SPiC-Aufgabe #8: mish

(22 Punkte, in Zweier-Gruppen)

Entwerfen und programmieren Sie eine Shell `mish` (**mi**ni **sh**ell), die Programme (im Weiteren als Kommandos bezeichnet) ausführen kann. Verwenden Sie für Ihre Implementierung den Dateinamen `mish.c`. Die Aufgabe ist in drei Teilaufgaben unterteilt und es wird Ihnen eine Vorlage zur Verfügung gestellt.

Hinweis: Sowohl Teilaufgabe b), als auch Teilaufgabe c) erfordert die Verwendung von Signalen. Die Funktionsweise von Signalen unter Linux wird in der zweiten Tafelübung zur `mish` genauer erklärt. Wir empfehlen deswegen in der ersten Aufgabenwoche Teilaufgabe a) zu bearbeiten und erst nach der zweiten Tafelübung, in den verbleibenden zwei Aufgabenwochen, die Bearbeitung der Teilaufgaben b) und c).

Teilaufgabe a): Grundfunktionalität

In dieser Teilaufgabe soll die Grundfunktionalität zur Interaktion mit der `mish` implementiert werden.

- 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 `mish` soll auf das Terminieren des Prozesses warten (`waitpid(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. Außerdem soll die `PID` des Prozesses ausgegeben werden:

  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
     ```

     Sie können die Ausgabe des Exitstatus mit den Programmen `spic-wait` und `spic-exit` testen.

- Nach der Ausgabe des Exitstatus soll die `mish` wieder eine neue Eingabe entgegennehmen. Die `mish` soll terminieren, wenn es beim Lesen vom Standardeingabekanal ein End-of-File (`CTRL-D`) erhält.

Teilaufgabe b): Signale

In dieser Teilaufgabe soll die `mish` um die Behandlung der Signale `SIGINT` und `SIGCHLD` erweitert werden.

- **Ignorieren des Signals** `SIGINT`

  Wenn Sie in einem Terminalfenster z.B. durch Drücken von `CTRL-C` ein Signal auslösen (in diesem Fall `SIGINT`), 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 `mish` so, dass das Signal `SIGINT` von der `mish` (jedoch nicht von den erzeugten Kindprozessen!) ignoriert wird (`sigaction(2)`). Sie können nun laufende Kindprozesse durch Eingabe von `CTRL-C` abbrechen, ohne jedoch dabei die `mish` zu beenden.

- **Behandlung des Signals `SIGCHLD`**

  In Vorbereitung auf Teilaufgabe c) sollen nur noch nicht-blockierende Aufrufe von `waitpid(2)` verwendet werden und völlig auf den Einsatz von `wait(2)` verzichtet werden. Wurde ein Kindprozess beendet, so wird der `mish` das Signal `SIGCHLD` zugestellt. Um auf die Beendigung des Kindprozesses zu warten, soll mit `sigsuspend(2)` auf ein Vorkommen von `SIGCHLD` gewartet werden.

  - Registrieren Sie mittels `sigaction(2)` eine eigene Signalbehandlungsfunktion für `SIGCHLD`. Ein Auftreten von `SIGCHLD` soll im zugehörigen Handler in einer Eventvariablen erfasst werden.
  - Vor Ausgabe des Prompts muss gewartet werden, bis sich der Kindprozess beendet hat. Hierfür kann es (ggf. mehrmals) notwendig sein, mittels `sigsuspend(2)` auf das Auftreten von `SIGCHLD` zu warten. Achten Sie auf eine korrekte Synchronisation mit Hilfe von `sigprocmask(2)`.
  - Wenn `SIGCHLD` aufgetreten ist, soll der beendete Hintergrundprozess mit `waitpid(2)` eingesammelt werden. Auch hier soll (wie bisher) der Existstatus ausgegeben werden.

Teilaufgabe c): Hintergrundprozesse

Erweitern Sie die `mish` so, dass Kommandozeilen die mit dem Zeichen '`&`' enden, als Hintergrundprozess ausgeführt werden. In diesem Fall soll die `mish` 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. Hintergrundprozesse sollen **nicht** durch das Zustellen von `SIGINT` beendet werden.

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

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

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

- Ist derzeit kein Vordergrundprozess aktiv, dann muss vor Ausgabe des Prompts 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.
- Achten Sie darauf, dass während des Einsammelns von terminierten Kindprozessen weitere Hintergrundprozesse terminieren könnten, die ebenfalls eingesammelt werden sollen.
- Ist ein Vordergrundprozess aktiv, muss vor Ausgabe des Prompts gewartet werden, bis sich der Vordergrundprozess beendet hat. Während des Wartens soll der Exitstatus zwischenzeitlich terminierender Hintergrundprozesse ausgegeben werden (`waitpid(2)`). Hierfür kann es (ggf. mehrmals) notwendig sein, mittels `sigsuspend(2)` auf das Auftreten von `SIGCHLD` zu warten, ehe der Vordergrundprozess beendet wurde. Auch hier soll (wie bisher) für jeden eingesammelten Kindprozess der Exitstatus ausgegeben werden.

Siehe Hinweise auf der nächsten Seite.

Hinweise:

- 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 und wartet dann unendlich, so dass Sie ihm einfach mit dem Kommando `kill(1)` ein beliebiges Signal zustellen können. Alternativ können Sie `spic-wait` auch eine Signalnummer übergeben, die das Programm an sich selbst schickt.
- Das Programm `/proj/i4spic/pub/aufgabe8/spic-exit` eignet sich zum Testen der Exitstatusausgabe. Das Programm beendet sich und nutzt dem ihm übergebenen Parameter als Exitstatus.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe8/` finden Sie eine Vorlage (`mish_vorlage.c`), die Ihnen einige Funktionen und Signaturen vorgibt. Dies soll Ihnen helfen Ihr Programm zu strukturieren.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe8/` befinden sich die Dateien `mish_teil_a`, `mish_teil_b` und `mish_teil_c`, welche eine Beispielimplementierung der jeweiligen Teilaufgabe enthalten.
- Begründen Sie die Verwendung von allen `volatile` Variablen. Wenn für mehrere Variablen die selbe Begründung gilt, dürfen Sie diese gemeinsam begründen.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe8/` befindet sich die Datei `mish`, welche eine Beispielimplementierung enthält.
- Achten Sie auf aussagekräftige Fehlermeldungen, die alle auf dem Standardfehlerkanal ausgegeben werden sollen. (`fprintf(stderr,...)(3)` / `perror(3)`)
- Testen Sie Ihr Programm auch mit `valgrind`. Dies kann bei der Suche nach Fehlern helfen. *suppressed Errors* können ignoriert werden. Weitergehende Fehlermeldungen erhalten Sie, wenn Sie `valgrind` mit den Flags `--leak-check=full --show-reachable=yes` aufrufen und das zu analysierende Binary mit Debug-Symbolen bauen.
- 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.
- Funktionen der `libc`, für die wir keine Fehlerbehandlung erwarten, sind online in der Linux `libc`-Doku entsprechend markiert.
- Sie können ein `Makefile` schreiben, das eine Anleitung für den Bau des Programms mit dem Tool `make` enthält. Hierfür legen Sie eine Datei `Makefile` in Ihrem Aufgabenordner (`aufgabe8/`) an. In die erste Zeile schreiben Sie:
  `CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3`
  Der Bau erfolgt im Terminal durch `make mish` oder dem `make` Button in der SPiC-IDE.

Abgabezeitpunkt

T01	27.01.2020	18:00:00
T02	28.01.2020	18:00:00
T03	30.01.2020	18:00:00