Bitte hier tackern!

SoSe 20 Systemprogrammierung

Aufgabenblatt 2

Abgabe von

Michelle Döring

Jannik Heinze

Iryna Novytska

Charlotte Fritsch

Aufgabe 2.3: Prozesse und Threads

a) Erklären Sie, was Prozesse sind und wofür sie benötigt werden. Gehen Sie dabei auf die Aussage "Prozesse sind instanziierte Programme" ein.

Prozessor und Speicher sind Ressourcen, die alle Anwendungen benötigen. Sie müssen also verteilt werden. Dafür erfolgt die Unterteilung der Anwendungen in Prozesse.
Ein Prozess ist ein dynamisches Objekt, das sequentielle Aktivitäten in einem System repräsentiert. Es ist ein virtueller Rechner, der auf die Ausführung eines bestimmten Programms spezialisiert ist. Eine Anwendung kann ohne Zuweisung auf einen Prozessor, der sie ausführt, und Speicher nicht durchgeführt werden.
[Quelle: Vorlesung 2, S.2]

Ein Prozess ist die Instanz eines Programms. Das Programm gibt die Funktionen und Arbeitsweise vor, der Prozess ist die Ausführung mit spezifischen Daten, Zeitpunkt und Ort. Demnach kann dasselbe Programm durch mehrere Prozesse mit unterschiedlichen Daten ausgeführt werden.
[Quelle: Vorlesung 2, S.3-4]

Damit erschließt sich auch die Bedeutung von der Aussage, dass Prozesse instanziierte Programme seien: ein Prozess ist eine als Instanz erstellte Kopie des zugehörigen Programms, ergänzt um Daten und Ressourcenzuteilung. Ein Prozess ist demnach genau einem Programm zugeordnet; ein Programm (meist) mehr als einem Prozess.
[Quelle: Vorlesung 2, S.7]

b) Erklären Sie die Begriffe Parallelität und Nebenläufigkeit.

Nebenläufigkeit ist die logisch/scheinbar simultane Verarbeitung von Operationsströmen, d.h. es wird der Eindruck erweckt, dass die Prozesse gleichzeitig ablaufen, obwohl sie es eigentlich nicht tun. Dies wird erreicht durch verzahnte Ausführung auf einem Einprozessorsystem.

Bei Parallelität hingegen werden die Operationsströme tatsächlich simultan ausgeführt. Dafür werden mehrfache Verarbeitungselemente, d.h. Prozessoren oder andere unabhängige Architekturelemente unbedingt benötigt.
[Quelle: Vorlesung 2, S.12]

Sowohl Nebenläufigkeit als auch Parallelität setzen voraus, dass der Zugang zu gemeinsamen Ressourcen kontrolliert erfolgt. Bei Nebenläufigkeit werden mehrere Prozesse zu mindestens einem Prozessor zugeordnet; bei Parallelität zu mindestens zwei Prozessoren. Demnach ist Nebenläufigkeit ein Spezialfall der Parallelität auf n Prozessoren. [Quelle: Vorlesung 2, S.13]

c) Erläutern sie den Nutzen der Datenstruktur Process Control Block. Gehen sie dabei besonders auf Prozessumschaltungen ein..

Durch die Datenstruktur PCB werden Prozesse im Betriebssystem implementiert. Er ist also ein verwaltungstechnischer Repräsentant des Prozesses den er implementiert. Er enthält unter anderem die Prozessidentifikation, eine Identifikation des Besitzers, Zustandsvariable, ...
[Quelle: Vorlesung 2, S.16]

Die Zustandsvariable eines Prozesses beschreibt in welchem der folgenden Zustände sich der Prozess derzeit befindet: Bereit, Laufend, Blockiert, Beendet. Der Zustand beschreibt, wie sich der Prozess beim Werben um die CPU verhält, bzw ob er gerade ausgeführt wird oder nicht.
Tritt ein Interrupt auf, kommt ein blockierender Systemaufruf, ist die Zeitscheibe verbraucht o.ä., so wird ein aktuell aktiver Prozess auf dem Zustand *Laufend* in einen anderen Zustand versetzt. Es werden die Registerinhalte in dem PCB des Prozesses abgelegt, inklusive dem Ort, an dem der Prozess unterbrochen wurde, und der Prozesszustand wird aktualisiert auf *Blockiert*/*Bereit*/.... nach einem Prozesswechsel müssen gegebenenfalls Attribute anderer Prozesse angepasst werden (Priorität etc.)
Befinden ich Kapazitäten in der CPU, so wird ein bereiter Prozess in den Zustand *Laufend* versetzt. Dabei muss der Prozesszustand aktualisiert werden, der virtuelle Adressraum wird gemäß der Konfiguration in dem PCB umgeschaltet, die Registerinhalte werden aus dem PCB geladen un der Prozess wird an dessen gespeichertem Befehlszähler fortgesetzt.
[Quelle: Vorlesung 2, S.18]

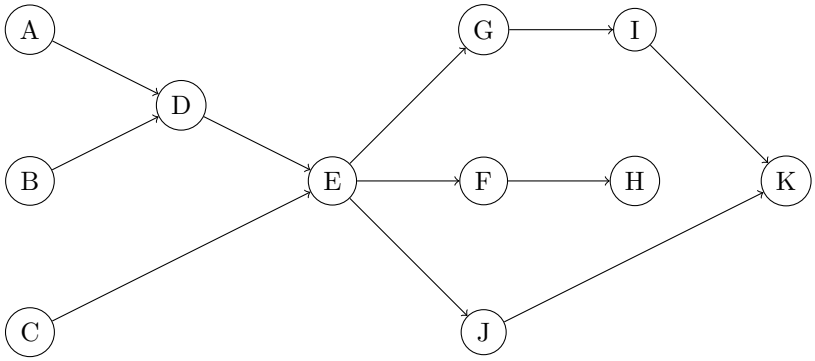
d) Was unterscheidet einen User Level Thread von einem Prozess? Worin liegen Vorteile des multithreadings?

User Level Threads sind im Benutzeradressraum realisierte Threads. Im Gegensatz zu einem Prozess, der ein ausführbares Programm darstellt, welches die Ressourcenverwaltung und das Dispatching (kurzfristiges Scheduling) vereint, will man mit einem Thread das Dispatching auslagern.
Der Prozess stellt nun die Bündelungseinheit (Task) dar und der Thread die Ausführungseinheit (Leichtgewichtsprozess). Ein Prozess ist also ein virtueller Adressraum mit Process image, dient dem Speicherschutz, hat Zugriff auf Quelltexte und Daten, etc.
Ein Thread enthält den Ausführungszustand, den Kontext, den Stack, hat Zugriff auf Prozessspeicher und Ressourcen und ist einem Prozess fest zugeordnet.

Beim multithreading Modell werden mehrere Threads einem Prozess zugeordnet und können parallel ausgeführt werden. Es erlaubt insbesondere die Ausnutzung von Multi-Prozessor-Architekturen. Blockierende Systemaufrufe haben keinen Einfluss auf übrige Threads. [Quelle: Vorlesung 2, S.25-37]
Desweiteren benötigt die Erzeugung eines Threads weniger Zeit und die Umschaltung zwischen Threads geht schneller als bei Prozessen. Auch die Terminierung ist schneller. Die Kommunikation zwischen Threads eines Prozesses ist ohne Einschaltung des Kernels möglich.
[Quelle: https://ti.tuwien.ac.at/cps/teaching/courses/osvo/bs-folien/bs02_processes.pdf]

Aufgabe 2.4: Parallelisierung II

a) Zeichnen Sie einen Prozessabhangigkeitsgraphen, der die Abhängigkeiten der einzelnen Jobs darstellt. Jede aufgerufene Funktion soll dabei einem Task bzw. Prozess/Thread entsprechen.



b)Schreiben Sie basierend auf dem Graphen ein Programm in Pseudocode mit fork/join, das die Jobs möglichst effizient abarbeitet.

Algorithm 1: fork/join Algorithmus zu 2.4

```
fork B
fork C
A
join B
D
join C
E
fork Z1
fork J
G
I
join J
K
join Z1

Z1:
  F
  H
  end
```

Aufgabe 2.5 Parallelisierung III

a) Schreiben Sie basierend auf dem Graphen zwei Programme in Pseudocode mit 1.fork/join und 2.parbegin/parend, welche die Jobs möglichst effizient abarbeiten.

1.

Algorithm 2: fork/join

```
fork Z_2
fork K
fork B
C
join B
D
E
fork F
join K
G
fork J
join 1
join F
I
join J
X

Z_2:
  A
  H
  end
```

2.

Algorithm 3: parbegin/parend

```
parbegin
    begin
        A
        H
    end
    begin
        parbegin
            K
            begin
                parbegin
                    B
                    C
                parend
                D
                E
            end
        parend
        parbegin
        F
        G
        parend
    end
parend
parbegin
I
J
parend
X
```