Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Reinhold Kröger
Prof. Dr. Martin Gergeleit

Betriebssysteme und Rechnerarchitektur
WS 2015/16
LV 3142

Übungsblatt 2
Bearbeitungszeit 2 Wochen
Abgabetermin: 16.11.2015, 4:00 Uhr

In dieser Übung verwenden Sie UNIX-Systemaufrufe, die das Prozesskonzept betreffen. Wie in der Vorlesung besprochen (vgl. Folie 3-22/23), stehen dazu i.w. zur Verfügung:

pid_t fork(); Hinweis: pid_t löst zum Typ „Signed Integer“ auf.	Erzeugen eines Sohnprozesses (Kopie)
int execve(char *filename, char *argv[], char *envp[]); Varianten: execl, execlp, execle, execv,... (C lib)	Überlagern des Programms des ausführenden Prozesses
pid_t wait(int *status); Variante: waitpid	Warten auf Terminieren eines Sohnes
void exit(int status); (ANSI C-Funktion, _exit() POSIX ohne Aufräumen)	beendet den ausführenden Prozess mit Status status an den Vater
pid_t getpid();	liefert eigene Prozessidentifikation (pid)
pid_t getppid();	liefert Prozessident. des Vaterprozesses

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

int sleep(int seconds); (C lib)	blockiere für n Sekunden
int system(char *string); (C lib)	führe ein Shell-Kommando aus

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

Aufgabe 2.1:

Komplettieren Sie im Übungsverzeichnis die folgenden drei C-Programme:

(a) Das erste Programm `vater.c` soll von einem (Vater-)Prozess ausgeführt werden. Er erzeugt zunächst zwei Sohnprozesse, von denen der erste „Sohn 1“ das Programm (b) und der zweite „Sohn 2“ das Programm (c) ausführt. Wenn er beide Söhne erzeugt hat, gibt er einmalig in einer Zeile den Text "Vater:", seine eigene Prozessidentifikation und die seiner beiden Söhne aus. Anschließend durchläuft er ITERATIONS=6 mal eine Schleife, in der er für SLEEP_TIME=2 Sekunden schläft und dann die aktuelle Zeit ausgibt (mittels des shell-Kommandos `date`). Abschließend wartet er auf die Beendigung seiner beiden Söhne, wobei der Vater keine Beendigungsreihenfolge annimmt, sondern prinzipiell jeder der beiden Söhne sich zuerst beenden kann und der Vater dieses auch so registriert. Für jeden beendeten Sohn gibt er einen entsprechenden Text "Sohn <x> beendet" sowie die Uhrzeit aus. Wenn beide Söhne beendet sind, beendet der Vater sich selbst.

(b) Der erste (Sohn-)Prozess gibt in seinem Programm `sohn1.c` einmalig den Text "Sohn_1", seine Prozessidentifikation und die seines Vaters sowie die Uhrzeit aus, blockiert dann für 10 sec und beendet sich.

(c) Der zweite (Sohn-)Prozess gibt in seinem Programm `sohn2.c` einmalig den Text "Sohn_2", seine Prozessidentifikation und die seines Vaters sowie die Uhrzeit aus, durchläuft anschließend 10 mal eine Schleife, in der er zuerst bis MAX_COUNT=100.000.000 zählt und bei Erreichen einen "." ausgibt und sich beendet. (Dieser Sohn verbraucht im Verhältnis zu Sohn_1 "viel" CPU-Zeit).

Aufgabe 2.2: (Erweiterung der Aufgabe 2.1):

(a) Passen Sie den Wert von MAX_COUNT so an, dass die Programmausführungsdauer des zweiten Sohn-Prozesses ebenfalls nahezu 10 sec beträgt.

(b) Der Vater gibt in seiner Schleife zusätzlich Informationen über alle Ihre Prozesse und deren Zustände und Prioritäten aus (mittels des shell-Kommandos `ps`).

(c) Stellen Sie den Prozessbaum beginnend mit dem Kommandointerpreter als Graph dar.

(d) Beschreiben Sie Ihre Beobachtungen in Hinblick auf die zeitlichen Veränderungen der Prozesszustände in einer Tabelle. Können Sie sich die Beobachtungen erklären?

Aufgabe 2.3: (optionale Erweiterung der Aufgabe 2.2):

Versuchen Sie, experimentell die Arbeitsweise des Schedulers zu ergründen! Denkbares Vorgehen:

(a) Finden Sie anhand einer Internet-Recherche heraus, wie der Scheduler Ihrer aktuell genutzten Linux-Plattform arbeitet.

(b) Können Sie die in (a) gefundene Arbeitsweise experimentell bestätigen?

Bewertung:

Aufgabe	Kriterien	Punkte
vater.c	Söhne erzeugen korrekt umgesetzt	3
	Warten auf Beendigung der Söhne korrekt umgesetzt	3
	Sonstige Programmlogik Vater	2
	Geforderte Ausgaben	1
sohn1/2.c	Überlagerung in Söhnen korrekt umgesetzt / exec()	3
	Sonstige Programmlogik Söhne	2
	Geforderte Ausgaben	1
2.2 (a)	MAX_COUNT angepasst	1
2.2 (b)	Erweiterung	1
2.2 (c)	Prozessbaum vorhanden	1
2.2 (d)	Beschreibung	1
2.2 (d)	Zombies erkannt	1
	Abzüge bei fehlender Return-Code-Behandlung	(-3)
	Abzüge bei Compiler-Warnungen	(-2)
	Abzüge Lesbarkeit / Kommentare	(-2)
	Gesamt	**20**