TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

Betriebssysteme und Sicherheit, WS 2022/23

1. Aufgabenblatt – Unix I

Bearbeitungszeitraum: 17.10.2022 – 21.10.2022

Ziel dieses Übungsblattes ist es diverse Grundlagen von Unix und der systemnahen Programmierung zu erlernen und zu festigen. Dabei werden wir uns beispielhaft mit der Entwicklung von C-Programmen befassen, und damit, wie typischerweise Programme den Hauptspeicher verwenden. Zudem soll die Unix-Prozesserzeugung betrachtet werden.

Aufgabe 1.1 Erläutern Sie die einzelnen Schritte des Übergangs von einem (gedanklich oder schriftlich vorliegenden) Algorithmus über die Implementierung in C bis hin zu einem lauffähigen Programm. Geben Sie dazu jeweils folgende Informationen an:

- Schritt bzw. ausführendes Programm
- Programmname in Unix
- Ergebnis (Bezeichnung und Art der erzeugten Datei, Beschreibung ihres Inhalts)
- Dateiendung unter Unix

Aufgabe 1.2 In der Vorlesung wurde die Adressraumstruktur von Unix eingeführt. Ordnen Sie für das links stehende Beispiel die in der Tabelle enthaltenen Symbole demgemäß ein.

```
#include <stdlib.h>
int a[20];
int x = 1;
void foo(void) {
  int b[20];
  void *p = malloc(100);
  free(p);
}
```

	Text	Data	BSS	Heap	Stack
`a`	☐	☐	☐	☐	☐
`x`	☐	☐	☐	☐	☐
`foo`	☐	☐	☐	☐	☐
`b`	☐	☐	☐	☐	☐
`p`	☐	☐	☐	☐	☐
`*p`	☐	☐	☐	☐	☐

Aufgabe 1.3

(a) Erklären Sie, warum nach `fork()` *parent* und *child* zum einen an der gleichen Stelle des Programms fortgesetzt werden, zum anderen dann aber unterschiedliche Wege gehen können.

(b) Welche Daten werden durch `fork()` von *parent* nach *child* kopiert und welche nicht? Welche Auswirkungen hat das auf den Ablauf von den beiden Prozessen?

(c) Erklären Sie, warum nach `fork()` eine im Hauptspeicher abgelegte globale Variable unterschiedliche Werte in *parent* und *child* haben kann.

Aufgabe 1.4 Entwickeln Sie ein C-Programm, das, ähnlich wie eine *shell*, in einer Endlosschleife auf die Eingabe von Befehlen von einem Nutzer wartet und diese dann als eigenständige Prozesse ausführt. Machen Sie sich dafür nochmals mit der Funktionsweise von `fork()`, `execve()` und `wait()` vertraut. Implementieren Sie das Programm so, dass es nach der Eingabe des Befehls sowohl diesen nochmals an den Nutzer ausgibt als auch die PID des neu gestarteten Prozesses, der den Befehl ausführen soll.

Erarbeiten Sie außerdem den Begriff *Zombie-Prozess* und erläutern Sie, wie er in diesem Zusammenhang entstehen kann.

Klausuraufgaben

Klausuraufgabe I

Beantworten Sie folgende Fragen zu Unix-Prozessen und Shells.

fork.c

```c
int x = 1;
void next() {
   printf("%d\n", x);
   x++;
}
int main() {
   next();
   pid_t pid = fork();
   if (pid > 0) { // Elternprozess
      wait(NULL);
      next();
      next();
   } else if (pid == 0) { // Kindprozess
      next();
   } else { // Fehlerfall
      next();
      printf("Fehler\n");
   }
   return 0;
}
```

a) **Unix-Systemaufrufe:** Welche Ausgabe druckt das obengenannte C-Programm in die Konsole? HINWEIS: Das Einbinden der Header-Dateien sowie ein Teil der Fehlerbehandlung wurden ausgelassen. Gehen Sie von einem fehlerfreien Ablauf aus.

b) **Fehlerbehandlung:** Wir nehmen nun an, dass unmittelbar nach Start des Programms (noch vor dem fork-Aufruf) die maximal erlaubte Prozesszahl des ausführenden Nutzers auf 1 beschränkt wird. Es darf also nur ein einziger Prozess dieses Nutzers gleichzeitig existieren. Geben Sie die Ausgabe an, die in diesem Szenario vom Programm ausgegeben wird.

Klausuraufgabe II

In der Vorlesung wurden verschiedene Prozessmodelle behandelt. Gegeben sei ein Programm, das durch mehrere, voneinander unabhängige Kontrollflüsse realisiert ist. Diese können als schwergewichtige, leichtgewichtige oder federgewichtige Prozesse ausgeführt werden. Geben Sie für jeden dieser Fälle an, ob die links aufgeführten Eigenschaften zutreffen, indem Sie in das entsprechende Feld ein J (ja) oder ein
N (nein) eintragen.

	schwergewichtige Prozesse	leichtgewichtige Prozesse	federgewichtige Prozesse
Mehrere Kontrollflüsse teilen sich einen Adressraum.			
Das Programm lässt sich durch die Verwendung von Multiprozessor-Hardware beschleunigen.			
Die Blockierung eines Kontrollflusses führt zur Blockierung des ganzen Programms.			