Vorlesung Betriebssysteme WS 2018/2019

Aufgabenblatt 3 vom 11. Dezember 2018

(Vorstellung der Lösungen bei den Tutoren bis zum 11.01.2019)

1. Untersuchen Sie die Eigenschaften des Betriebssystems Windows mit Hilfe des *Performance Monitors* (`perfmon4.exe`). Beobachten Sie die Zähler für `privileged time`, `processor time` und `user time` für eine gestartete Anwendung, sowie zusätzlich pro Prozessor `interrupts / second`.

   Als Anwendungen vergleichen Sie `notepad.exe` mit einer rechenintensiven Anwendung, wie z.B. `certutil.exe` bei der Hashberechnung[1] einer *großen* Datei.

   Beantworten Sie die folgenden Fragen:

   (a) Beschreiben Sie die gemessenen Werte und erklären Sie deren Bedeutung. Welche Charakteristiken des Betriebssystems werden erkennbar?

   (b) Erklären Sie Veränderungen in den Werten, die durch Maus- oder Tastaturereignisse, sowie Ereignissen im Fenstersystem auftreten. Wie können Sie die gemessenen Werte maximieren?

   Hinweis: Auf Windows 10 erlaubt `perfmon` nicht mehr Messungen genauer als eine Sekunde aufzulösen. Sie können u.U. aussagekräftigere Ergebnisse auf älteren Versionen erhalten.

2. Untersuchen Sie die Eigenschaften des Betriebssystems Linux. Lesen Sie dazu die Manualseiten `top(1)` und `ps(1)`. Beantworten Sie die folgenden Fragen.

   (a) Welche Informationen zeigen `top` und `ps` an? Wie unterscheidet sich das Verhalten der beiden Programme voneinander?

   (b) Woran erkennt man die Prozesse, die als Dämonen laufen?

   (c) Wo finden Sie die von `ps` angezeigten Informationen im Dateisystem wieder?

[1] `https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/certutil#BKMK_hashfile`

3. Machen Sie sich mit der Semantik der Unix-Systemaufrufe `fork` und `wait` vertraut. Betrachten Sie anschließend das folgende Programm. Welche Prozesse existieren zu den Zeitpunkten $t \in \{10s, 30s, ..., 110s\}$? Fertigen Sie einen Prozessgraphen wie in Abbildung 1 gezeigt. Nehmen Sie an, dass zur Ausführung der Systemaufrufe `wait` und `fork` selbst keine CPU Zeit verbraucht wird.

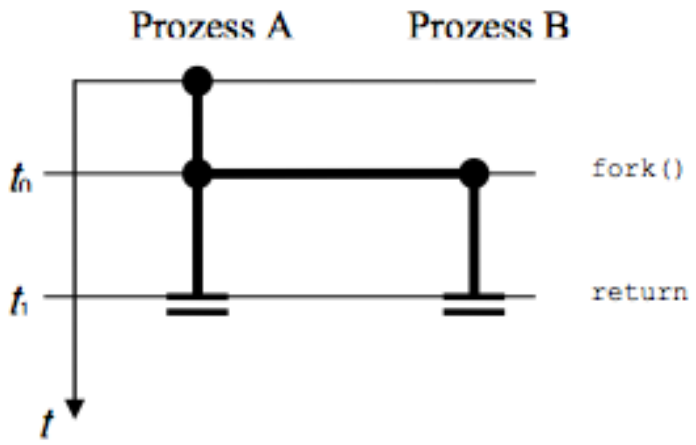

Abbildung 1: Prozessdiagramm für ein einfaches Programm

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int
main(void)
{
  sleep(20);
  fork();
  sleep(20);
  fork();
  sleep(20);
  fork();
  wait(NULL);
  sleep(20);
  fork();
  sleep(20);
}
```

4. Eine Kommunikation zwischen zwei Prozessen erreichen Sie mit einer *Pipe*. Dabei wird die *Standardausgabe* des ersten Prozesses in die *Standardeingabe* des zweiten Prozesses umgeleitet.

   ```
   $ ls | less
   ```

   Schreiben Sie in der Programmiersprache C ein Programm, welches 2 Programme als Kommandozeilenparameter entgegen nimmt und diese mit einer Pipe verbunden startet. Beispiel:

   ```
   $ pipe_example date wc
   ```

   Das Beispiel führt `date` aus und leitet die Ausgabe als Eingabe für `wc` um (`date | wc`). Kommandozeilenparameter für die kombinierten Programme müssen nicht unterstützt werden. Ihr C-Programm soll sowohl unter Windows als auch unter Linux funktionieren. Achten Sie außerdem wie üblich auf korrekte Behandlung und sinnvolle Ausgabe möglicherweise auftretender Fehler.

   Laden Sie Ihren Quelltext bis spätestens 24 Studenten vor Tutoriumstermin in das Abgabesystem hoch.

5. In der Vorlesung wurde der *Bakery-Algorithmus* zur Lösung des *Critical Section* Problems vorgestellt. Implementieren Sie den Algorithmus innerhalb des bereitgestellten Programmrahmens (`bakery.zip`) unter Windows.

   (a) Vervollständigen Sie die Definition von `critical_section_t` in `bakery.h`.

   (b) Vervollständigen Sie die Implementierungen der Funktionen in `bakery.c`.

   Verpacken Sie alle Quellcode-Dateien mit einem Makefile in ein ZIP-Archiv und laden Sie dieses Archiv im Abgabesystem hoch.

6. **(Zusatz)** Spielen Sie Scheduler im *Deadlock Empire*[2].

   Zeigen Sie Ihrem Tutor den Spielfortschritt auf dem Dashboard und erklären Sie ausgewählte Aufgaben.

[2] `https://deadlockempire.github.io`