CAuBS Einsendeaufgaben 2 – Deadline 02.12.2018 23:55 Uhr

Aufgabe 1: 3.1.3 Zwischen Benutzer und Hardware: Was wäre wenn... I+II

Was wäre, wenn es keine Betriebssysteme gäbe? Die Abbildung oben bestünde dann nur aus den Ebenen *Benutzer*, *Anwendungsprogramm* und *Hardware*.

a. Was bedeutet dies für die "normalen Nutzer" der Anwendungssoftware?
(z.B. für den Studierenden, der seine/ihre Abschlussarbeit mit einer Textverarbeitung erstellt? Oder für den Manager, der seine E-Mails abruft?)

Das würde für den Benutzer bedeuten, dass er nur Programme erwerben kann, die auf seine aktuelle Hardware angepasst sind, er müsste mit jeder neuen Hardware seine komplette Software austauschen. Der zusätzliche Aufwand in der Programmierung würde die Programme deutlich verteuern.

Für jeden Programmwechsel müsste er z.B. wie bei Konsolenspielen die Catridge wechseln und das Übertragen von Daten zwischen Anwendungen wäre ohne definierten Standard schwierig.

Sollte tatsächlich ein Wechsel zwischen Programmen möglich sein, so müsste jedes Programm wissen wo die anderen Programme liegen und wann es dahin springen soll.

b. Was bedeutet dies für die Programmierer der Anwendungssoftware?

Statt auf vordefinierte APIs zuzugreifen müssten die Programmierer für jede Anwendung alles selbst entwickeln und für jede Hardwareversion anpassen was finanziell oft jedes Budget sprengen würde.

Gerade in höheren Sprachen wie Delphi ist keinerlei Kenntnis von Prozessorarchitekturen, Gerätecontrollern Speicheraddressierung, Ausgabetreibern… notwendig, da zum Einlesen ein einfaches „read(variable)“ ausreicht und die Speicherverwaltung und Eingabegerätsteuerung „von alleine“ passiert. Somit muss der Programmierer im Grunde genommen keinerlei Zeit für das Verständnis der Computerarchitektur verwenden und kann direkt sein eigentliches Problem bearbeiten.

c. Und welche Auswirkungen hätte dies auf die Sicherheit des Systems, insbesondere wenn mehrere Programme quasi-parallel ausgeführt werden?

Da die Programme ohne Betriebssystem die Hardware selbst verwalten müssten und damit auch die Übergabe an Nachbarprogramme selbst steuern müssten, ist die Chance erheblich größer, dass ein Fehler in einem Prozess alle anderen Programme mit niederreißt.

Es wäre auch kein echtes Hindernis, mutwillig Stack und Basisregister aus der Anwendung heraus zu manipulieren, da bei bekanntem Quellcode einfach abgezählt werden kann, wo sensible Daten einer anderen Anwendung liegen, die man auslesen möchte.

Was wäre, wenn es keine Betriebssysteme und keine Anwendungsprogramme gäbe? Die Abbildung oben bestünde dann nur aus den Ebenen *Benutzer* und *Hardware*.

d. Was würden Studierende dann in Fächern wie "Grundlagen der Programmierung" lernen müssen?
Ohne Programme wäre es für die meisten Probleme wahrscheinlich einfacher das Problem auf Papier zu lösen. Somit müssten die Studenten lernen, das erste Anwendungsprogramm zu schreiben. Das Lesen und direkte Eingeben von Maschinencode ist zu aufwendig und fehleranfällig, dass es besser wäre, damit gar nicht erst anzufangen.

Aufgabe 2: 3.1.5 Betriebsmittel sind Prozessen zugeordnet: Datei nur exklusiv oder gemeinsam nutzbar?

Ob eine einzelne Datei ein *exklusiv* oder ein *gemeinsam nutzbares Betriebsmittel* ist, kommt darauf an.

a. Erläutere: Worauf kommt es an?

Im Grunde genommen kommt es nur auf die Sichtweise des Fragestellers an. Letztlich ist eine Datei immer ein gemeinsam nutzbares Betriebsmittel, was nur aus (zumeist sinnvollen) Gründen unterbunden wird wo es erforderlich ist. Da z.B. eine Protokolldatei die vom Benutzer System geschrieben wird, ständig aktualisiert wird, wäre es unschön wenn der Benutzer sie zum öffnet (Datei-> Speicher), Kaffee trinkt, mit Kollegen redet und dann aus Gewohnheit speichern und Schließen klickt und somit alles, was Benutzer System eingetragen hat, überschreibt. Tatsächlich schlecht wäre es, wenn zwei Prozesse „Gehalt einzahlen“ und „Tanken abbuchen“ gleichzeitig meinen Kontostand laden und „Tanken abbuchen“ erst als zweiter das Gehalt überspeichert, da ich dann kein Geld mehr für Kuchen habe.

Unter Windows rufen daher Programme, die exklusiven Zugriff auf eine Datei haben sollen, die API „Lockfile“ auf, wodurch die eigentlich immer gemeinsame Nutzung eingeschränkt wird. In Linux Systemen gibt es dazu ein passendes Gegenstück, mit dem ich mich aber bisher nie beschäftigt habe.

Programmdateien werden zur Laufzeit standardmäßig auch vom Betriebssystem gegen Veränderung gelockt.

Da beide Fälle aber durchaus „gewaltsam“ umgangen werden können (z.B. indem man den Sicherheitstoken des angemeldeten Administrators via Powershell so modifiziert, dass jegliche Dateisystemrechte und locks keine Rolle mehr spielen), gibt es in dem Sinne nur softwareseitig definiert Dateien, die exklusive Betriebsmittel sind.

Das spielt z.B. dort eine Rolle, wo nachts ein Restore Task Teile einer (Schulungs)Umgebung auf einen definierten Stand bringt.

Daher sollte man bei der Programmierung auch immer genauestens darauf achten, wo es sinnvoll ist, einen lock zu setzen.

b. Finde ein Beispiel, bei dem eine Datei nur exklusiv nutzbar ist.

Eine Datei sollte immer dann als nur exklusiv nutzbar gesetzt werden, wenn eine gleichzeitige Bearbeitung mit gegenseitigem Überspeichern negative Folgen hätte (Datenüberspeicherung//Wegfall von Daten).

Das bekannteste Beispiel sind vermutlich Word-Dateien, die der Kollege morgens geöffnet hat und den ganzen Tag geöffnet lässt oder ein geöffnetes Programmfenster. Lokale Anwendungen, die sich auf Access-Datenbankdateien beziehen, sperren auch diese Datenbank sodass ein weiteres Programmodul auf dem selben Rechner keine gleichzeitigen Änderungen vornehmen kann.

c. Finde ein weiteres Beispiel, bei dem eine Datei von mehreren Prozessen gemeinsam nutzbar ist.

Grundsätzlich ist es kein Problem eine Datei beliebig oft zu lesen.

Es ist ohne weiteres möglich z.B. eine Protokolldatei alle 10 Sekunden zu versenden, währenddessen die Einträge sekündlich zu lesen und in der Anwendung anzuzeigen, zusätzlich eine Sicherung zu schreiben und nebenher von einem Statistikprogramm die Inhalte der Protokolldatei auszuwerten.

Aufgabe 3: 3.2.4.1 Fork: Ein anderes Programm mittels fork starten

Schaust du dir die Unterschiede zwischen *fork* und *CreateProcess* genauer an, so fällt auf, dass bei *CreateProcess* eine andere Anwendung gestartet wird, während *fork* lediglich eine Kopie eines bereits existierenden Prozesses erzeugt. Wie kann mittels *fork* ein anderes Programm gestartet werden? Erläutere mit deinen eigenen Worten die Vorgehensweise dabei!
Achilles 2006 gibt Hinweise dazu in Kapitel 3.2.1 und 3.2.2.

Im Programmcode des Kinds wird mittels im Kind vorhandener exec Anweisung, die den Aufruf des anderen Programms enthält, das Programm in den Adressraum des Kindes geladen und dieses dadurch überschrieben.

Aufgabe 4: 3.2.9 Threads: Geschwindigkeitsvorteil

Warum benötigt ein Thread-Kontextwechsel weniger Zeit als ein Prozess-Kontextwechsel? Erläutere!

Ein Thread ist dem Prozess untergeordnet und nutzt alle seine Betriebsmittel. Dadurch müssen bei einem Threadwechsel weniger neue Daten nachgeladen werden.

Aufgabe 5: 3.2.9 Threads: Vor- und Nachteil des Betriebsmittelzugriffs

Warum ist die Zugriffsmöglichkeit auf alle Betriebsmittel eines Prozesses durch die verschiedenen Threads dieses Prozesses gleichzeitig ein Vor- und ein Nachteil? Erläutere!

Von Vorteil ist, dass weniger Ressourcen verwendet werden und Daten gemeinsam bearbeitet werden können.
Nachteilig ist, dass bei ungenauer Programmierung ein Thread dem anderen die Daten ändern kann und der dann von falschen Voraussetzungen bei der weiteren Verarbeitung ausgeht.

Aufgabe 6: 3.2.9.1 Java-Beispiel mit Threads: Starte die Threads!

Übertrage den Java-Quellcode in eine Entwicklungsebene deiner Wahl, kompiliere und starte ihn.

a. Was passiert?

   Die Ausgaben der beiden Threads werden durcheinandergewürfelt ausgegeben.

b. Welche Veränderung kannst du bei der Ausgabe erkennen, wenn du das Programm mehrmals startest?

   Die Zuteilung der Arbeitszeit zu den Threads wirkt zufällig. Je weniger Last auf dem System ist desto mehr zusammenhängende Daten werden ausgegeben.

Falls du keine Veränderung erkennst: Sorge auf deinem Rechner mal für etwas mehr Arbeitslast: Starte viele Programme, lass ein Video laufen und gleichzeitig Musik abspielen. Und starte immer wieder das Programm.

Aufgabe 7: 3.2.10.1 Scheduling-Ziele: Scheduling-Ziele

Mandl 2013 erläutert in Kapitel 5.1 (Scheduling-Kriterien) verschiedene Scheduling-Ziele.

a. Welche Ziele sind das?

Fairness, Effizienz, minimierte Antwortzeit, minimierte Wartezeit, optimierter Durchsatz, minimierte Durchlaufzeit, Vorhersehbarkeit

b. Erläutere jedes Ziel kurz.

Fairness soll dafür sorgen, dass jeder Prozess eine gewisse Rechenzeit erhält.

Effizienz soll dafür sorgen, dass es möglichst wenig Leerlaufzeit gibt.

Minimierte Antwortzeit soll dafür sorgen, dass die Verarbeitung möglichst Echtzeitnah wirkt

Minimierte Wartezeit soll dafür sorgen, dass ein Prozess möglichst bald Rechenzeit zugeteilt bekommt.

Optimierter Durchsatz soll dafür sorgen, dass möglichst viele Daten auf einmal verarbeitet werden können.

Minimierte Durchlaufzeit soll dafür sorgen, dass die Aufgabe möglichst schnell abgearbeitet wird.

Die Vorhersehbarkeit soll dafür sorgen, dass die benötigte Zeit der Ausführung planbar ist.

c. Wie steht es mit der gleichzeitigen Erfüllung aller Ziele?

Es muss entschieden werden, welches Ziel die höchste Priorität hat oder ein Kompromiss aus allem gefunden werden.

- *Der Durchsatz, die Antwortzeit, die Wartezeit und die Durchlaufzeit sind nur dann vollständig optimiert wenn der Prozess/Thread sobald er erstellt wurde vollständig am Stück ausgeführt wird und erst am Ende wieder andere Aufgaben erfüllt werden. Das widerspricht allerdings der Fairness, der Effizienz (Leerlaufzeit während des Wartens auf E/A Geräte) und der Vorhersehbarkeit für alle anderen Aufgaben.*

Aufgabe 8: 3.2.10.2.1 First Come First Serve: FCFS auf deinem Rechner

Du ahnst bestimmt schon, dass das Betriebssystem deines Computers nicht nach der FCFS-Scheduling-Strategie arbeitet, oder? Welche Auswirkungen wären zu befürchten, wenn der Scheduler deines Betriebssystems plötzlich auf FCFS umstellen würde?

Es würde dazu führen, dass das System oft nicht reagiert.
Es wäre z.B. nicht möglich in einem Musikprogramm wie Cubase einen Mixdown von vielen Spuren zu starten und während der Verarbeitung im Internet zu surfen, da der Browser erst nach Abschluss der ersten Aufgabe Prozessorzeit erhalten würde.

Aufgabe 9: 3.2.10.2.2 Shortest Job First: SJF auf deinem Rechner

Was denkst du, wie sich SJF auf deinem PC oder Laptop mit grafischer Oberfläche auswirkt? Nenne mindestens zwei Auswirkungen! Sind diese Auswirkungen positiv oder negativ?

Es hätte auf den Rechner die gegenteilige Auswirkung wie beim FCFS, da ein sehr langer Prozess bei entsprechender Grundlast kurzer Prozesse unter Umständen niemals zur Ausführung kommen würde.(Negativ)
Wenn es sehr viele kurze Jobs gibt, kann der PC des öfteren ruckeln, da die Wiedergabe eines Videos auf einer Webseite unter umständen aufwendiger ist als Hintergrundaufgaben. (Negativ)

Aufgabe 10: 3.2.10.2.4 Round Robin: Zu kurz oder zu lang

Es ist denkbar, dass das Betriebssystem deines PCs oder Laptops nach dem RR-Verfahren arbeitet. Was ist zu befürchten,

a. wenn das Quantum zu kurz gewählt ist?

Wenn die Prozesse sehr häufig wechseln, wird viel Arbeitszeit an den Prozess/Threadwechsel verschwendet sodass der PC mehr mit der Eigenorganisation als mit der Abarbeitung beschäftigt ist.

b. wenn das Quantum zu lang gewählt ist?

Das Betriebssystem scheint des Öfteren zu hängen, da die Echtzeiterfahrung dadurch beeinträchtigt wird, dass zu lange auf die Abarbeitung eines Prozesses gewartet werden muss.

Aufgabe 11: 3.2.10.4 Vergleichskriterien: Scheduling-Kriterien

Mandl 2013 erläutert in Kapitel 5.3 (Vergleich ausgewählter Scheduling-Verfahren) verschiedene Scheduling-Kriterien.

a. Welche Kriterien sind das?

Durchlaufzeit, Wartezeit, Bedienzeit, Antwortzeit, Durchsatz, CPU-Auslastung

b. Erläutere jedes Kriterium kurz.

- *Die Durchlaufzeit besteht aus den Arbeits- und den Wartezeiten – also der gesamten Zeit, die ein Prozess bis zu seinem Abschluss benötigt.*
- *Die Wartezeit ist die Zeit, die ein Prozess auf seine Ausführung warten muss.*
- *Die Bedienzeit ist die Zeit, für die ein Prozess die CPU verwenden kann.*
- *Die Antwortzeit ist die Zeit, die ein Anwender auf die Auftragsbearbeitung warten muss.*
- *Beim Durchsatz handelt es sich um die Anzahl an Prozessen, die das System in einer bestimmten Zeit bearbeiten kann.*
- *Bei der CPU-Auslastung handelt es sich um die Auslastung in Prozent während der Bearbeitung von Prozessen.*