Betriebssysteme
bs@ibr.cs.tu-bs.de
Wintersemester 2025/26

Technische Universität Braunschweig

Übungsblatt 4: Shell und Prozesse

Abgabetermin: Donnerstag, 18.12.2025 (23:59 Uhr)

Allgemeine Hinweise zu den BS Übungen

- Die Aufgaben sind alleine zu bearbeiten, allerdings ist Partnerarbeit (maximal zu zweit) möglich. Beide Partner müssen aber in ihrem Git-Repository abgeben und eine `PARTNER`-Datei mit dem Name des Partners im Repository anlegen.
- Jeder Teilnehmende muss dabei in der Lage sein, die gesamte Lösung zu erläutern.
- Zur Versionsverwaltung wird jedem Teilnehmenden ein Git-Repository zur Verfügung gestellt, welches auch zur Abgabe verwendet wird. Der Zugriff auf das Repository erfolgt über das IBR GitLab unter `https://gitlab.ibr.cs.tu-bs.de`.
- Die Abgabe ist spätestens zur Deadline durch ein `git push` zu tätigen. Es sollte sichergestellt werden, dass die Änderungen vorher auch durch ein `git commit` gesichert wurden.
- Eine Aufgabe kann beliebig oft abgegeben werden; es gilt die letzte *rechtzeitige* Abgabe (der letzte rechtzeitige Commit auf dem `main` Branch).
- Die abgegebenen Programme werden auf Ähnlichkeit mit anderen Programmen desselben Semesters und früherer Semester überprüft. Bei starken Übereinstimmungen behalten wir uns ein Ausschluss vom Übungsbetrieb und eine Meldung an den Prüfungsausschuss vor.
- Die Lösungen der Programmieraufgaben müssen im GitLab CI/CD lauffähig sein. Zusätzlich kann das System `x1.ibr.cs.tu-bs.de` zum Testen genutzt werden. Auf diesem können Sie sich per SSH mit ihrem IBR Account anmelden.
- Für die Bearbeitung kann auf einem beliebigen Linux-System gearbeitet werden. Allerdings empfehlen wir das frühzeitige Testen der Abgabe durch ein `git push` und die Inspektion der Test Pipeline im GitLab.
- Die Programmieraufgaben werden automatisch durch das bereitgestellte Testsystem per GitLab CI/CD ausgewertet. Es müssen mindestens $50\,\%$ der verfügbaren Testcase-Punkte erreicht werden.
- Die Testcase-Punkte können kontinuierlich über die Ausgabe des `run-tests` Jobs aus der GitLab CI/CD Pipeline eingesehen werden. Gewertet wird der letzte Pipeline-Status auf dem von uns zur Abgabe angelegten Abgabebranch (z.B. `abgabe1`), überprüfen Sie diesen gegebenenfalls.
- Sollten Sie $50\,\%$ der Punkte erreicht haben werden Sie ggf. für ein Code-Interview ausgewählt. Sollten Sie zum Code-Interview nicht erscheinen oder ihre Lösung nicht ausreichend erklären können wird der Zettel als „ungenügend“ bewertet.
- **KI Richtline**: Der Einsatz von KI-gestützten Tools ist bei der Bearbeitung der Hausaufgaben streng untersagt, da die eigenständige Erarbeitung der Lösungen inhärentes Lernziel ist. Es ist zu beachten, dass wir alle Abgaben als Prüfungsdokumente archivieren und es auch bei einer nachträglichen Prüfung zu einer rückwirkenden Aberkennung der Studienleistung kommen kann.

Aufgabe 1: clash

Implementieren Sie ein Programm `clash` (**C l**anguage **a**pprentice's **sh**ell), das Programme (im Weiteren als Kommandos bezeichnet) ausführt. Sofern nicht weiter angegeben, ist die Hauptfunktionalität in der Datei `clash.c` zu implementieren.

(a) Basisfunktionalität

Die `clash` liest zeilenweise Kommandos von der Standardeingabe ein. Die eingelesene Zeile wird in Kommandonamen und Argumente zerlegt, als Trennzeichen dienen Leerzeichen und Tabulatoren (**strtok(3)**). Das Kommando wird dann in einem neu erzeugten Prozess (**fork(2)**) mit korrekt übergebenen Argumenten ausgeführt (**exec(3)**).

Die `clash` wartet auf das Terminieren der Kommandoausführung (**waitpid(2)**) und gibt den Exitstatus gemeinsam mit der zugehörigen Befehlszeile auf der Standardfehlerausgabe aus. Die Ausgabe soll wie folgt aussehen:

```
/home/student/tempel: echo test
test
Exitstatus [echo test] = 0
```

Nach der Ausgabe des Exitstatus nimmt die Shell wieder eine neue Eingabe entgegen.
Die `clash` terminiert sobald das Ende der Eingabe (end of file) erreicht ist. Dies kann mittels (Ctrl-D) erzeugt werden.

(b) Prompt

Die `clash` gibt als Promptsymbol das aktuelle Verzeichnis (**getcwd(3)**) gefolgt von einem Doppelpunkt auf dem Standardfehlerausgabekanal (`stderr`) aus, bevor sie eine neue Zeile auf der Standardeingabe liest.

(c) Verzeichniswechsel

Implementieren Sie nun einen Verzeichniswechsel (**chdir(2)**). Wird als Kommando `cd` eingegeben, so soll der `clash`-Prozess sein Arbeitsverzeichnis auf den im nachfolgenden Argument angegebenen Pfad setzen.

(d) Hintergrundprozesse

Basierend auf einer funktionierenden Implementierung der vorherigen Aufgabe soll die `clash` nun um Hintergrundprozesse erweitert werden. Endet eine Kommandozeile mit dem Token „&“, so wird das Kommando in einem Hintergrundprozess ausgeführt. In diesem Fall wartet die Shell nicht auf die Beendigung des Prozesses, sondern zeigt sofort einen neuen Prompt zur Entgegennahme weiterer Kommandos an.

Jeweils vor Anzeige eines neuen Prompts sammelt die Shell alle bis zu diesem Zeitpunkt terminierten Hintergrundprozesse (Zombies) auf und gibt deren Exitstatus analog zu den Vordergrundprozessen aus. Hierzu muss **waitpid(2)** mit den passenden Optionen aufgerufen werden. Merken Sie sich hierfür beim Erzeugen eines Hintergrundprozesses dessen PID und Kommandozeile in einer verketteten Liste.
Die Implementierung dieser verketteten Liste ist im Modul `plist` vorgegeben. Die Verwendung ist im entsprechenden Header dokumentiert.

(e) Anzeige laufender Hintergrundprozesse

Implementieren Sie ein Kommando `jobs`, welches in jeweils einer Zeile PID und Kommandozeile aller aktuell laufenden Hintergrundprozesse auf die Standardausgabe ausgibt.

Dazu ist es notwendig, das vorgegebene `plist_walklist.c`-Modul um die Funktion `walkList()` zu erweitern, die für jedes Element der verketteten Liste eine Callback-Funktion aufruft. Startet man die `walkList`-Funktion dann mit entsprechender Callback-Funktion, wird die Ausgabe der laufenden

Prozesse erzeugt. Gibt die Callback-Funktion den Wert 0 zurück, wird das nächste Listenelement behandelt. Dazu wird der `walkList`-Funktion ein sogenannter *Funktionszeiger* übergeben, also ein Zeiger der auf eine Funktion verweist. Die Funktion kann über den Funktionzeiger mit den entsprechenden Argumenten aufgerufen werden. Gibt sie einen Wert ungleich 0 zurück, wird das Durchlaufen der Liste abgebrochen.

Weitere Spezifikationen

- Das C-Programm muss dem ANSI-C11-Standard entsprechen und mit dem GNU-C-Compiler auf dem Referenzsystem kompilieren. Dazu ist der Compiler mit folgenden Parametern aufzurufen. Das mitgelieferte Makefile enthält bereits korrekte Aufrufe.
  `gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -o hello hello.c`
- Die Modulschnittstelle der *plist* ist vorgeschrieben und darf nicht verändert werden. Hilfsfunktionen dürfen nicht Teil der Schnittstelle werden ($\rightarrow$ `static`).
- Der Aufruf der Funktion **getcwd(3)** mit `NULL` als erstem Parameter (z.B. `getcwd(NULL, 0);`) ist **nicht** Teil der POSIX-Spezifikation. An dieser Stelle weichen wir einmal explizit von der POSIX-Spezifikation ab und erlauben diesen Aufruf.
- Sollte bei der Ausführung einer verwendeten Funktion (z.B. `getcwd(3)`) ein Fehler auftreten, muss der Fehler gemäß der Aufgabenstellung sinnvoll behandelt werden.
- Die Funktion **sysconf(3)**, aufgerufen mit dem Parameter `_SC_LINE_MAX`, gibt die maximale Länge einer Eingabezeile an, die Ihre `clash` verarbeiten können soll. Sollten mehr Zeichen eingeben werden, soll die gesamte (überlange) Zeile mit einer Warnung verworfen werden.
- Unterprogramme und globale Variablendefinitionen sind ausreichend zu kommentieren. Achten Sie bitte außerdem auf saubere Gliederung des Quellcodes!

Fehlerbehandlung

- Für jeden Funktionsaufruf, der fehlschlagen kann, muss eine Fehlerbehandlung geschehen.
- Es soll eine Fehlerausgabe mittels **perror(3)** getätigt werden.
- Betrifft der Fehler nur einen Kindprozess, so soll die Shell weiterhin funktionsfähig bleiben und nur der Kindprozess beendet werden.
- Kann die Aktion durch wiederholte Nutzereingabe wiederholt werden, so soll der Fehler nicht zum Programmabbruch führen.
- Andernfalls soll das Programm mit Fehlerstatus (**exit(3)**) beendet werden.

Weitere Hinweise

- In `/ibr/courses/ws2526/bs/blatt4/clash` finden Sie die Referenzimplementierung zum Vergleichen bzw. Testen.
- Hilfreiche *Manual-Pages*: **fork(2)**, **exec(3)**, **waitpid(2)**, **getcwd(3)**, **chdir(2)**, **strtok(3)**, **sysconf(3)**, **perror(3)**