Jonas Kuß, Anton Jürß, Daniel Nikulin, Robin Sadeghpour

Aufgabe 2.3

a)

Prozesse representieren sequentielle Aktivitäten innerhalb eines Systems.
Sie sind dynamische Objekte. Ein Prozess ist eine Art virtueller Rechner für eine bestimmte Ausführung eines Programms.
Ein Prozess wird definiert durch den Besitz von Ressourcen, durch eine Arbeitsvorschrift und durch einen Aktivitätsträger, welcher die Verarbeitungsvorschrift ausführt.
Die Prozesse schaffen Struktur und Ordnung innerhalb eines Programms (Prozesshirarchie) und ermöglichen neben sequentieller Arbeit auch parallele Arbeit (Nebenläufigkeit und Parallelität). Somit ermöglichen Prozesse eine effizientere Nutzung von Ressourcen. Prozesse werden von einem laufendem Programm erzeugt und erhalten von diesem Eingaben, welche sie verarbeiten und liefern demzufolge Ausgaben. Sie sind somit einem Programm eindeutig zugewiesen und können somit als eine Instanz dessen aufgefasst werden.

Quellen: Vorlesungsfolien Kapitel 2, Folien 1-14

b)

Nebenläufigkeit ist eine "logisch simultane Verarbeitung von Operationsströmen" (VL Kapitel 2 Folie 12), wobei die Prozesse nicht tatsächlich simultan, bzw. parallel Ablaufen sondern verzahnt auf einem Einprozessorsystem.
Dabei werden mehrere Prozesse mindestens einem Prozessor zugeordnet.

Parallelität ist eine tatsächlich simultane Ausführung von Prozessen.
Es werden mehrere Prozesse auf mindestens zwei Prozessoren zugeordnet.
Somit ist die Parallelität eine Teilmenge der Nebenläufigkeit. Zudem sind mehrfache Verarbeitungselemente notwendig (z.B. Prozessoren).

Quellen: Vorlesungsfolien Kapitel 2, Folien 12-14

c)

Prozesse können im Betriebssystem dank der Datenstruktur des Process Control Block (PCB) implementiert werden. Der PCB ist ein "verwaltungstechnischer Repräsentant des Prozesses" (VL Kapitel 2 Folie 16). Der PCB enthält Information über den Prozess, unter anderem die Prozessnummer, oder die Zustandsvariable.
Die Zustandsvariable bezeichnet in welchen Zustand der Prozessor sich befindet (Bereit, Laufend, etc.).
Falls ein Zustandswechsel statt findet, werden jenachdem die aktuellen Registerinhalte in dem PCB abgelegt oder von dem PCB geladen, der virtuelle Adressraum innerhalb des PCB umgeschaltet und der Prozesszustand aktualisiert.

Quellen: Vorlesungsfolien Kapitel 2, Folien 16-18

d)

Ein Thread ist eine Ausführungseinheit eines Prozesses. Mehrere Threads können einem Prozess zugeteilt werden. Threads entsprechen einem Kontrollfluss von einem oder mehreren Prozessen. Ein Thread verfügt über die gleichen Betriebsmittel und befindet sich im selben virtuellen Adressraum wie die anderen Threads des Prozesses, somit können mehrere Threads sich gegenseitig beeinflussen. Bei Prozessen ist das nicht der Fall, da jeder Prozess über eigene Ressourcen verfügt. Threads werden im Allgemeinen in User-Level (UL) Threads und Kernel-Level (KL) Threads unterschieden. UL-Threads werden vollständig im Adressraum der Anwendung realisiert. Somit sind diese dem Betriebssystem völlig unbekannt. Prozesse dahingegen sind dem Betriebssystem (BS) bekannt. Aus diesem Grund dispatcht das BS mindestens einen KL Thread als Träger für die UL Threads. Prozesse interagieren mit dem BS, wobei dies bei UL Threads nicht der Fall ist. Falls ein Prozess blockiert ist, können andere Prozesse auch nicht ausgeführt werden bis zur Fehlerbehandlung des blockierten Prozesses. Bei multithreading Modellen (Verwendung von UL Threads) können Threads trotz eines blockierten Threads dennoch weiter laufen.

Vorteile von multithreading

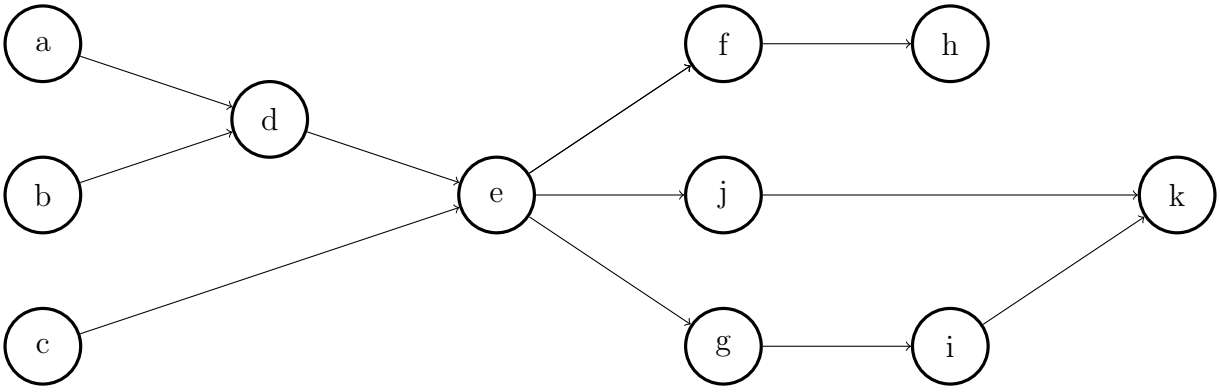
- Mehrere Threads können parallel ausgeführt werden
- Auslagerung von Aufgabenteilen auf mehrere Threads kann diese Aufgabe effizienter lösen
- Kürzere Leerläufe bei Lese- und Schreibevorgängen
- Blockierender Systemaufruf muss nicht unbedingt alle Threads blockieren (Abhängig vom multithreading Modell)
- Bessere Fehlerbehandlung
- Scheduling Algorithmus beliebig

Quellen:

- Vorlesungsfolien Kapitel 2, Folien 25-40
- tutorialspoint.com

Aufgabe 2.4

a)



b)

```
fork b
fork c
a
join b
d
join c
e
fork PartOne
fork PartTwo
j
join PartTwo
k
join PartOne
end

PartOne:
f
h
end

PartTwo:
g
i
end
```

Aufgabe 2.5

a) fork/join

```
fork Part
fork B
fork K
C
join B
D
E
fork F
join K
G
fork J
join F
Join Part
I
join J
X
end

Part:
A
H
end
```

a) parbegin/parend

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