P10
```

fork / join



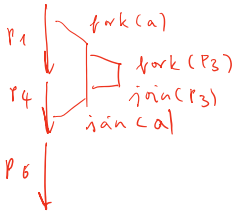
Beispiel



a: P2
P5

```
P1
fork (a)
fork (P3)
P4 →
join (a)
join (P2)
P6
```

wieso P4?
muss sowieso schon laufen, man kann aber auch fork (P4) machen ⇒ langsamer



Aufgabenblatt:

```
P1
fork (a)
P2
fork (P3)
P5
fork (P4)
join (P6)
P8
join (P4)
P9
join (a)
P10
```

a: P3
P7

b: P5
fork(P4)