



Betriebssysteme und Sicherheit, WS 2023/24

13. Aufgabenblatt – Unix II

Bearbeitungszeitraum: 22.01.2024 – 26.01.2024

Ziel dieses Aufgabenblatts ist es die Funktionsweisen von `fork()`, `execve()` und `wait()`, sowie die Beziehungen zwischen den Prozessen nach einer solchen Operation an praktischen Beispielen zu erarbeiten.

Aufgabe 13.1 Gegeben sei ein C-Programm, das in seiner `main()` Funktion folgenden Code ausführt:

```
int i, pid;

pid = fork();
if (pid < 0) {
    perror("Error_during_fork()");
    exit(1);
}
if (pid) {
    for (i = 0; i < 4; i++)
        puts("parent");
} else {
    for (i = 0; i < 4; i++)
        puts("child");
}

return 0;
```

- Kann das gegebene Programmstück zu verschiedenen Ergebnissen führen? Begründen Sie Ihre Antwort und nennen Sie das Ergebnis bzw. einige Ergebnisse. Dabei bewirkt `puts()` die unformatierte zeilenweise Ausgabe der als Parameter angegebenen Zeichenkette.
- Wie würde sich die Ausgabe verändern, wenn anstelle auf `stdout` die Zeichenketten in eine *vor dem* `fork()` geöffnete Datei `foo.txt` geschrieben würde?
- Wie würde sich die Ausgabe verändern, wenn anstelle auf `stdout` die Zeichenketten in eine *nach dem* `fork()` geöffnete Datei `foo.txt` geschrieben würde?

Aufgabe 13.2 Unix bietet die Systemaufrufe **`creat(2)`**, **`unlink(2)`**, **`write(2)`** und **`close(2)`** respektive zum Anlegen/Öffnen, Löschen, Schreiben und Schließen einer Datei an (Details siehe zugehörige Manpages). Was geschieht beim Ausführen der folgenden Codesequenz?

```
int fd = creat("/tmp/test", S_IRWXU);
unlink("/tmp/test");
write(fd, "Hello_world\n", 12);
close(fd);
```

Wann wird die Datei `test` gelöscht und ist damit für andere nicht mehr sichtbar? Kann `write()` erfolgreich abgeschlossen werden? Was geschieht beim Aufruf von `close()`?

Aufgabe 13.3 Welche Voraussetzungen muss ein Programm erfüllen, das in einer Kommandosequenz wie `ls | grep ps | sort` verwendet werden soll, oder anders gefragt, warum kann das Beispiel `gcc -o foo foo.c | objdump -d` so nicht funktionieren? Informieren Sie sich zuvor mit Hilfe der Manpages über die Funktionalität der auftretenden Kommandos.

Klausuraufgaben

Klausuraufgabe I

In einem Unix-ähnlichen System existieren im Verzeichnis `/tmp` die ausführbaren Dateien `Alpha` und `Omega`. Den beiden Programmen liegt folgender C-Code zugrunde:

Alpha	Omega
<pre> 1 int main() { 2 int pid; 3 pid = fork(); 4 5 if (pid < 0) { printf("F"); } 6 else { 7 if (pid == 0) { exec("/tmp/Omega"); } 8 else { unlink("/tmp/Omega"); } 9 } 10 printf("A"); 11 return 0; 12 }</pre>	<pre> 1 int main() { 2 printf("M"); 3 return 0; 4 }</pre>

- a) Nennen Sie zwei mögliche Ursachen dafür, dass die Funktion `exec` fehlschlägt.
- b) Das Programm `Alpha` wird nun ausgeführt. Der aktuelle Benutzer verfügt dabei über alle erforderlichen Rechte an den beteiligten Dateien und Verzeichnissen. Skizzieren Sie kurz *alle* möglichen Programmabläufe und geben Sie jeweils an, welche Ausgaben dabei auf der Konsole erscheinen.
HINWEIS: Beachten Sie, dass Systemaufrufe fehlschlagen können.
- c) Welche Auswirkungen hat es, wenn innerhalb des Quelltextes von `Alpha` in den Zeilen 7 und 8 „`Omega`“ durch „`Alpha`“ ersetzt wird?

Klausuraufgabe II

Betrachtet werde das folgende Programm:

```

1 int fd1, fd2, pid1, pid2;
2
3 fd1 = creat("foo.txt");
4 pid1 = fork();
5
6 if (pid1 == 0) {
7     fd2 = creat("bar.txt");
8     write(fd1, "Lorem", 5);
9
10    pid2 = fork();
11    if (pid2 == 0) {
12        write(fd2, "Ipsum", 5);
13        wait();
14    }
15    exit();
16 }
17
18 printf("X");
19 wait();
20 exit();
```

HINWEIS: Die Funktion...

`creat` legt die angegebene Datei an und öffnet sie. Sollte die Datei bereits existieren, wird ihr Inhalt beim Öffnen vollständig gelöscht.

`wait` blockiert den aufrufenden Prozess so lange bis sich ein beliebiger Kindprozess beendet. Existiert kein solcher, so kehrt die Funktion sofort erfolgreich zurück.

`exit` beendet den aufrufenden Prozess sofort.

Ein Nutzer führt das Programm nun aus. Alle auftretenden Funktionsaufrufe sind dabei erfolgreich.

- a) Was wurde auf der Konsole ausgegeben unmittelbar nachdem sich der Hauptprozess beendet hat? Skizzieren Sie kurz wie es zu dieser Ausgabe kommt. Sind auch andere Ausgaben möglich? Falls ja, geben Sie die Ursache dafür an und nennen Sie eine zweite mögliche Ausgabe!
- b) Welchen Inhalt haben die beiden Dateien `foo.txt` und `bar.txt` unmittelbar nachdem sich der Hauptprozess beendet hat? Gibt es mehrere Möglichkeiten, so geben Sie die Ursache dafür an und nennen Sie einen zweiten möglichen Dateiinhalt!
- c) Durch das Entfernen einer einzelnen Zeile des obigen Codes lässt sich erreichen, dass beim Beenden des Hauptprozesses *immer genau drei X* ausgegeben wurden. Um welche Zeile (Angabe der Zeilennummer genügt) handelt es sich? Erläutern Sie kurz wieso damit das beschriebene Ergebnis erreicht wird!

- d) Welches Ergebnis hätte ein Aufruf von `read(fd1, buf, 10)` – also das Einlesen von bis zu 10 Byte aus `fd1` in den (zuvor angelegten) Puffer `buf` – unmittelbar vor Zeile 20?

Klausuraufgabe III

Beantworten Sie folgende Fragen zu Unix-Prozessen und Shells.
fork.c

```
1 int x = 1;
2 void next() {
3     printf("%d\n", x);
4     x++;
5 }
6 int main() {
7     next();
8     pid_t pid = fork();
9     if (pid > 0) { // Elternprozess
10        wait(NULL);
11        next();
12        next();
13    } else if (pid == 0) { // Kindprozess
14        next();
15    } else { // Fehlerfall
16        next();
17        printf("Fehler\n");
18    }
19    return 0;
20 }
```

- a) **Unix-Systemaufrufe:** Welche Ausgabe druckt das obengenannte C-Programm in die Konsole? **HINWEIS:** Das Einbinden der Header-Dateien sowie ein Teil der Fehlerbehandlung wurden ausgelassen. Gehen Sie von einem fehlerfreien Ablauf aus.
- b) **Fehlerbehandlung:** Wir nehmen nun an, dass unmittelbar nach Start des Programms (noch vor dem `fork`-Aufruf) die maximal erlaubte Prozesszahl des ausführenden Nutzers auf 1 beschränkt wird. Es darf also nur ein einziger Prozess dieses Nutzers gleichzeitig existieren. Geben Sie die Ausgabe an, die in diesem Szenario vom Programm ausgegeben wird.
- c) **Unix-Shell:** Geben Sie die Auswirkungen der beiden Befehle `„ls > wc“` sowie `„ls | wc“` an und erläutern Sie anhand dessen den Unterschied zwischen den Shell-Operatoren `„>“` und `„|“`.