Labor Client/Server-Programmierung

Teil 1: Prozesse und Interprozesskommunikation

1. Hintergrundliteratur

- Vorlesungsskript. Insbesondere Kapitel zu Prozessen und Interprozesskommunikation.
- Linksammlung auf `jochenkoegel.de/dhbw` mit
    - Anleitung für die Einrichtung von Virtualbox
    - Grundlegende Linux-Befehle
    - Link zu „Beej's Guide to Unix Interprocess Communication“
    - Link zu Linux-Manpages
    - Programmgerüste für Aufgaben

2. Vorbereitungen im Labor

- Richten Sie die Virtualbox nach Anleitung ein
- Machen sie sich mit grundlegenden Linux-Befehlern vertraut, falls Sie Ihnen noch nicht bekannt sind (Dokument siehe Linksammlung, Abschnitte 1-6)

3. Aufgabe 1: Signale

3.1. Hintergrund

Signale stellen die einfachste Art der Interprozesskommunikation dar. Sie werden häufig zur Steuerung des Prozesszustands verwendet. Signale werden über das Betriebssystem an Prozesse gesendet. Dabei werden gegebenenfalls vorhandene Signal-Handler aufgerufen.

Über die Betriebssystem-API verwendete Systemaufrufe

- `signal` (signal-ID, handler): registrieren eine Signal-Handlers
- `kill` (PID, signal-ID): senden eines Signals

Die untenstehende Abbildung zeigt folgende Aktionen

- Prozess 42 registriert beim Betriebssystem einen Signal-Handler für das Signal 2
- Prozess 21 sendet per `kill`-Systemaufruf an das Betriebssystem das Signal 2 an Prozess 42
- Das Betriebssystem ruft den registrierten Signal-Handler bei Prozess 42 auf und übergibt als Parameter die Signalnummer.

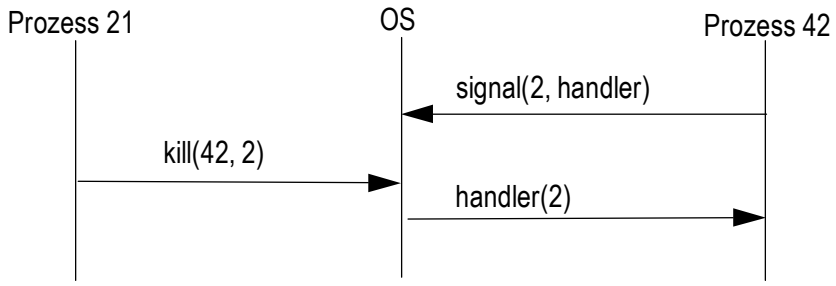

Abbildung 3.1 Registrierung von Signal-Handlern und Senden von Signalen

3.2. Vorbeitungsaufgaben

1. Können Prozesse auf Speicherbereiche von anderen Prozessen zugreifen? Wie verhält sich dies bei Threads?
2. Nennen Sie 3 Mechanismen zur Interprozesskommunikation.
3. Recherchieren Sie die Linux-Signalnummern und -namen. Nennen Sie die Nummern und Bedeutung der Signale SIGINT, SIGKILL, SIGTERM, SIGCHLD, SIGALRM

3.3. Hinweise für die Programmierung

- Versehen Sie alle Ihre Programme mit aussagekräftigen Debug-Ausgaben (`printf`)
- Orientieren Sie sich an den Beispielen aus „Beej's Guide to UNIX IPC“
- Schlagen Sie weitere benötigte Informationen in den man-pages nach

3.4. Laboraufgaben

1. Kompilieren Sie das Programmgerüst von `http://jochenkoegel.de/dhbw/code/signals.c` . Achten Sie darauf, dass die ausführbare Datei „signals“ erstellt wird und starten Sie das kompilierte Programm.
2. Machen Sie sich anhand des Programms „signals“ mit der Prozesssteuerung vertraut
   - Stoppen mit Strg-Z
   - Auflisten von Prozessen mit `ps`
   - Foreground/Background mit `fg, bg`
   - Beenden mit Strg-C, `kill, killall`
3. Erweitern Sie das Programm um einen Signal-Handler für das Signal SIGTERM
4. Nach fünfmaligem Empfangen von SIGINT soll sich das Programm beenden. Testen Sie diese Funktionalität mit Strg-C und `kill`.
5. Wie kann das Programm ohne das SIGINT-Signal beendet werden?
6. Erweitern Sie das Programm um einen Handler für SIGKILL und testen Sie entsprechend.

4. Aufgabe 2: Prozesserzeugung und -steuerung

4.1. Szenario

Nun soll das Erzeugen von Prozessen und deren Steuerung durch Systemaufrufe betrachtet werden. Ein Vaterprozess soll eine einstellbare Anzahl von Kindprozessen erzeugen und über deren PIDs und entsprechende Systemaufrufe diese Kindprozesse steuern.

Es soll folgendes Verhalten realisiert werden

- Empfängt der Vaterprozess SIGINT, so sendet er einem seiner noch aktiven Kindprozesse SIGTERM.
- Empfängt ein Kindprozess SIGTERM, beendet er sich.
- Hat der Vaterprozess alle Kindprozesse beendet, beendet sich auch der Vaterprozess.

4.2. Vorbeitungsaufgaben

1. Was geschieht beim Aufruf des Systemaufrufes `fork`?
2. Was bedeutet der Rückgabewert von `fork`?

4.3. Laboraufgaben

3. Kompilieren Sie das Programmgerüst von `http://jochenkoegel.de/dhbw/code/fork.c` und starten Sie es.
4. Erweitern Sie das Programm so, dass eine einstellbare Anzahl von Kindprozessen erzeugt werden kann und der Vaterprozess sich erst beendet, wenn alle Kindprozesse beendet sind. Auf das Ende eines Kindprozesses kann mit `wait(int *status)` gewartet werden. Der Rückgabewert von wait ist die PID des Kindprozesses, der sich beendet hat. In der übergebenen status-Variable wird der exit-status des Kindprozesses gespeichert.
5. Implementieren Sie die restlichen benötigten Signal-Handler für die oben beschriebene Funktionialität.

5. Aufgabe 3: Pipes und Shared Memory

5.1. Szenario

Es soll ein Programm erstellt werden, bei dem der Vaterprozess eine Datei einliest und die Buchstaben über Pipes an jeden seiner Kindprozesse weitergibt. Die Kindprozesse zählen jeweils das Auftreten eines Buchstabens (erstes Kind: „a“, zweites Kind: „b“, usw.) und legen das Ergebnis im shared memory ab, sodass der Vaterprozess auf das Ergebnis zugreifen kann. Die Kindprozesse werden wie in der vorigen Aufgabe durch Signale beendet.

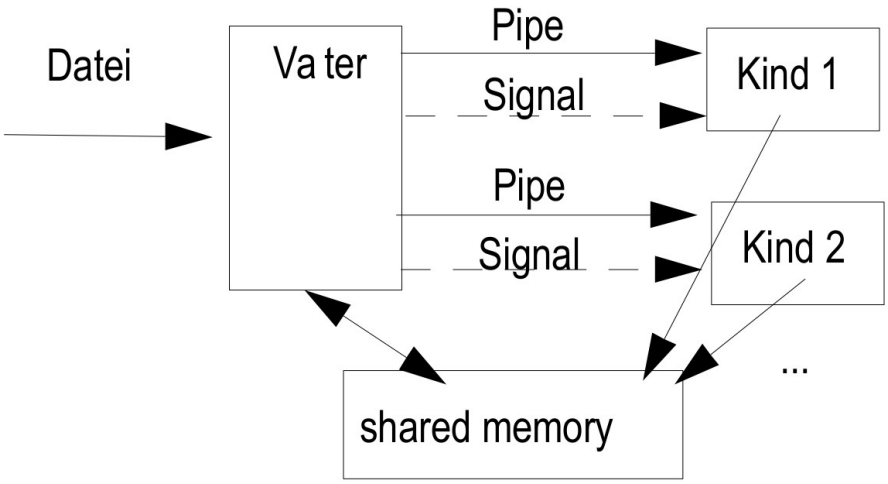

Abbildung 5.1: IPC-Mechanismen zwischen Vater- und Kindprozessen

5.2. Vorbeitungsaufgaben

1. Wie wird Shared Memory vom Betriebssystem angefordert?
2. Ist der gleichzeitige Zugriff auf Shared Memory von verschiedenen Prozessen erlaubt?

5.3. Laboraufgaben

1. Erstellen Sie das Programm und verwenden Sie wie beschrieben Pipes zwischen den Prozessen. Beim Beenden sollen die Kindprozesse das Ergebnis zunächst auf der Konsole ausgeben. Sie können als Eingabedatei den Text aus der Linksammlung verwenden.
2. Erweitern Sie das Programm um die Funktionalität mit shared memory.
3. Erweitern Sie das Programm so, dass die Prozesse, die Buchstaben zählen nicht mehr als Kindprozesse realisiert werden.
4. Die Zählprozesse sollen sich zur Laufzeit am Hauptprozess anmelden
5. Verwenden Sie geeignete IPC-Mechanismen (shared memory, messages, ...)