

Betriebssysteme, WS 2026/27

2. Aufgabenblatt – Prozesse und Parallelität

Besprechungszeitraum: 03.11. bis 06.11.

Prozess (nach Nehmer) dynamisches Objekt, das selbständige, voneinander isolierte sequentielle Aktivitäten bezüglich des Anforderns und Besitzes von Betriebsmitteln in einem Rechensystem repräsentiert.

Ein Prozess ist gegeben durch:

Adressraum Behälter für die Aufnahme abgegrenzter Programme mit zugeordneten Daten, Abstraktion des physischen Speichers

Handlungsvorschrift oder Code im Adressraum in Form eines sequentiellen Programms gespeichert

Aktivitätsträger oder Thread führt Handlungsvorschrift aus, Abstraktion des physischen Prozessors

Aufgabe 2.1

- (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 der beiden Prozesse?
- (c) Erklären Sie, warum nach `fork()` die Veränderung einer im Hauptspeicher abgelegten globalen Variable unterschiedliche Werte dieser Variable in *parent* und *child* hervorruft.

Aufgabe 2.2 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()`, `exec()` 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.

Aufgabe 2.3 Betrachtet werde ein Mail-Server, der in einer Liste vorliegende Nachrichten an den jeweiligen Empfänger versenden soll; jede Nachricht habe nur einen einzigen Empfänger. Der Server arbeite gemäß dem nebenstehenden Pseudocode. Dabei werde angenommen, dass stets zu versendende Nachrichten vorliegen.

```
while(true) {  
    Nachricht ← Entnehmen(Liste)  
    Empfänger ← Extrahieren(Nachricht)  
    IP ← IP_Ermitteln(Empfänger)  
    Verbindung ← Verbindung_aufbauen(IP)  
    Senden(Nachricht, Verbindung)  
    Verbindung_schließen(Verbindung)  
}
```

- (a) Warum ist diese Lösung ineffizient? Welche Möglichkeiten gibt es, sie zu verbessern? Bietet sie dennoch Vorteile?
- (b) Beschreiben Sie eine Implementation, bei der mehrere Threads innerhalb eines Prozesses genutzt werden können. Welche Vor- und Nachteile hat dieses Vorgehen?
- (c) Welche Konsequenzen hat es, wenn anstelle der Threads selbst wieder sequentielle Prozesse verwendet werden?

Klausuraufgaben

Klausuraufgabe I

```

1  int main() {
2      int pid = fork();
3      printf ("A") ;
4
5      if (pid < 0) {
6          printf("F");
7          exit(1);
8      } else if (pid == 0) {
9          printf("B");
10         wait();
11         exit(0);
12     } else {
13         printf("C");
14         wait();
15     }
16
17     printf("D");
18     exit(0);
19 }
```

Wählen Sie aus den vorgegebenen Ausgaben alle aus, die bei Ausführung des nebenstehenden Unix-Programms entstehen können und notieren Sie die zugeordneten Buchstaben.

Hinweis:

- Die Funktion wait blockiert den aufrufenden Prozess so lange, bis sich einer seiner Kindprozesse beendet. Existiert kein solcher, so kehrt die Funktion sofort zurück.
- Die Auswahl falscher Ausgaben führt zu Punktabzug innerhalb dieser Teilaufgabe!

(a) ABACD	(e) ACABD	(i) ACADB
(b) AF	(f) AFD	(j) AACBD
(c) AACDD	(g) ABBA	(k) ACDAB
(d) AABCD	(h) AFAF	(l) ABABDD

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			