Prof. Dr. Franz Schweiggert Institut für Angewandte Informationsverarbeitung 13. Mai 2009
Michael Mattes Blatt 4

Systemnahe Software (II)
im Sommersemester 2009

Abgabetermin: 20. Mai 2009

10 Prozessgruppen (8 Punkte)

Um in der Praxis zusätzlich Schutz vor Fork-Lawinen zu erhalten, kann man Prozessgruppen als Vorsichtsmaßnahme wie folgt verwenden:

Ein Prozess erzeugt, bevor er irgendetwas anderes macht, einen Kindprozess. Dieses Kind macht sich zum Prozessgruppenführer einer neuen Prozessgruppe und erzeugt erst dann weitere Kindprozesse. Alle Prozesse der Prozessgruppe verrichten nun irgendwelche Aufgaben. Vermutet der ursprüngliche Elternprozess nun ein unerwünschtes Prozessverhalten innerhalb der Prozessgruppe, so kann er alle Prozesse der Prozessgruppe gemeinsam "killen" (d.h. ihnen Signale schicken), so dass diese terminiert werden.

Setzen Sie dieses Szenario um. Es genügt, wenn die neue Prozessgruppe aus drei Prozessen besteht (indem der Kindprozess lediglich noch zweimal forkt). Die Aufgaben sollen aus der Ausgabe der eigenen PID und der des Erzeugers sowie einem Aufruf von `sleep(100)` bestehen.

In dieser Aufgabe soll die Erkennung von unerwünschtem Verhalten sehr einfach realisiert werden: Der Elternprozess soll einfach nach 10 Sekunden von einer unerwünschten Prozesslawine ausgehen und daraufhin der Prozessgruppe zuerst das Signal SIGTERM (15) und – um ganz sicher zu gehen – einige Sekunden später auch noch das Signal SIGKILL (9) senden.

11 Auffrischung der stralloc-Bibliothek (4 Punkte)

Machen Sie sich noch einmal mit der stralloc-Bibliothek vom vorigen Semester vertraut (Kapitel "Dynamische Strings" im Soft1-Skript). Ihr Tutor wird jedem Mitglied Ihrer Gruppe eine Frage dazu stellen.

Mögliche Fragen wären z.B., wie man einen `stralloc*`-String an einen anderen anhängen kann, wie man eine stralloc-Struktur initialisieren kann, wie man den Speicher dafür wieder freigibt oder auch, welchen Zweck eine bestimmte `stralloc_...()`-Funktion erfüllt.

Das Environment

Bisher wurden zwei Möglichkeiten vorgestellt, mit denen ein Elternprozess dem Kindprozess vor dem Start Daten geben kann. Erstens geht dies natürlich über die Verwendung der kopierten Variablen des Elternprozesses, zweitens bietet sich `argv` an. Die erste Möglichkeit hat übrigens den Nachteil, dass damit keine Informationen an Kinder gegeben werden können, die weitere Programme via `exec()` ausführen.

Nun betrachten wir eine weitere Möglichkeit, einem Programm Daten zukommen zu lassen: Das *Environment* ("Umgebung"). Es handelt sich hierbei um eine Liste von Schlüssel–Wert–Zuordnungen, die als Array von Strings modelliert ist. Die Trennung von Schlüssel und Wert geschieht üblicherweise mittels eines Gleichheitszeichens.

In C ist das Environment über die globale Variable `char **environ` zugänglich, die Sie selbst im eigenen Programm als externe Variable deklarieren (nicht definieren!) müssen. Weitere Informationen bekommen Sie, wenn Sie in exec(2) nach `environ` suchen oder unter Linux environ(7) lesen.

Per `exec()` ausgeführte Programme übernehmen das aktuelle Environment, es sei denn, es wird eine der `exec()`-Varianten mit e am Ende verwendet, also `execve()` oder `execle()`. Diese Varianten erwarten ein NULL-terminiertes Array von neuen Environment-Variablen.

12 Umgang mit dem Environment in der Shell (6 Punkte)

Finden Sie heraus, wie Sie in der `bash`-Shell das Environment (also "Environment variables" bzw. "Umgebungsvariablen") setzen und anzeigen lassen können. Wie geht dies unter Windows?

Erklären Sie Ihrem Tutor/Ihrer Tutorin, welche Bedeutung folgende Umgebungsvariablen haben: HOME; PWD; PATH; PS1 (unter Unix) sowie SystemDrive; SystemRoot; AppData; PATH (unter Windows).

13 Ausgabe des Environments (4 Punkte)

Schreiben Sie ein Programm `envprint.c`, welches das aktuelle Environment komplett ausgibt.

14 Exit-Status durch Environment (4 Punkte)

Schreiben Sie ein Programm `envstatus.c`, welches den Wert der Umgebungsvariable EXITVAL als Exit-Status zurückliefert.

15 Test des Exitstatus (4 Punkte)

Schreiben Sie ein Programm `teststatus.c`, welches das Programm aus der vorigen Aufgabe mit zufälligen Werten testet. Hierzu muss `teststatus.c` forken, `envstatus.c` ausführen und dessen Exitstatus auswerten. Wenn Sie fertig sind, führen Sie statt `envstatus.c` testweise `envprint.c` aus.

Viel Erfolg!