Prof. Dr. Franz Schweiggert Institut für Angewandte Informationsverarbeitung
Michael Mattes

Systemnahe Software (II) im Sommersemester 2009

Abgabetermin: 10. Juni 2009

Dies ist die Gesamtversion des Blatts 6, bestehend aus Teil a und b.

tinysh

Die folgenden Aufgaben befassen sich weiter mit der tinysh. Zum Bearbeiten der Aufgaben empfiehlt es sich, nicht die eigene bearbeitete Version der tinysh zu verwenden, sondern sich in die Beispiellösung des letzten Aufgabenblatts einzuarbeiten.

22 Verbesserter Parser (5 Punkte)

Bisher konnte die tinysh nur dann Eingabeumlenkung durchführen, wenn kein Trennzeichen zwischen den Umlenkungspfeilen standen. Erweitern Sie den Parser der tinysh so, dass die Umlenkung auch mit Trennzeichen zwischen dem Dateinamen und dem Umlenkungspfeil funktioniert.

Ein Beispiel für die alte Variante ist

```
% /bin/echo Hallo >datei
```

Nun soll auch dies funktionieren:

```
% /bin/echo Hallo > datei
```

23 Jobkontrolle (Extrapunkte)

Hinweis: Diese Aufgabe ist freiwillig und wird gesondert bewertet. Schicken Sie Ihre Lösung direkt per E-Mail an den Übungsleiter. Die schönste Lösung wird in den Übungen vorgestellt werden.

Die tinysh besitzt bisher keine Jobkontrolle – es ist also nicht möglich, in der tinysh einen Prozess im Hintergrund zu starten. Bauen Sie diese Möglichkeit ein.

Es bietet sich an, für die Jobverwaltung ein `struct job` oder ähnliches zu definieren, in dem einige Daten eines gestarteten Prozesses verwaltet werden. Hier müssten zumindest die PID des Kindprozesses und eine Jobnummer verwaltet werden.

Außerdem müssen Sie Builtins zur Jobkontrolle schreiben und auch die Möglichkeit schaffen, Jobs mit `&` im Hintergrund zu starten. Hinweise liefert auch die `bash`-Manpage unter dem Stichwort JOB CONTROL.

Signale

Beachten Sie beim Umgang mit Signalbehandlern immer die Hinweise im Skript ab Seite 63 ("Zur Speicherklasse `volatile`") und verwenden Sie nur als `volatile` deklarierte Variablen, lokale Variablen sowie "Async-Signal-Safe"-Funktionen wie `signal()`, `abort()` und `_Exit()`.

24 Signale senden und auswerten (7 + 3 Punkte)

Schreiben Sie ein Programm `sigsend.c`, welches einen Kindprozess erzeugt, kurz wartet und danach erst das Signal SIGHUP und dann das Signal SIGINT an den Kindprozess sendet.

Der Kindprozess soll für beide Signale einen *Signalhandler* einrichten und danach auf ein Signal warten (siehe Manpage `pause(2)`). SIGHUP soll ignoriert werden und der Kindprozess soll sich beim Erhalt von SIGINT beenden.

Wenn dies funktioniert, erweitern Sie das Programm so, dass der Erzeugerprozess ermittelt, welches Signal zur Beendigung des Kindes geführt hat. Falls der Kindprozess nicht durch ein Signal beendet wurde, soll statt dessen der Exitstatus des Kinds ausgegeben werden.

25 Signalbehandlung (6 + 4 + 5 Punkte)

Erweitern Sie das Programm aus der vorherigen Aufgabe so, dass der Kindprozess die Signalnummer des jeweils empfangenen Signals ausgibt. Beachten Sie dabei die Hinweise weiter oben. Der Elternprozess soll nun zusätzlich noch das Signal SIGQUIT senden können (und das Kind dieses abfangen).

Wenn Sie dies geschafft haben, soll der Kindprozess auch noch das Signal SIGTERM abfangen können, dessen Signalnummer ausgeben und danach terminieren.

Zum Schluss sollten Sie Ihrem Tutor erklären können, warum die Einrichtung eines Signalhandlers für die Signal SIGKILL und SIGSTOP nicht funktionieren kann.

Erweiterte Signalbehandlung mit sigaction (Blatt 6b)

Da die Signalbehandlungsöglichkeiten nach dem C-Standard teilweise eingeschränkt sind, gibt es im POSIX-Standard eine umfangreichere Schnittstelle zur Signalbehandlung. Diese wird über den Systemaufruf `sigaction(2)` angesteuert. *Hinweis:* Um `sigaction()` nutzen zu können, muss das Makro `__EXTENSIONS__` definiert werden. In gcc kann man dies mit dem Kommandozeilenparameter `-D__EXTENSIONS__` erreichen.

26 Holzmichl (6 Punkte)

Schreiben Sie ein Programm `holzmichl.c`, welches nur auf das Signal SIGUSR2 wartet und dem Prozess, der es gesendet hat, ein SIGUSR1 zurückschickt. *Hinweis:* Sie müssen hier `sigaction()` verwenden, da Sie sonst nicht wissen können, welcher Prozess das Signal SIGUSR1 verschickt hat.

27 De Randfichten (6 Punkte)

Schreiben Sie ein Programm `randfichten.c`, welches über die Umgebungsvariable MICHL die PID eines `holzmichl.c`-Prozesses erhält und prüft, ob dieser noch "lebt". Hierzu soll das Signal SIGUSR2 geschickt werden und geprüft werden, ob ordnungsgemäß SIGUSR1 zurückgesendet wird. Verwenden Sie auch in `randfichten.c` den Aufruf `sigaction()`.

28 Verwendung von Makefiles (3 Punkte + 3 Extrapunkte)

Schreiben Sie ein Makefile, welches sowohl `holzmichl.c` als auch `randfichten.c` kompilieren kann. Die Extrapunkte gibt es, wenn das Makefile unter der Angabe von `make michltest` in der Lage ist, erst `holzmichl.c` zu starten und danach sogar `randfichten.c` mit der passenden Umgebungsvariable ausführen kann.
Viel Erfolg!