Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Robert Kaiser
Sebastian Flothow
Alexander Schönborn
Daniel Schultz

Betriebssysteme

WS 2020/21
LV 3122

Aufgabenblatt 2
Bearbeitungszeit 2 Wochen
Abgabetermin: 07.12.2020, 4:00 Uhr

In dieser Aufgabe verwenden Sie UNIX-Systemaufrufe, die das Prozesskonzept betreffen. Wie in der Vorlesung besprochen (vgl. Folie 3-26/27), stehen dazu im Wesentlichen zur Verfügung:

`pid_t fork(void);`	Erzeugen eines Kindprozesses (Kopie)
`int execve(char *filename, char *argv[], char *envp[]);` Varianten: `execl`, `execlp`, `execle`, `execv`,... (C lib)	Überlagern des Programms des ausführenden Prozesses
`void exit(int status);` (ANSI C-Funktion, `_exit()` POSIX ohne Aufräumen)	Beendet den ausführenden Prozess mit Status `status` an den Elternprozess
`pid_t wait(int *status);` Variante: `waitpid`	Warten auf Terminieren eines Kindes, Rückgabe des Status des Kindprozesses
`pid_t getpid(void);`	Liefert eigene Prozessidentifikation (pid)
`pid_t getppid(void);`	Liefert Prozessidentifikation des Elternprozesses

Weitere sinnvolle C-Library-Funktionen für diese Aufgabe sind:

`int sleep(int seconds);` (C lib)	Blockiere für n Sekunden
`int system(char *string);` (C lib)	Führe ein Shell-Kommando aus
`int fflush(FILE *datenstrom);` (C lib)	Alle bisher gepufferten Ausgaben schreiben

Zur Beschreibung der Systemaufrufe sei auf die Manual Pages (z.B. `man 2 fork`) verwiesen.

Hilfreich bei dieser Aufgabe sind einige Kommandozeilenbefehle, welche den aktuellen Zustand aller Prozesse oder den Prozessbaum ausgeben können. Diese können Sie in einer separaten Shell eingeben oder auch direkt in Ihren Programmen mittels `system()` aufrufen:

`$ ps f`	Alle Prozesse in der aktiven Konsole als Prozessbaum anzeigen
`$ ps axf` Alternativ: `$ ps -Hej -forest` Oder: `$ pstree`	Alle Prozesse im gesamten System als Prozessbaum anzeigen
`$ date`	Aktuelles Datum und aktuelle Uhrzeit ausgeben

Aufgabe 2.1 (Eltern- und Kindprozesse):

In dieser Aufgabe üben wir die Mechanismen zur Prozesserzeugung und -kontrolle:

- Ein Prozess erzeugt zwei Kindprozesse. Dieser Prozess wird somit zum Elternprozes der beiden Kindprozesse. Die beiden Kindprozesse werden danach parallel ausgeführt.
- Daraufhin gibt der Elternprozess gibt den Text „Elternprozess“, seine eigene Prozess-ID (PID) und die Prozess-IDs der beiden erzeugten Kindprozesse aus.
- Danach schläft der Elternprozess 15 mal für jeweils eine Sekunde und gibt nach der einen Sekunde Wartezeit jeweils die aktuelle Systemzeit (mittels `system("date")`) und den Zustand der beteiligten Prozesse aus (`system("ps f")`).
- Sobald insgesamt 15 Sekunden Wartezeit verstrichen sind, wartet der Elternprozess danach auf die Beendigung der beiden Kindprozesse.
- Wenn sich ein Kindprozess beendet hat, gibt der Elternprozess die aktuelle Systemzeit aus und ob es jeweils der erste oder zweite Kindprozess war (Prozess-IDs prüfen!).
- Danach beendet sich der Elternprozess.
- Der erste Kindprozess gibt am Anfang den Text „Kind1 Start“, seine eigene Prozess-ID und die Prozess-ID des Elternprozesses aus. Danach schläft er für 10 Sekunden, gibt den Text „Kind1 Ende“ aus und beendet sich.
- Der zweite Kindprozess gibt ebenfalls einen eindeutigen Text und die Prozess-IDs aus. Dann führt er wichtige Berechnungen aus, die viel Rechenzeit benötigen, z.B. Bitcoin-Mining. Wir simulieren das durch eine leere Zählschleife, die viel Rechenzeit verbraucht. Passen Sie die Laufzeit der Schleife so an, dass die Ausführung auch ungefähr 10 Sekunden beträgt. Danach gibt der Prozess „Kind2 Ende“ aus und beendet sich.

  Hinweis: Verwenden Sie für die Zählschleife eine Zählvariable vom Typ `volatile uint64_t`. Das Schlüsselwort `volatile` verhindert dabei Compiler-Optimierungen, die eine leere Schleife wegoptimieren würden.

Komplettieren Sie dazu im Aufgabenverzeichnis die folgenden drei C-Programme:

a) Implementieren Sie zunächst in `parent.c` die oben beschriebene Ablauflogik des Elternprozesses *und* der beiden Kindprozesse. Die beiden Kindprozesse werden dabei jeweils im gleichen Programm wie der Elternprozess ausgeführt.

b) Beobachten Sie während der Ausführung des Elternprozesses die Spalte `STAT` in den Ausgaben von `ps`. Können Sie sich die Prozesszustände der Kindprozesse erklären?

c) Lagern Sie die Programmlogik der beiden Kindprozesse in die jeweiligen Programme `child1.c` und `child2.c` aus. Implementieren Sie die Ausführung dieser Kindprogramme durch Überlagerung mittels `execve()` oder Varianten, z.B. `execl()` oder `execv()`, im Elternprogramm.

Aufgabe 2.2 (Zeitmessung):

Implementieren Sie eine Variante des Dienstprogramms `time`!

a) Das Programm `mytime.c` soll das im ersten Argument übergebene Programm als Kindprozess ausführen und alle weiteren Argumente an das Programm durchreichen. Nach Beendigung des ausgeführten Kindprozesses soll `mytime.c` die Prozessidentifikation, den Beendigungsstatus von `exit()` sowie die Ausführungszeit des Kindprozesses in Mikrosekunden ausgeben:

```
$ ./mytime
Fehler: Aufruf: mytime <programm> [<arg> [<...>]]
$ ./mytime /bin/ls -l
insgesamt 80
-rwx------ 1 studi hsrm 11622 Okt 26 17:14 mytime
-rw------- 1 studi hsrm  1065 Okt 26 17:14 mytime.c
...
Prozess 29259 endete mit Status 0 nach 4733 us
$
```

Beachten Sie bei Ihren Tests, dass `execve()` relative oder absolute Pfadangaben erwartet und die Umgebungsvariable `PATH` nicht auswertet. Benutzen Sie `clock_gettime()` zur Zeitmessung.

Hinweis: Sie können längere die korrekte Berechnung der Ausführungszeit mit einem länger laufenden Programm testen, z.B.:

```
$ ./mytime /bin/sleep 3
```

b) Führen Sie das Programm mindestens 5 mal mit einem Beispiel aus und notieren Sie die Ausführungszeiten jeder Progammausführung. Warum unterscheiden sich die Ausführungszeiten, wenn das Programm eigentlich immer das gleiche macht?