Institut für Informatik
Arbeitsgruppe Software Engineering
Barry Linnert

Betriebssysteme WiSe 2016/17· Übungsblatt 5
Bearbeitungszeit: 05.01.2017 – 19.01.2017 um 09:59 Uhr

Bereiten Sie Ihre **Lösungen** grundsätzlich so vor, dass Sie diese in der Übung Ihren Kommilitonen in geeigneter Form **zeigen** und **diskutieren** können. Geben Sie bitte stets Ihre verwendeten **Quellen** an.

Die Abgabe erfolgt elektronisch über das KVV.

Aufgabe 5-1 (Gerätezugriff) – 5 Punkte

Erläutern Sie die Begriffe:

- memory mapped I/O,
- DMA (direct memory access) und polling.

Gegeben sei nun ein Prozessor mit einer 200MHz Taktfrequenz. Eine Polling-Operation benötige 400 Taktzyklen. Berechnen Sie die prozentuale CPU-Auslastung durch das Polling für die folgenden Geräte. Bei 2 und 3 ist dabei die Abfragerate so zu wählen, dass keine Daten verloren gehen.

1. Maus mit einer Abfragerate von 30/sec,
2. Diskettenlaufwerk: Datentransfer in 16-bit-Wörtern mit einer Datenrate von 50KB/sec,
3. Plattengerät, das die Daten in 32-bit-Wörtern mit einer Rate von 2MB/sec transportiert.

(Hinweis: Wie viele Taktzyklen gibt es pro Sekunde? Wie viele Taktzyklen werden für eine Abtastrate von 30/sec benötigt? Was für einer Abtastrate entsprechen 50KB/sec bei 16-bit-Wörtern ...)

Für das Plattengerät soll jetzt DMA eingesetzt werden. Wir nehmen an, dass für das Initialisieren des DMA 4000 Takte, für eine Interrupt-Behandlung 2000 Takte benötigt werden und dass per DMA 4KB transportiert werden. Das Plattengerät sei ununterbrochen aktiv. Zugriffskonflikte am Bus zwischen Prozessor und DMA-Steuereinheit werden ignoriert.

Wie hoch ist nun die prozentuale Belastung des Prozessors? (Hinweise: Wie viele DMA-Transfers pro Sekunde sind bei 2MB/sec und 4KB Transfergröße notwendig?)

Aufgabe 5-2 (User-/Kernel-Interface) – 15 Punkte

In dieser Aufgabe soll Ihr Betriebssystem um eine einfache Koordination einerseits und andererseits um eine effizientere Ressourcen-Nutzung erweitert werden.

Definieren Sie hierzu eine ordentliche Schnittstelle zwischen Anwendungen und Betriebssystem und führen Sie blockierende Systemrufe ein, damit wartende Threads nicht unnötig die CPU blockieren.

1. Legen Sie eine Aufrufkonvention für Systemrufe mittels SWI fest.
2. Definieren Sie Systemrufe für die Aufgaben „Zeichen ausgeben“, „Zeichen einlesen“, „Thread beenden“, „Thread erzeugen“ und „Thread für bestimmte Zeitspanne verzögern“.
3. Implementieren Sie die Systemrufe auf Seite des Kernels. Dabei sollen „Zeichen einlesen“ und „Thread verzögern“ blockierend arbeiten. Das heißt, der Thread wird so lange nicht mehr durch den Scheduler erfasst, bis ein Zeichen da ist bzw. die vorgegebene Zeitspanne vorbei ist. Überlegen Sie sich, ob ein aufwachender oder neuer Thread dem gerade laufenden Thread vorgezogen werden sollte oder nicht.
4. Implementieren Sie eine „Bibliothek“, die für Anwendungen eine leichter zu benutzende Schnittstelle zur Verfügung stellt, als direkt SWIs aufzurufen. Idealerweise sind der Code für Anwendungen/Anwendungsbibliotheken und der Code für das Betriebssystem/Betriebssystembibliotheken zum Schluss vollständig unabhängig voneinander.

Durch die SWI-Schnittstelle wird zum einen eine logische Trennung von Anwendungen und Betriebssystem erzeugt. Zum anderen wird – da nur einen Kern vorhanden ist – automatisch ein gegenseitiger Ausschluss sämtlicher Kernel-Funktionen realisiert (sofern Sie in privilegierten Modus [Modi] die Interrupts nicht demaskiert, siehe Zusatzaufgabe).

Die Demonstration ist ähnlich wie in der letzten Aufgabe zu implementieren, diesmal jedoch mit eigenem Thread:

5. Schreiben Sie eine Anwendung, die stets auf Zeichen wartet. Wann immer ein Zeichen empfangen wird, erzeugt diese Anwendung einen neuen (Anwendungs-)Thread und gibt diesem das empfangende Zeichen als Parameter mit.
6. Der neu erzeugte Thread soll das Zeichen wiederholt (aber nicht endlos) mit kleinen Pausen ausgeben und sich anschließend beenden. Handelt es sich bei dem Zeichen um einen Großbuchstaben wird zur Erzeugung der Pause aktiv gewartet (wie bisher), bei anderen Zeichen wird die Pause durch den neu eingeführten Systemruf erzeugt.

Bei den daraus resultierenden Ausgaben ist insbesondere das zeitliche Verhalten spannend. Die folgenden Beispiele gehen davon aus, dass die Rechenzeit beim aktiven Warten genauso lang ist, wie die Dauer der Verzögerung via Systemruf. (Und beides ist ein Vielfaches der Zeitscheibenlänge.)

2x aktives Warten:	ABABABABABABABABABAB
1x aktiv + 1x passiv:	AbAbAbAbAbAbAbAbAbAb
2x passiv:	abababababababababab
1x aktiv + 2x passiv:	AbcAbcAbcAbcAbcAbcAbcAbcAbcAbc
2x aktiv + 1x passiv:	ABcAcBcAcBcAcBcAcBcAcBABABABAB