## Abgabefrist Übungsaufgabe 1

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – eigene Übung U1 in KW 18 (29.-02.05.): Abgabe bis **Mittwoch, den 08.05.2024 23:59 Uhr (Neu ab diesem Blatt)**
- **W2** – eigene Übung U1 in KW 19 (06.-08.05.): Abgabe bis **Montag, den 13.05.2024 10:00 Uhr**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

## Allgemeine Hinweise zu den BS-Übungen

- Ab jetzt ist es **nicht** mehr möglich, Einzelabgaben in AsSESS zu tätigen. Falls ihr (statt in einer Dreiergruppe) zu zweit oder zu viert abgeben möchtet, klärt das bitte **vorher** mit eurem Übungsleiter! Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die Gruppenmitglieder sollten nach Möglichkeit gemeinsam an der gleichen Tafelübung teilnehmen. Es sind aber auch Abgaben mit Studierenden aus verschiedenen Übungsgruppen zulässig. Es gilt dabei immer die früheste Abgabefrist. Die Lösung wird jeweils komplett bewertet und den Gruppenmitgliedern gleichermaßen angerechnet.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.
- Die Aufgaben sind über AsSESS (`https://sys-sideshow.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt **ein** Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder (Matrikelnummer, Vor- und Nachname erforderlich!). Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sobald eine Abgabe korrigiert wurde, kann das Ergebnis ebenfalls im AsSESS eingesehen werden.
- (Neu ab diesem Blatt) Ihre Programme müssen von `gcc` (also kein C++) mit der Option `-Wall` kompilierbar sein (als Referenzumgebung dienen die Poolrechner, wie empfehlen zusätzlich `-Werror`). Sollten Warnungen entstehen können ihnen dafür Punkte abgezogen werden. Programm die nicht übersetzt werden können werden nicht als Lösung akzeptiert und mit maximal einem Punkt bewertet.
- (Neu ab diesem Blatt) Nutzen sie in ihrem Quelltext eine sinnvolle Einrückung für Blöcke und vermeiden sie mehrere Anweisungen in einer Zeile. In Fällen von unleserlicher Formatierung können ihnen Punkte abgezogen werden.

**Hinweis zu Übungen an Feiertagen (1.5. und 9.5.):**

Bitte besuchen sie ausnahmsweise einen der anderen Termine. Sie können auch Termine in der jeweils anderen Woche besuchen.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8, keinesfalls PDF o.Ä.

# Aufgabe 1: Prozessverwaltung und fork (10 Punkte)

Lernziel dieser Aufgabe ist die Verwendung der UNIX-Systemschnittstelle zum Erzeugen und Verwalten von Prozessen.

## 1.1 Theoriefragen: Prozesse / Shell (4 Punkte)

1. Erklärt den Unterschied zwischen **fork(2)** und **vfork(2)**? Gibt es zwischen den beiden heutzutage effektiv noch einen Unterschied?
2. Erklärt den Unterschied zwischen **execv(3)** und **execvp(3)**.
3. Betrachtet folgendes C-Programm:

```
int main() {
    int a = 10;
    pid_t pid = fork();
    if (pid == 0) {
        a += 5;
        printf("ch a: %d\n", a);
    } else {
        a -= 5;
        printf("pa a: %d\n", a);
    }

    printf("a: %d\n", a);
}
```

   Welche Ausgabe erfolgt beim Ausführen des Programms? Wie kommt die Reihenfolge der Ausgabe zustande?
   Das Einbinden der Header-Dateien wie auch die Fehlerbehandlung wurden zur besseren Übersicht weggelassen.

⇒ `antworten.txt`

## 1.2 Programmierung: Menü / Shell (6 Punkte)

In dieser Aufgabe soll ein einfaches Menü implementiert werden. Gebt eure Lösung in einer Datei ab. Um euch die Arbeit zu erleichtern, haben wir die Aufgaben in kleine Schritte unterteilt.

Achtet bei jedem System- bzw. Bibliotheksaufruf auf eine geeignete Fehlerbehandlung – sofern erforderlich. Dies schließt auch den korrekten Umgang mit falschen Nutzereingaben ein. Lest euch im Zweifel die Man-Page dazu durch oder schaut in die Übungsfolien.

⇒ `shell_menue.c`

**a) Einlesen der Standardeingabe (1 Punkt)** Implementiert zunächst ein einfaches Menü, das nummerierte Programmnamen auf der Kommandozeile ausgibt. Das Menü soll den Benutzer auffordern, eine Zahl einzugeben, um einen von zwei möglichen Menüeinträgen auszuwählen. Anschließend soll lediglich ausgegeben werden, welche Wahl getroffen wurde. Die Programmeinträge sollen *ls -a* sowie *cat fish.txt* lauten.

**Hinweis:** Zur Ausgabe soll **printf(3)** und für die Eingabe **scanf(3)** verwendet werden.

**b) Starten von Programmen (3 Punkte)** Nun sollen nicht mehr nur die Programmnamen ausgegeben werden, sondern die jeweiligen Programme auch gestartet werden. Die Ausführung des ausgewählten Programms soll in einem neuen Kindprozess geschehen. Der Elternprozess gibt *nach* der Terminierung des Kindes die PID des Kindes aus. Beachtet, dass es für beide Programmeinträge einen Programmnamen wie auch jeweils ein Argument gibt, das ihr übergeben müsst.

**Hinweis:** Zum Erzeugen der Prozesse soll **fork(2)** und **execlp(3)** verwendet werden. Achtet darauf, keine verwaisten Prozesse oder Zombies zu hinterlassen.

**Hinweis:** Die Datei *fish.txt* findet ihr im Moodle zum Herunterladen. Diese braucht ihr, damit der Menüeintrag *cat fish.txt* erfolgreich ausgeführt werden kann.

**c) Schleife (1 Punkte)** Es soll jetzt möglich sein, mehrere Programme nacheinander zu starten. *Nach* dem Ende eines Kindprozesses soll dazu erneut das Menü mit der Auswahl erscheinen. Zum Beenden des Menüs soll ein dritter „Programmeintrag“ *exit* dienen.

**d) Exit-Status abfragen und ausgeben (1 Punkt)** Sobald Teilaufgabe c) funktioniert, erweitert die Funktionalität so, dass der Exit-Status eines Kindprozesses durch den Elternprozess abgefragt wird. Dieser soll anschließend mit einer Meldung ausgegeben werden. Sollte der Kindprozess nicht ordnungsgemäß terminiert worden sein, gebt dies mit einer zusätzlichen Meldung aus.

## Zusatzaufgabe 1: Variable Argumente (2 Punkte)

Ändert das Programm so ab, dass nach Auswahl des Befehls *cat fish.txt* zusätzlich nach dem Dateinamen der Datei, die auszugeben ist, gefragt wird. Offensichtlich muss hier das Argument *fish.txt* für das Programm *cat* durch die gerade eingelesene Zeichenkette ersetzt werden. Nutzen sie **scanf(3)** und vermeiden sie unsichere Speicherzugriffe.

⇒ `shell_menue_extra.c`

**Tipps zu den Programmieraufgaben:**

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denkt daran, dass viele Systemaufrufe fehlschlagen können! Fangt diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), gebt geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beendet euer Programm danach ordnungsgemäß.
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist dazu mit folgenden Parametern aufzurufen:
  `gcc -std=c11 -Wall -o ziel datei.c`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind:
  `-Wpedantic -Werror -D_POSIX_SOURCE`
- Achtet darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt; z.B. durch Nutzung von `-Werror`.