Aufgabe 3: rush (12.0 Punkte)

Programmieren Sie basierend auf der vorgegebenen `clash` eine Shell, die in ihrer Funktionalität um die Umleitung von Ein- und Ausgabekanal und um Jobverwaltung erweitert ist: `rush` (**R**evamped **U**NIX **Sh**ell).

a) Makefile

Erstellen Sie von der vorgegebenen Datei `clash.c` eine Kopie namens `rush.c` und schreiben Sie ein Makefile, das daraus unter Verwendung der vorgegebenen Module `plist.o` und `shellutils.o` ein ausführbares Programm `rush` erstellt. Achten Sie darauf, alle Abhängigkeiten inklusive der Header-Dateien korrekt anzugeben! Das Makefile soll außerdem Regeln für die beiden Standard-Pseudotargets `all` und `clean` enthalten.

b) Umleiten des Standardein-/ausgabekanals

Die Shell soll das Umleiten der Standardein- und ausgabe von gestarteten Programmen erlauben (**dup2(2)**). In die Standardeingabe wird der Inhalt der Datei *inFile* umgeleitet, falls das Token *<inFile* auftritt. Die Standardausgabe wird in eine Datei *outFile* umgeleitet, falls das Token *>outFile* auftritt. Dabei wird *outFile* angelegt, falls es noch nicht existiert, und überschrieben, wenn es bereits existiert. Das Umleitzeichen < bzw. > ist immer jeweils mit dem Dateinamen verbunden.

c) Signalbehandlung

Beim Drücken der Tastenkombination Ctrl-C (*Prozess beenden*) bzw. Ctrl-Z (*Prozess stoppen*) stellt der Terminal-Treiber der Shell und allen ihren Kindprozessen ein Signal vom Typ `SIGINT` bzw. `SIGTSTP` zu. Sorgen Sie dafür, dass diese beiden Signale von der `rush` und von ihren Hintergrundprozessen ignoriert werden (**sigaction(2)**) und nur von Vordergrundprozessen behandelt werden (mit Standardverhalten).

d) Sofortiges Aufsammeln von Zombieprozessen

Die `clash` sammelt angefallene Zombieprozesse jeweils erst vor der Ausgabe eines Prompt-Symbols auf. Ändern Sie dieses Verhalten so, dass Zombieprozesse immer unmittelbar nach ihrem Entstehen aufgesammelt werden, aber die Ausgabe des Ereignisses weiterhin vor dem Prompt stattfindet. Schreiben Sie hierfür eine Signalbehandlungsfunktion für `SIGCHLD` (**sigaction(2)**) und benutzen Sie die Funktionen aus dem `plist`-Modul, um über den Zustand der Kindprozesse Buch zu führen. Beachten Sie, dass das Warten auf Vordergrundprozesse nun mittels **sigsuspend(2)** erfolgen muss.

e) Jobverwaltung

Passen Sie die Shell so an, dass sie nicht nur das Terminieren eines Kindprozesses an den Benutzer meldet, sondern auch das Stoppen und das Fortsetzen. Wird der aktuelle Vordergrundprozess gestoppt, soll die Shell wieder Benutzereingaben entgegennehmen. Implementieren Sie ferner ein Shell-Kommando `cont <pid>`, das den gestoppten Kindprozess mit der angegebenen PID fortsetzt (**kill(2)**). Wurde dieser Prozess ursprünglich im Vordergrund gestartet, so soll er erneut im Vordergrund weiterlaufen.

f) Nebenläufigkeitsprobleme

Durch den nebenläufigen Ablauf von Signalbehandlungen und dem Hauptprogramm kann es an mehreren Stellen zu sogenannten Race-Conditions kommen. Identifizieren Sie diese Stellen und lösen Sie alle potenziellen Nebenläufigkeitsprobleme in Ihrer Implementierung (**sigprocmask(2)**). Achten Sie insbesondere auch auf Fälle, in denen die Shell die Zustandsänderung eines Kindprozesses herbeiführt.

Hinweise zur Aufgabe:

- Die für die Umleitung der Standardausgabe angelegte Datei soll standardmäßig mit Lese- und Schreibrechten für den Besitzer sowie mit Leserechten für die Besitzergruppe und alle anderen Benutzer erzeugt werden (`rw-r--r--`).
- Für die vorgegebenen Hilfsmodule finden Sie eine Doxygen-Dokumentation auf der Übungswebseite.

Hinweise zur Abgabe:

Erforderliche Dateien: `Makefile, rush.c`
Bearbeitung: Zweiergruppen
Bearbeitungszeit: 10 Werktage
Abgabezeit: 17:30 Uhr