SPiC-Aufgabe #8: mish

(22 Punkte, in Zweier-Gruppen)

Entwerfen und programmieren Sie eine Shell `mish` (mini shell), die Programme (im Weiteren als Kommandos bezeichnet) ausführen kann. Verwenden Sie für Ihre Implementierung den Dateinamen `mish.c`.

Teilaufgabe a (14 Punkte)

- **Grundfunktionalität**
  - Ihr Programm soll als Promptsymbol zunächst den String

    ```
    mish>
    ```

    ausgeben und anschließend auf die Eingabe von Kommandos warten.
  - Die eingelesene Zeile soll in Kommandoname und Argumente zerlegt werden, wobei Leerzeichen und Tabulatoren als Trennzeichen dienen (`fgets(3)`, `strtok(3)`).
  - Das Kommando soll dann in einem neu erzeugten Prozess (`fork(2)`) mit korrekt übergebenen Argumenten ausgeführt werden (`execvp(3)`).
  - Die Shell soll auf das Terminieren des Prozesses warten (`wait(2)`) und den Exitstatus auf dem Standardfehlerkanal ausgeben. Bei der Statusausgabe soll unterschieden werden, ob der Prozess sich selbst beendet hat (`WIFEXITED`, `WEXITSTATUS`), oder ob der Prozess durch ein Signal (`WIFSIGNALED`, `WTERMSIG`) beendet wurde:
    1. Fall: Prozess beendet sich selbst (in diesem Beispiel mit Exitstatus 0):

       ```
       mish> ls -l
       ...
       Exit status [2110] = 0
       ```

    2. Fall: Prozess wird durch ein Signal beendet (in diesem Beispiel ein Interrupt-Signal (`SIGINT=2`) durch Drücken von `CTRL-C`):

       ```
       mish> sleep 10
       Signal [1302] = 2
       ```

       Beachten Sie, dass dieser Fall erst getestet werden kann, nachdem Sie das Ignorieren des `INT`-Signals (siehe unten) implementiert haben.
  - Nach der Ausgabe des Exitstatus soll die Shell wieder eine neue Eingabe entgegennehmen. Das Shell-Programm soll terminieren, wenn es beim Lesen vom Standardeingabekanal ein End-of-File (`CTRL-D`) erhält.
- **Ignorieren des `INT`-Signals**

  Wenn Sie in einem Terminalfenster z.B. durch Drücken von `CTRL-C` ein Signal auslösen, so wird dieses Signal allen Prozessen in der Prozessgruppe des Terminalfensters zugestellt, also insbesondere sowohl der `mish` als auch einem evtl. gerade laufenden Kindprozess.
  Erweitern Sie die Shell so, dass das `INT`-Signal (erzeugt z.B. durch Drücken von `CTRL-C`) von Ihrer Shell (jedoch nicht von den erzeugten Kindprozessen!) ignoriert wird (`sigaction(2)`). Sie sollten nun laufende Kindprozesse durch Eingabe von `CTRL-C` abbrechen können, ohne jedoch dabei Ihre Shell zu beenden.

Teilaufgabe b (8 Punkte)

Erweitern Sie die in der vorherigen Teilaufgabe entstandene Shell nun um folgende Funktionalität:

- **Unterstützung von Hintergrundprozessen**

  Erweitern Sie `mish` so, dass Kommandozeilen die mit dem Token „&“ enden, als Hintergrundprozess ausgeführt werden. In diesem Fall soll die Shell nicht auf die Beendigung des Prozesses warten, sondern stattdessen die Prozess-ID des Hintergrundprozesses ausgegeben und sofort einen neuen Prompt zur Entgegennahme weiterer Kommandos anzeigen.

  Die Ausführung eines Hintergrundprozesses könnte in der `mish` demnach wie folgt aussehen:

  ```
  mish> sleep 10 &
  Started [2110]
  mish> ...
  ...
  Exit status [2110] = 0
  ```

- **Erweiterte Behandlung von Zombieprozessen**

  Um sowohl Vorder- als auch Hintergrundprozesse zu unterstützen muss die Behandlung der Zombieprozesse überarbeitet werden. Die Behandlung soll weiterhin vor Ausgabe des Prompts stattfinden, nun aber wie folgt ablaufen:

  – Wurde ein Kindprozess beendet, so wird der `mish` das `SIGCHLD`-Signal zugestellt, dessen Auftreten im zugehörigen Handler entsprechend zu erfassen ist. Registrieren Sie hierfür mit `sigaction(2)` eine eigene Signalbehandlungsfunktion für `SIGCHLD`.

  – Ist derzeit kein Vordergrundprozess aktiv, dann muss vor Ausgabe des Prompt lediglich geprüft werden, ob seit dem letzten Einsammeln der Zombieprozesse das `SIGCHLD`-Signal aufgetreten ist. Ist dies der Fall, dann sollen alle angefallenen Zombieprozesse mit `waitpid(2)` eingesammelt werden und ihr Exitstatus entsprechend ausgegeben werden.

  – Ist ein Vordergrundprozess aktiv, muss vor Ausgabe des Prompt gewartet werden, bis sich der Vordergrundprozess beendet hat. Hierfür kann es (ggf. mehrmals) notwendig sein, mittels `sigsuspend(2)` auf das Auftreten von `SIGCHLD` zu warten, ehe der Vordergrundprozess beendet wurde und alle Zombieprozesse mit `waitpid(2)` eingesammelt wurden. Auch hier soll (wie bisher) für jeden eingesammelten Kindprozess der Exitstatus ausgegeben werden.

  – Verwenden Sie lediglich nicht-blockierende Aufrufe von `waitpid(2)` und verzichten Sie auf den Einsatz von `wait(2)`. Achten Sie auf eine korrekte Synchronisation mit Hilfe von `sigprocmask(2)` und `sigsuspend(2)`.

Hinweise:

- Ihr Programm muss mit dem folgendem Aufruf übersetzen:
  `gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o mish mish.c`
  Diese Konfiguration wird zur Bewertung herangezogen.
- Sie können vereinfachend davon ausgehen, dass die Länge einer Kommandozeile maximal 1023 Zeichen beträgt. Alle anderen Fälle dürfen mit einer Fehlermeldung behandelt werden.
- Das Programm `/proj/i4spic/pub/aufgabe8/spic-wait` eignet sich zum Testen der Reaktion auf Signale. Das Programm gibt nach dem Start seine Prozess-ID aus, sodass Sie ihm einfach mit dem Kommando `kill(1)` ein beliebiges Signal zustellen können.
- Sie können die Exitstatusausgabe testen, indem Sie das Kommando `/proj/i4spic/pub/aufgabe8/spic-exit` mit dem entsprechenden Status als Parameter aufrufen.

Abgabezeitpunkt

T01	08.07.2018	18:00:00
T02	08.07.2018	18:00:00
T03	08.07.2018	18:00:00
T04	09.07.2018	18:00:00
T05	03.07.2018	18:00:00
T06	03.07.2018	18:00:00
T07	04.07.2018	18:00:00
T08	04.07.2018	18:00:00
T09	04.07.2018	18:00:00