SPiC-Aufgabe #8: `tbsh`

(22 Punkte, Montag, 12.01.2015, 10:00, keine Gruppen)

Entwerfen und programmieren Sie eine Shell `tbsh` (token bucket shell) in einer Datei `tbsh.c`, die Programme (im Weiteren als Kommandos bezeichnet) ausführen kann.

Teilaufgabe a (12 Punkte)

- Ihr Programm soll als Promptsymbol den String
  `tbsh>`
  ausgeben.
- 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 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):

     ```
     shname> ls -l
     ...
     Exitstatus [ ls -l ] = 0
     ```

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

     ```
     shname> sleep 10
     Signal [ sleep 10 ] = 2
     ```

- 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.

Hinweise zu Teilaufgabe a:

- 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 tbsh als auch einem evtl. gerade laufenden Kindprozess. Dieses Verhalten wird Tailaufgabe b behandelt.

Teilaufgabe b (10 Punkte)

Entwickeln Sie auf Basis der `tish` aus der vorherigen Aufgabe nun eine Shell `tbsh` (token bucket shell), die um folgende Funktionalität erweitert werden soll:

1. Ignorieren des INT-Signals

   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.

2. Begrenzung der maximalen Prozesszahl pro Zeiteinheit

   Die Shell soll nun einen Mechanismus implementieren, der die Zahl an Prozessen, die im Zuge einer Kommandoausführung erzeugt werden, begrenzen soll. Hierzu vergibt die Shell in bestimmten Zeitabständen (refresh interval) Token an den Benutzer. Ein Token erlaubt die Ausführung eines Prozesses und wird hierbei verbraucht. Die maximale Zahl an Tokens (`max_tokens`), die ein Benutzer ohne einen Prozess zu starten durch Warten ansparen kann, ist begrenzt. Der Mechanismus soll im Detail wie folgt funktionieren:

   - Die beiden Parameter refresh interval und max tokens können der Shell als Kommandozeilenparameter übergeben werden (`atoi(3)`). Es müssen entweder beide Werte oder keiner der Werte übergeben werden. In letzterem Falle ist die Zahl der Prozesse wie schon in der `tish` nicht begrenzt. Ansonsten steht zum Programmstart *ein* Token zur Verfügung.
   - Wird von dem Mechanismus Gebrauch gemacht, aktiviert die Shell einen Systemalarm (`alarm(2)`), der nach refresh interval Sekunden abläuft und der Shell dann ein ALRM-Signal zustellt.
   - Die von der Shell für das ALRM-Signal installierte Signalbehandlungsroutine (`sigaction(2)`) erhöht die Zahl der verfügbaren Tokens um 1, falls die maximale Zahl max tokens noch nicht erreicht ist.
   - Vor dem Start eines Prozesses wird überprüft, ob noch Token zur Verfügung stehen. Sind derzeit alle Token verbraucht, gibt die Shell eine entsprechende Fehlermeldung aus und führt das Kommando nicht aus. Ansonsten wird das Kommando gestartet und ein Token verbraucht. Achten Sie hierbei auf evtl. bestehende Nebenläufigkeitsprobleme mit Signalbehandlungen und synchronisieren Sie entsprechend (`sigprocmask(2)`).

3. Shell-Prompt

   Zuletzt soll noch das Prompt-Symbol der Shell angepasst werden. Wird vom time-quota-Mechanismus Gebrauch gemacht, so soll die Shell einen Prompt der Form `tbsh[num_tokens]>` ausgeben, wobei an Stelle von `num_tokens` die zum Zeitpunkt der Promptausgabe verfügbare Zahl von Tokens eingesetzt wird. Wird der Mechanismus nicht verwendet, so wird an Stelle der Tokenzahl `unlimited` ausgegeben.

Hinweise:

- Ihr Programm muss mit folgenden Flags warnungs- und fehlerfrei übersetzen:

  ```
  gcc -std=c99 -pedantic -Wall -Werror -D_XOPEN_SOURCE=500 -O2 -o tbsh tbsh.c
  ```

- Sie können vereinfachend davon ausgehen, dass die Länge einer Kommandozeile maximal 1023 Zeichen beträgt. Alle anderen Fälle dürfen mit eine 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.