Übungsblatt 3

Abgabe bis spätestens 21.11.2016 in Stud.IP

Aufgabe 1 (4 Punkte)

Eine Benutzerin gibt in ihrer Shell das Kommando „`ls -lR / >ausgabe`“ ein. Während der Prozess läuft, drückt sie die Tastenkombination **Strg-Z**, um den Prozess anzuhalten. Sie setzt ihn anschließend mit dem Shell-Kommando `bg` im Hintergrund fort. Nach einiger Zeit ist das Kommando fertig, und das Ergebnis wurde vollständig in die Datei mit dem Namen „ausgabe“ geschrieben.

Beschreibt kurz, an welchen Stellen Traps, Interrupts oder Signale ausgelöst werden (sollten mehrere gleichartige Ereignisse unmittelbar aufeinander folgen, könnt ihr diese zusammenfassend beschreiben). Welche Zustandsänderungen werden dadurch für die hier beteiligten Prozesse bewirkt?

Aufgabe 2 (2 Punkte)

Schreibt ein einfaches C++-Programm mit dem Namen `sigserver`, das mittels `sigaction` einen Handler für das Signal `SIGUSR1` installiert. In eurem Signalhandler soll die Nummer des empfangenen Signals ausgegeben werden sowie die Prozess-ID und die User-ID des sendenden Prozesses. Nach dem Setzen des Signalhandlers soll euer Programm in einer Endlosschleife mittels `pause` auf eingehende Signale warten.

Hinweis: Benutzt in eurer Lösung die Variante `sa_sigaction` zum Setzen eures Handlers, d. h. `sa_flags` muss auf `SA_SIGINFO` gesetzt werden (siehe `man sigaction`).

Aufgabe 3 (4 Punkte)

Ein hypothetischer Scheduler verwendet den folgenden Algorithmus (an den vorgestellten allgemeinen UNIX-Scheduler angelehnt):

- Scheduling findet alle 200 ms statt (d. h. die ermittelten Prioritäten bleiben für diese Zeitspanne gültig).
- Gibt ein Prozess seine Zeitscheibe vorzeitig ab, so kommt sofort der Prozess mit der nächstbesten Priorität (ermittelt zu Beginn der betreffenden Zeitscheibe) an die Reihe.
- Jeder Prozess hat ein CPU-Konto, das folgendermaßen verwaltet wird: Bei jedem Lauf des Schedulers wird die in der soeben vergangenen Zeitscheibe für diesen Prozess verbrauchte CPU-Zeit in Prozent (der Zeitscheibe mit der Länge 200 ms) auf das Konto aufgeschlagen. Anschließend wird der aktuelle Kontostand halbiert (ein ggf. auftretender Nachkommaanteil wird abgeschnitten).

- Die Priorität eines Prozesses errechnet sich als Summe aus der (voreingestellten) Basispriorität (*nice*-Wert zwischen 0 und 100) und dem aktuellen CPU-Kontostand (nach dem Halbieren). Je kleiner der ermittelte Wert ist, desto höher ist die Priorität.

Fragen hierzu:

a) Zu welchen unangenehmen Effekten würde dieser Scheduling-Algorithmus führen, wenn der CPU-Kontostand der Prozesse nicht nach jeder Zeitscheibe halbiert würde?
b) A und B sind immer lauffähige Prozesse. A hat die Basispriorität 56, B dagegen die Basispriorität 10. Wann laufen diese Prozesse? Berechnet eine Tabelle für die ersten zwölf Zeitscheiben der Laufzeit. Lässt das Ergebnis eine Regelmäßigkeit erkennen? Wenn ja: welche?
c) A und B sind immer lauffähige Prozesse. A hat die Basispriorität 0. Ab welcher Basispriorität würde B die CPU niemals erhalten?
d) A hat die Basispriorität 0, B die Basispriorität 15. A ist immer lauffähig. B ist ein Backupprogramm, das im Hintergrund den gesamten Inhalt eines Dateisystems auf Band sichern soll. Die dazu genutzte Systemfunktion `write` blockiert nach jedem Aufruf solange, bis eine bestimmte Datenmenge erfolgreich geschrieben worden ist. Vereinfacht soll angenommen werden, dass B nach jedem Aufwachen 55 ms arbeitet und dann den nächsten `write`-Aufruf absetzt. Dieser blockiert B jeweils für 250 ms. Wie sieht das Scheduling für die ersten zwölf Zeitscheiben aus?

Weitere Aufgaben

1. Durch welche „Qualitätsmerkmale“ sollten Betriebssysteme gekennzeichnet sein? Nenne Beispiele für konkurrierende Anforderungen.
2. Worin unterscheidet sich der Kernel-Mode vom User-Mode (in Unix)? Warum wird diese Unterscheidung getroffen?
3. Was passiert in etwa bei einem Systemaufruf? (Reihenfolge der Arbeitsschritte.)
4. Was ist ein Interrupt? Nenne Beispiele für mögliche Interrupt-Quellen. Warum werden sie unterschiedlich priorisiert? Wie wird ein Interrupt in etwa behandelt?
5. Was ist ein Trap? Nenne Beispiele. Inwiefern unterscheiden sich Traps von Interrupts?
6. Was ist ein Signal? Nenne Beispiele für mögliche Signalquellen. Wie kann ein Prozess auf ein Signal reagieren?
7. Beschreibe kurz einige Zustände, in denen sich ein (Unix-)Prozess befinden kann.
8. Nenne einige Randbedingungen, auf die man beim Entwurf eines Schedulers achten sollte. Wie sollten rechenintensive bzw. Ein-/Ausgabe-intensive Prozesse dabei behandelt werden?
9. Wie könnte man mit Hilfe eines Round-Robin-Schedulers Prozessprioritäten „simulieren“?
10. Warum bestehen die *Sleep-Queue* und die *Run-Queue* in Unix nicht aus jeweils einer einzigen Warteschlange? Wie sind sie stattdessen organisiert?

Diese Aufgaben müssen nicht abgegeben werden. Sie dienen als Vorbereitung auf das Fachgespräch und werden im Tutorium besprochen.

Abgabe

Bis 24:00 Uhr am 21.11.2016 digital in Stud.IP und – wenn nicht anders mit Eurem Tutor vereinbart – ausgedruckt in unser Postfach in der MZH-Ebene 6 oder im nächsten Tutorium. Es gelten die vereinbarten Scheinbedingungen (siehe Stud.IP). Bitte beachtet unsere ergänzenden Hinweise ebenda.

Packt die abgaberelevanten Dateien in eine Archivdatei. Die Dateinamen innerhalb des Archivs wie auch der Archivname selbst müssen den in den Scheinbedingungen festgelegten Namenskonventionen folgen. Abgaben, die diese Konventionen nicht einhalten, gelten als nicht erfolgt.

Eure Ansätze und der gewählte Lösungsweg müssen nachvollziehbar sein. Achtet insofern auf eine saubere Dokumentation im Quelltext. Benennt alle von euch verwendeten Quellen, auch Zusammenarbeit mit anderen Gruppen und verwendete Unterlagen aus früheren Jahrgängen.

Für Programmieraufgaben ist die Korrektheit der Lösung bzw. deren Grenzen grundsätzlich nachzuweisen. Dies geschieht neben der Dokumentation des Programmcodes durch geeignete Tests, deren Auswahl und Eignung begründet werden muss.