TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

Betriebssysteme und Sicherheit, WS 2024/25

13. Aufgabenblatt – Unix II und Dateisysteme

Besprechungszeitraum: 04.02.2025 – 07.02.2025

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;
```

(a) 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.

(b) Wie würde sich die Ausgabe verändern, wenn anstelle auf `stdout` die Zeichenketten in eine *vor dem* `fork()` geöffnete Datei `foo.txt` geschrieben würde?

(c) 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 In einem Unix-Dateisystem beträgt die Blockgröße 4 KiB. Eine Allokations-Bitmap dient der Verwaltung belegter Blöcke. In diesem Dateisystem befindet sich eine Datei `/tmp/A` mit einer Länge von 3000 Byte, welche von einem Programm zum Schreiben geöffnet wird. Das Programm hängt an die bereits vorhandenen Inhalte der Datei weitere 16 KiB Daten an und schließt die Datei danach wieder.

(a) Welche Blöcke muss das Dateisystem lesen, wenn die Datei wie oben beschrieben geöffnet wird?

(b) Welche Änderungen müssen an den Metadaten in den Dateisystemstrukturen vorgenommen werden, um die neu geschriebenen Inhalte im Dateisystem abzulegen?

Klausuraufgabe I

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 II

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 „|“.

Klausuraufgabe III

Für diese Aufgabe soll eine SSD mit einer Blockgröße von 4 KiB betrachtet werden. Auf der betrachteten SSD befindet sich das unten skizzierte Dateisystem. Das Dateisystem nutzt eine Blockgröße von 4 KiB. Die dargestellten Blöcke enthalten entweder Inodes (IB), Superblöcke (SB), Verzeichnisblöcke (VB) oder eine Allokations-Bitmap für Blöcke (BA) bzw. Inodes (IA). Weitere Blöcke (z.B. für Dateiinhalte) existieren, sind aber nicht dargestellt. Einträge in den Inodeblöcken haben dabei die Form:

`<Inode-Nummer>` → `<Liste der Datenblöcke> /<Dateigröße>`

Block-ID	0	1	2	3	4	5
Typ	SB	BA	IA	IB	VB	VB

Inhalt	free: 90
	root: `I0`
		0–9	0,1,
			2,3
				`I0` → `4 / 32`
				`I1` → `5 / 32`
				`I2` → `6,7,8,9 / 14000`
					`users` → `I1`	`anthony` → `I2`

a) Welche Blöcke des Dateisystems müssen in welcher Reihenfolge gelesen werden, wenn aus der Datei `/users/anthony` 8 KiB ab einem Offset von 4 KiB eingelesen werden sollen? Nehmen Sie dabei an, dass die Inhalte eines Blockes immer vollständig gelesen werden (also kein Block ein zweites Mal gelesen werden muss).

b) Im Folgenden werden 3000 Bytes Daten an die Datei `/users/anthony` angehangen. Auf wie viele Datenblöcke erstreckt sich die vergrößerte Datei? Wie viele Bytes werden beim Speichern der neu angehängten Daten auf die SSD geschrieben (ohne Updates der Metadaten)?