Übungen zu Vorlesung
Betriebssysteme I
SommerSemester 1999

AG BS
Oliver Meyer/ Jan Peleska

Serie 3

Ressourcen-Verwaltung

Algorithmus zur Verwaltung verteilter Ressourcen Im folgenden wird ein Algorithmus zur Verwaltung verteilt bearbeiteter Ressourcen nach einer Idee von Herrn Kouvaras, Daimler-Benz Aerospace beschrieben. Die Motivation für den Algorithmus ist durch Anwendungen gegeben, in denen ein Betriebsmittel von mehreren kooperierenden Prozessen gleichzeitig genutzt werden darf, jeder einzelne Prozeß aber nur für sich selber entscheiden kann, wie lange er die Ressource noch bearbeitet. Eine Freigabe der Ressource und Verwendung durch eine neue Gruppe von Prozessen darf erst dann erfolgen, wenn sie auch vom letzten der kooperierenden Prozesse nicht mehr benötigt wird.

Die Betriebsmittel werden zentral von einem Ressource Manager $RESMGR$ verwaltet. Die Anzahl der Betriebsmittel ist begrenzt durch $maxRes > 0$. $RESMGR$ verwaltet eine Tabelle mit Einträgen $(resId, retValue)$, wobei der erste Eintrag ein Betriebsmittel eindeutig kennzeichnet ($resId \geq 0$) und der zweite eine "Rückgabezahl" darstellt, aus welcher der Ressource Manager feststellen kann, ob das Betriebsmittel frei ($retValue = maxRet$) oder belegt ($0 \leq retValue < maxRet$) ist.

Jeder Prozeß darf bei $RESMGR$ eine neue Ressource anfordern. Bei erfolgreicher Anforderung (es gibt noch freie Ressourcen) erhält er von $RESMGR$ ein Paar ($resId \geq 0, retValue = maxRet$) als Antwort, andernfalls eine Kennung ($resId = -1, retValue = -1$) als Zeichen, daß die Anforderung erfolglos war. Im Erfolgsfall setzt $RESMGR$ die der Ressource zugeordnete Rückgabezahl in seiner Tabelle auf $retValue := 0$.

Ein Prozeß P darf "legal" auf eine Ressource zugreifen, sobald ihm deren Kennung $resId$ zusammen mit einer Rückgabezahl $retValue$ entweder direkt von $RESMGR$ oder von einem anderen Prozeß, dem die Ressource bereits bekannt war, übergeben wurde. Der Wert von $retValue$ kann sich im Laufe der Bearbeitungszeit ändern, sobald ein Prozeß P nämlich die Ressource k anderen Prozessen zur weiteren Nutzung übergeben möchte. In diesem Fall sendet P an diese jeweils das Paar $(resId, retValue/(k+1))$ und arbeitet auch selbst mit $retValue := retValue/(k+1)$ weiter. Es kann auch vorkommen, daß P von einem anderen Prozeß eine Ressource erhält, die ihm bereits bekannt ist. In diesem Fall addiert P die mit der Sendung erhaltenen Rückgabezahl zur eigenen bisher für das Betriebsmittel lokal vorliegenden Zahl. Nach Gebrauch sendet er an $RESMGR$ das Paar $(resId, retValue)$ mit aktueller Rückgabezahl zurück und zeigt damit an, daß für ihn das Betriebsmittel nicht mehr erforderlich ist. Der Ressource Manager $RESMGR$ addiert bei jeder Rückgabe die zugehörige Rückgabezahl zum aktuellen Wert in der Tabelle. Sobald die in der Tabelle gespeicherte Rückgabezahl wieder den Wert $retValue = maxRet$ erreicht hat, ist das Betriebsmittel frei und kann erneut an einen Anforderer vergeben werden.

Damit die Rückgabezahl in jedem Fall mit Integer-Arithmetik manipuliert werden kann, wird festgelegt, daß jeder Prozeß eine Ressource $(resId, retValue)$ nur dann an k Folgeprozesse übergeben darf, wenn $retValue/(k+1)$ ganzzahlig ist. Ist $retValue = 1$, darf er nur noch lokal mit $resId$ arbeiten.

Aufgabe 1:

Für eine kontrollierte Manipulation der Betriebsmittel kann es günstig sein, nicht nur die Vergabe, sondern auch den Zugriff auf die Ressourcen durch den Ressource Manager $RESMGR$ zu kapseln. Wie läßt sich der Algorithmus geeignet erweitern (die Erweiterung muß implementierbar sein!), damit $RESMGR$ den illegalen Zugriff von einem Prozeß auf ein Betriebsmittel erkennen kann?

Aufgabe 2:

Programmieren Sie (in C oder C++) eine Implementierung des Algorithmus nach folgenden Randbedingungen:

1. Die Ressourcen sind Datenblöcke, die String-Telegramme fester Länge aufnehmen können.

2. $RESMGR$ vergibt die Ressourcen und bietet eine Manipulationsschnittstelle WRITE/READ $resId$ an. $RESMGR$ implementiert Ihre Lösung von Aufgabe 1 zum Schutz gegen illegale Zugriffe.

3. $RESMGR$ und die Anwendungsprozesse liegen auf verschiedenen Rechnern, die Kommunikation erfolgt über Berkeley Sockets, Connection-oriented Protocol, Datagrammverkehr. Jeder Prozeß soll unter Solaris und Linux, nach Möglichkeit auch unter Windows-NT lauffähig sein.

4. Ein Anwendungsprozeß läuft interaktiv gesteuert und bietet dem Nutzer folgende Befehle an:

   - neue Ressource anfordern (ein Anwendungsprozeß kann mehr als eine Ressource gleichzeitig manipulieren!),
   - vorhandene Ressource zurückgeben,
   - vorhandene Ressource lesen (Inhalt des bei $RESMGR$ gespeicherten Telegramms),
   - vorhandene Ressource ändern (neuer Telegrammtext wird an $RESMGR$ mit WRITE-Befehl geschickt),
   - Weitergabe einer vorhandenen Ressource an einen oder mehrere andere Prozesse.

Abgabe: Bis Mo, 14. Juni 99, im Tutorium