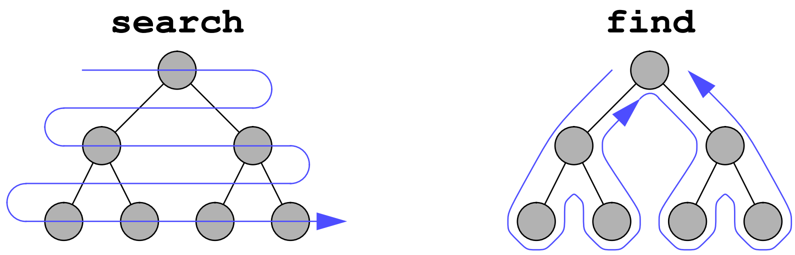
Systemprogrammierung I

Nachfolgend finden Sie die drei Aufgaben, die Sie als Zulassungsvoraussetzung für das Scheinkolloquium am Ende des Semesters lösen und abgeben müssen. Sollten Fragen zu den Aufgabenstellungen und/oder zu den Lösungsmöglichkeiten auftauchen, dann stehen wir Ihnen jederzeit zur Verfügung.

Aufgabe 1:

Schreiben Sie ein Programm **search**, mit dem sie ähnlich wie mit dem UNIX-Kommando **find**(1) Dateibäume rekursiv durchsuchen können. Der Name des Verzeichnisses, in dem die Suche startet, wird dabei in der Kommandozeile übergeben. Von diesem Verzeichnis aus wird rekursiv in Unterverzeichnisse abgestiegen, wobei Symbolic-Links nicht verfolgt werden.

Im Gegensatz zum UNIX find soll search aber den Dateibaum nicht zuerst in die Tiefe verfolgen, sondern in die Breite, d.h. es wird nicht sofort bis zu den Blättern abgestiegen, sondern schichtenweise:



Das Programm soll außerdem folgende Kommandozeilen-Optionen verstehen, durch die es in seiner Funktionalität beeinflußt wird:

a) **-print**: Die Namen aller Dateien im Startverzeichnis und allen Unterverzeichnissen werden auf den Standardausgabekanal geschrieben. Hinter dem Dateinamen steht in Klammern immer der Hinweis um welchen Typ von Datei es sich handelt, also etwa (R) für eine reguläre Datei, (D) für Directory usw. Verwenden Sie dazu dieselben Buchstaben wie das Kommando **ls(1)**.

b) **-depth** *n*: Statt eines unbegrenzten Abstiegs wird hier die Suche auf *n* Ebenen beschränkt. Sollten weitere Unterverzeichnishierarchien existieren, so werden sie durch das Programm ignoriert.

c) **-user** *uname*: Es werden nur diejenigen Dateien beachtet, die dem Benutzer *uname* gehören. Der Benutzer kann dabei entweder durch seine UID oder durch seinen Benutzernamen spezifiziert werden.

d) **-group** *gname*: Wie bei der Option -user, nur stattdessen mit den Gruppennamen, bzw. der GID.

e) **-exec** *command [options]*: Für jede gefundene reguläre Datei wird das Kommando *command* aufgerufen. Dem Kommando werden dabei die nachfolgenden Optionen (*options*), sowie der Name der gefundenen Datei als Kommandozeile übergeben. **"-exec"** darf deshalb immer nur als letzte Option für **search** angegeben werden.

f) **-name** *file*: Es wird nach der Datei *file* gesucht und ihr kompletter Pfad ausgegeben, wenn sie existiert.

Beachten Sie, daß alle Optionen gleichzeitig angegeben werden können und sich dann ergänzen. So werden etwa bei "print" dann nur diejenigen Dateien angezeigt, die allen angegebenen Kriterien genügen.

Aufgabe 2:

Entwerfen und programmieren Sie ein Programm **yash** (yet another shell), das ähnlich einer einfachen UNIX-Shell zum Starten von Programmen verwendet werden kann.

Ihre Shell soll dazu das Prompt **yash>** ausgeben und auf die Eingabe durch den Benutzer warten. Sobald dieser seine Eingabe mit **return** bestätigt, wird das angegebene Kommando mit allen Optionen durch Ihre Shell gestartet. Die Shell wartet dabei üblicherweise bis das Kommando beendet ist und fragt erst anschliessend nach dem nächsten Kommando.

Durch die Eingabe von Ctrl-Z (Stop-Signal, SIGTSTOP) von der Tastatur soll der gerade im Vordergrund laufende Prozeß in den Hintergrund geschickt werden und die Shell mit dem Promptsymbol ein neues Kommando anfordern. Alternativ dazu kann ein Kommando auch sofort als Hintergrundprozeß gestartet werden, indem in der Eingabezeile als letztes Zeichen ein **&** angegeben wird.

Weiterhin existieren die folgenden Kommandos, die von der Shell direkt interpretiert werden, d.h. keine anderen Programme starten:

a) **jobs**: Die Shell gibt eine Liste aller noch laufenden Hintergrundprozesse sortiert nach ihren Startzeiten aus, wobei die Prozesse mit Nummern versehen werden sollten.

b) **fg** *[n]*: Der Hintergrundprozeß *n* wird in den Vordergrund geholt. Falls keine Nummer spezifiziert wurde, wird der zuletzt gestartete Hintergrundprozeß ausgewählt.

c) **kill** *[-signal] jobnumber*: Dem Hintergrundprozeß mit der angegebenen Jobnummer wird ein Signal zugestellt. Falls eine Signalnummer spezifiziert wurde, wird dieses Signal zugestellt, sonst standardmäßig SIGTERM.

d) **exit**: Die Shell terminiert.

Aufgabe 3:

Programmieren Sie ein Erzeuger-Verbraucher-System, das Daten über einen Ringpuffer überträgt. Der Erzeugerprozeß liest von der Standard-Eingabe und schreibt in den Ringpuffer, der Verbraucherprozeß liest aus dem Ringpuffer und schreibt auf die Standard-Ausgabe. Im einzelnen sollen Erzeuger und Verbraucher die folgende Funktionalität haben:

Erzeuger:

Der Prozeß erzeugt ein Shared-Memory-Segment und die für die Koordinierung benötigten Semaphore. Der Key für diese Datenstrukturen soll mittels ***ftok(3)*** aus dem Homedirectory gewonnen werden.

Am Anfang des Shared-Memory-Segments soll eine Verwaltungsdatenstruktur angelegt werden, die alle Daten enthält, die für die Verwaltung der Kommunikation benötigt werden. Außerdem liegt der Kommunikationspuffer im Shared-Memory. Der Puffer soll als Ringpuffer (Erzeuger schreibt bis zum Ende des Puffers und fängt dann wieder vorne an) benutzt werden. Die Größe des Ringpuffers soll über eine Macro-Definition in einer include-Datei festgelegt werden. Testen Sie Ihr Programm mit den Pufferlängen 1024, 12, 2 und 1. Die Schreiboperationen auf dem Ringpuffer müssen über Semaphore geeignet mit den Leseoperationen des Verbrauchers koordiniert werden!

Sollte bereits ein Erzeuger-Prozeß laufen, sollen weitere Erzeuger-Starts mit einer Fehlermeldung abbrechen.

Shared-Memory-Segment und Semaphore bleiben auch nach dem Ende des Programmlaufs, in dem sie erzeugt wurden, im Betriebssystem liegen. Sie müssen deshalb dafür sorgen, daß bei Beendigung des Erzeugers (auch bei Abbruch durch die Signale SIGHUP, SIGINT, SIGQUIT und SIGTERM!) aufgeräumt wird (***shmctl(2)***, ***semctl(2)***). Ein evtl. vorhandener Verbraucherprozeß soll im Rahmen der Aufräumaktion mit dem Signal SIGPIPE beendet werden. Dabei muß allerdings gewartet werden, bis der Verbraucher alle Daten aus dem Ringpuffer gelesen und ausgegeben hat.

Mit Hilfe des Kommandos ***ipcs(1)*** können Sie sich ausgeben lassen, welche dieser Betriebsmittel angelegt wurden, mit ***ipcrm(1)*** können Sie im Ernstfall (z. B. wenn Ihr Programm mit Segmentation Fault abgebrochen ist) über eine Kommandozeile aus der Shell aufräumen.

Achten Sie beim Erzeugen von Shared-Memory und Semaphoren darauf, daß Sie ausreichend Zugriffsrechte vergeben (am besten Read für alle).

Verbraucher:

Der Verbraucherprozeß *attached* das Shared-Memory-Segment und überprüft, daß noch kein Verbraucher existiert. Existiert bereits ein Verbraucher, terminiert er mit einer Fehlermeldung. Existiert noch kein Verbraucher, schreibt er seine Prozeß-Id in die Verwaltungsstruktur (Koordinierung hier nicht vergessen, es könnten mehrere Verbraucher gleichzeitig loslaufen!) und beginnt aus dem Ringpuffer zu lesen und die Daten auf dem Standardausgabekanal auszugeben. Am Ende der Datenübertragung wird er vom Erzeuger mit dem Signal SIGPIPE terminiert. Er muß sich nicht weiter um Aufräumarbeiten kümmern.

Es soll möglich sein, den Verbraucher mit dem Signal SIGINT abzubrechen. In diesem Fall meldet sich der Verbraucher ab (er signalisiert mit Prozeß-Id "-1" in der Verwaltungsstruktur, daß kein Verbraucher mehr vorhanden ist!) und terminiert. Danach soll es möglich sein, das Verbraucher-Programm neu zu starten. Dieser Prozeß klemmt sich wieder am Puffer an und gibt dessen Inhalt weiter aus.

Beachten Sie hierbei, daß SIGINT an beliebiger Stelle eintreffen kann, auch während gerade Daten aus dem Ringpuffer gelesen werden. Die Verwaltungsdaten (z. B. Schreib/Lese-Indizes/Semaphore) dürfen dabei auf keinen Fall in einen inkonsistenten Zustand geraten!