Aufgabenblatt 4

Tobias Petsch

Aufgabe 1

fork() erzeugt ein Kindsprozess, welcher einer PID bekommt und von dieser von seinem Elternprozess unterscheidet. Er erstellt also einen neuen Thread.

Aufgabe 2

a)

Nur der Textbereich und Data liegen tatsächlich auf dem Festplattenspeicher, der Rest wird nach Start im Arbeitsspeicher abgelegt.

b)

text	298
data	4
bss	8
dec	310
hex	136

c)

- a: bss
- b: data
- c: stack
- d: heap
- e: bss

d)

Die Ergebnisse passen zusammen, a und e werden im bss Segment gespeichert und sind jeweils 4 Byte groß. Und b wird im Data Segment abgelegt was ebenfalls übereinstimmt.

e)

Da der heap und der Stack erst während der Laufzeit gefüllt werden, bleiben diese vor der Ausführung des Programms leer.

f)

Es wurde insgesamt 1028 Bytes auf dem Heap aus zwei mallocs alloziert. Davon wurde nur ein malloc wieder freigegeben.

4.3

Der Identifikator speichert die PID und ermöglicht dem Betriebssystem Prozesse zu unterscheiden. Der Status speichert den aktuellen Zustand des Prozesses (ready, running, waiting, terminated) und sagt so dem Betriebssystem wie es mit dem Prozess umzugehen hat. Die Priorität beschreibt die Wichtigkeit des Prozesses damit das Scheduling damit arbeiten kann. Der Programmzähler enthällt die Adresse der nächsten Anweisung und ermöglicht das korrekte Fortfahren des Prozesses. Der Speicherzeiger speichert die Speicherbereiche die dem Prozess zugeordnet sind und hilft so bei der Speicherverwaltung der Prozesse. Die Kontextdaten enthalten alle CPU-Registerinhalte und werden ebenfalls benötigt damit ein Prozess korrekt fortfahren kann. E/A-Statusinformationen enthalten Informationen über geöffnete Datein, Geräte, etc. und ermöglichem dem Betriebssystem E/A Aktivitäten zu koordinieren. Die Accounting-Informationen beinhalten statistische Daten über Laufzeit, Speicherverbrauch und Startzeit.

4.4

Der Prozess startet im New Zustand wo er erstellt wird wird und dann an Not Running Admited wird. Nun ist er Ready und wird an Running Dispateched wo er ausgeführt wird. Beim malloc, puts und printf Befehl wird der Prozess geblocked und wartet auf die Beendigung der Befehle. Beim printf wechselt der Prozess nach jedem int print den Zustand zu Running und wieder zu blockiert bis alle ints abgeschlossen sind, danach beendet das Programm und der Prozess kommt zum Zustand Exit.

4.5

a)

Threads verbrauchen im Vergleich zu Prozessen weniger Ressourcen und sind sehr schnell im Kontextwechsel. Da sie über einen shared Memory Bereich verfügen können Daten ebenfalls schneller ausgetauscht werden. Threads sind jedoch nicht isoliert und sie müssen sich den Speicher vom Prozess teilen. Das führt ebenfalls zu komplexerer Synchronisation und kann zu Deadlocks oder Race Conditions führen.

b)

User-level Threads:

Vorteile:

Schneller Kontextwechsel, da kein Wechsel in den Kernel-Modus erforderlich ist. Plattformunabhängig, da sie vollständig in der Benutzerbibliothek implementiert sind. Benutzerdefinierte Thread-Scheduling-Strategien können implementiert werden.

Nachteile:

Keine echte Parallelität auf Multiprozessorsystemen, da der Kernel nur den Prozess und nicht die Threads kennt. Blockiert ein Thread, blockiert der gesamte Prozess, da der Kernel keine Kenntnis von den Threads hat.

Kernel-level Threads:

Vorteile:

Echte Parallelität auf Multiprozessorsystemen möglich, da der Kernel die Threads kennt. Blockiert ein Thread, können andere Threads desselben Prozesses weiterlaufen.

Nachteile:

Langsamerer Kontextwechsel, da ein Wechsel in den Kernel-Modus erforderlich ist. Höherer Ressourcenverbrauch, da der Kernel die Threads verwalten muss.

4.6

a)

Amdahl's Law beschreibt, wie sich die maximale Beschleunigung eines Programms durch Parallelisierung verhält. Es basiert auf der Annahme, dass ein Programm aus einem parallelen und einem seriellen Anteil besteht. Der serielle Anteil kann nicht parallelisiert werden, während der parallele Anteil auf mehrere Prozessoren verteilt werden kann.

Die Beschleunigung S wird durch die Formel beschrieben:

$$S = \frac{1}{(1 - P) + \frac{P}{N}}$$

wobei P der parallele Anteil des Programms ist, $1-P$ der serielle Anteil und N die Anzahl der Prozessoren.

Die Herleitung basiert darauf, dass der serielle Anteil unabhängig von der Anzahl der Prozessoren gleich bleibt, während der parallele Anteil durch die Anzahl der Prozessoren geteilt wird. Dadurch ergibt sich eine theoretische Obergrenze für die Beschleunigung, da der serielle Anteil die Gesamtleistung limitiert.

b)

Programm	Kerne	Beschleunigung S	Faktor
A	4	$\frac{1}{(1-0.1)+\frac{0.1}{4}}$	≈ 1.18
A	16	$\frac{1}{(1-0.1)+\frac{0.1}{16}}$	≈ 1.23
B	4	$\frac{1}{(1-0.5)+\frac{0.5}{4}}$	≈ 1.6
B	16	$\frac{1}{(1-0.5)+\frac{0.5}{16}}$	≈ 1.94
C	4	$\frac{1}{(1-0.9)+\frac{0.9}{4}}$	≈ 3.08
C	16	$\frac{1}{(1-0.9)+\frac{0.9}{16}}$	≈ 5.26