Schriftliche Ausarbeitung der praktischen Übung in Betriebssysteme an der HS-Mannheim im SS2011

Mark Albert, Henning Rohr, Patrick Beedgen, Dennis Cohen, Maiwand Haschmi

Fakultät für Informatik
Hochschule Mannheim
Paul-Wittsack-Straße 10
68163 Mannheim

Zusammenfassung Bei vorliegendem Programm sollen verschiedene Scheduling-Algorithmen anschaulich demonstriert werden. Mit Scheduling werden Verfahren zur Aufteilung eines Prozessors unter mehreren Prozessen bezeichnet.

In den grauen Vorzeit der Informatik belegte ein Rechenauftrag für seine komplette Laufzeit die Rechenanlage. Da die Anlagen sehr teuer waren, entstand großer Bedarf an speziellen Verfahren, die die Maschine unter Aufträgen erlauben würden.

> In jedem Fall ist Fairness wichtig. [1]

Je nach Art der Aufträge und Art der Nutzung der Anlage, benötigt man verschiedene Verfahren, die Scheduling-Algorithmen genannt werden.

1 Gründe für die Plattformwahl

Bei der Plattformwahl wurden drei Plattformen in die engere Auswahl genommen. Zur Wahl standen die Android-Plattform, die Java-Plattform oder eine Webanwendung. Unsere Wahl fiel ziemlich schnell auf die Java Plattform. Die Gründe hierfür liegen auf der Hand. Was die Erfahrung in Programmiersprachen angeht, hatte Java eindeutig die stärkeren Argumente. Zwar sind wir mit weiteren Programmiersprachen vertraut, jedoch bei weitem nicht so stark wie das bei Java der Fall ist. Ein weiterer Grund für die Java-Plattform ist die zu benutzende Entwicklungsumgebung. Unsere bevorzugte Umgebung ist Eclipse. Vor allem die umfangreiche Debugger Funktion von Eclipse hilft uns in der späteren Testphase schnell und sicher Fehler auszumerzen.

Desweiteren gab es Programmierspezifische Gründe für diese Wahl. Hauptteil der gestellten Aufgabe ist die Implementierung von sieben verschiedenen Scheduling-Verfahren. Diese Algorithmen haben teilweise dieselben oder ähnliche Eigenschaften und unterscheiden sich in kleinen Änderungen. Zum Beispiel muss bei Shortest-Job-First aus einer Liste mit Prozessen denjenigen mit der kürzesten Laufzeit suchen und zurückgeben. Bei dem Algorithmus des Longest-Job-First hingegen, sucht man den Prozess mit der längsten Laufzeit. Somit bieten sich die programmiersprachenspezifischen Eigenschaften von Java an. Mit der Vererbung, abstrakten Klassen und Methoden lassen sich diese Gemeinsamkeiten und oftmals kleinen Unterschiede der Algorithmen sehr gut implementieren ohne doppelten Code zuschreiben.

Java funktioniert nach dem Konzept Write Once, Run Anywhere (deutsch: „Einmal schreiben, überall ausführen“). Das bedeutet, dass man ein Programm, das in Java programmiert wurde, theoretisch nur einmal zu kompilieren braucht und es auf allen anderen Systemen läuft, die eine Java-Laufzeitumgebung (Java Runtime Environment bzw. JRE) besitzen. Dies wird dadurch erreicht, dass Java zunächst in Bytecode kompiliert wird, dieser wird von der JRE beim Starten des Programms erst in die Maschinensprache kompiliert (Man spricht hier von einem JIT-Compiler). Somit ist Java auf verschiedenen Zielplattformen einsetzbar. Um das Programm auf der Android-Plattform einzusetzen, müsste man für die GUI nur eine anderen Mechanismus (HTML/XML) als Java-Swing verwenden.

2 Architektur des Programms

Das Program folgt der Zweischicht Architektur. Hierbei existieren eine Daten Anzeigende und eine datenverarbeitende Schicht.

Die Anzeigende Schicht (GUI) stellt die Schnittstelle zum Benutzer dar. Hierbei werden einerseits die Daten in ein vom Menschen lesbares Format aufgearbeitet, andererseits existieren optische Eingabe-Mechanismen zum steueren des Programms durch den Benutzer. Im Fall des Visual-Schedulers hat das GUI die Aufgabe den Benutzer Prozess-Informationen eingeben und ihm die daraus resultierenden Scheduling-Ergebnisse graphisch darzustellen.

Die datenverarbeitende Schicht (Processing oder Buissness-Tier) hat (im Kontext der Zweischichtarchitektur) die Aufgabe Daten zu halten und zu verarbeiten. Man kann sie als Zusammenschluss des Processesing und dem Datakeeping (oder Data-Tier) einer 3-Schicht-Architektur betrachten. Im Fall des Visual Schedulers haellt das Processing die Prozess-Definition Daten sowie das Scheduler-Resultat falls vorhanden.

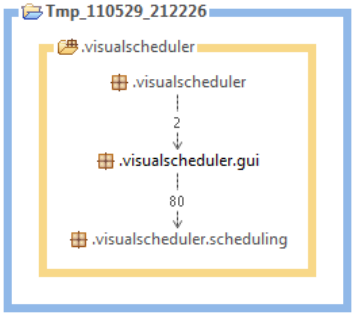

Abbildung 1. Packages

3 Schnittstelle GUI/Processing

Die Architektur der GUI-Processing Schnittstelle ist breit. Im Gegensatz zur gekapselten Schnittstellen Architektur die Interaktion der Schichten auf wenige Klassen beschraenken, existieren im Visual Scheduler viele GUI-Klassen mit genau einem Gegenstück.

Beispiele Hierfür sind u.a. das Process-Table-Model. Das Tabellen-Steuerelement JTable zeigt die Daten eines Tablemodels an. Um die Daten einfach zu halten beinhaltet die Klasse eine Listen mit den Prozessen. Damit wird die Klasse einerseits als Speicher verwendet andererseits ist sie direkt anzeigbar ohne das eine Adapterklasse nötig wird. Das Tablemodel interface beinhaltet methoden wie getColumnCount() und getValue(x,y) und stellt dem Table eine logische Matrix oder Tabelarische Struktur zur verfuegung welche dieser Anzeigt.

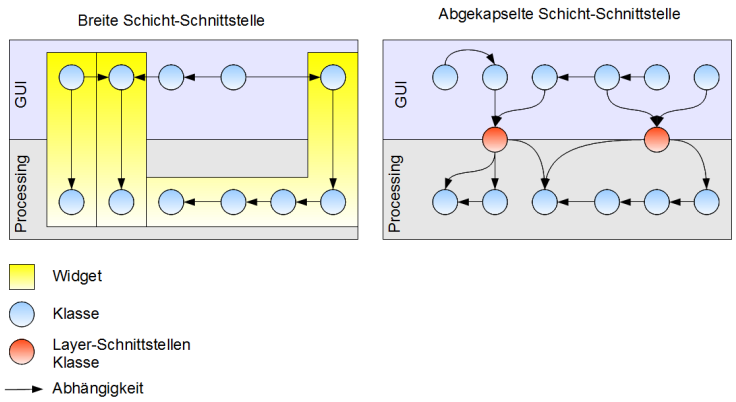

Abbildung 2. Zweischicht Architekturen

4 GUI - Überblick

Das GUI unterteilt sich in 2 Teile. Einen Teil zur Definition der Prozess-Informationen und einen Teil zur Presentation der Ergebnisse. Zwischen den Bereichen wird per Previous/Next navigiert. Damit verhält sich das Program wie ein Wizard und erlaubt auch unerfahrenen Benutzern die Nutzung der Software. Auch die verfuegbaren Buttons sind auf ein Minimum reduziert um die Komplexität gering zu halten.

Das Program benutzt das Substance Look And Feel (LaF) als 'Skin'. Java beinhaltet das Pluggable-Look-And-Feel Schema. Das Aussehen aller Komponenten des Programs kann durch setzen eines LaFs geändert werden. Eigene Swing-Komponenten müssen in der Regel über die LaFs 'bescheit wissen' um von diesen effektiv gestyled zu werden. Das Substance-LaF beinhaltet die Trident Bibliothek mit welcher Swing (Teilweise AWT) Komponeneten animiert werden konnen. Substance benutzt diese Lib um weiche Mouse-Over-Effekte für Swing Elemente (wie JTable) zur Verfügung zu stellen. Desweiteren besteht die Möglichkeit Wasserzeichen zu setzen. Damit wird der Metallik-Effekt des GUIs erzeugt.

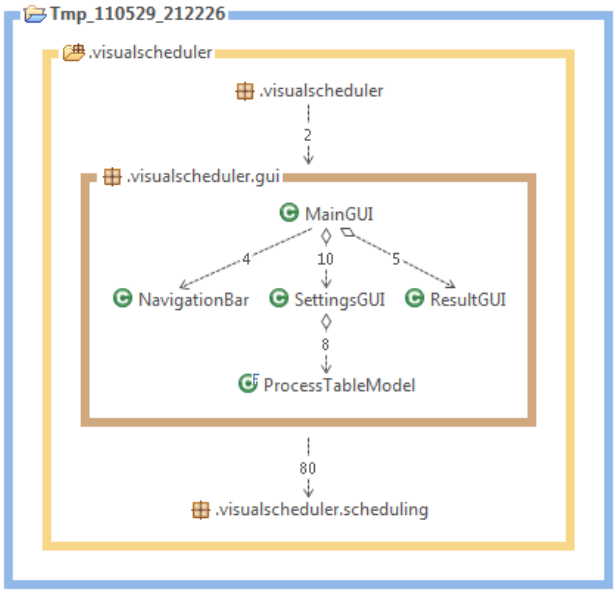

Abbildung 3. GUI

5 GUI - Process-Definition-View

Diese Sicht sieht der Benutzer wenn er das Program startet. Prozess-Informationen werden in einer Tabelle Dargestellt. Alle Werte können bearbeitet werden. Diese Möglichkeit bietet JTable wenn das TableModel die setValue() implementiert. Zellen werden wie in Tabellen-Kalkulationsprogrammen angewählt und können in-place bearbeitet werden. Hierbei findet eine Verifikation der Eingaben statt sofern das TableModel getClass(int column) Sinnvoll implementiert. Trägt der benutzer einen String für eine Spalte die Integer als Klasse hat ein, färbt sich der Rahmen der Zelle rot. Der Keyboard-focus bleibt in der Zelle gefangen bis ein gültiger Wert eingetragen wird oder der Benutzer die Bearbeitung mit drücken von ESC abbricht. In dem Fall wird die Zelle auf das letzte gültige Resultat zurückgesetzt. Desweiteren erlaubt JTable das sortieren der Spalten mit Drag and Drop sowie Klassensensitive (sofern getClass implementiert ist) sortierung der Reihen nach beliebig vielen hintereinandergeschalteten Sortierreihenfolgen. All diese Möglichkeiten bietet JTable (in JRE 1.6 was wir benötigen) ohne viel zusätzliche arbeit an. Es muss lediglich ein sinnvollen Tablemodel erstellt werden. Um neue Prozesse zu definieren steht eine art Form zur Verfügung. Wird diese ausgelöst übergibt sie die neuen Daten an das ProzessTable-Model welches wiederum eine Change-Event auslöst was ein Resort/Redraw des JTables zur folge hat. Das löschen ist möglich indem eine Reihen im Table selektiert wird un der Benutzer anschliessend auf den Löschen Button klickt. JTable bietet Reihen-, Spalten- und Zellen-Selektion an. Der Visual Scheduler benutzt nur Reihenselection. Der Benutzer kann irgenwo in die Reihe klicken um sie auszuwählen. Substance sorgt hierbei für eine schönen Animation.

6 GUI - Scheduling-Result-View

Hier wird das Ergebnis des Schedulings Präsentiert. Die Darstellung erfolgt als modifiertes Gant-Chart. Das Steuerelement benutz eine eigenes PLaF Konzept. Die Balken wiederholen ein Bitmap-Muster welches im Programm hinterlegt ist. Um das Program zukünftig um weitere Styles zu erweitern müsste ein Style als Paar aus Substance-LaF und Scheduling-Result-LaF definiert werden. Die Aktuelle Version beinhaltet nur einen Style wobei beide SSubstylesäufeinader abgestimmt sind. Das innere des Scheduling-Result-Viewers überschreibt die paint() methode um ein Costum paint zu ermöglichen. Anstatt das Element mit weiteren sub-elementen zu befüllen werden die Daten direkt auf das element gezeichnet. Das Element sitzt in eine JScrollPane. Ein Scrollpane ist eine Sicht auf eine anderes Steuerelement das zu groß ist um direkt angezeigt zu werden. Das Scrollpane stellt Scroll-bars zur verfügung um den Sichtaberen Ausschnitt des beinhalteten Steuerelements zu ändern. Die besonderheit des JScrollpanes im vergleich zu Scrollpanes in anderen Sprachen, ist die Tatsache, dass Skalen sehr einfach zu implementieren sind. Im Result-Viewer wird oben ein Zeit-Skala angezeigt. Horizonales Scrollen (Links/Rechts) verschiebt diese Skala mit dem Inhalt. Vertikales Scrollen (oben/unten) ändert die Position der Skala nicht. Wird nach

unten gescrollt bleibt die Skala trotzdem ganz oben und der Inhalt scheint dahinter zu verschwinden. Gleiches gilt für die Prozess Liste Links. Diese beleibt immer links. Bei Hoch/Runter Scrollen bewegt sie sich hingegen mit.

7 Processing - Überblick

Neben der schon vorgestellten Process-TableModel Klasse besteht das Processing hauptsächlich aus den Schedulern. Alle Scheduler erweitern die Abstrakte Basisklasse AbstractScheduler. Arbeitschritte die bei allen Schedulern gleich sind werden hier behandelt. Spezifischer bekommt der AbstractScheduler eine Liste mit Prozess-Definitionen aus den das Scheduling abgeleitet werden sollt. Der AbstractScheduler bricht die Aufgabe in Teilaufgaben auseinander wobei für jede Teilaufgabe die Scheduler iMplemementierung konsultiert wird. Die Ergebnisse werden vom Abstract Scheduler gesammelt und zu einen gesamtergebniss zusammengesetzt.

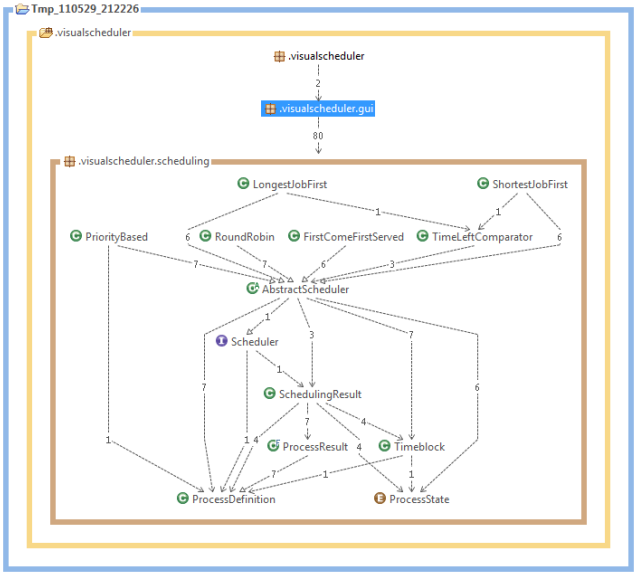

Abbildung 4. Processing

8 Processing - Abstract-Scheduler

Der AbstarctScheduler Durchläuft eine Schleife bis alle Prozesse in der Prozess-Liste abgearbeitet wurden. Dabei wird zu nächst der Zeitpunkt 0 betrachtet. Der AbstractScheduler erzeugt eine Liste aus allen Prozess die zum betrachteten Zeitpunkt bereits gestartet wurden und noch nicht beendet sind. Aus dieser Liste wählt die Scheduler-Implementierung einen Eintrag aus und gibt an wie lange daran gearbeitet werden sollt. Damit verbirgt der Abstract Scheduler einige Komplexität von den Implementierungen. Es gibt Fälle in denen die Liste leer ist (Wenn zwischendurch alles erledigt ist aber später noch mehr kommt). In dem Fall inkrementiert der AbstractScheduler seinen Zeit-wert und fährt fort ohne die Scheduler Implementierung zu konsultieren. Damit ist garantiert, dass die Auswahliste welche die Implementierungen sehen immer mindestens ein Element hat. Desweiteren werden die Prozesse in Reihenfolge ihrer Ankunft sortiert.

Auch das Preemptive Scheduling wird vom Abstract Scheduler behandelt. Ist ein Algorithmus preemptiv wird die Arbeitszeit welche die Scheduler Implementierung zurückgibt vom AbstractScheduler auf die Ankuftzeit des nächsten Prozesse (falls nötig) zurückgesetzt.

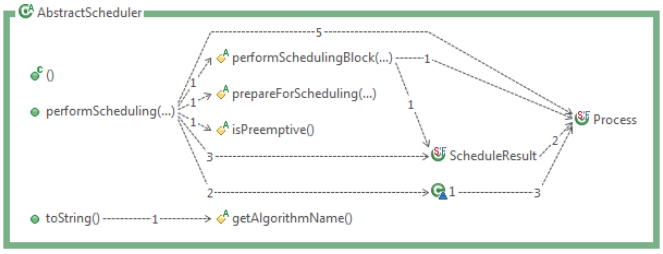

Abbildung 5. AbstractScheduler

9 Scheduling-Algorithmen

Dieser Abschnitt listet in der Anwendung vorgeführte Scheduling Algorithmen auf. Eine Übersicht bietet Tabelle 1.

Tabelle 1. Scheduling-Algorithmen

Algorithmus	Präemptiv	Laufzeit bekannt	Batch/Interaktiv
Shortest-Job-First	Nein	Ja	Batch
Longest-Job-First	Nein	Ja	Batch
First-Come-First-Serve	Nein	Nein	Batch
Last-Come-First-Serve	Nein	Nein	Batch
Round Robin	Nein	Nein	Interaktiv
Priorität-Scheduling	Beides Möglich	Nein	Batch
Longest Remaining Time First	Ja	Ja	Batch
Shortest Remaining Time First	Ja	Ja	Batch

9.1 Shortest Job First

Shortest-Job-First (SJF) ist ein nonpräemptives Scheduling-Verfahren, das eingesetzt wird, um rechenwillige Threads oder/und Prozesse auf die physischen Prozessoren des Rechners zu verteilen. Das SJF ist ähnlich dem FIFO-Verfahren und setzt die Prioritäten der rechenwilligen Prozesse anhand der Länge des CPU-Bursts(Rechenzeit). Threads mit der kürzesten Laufzeit werden zuerst der CPU zugeteilt. Kommen zwei Threads mit derselben Rechenzeit an wird sie anhand des First In First Out(FIFO)-Verfahren die Prozesse zugeteilt. Das SFJ hat den Vorteil, dass es die Wartezeit optimal zuteilt. Die Nachteile des SJF sind zum Einen, dass Prozesse mit hohem Burst eventuell nie der CPU zugeteilt werden, zum anderen dass der Algorithmus nur sehr schwer zu implementieren ist, da die Rechenzeiten der Prozesse nur selten bekannt sind. Implementierungen sind somit nur näherungsweise möglich.

Die Implementierung dieses Verfahrens ist sehr einfach. Dies geschieht in der Klasse ShortestJobFirst. Die von der Abstrakten Klasse AbstractScheduler geerbten Methode ScheduleResult() wird überschrieben um nach dem SJF zu sortieren. Die Methode nimmt eine ArrayList entgegen welche die aktuellen Prozesse enthält. Von dieser Liste wird eine flache Kopie erstellt. Danach wird anhand der Methode Collections.sort() aus dem Interface der Collections die Liste nach aufsteigender Restzeit sortiert. Danach wird das erste Element der Liste als neues ScheduleResult zurückgegeben.

9.2 Longest-Job-First

Longest-Job-First (LJF) ist auch ein nonpräemtives Scheduling-Verfahren, welches das Gegenteil zu SJF stellt. Das LJF legt die Prioritäten der rechenwilligen Prozesse anhand Länge des CPU-Bursts fest. Threads mit der längsten Laufzeit werden zuerst der CPU zugeteilt. Die Nachteile des SJF sind dieselben wie bei SJF. Prozesse mit kurzer Abarbeitungszeit haben eventuell sehr hohe Wartezeiten oder können sogar verhungern. Somit ist dieses Verfahren genauso nicht-fair wie SJF. Ebenfalls sind hier die Laufzeiten der Prozesse unbekannt. Somit gestaltet sich auch hier die Implementierung schwierig. Somit kann man wie bei SJF die Bursts nur näherungsweise bestimmen. Die Schätzung erfolgt dann wie bei SJF.

Die Implementierung erfolgt in der Klasse LongestJobFirst. Die von der Abstrakten Klasse AbstractScheduler geerbten Methode ScheduleResult() wird überschrieben um nach dem LJF zu sortieren. Die Methode nimmt eine ArrayList entgegen, welche die aktuellen Prozesse enthält. Von dieser Liste wird eine flache Kopie erstellt. Danach wird anhand der Methode Collections.sort() und Collections.reverse() aus dem Interface der Collections die Liste nach absteigender Restzeit sortiert. Danach wird das erste Element der Liste als neues ScheduleResult zurückgegeben. Dieses Ergebnis beinhaltet den Prozess mit der längsten Laufzeit.

9.3 Shortest/Longest Remainder First

Diese Algorithmen sind mit Longest Job First bzw. Shortest Job first verwandt. Der einzige Unterschied ist, dass Shortest/Longest Remainder preemptiv sind. Da ein Algorithmus im Visual Scheduler mittels einer abstrakten Funktion angibt ob er preemptiv ist oder nicht, werden Longest Job/Remainder und Shortest Job/Remainder durch die jeweils gleichen Klassen abgebiltet. Ein bool Variable gibt an ob preemptiv oder nicht.

9.4 Priority based Scheduling

Beim prioritätenbasierten Scheduling wird jedem Prozess eine Wichtigkeitsstufe, die Priorität, zugeordnet. Echtzeit-Systemkernels beispielsweise unterstützen im allgemeinen 256 Prioritäten-Ebenen, wobei 0 die höchste und 255 die niedrigste darstellt. Der Prozessor führt stets den Prozess mit der höchsten Priorität aus. Der Prozess mit der höchsten Priorität verdrängt also Prozesse mit niedrigerer Priorität. Dies geschieht dadurch, dass den anderen Prozessen der Prozessor entzogen wird weil der Prozess mit der höheren Priorität diesen benötigt. Haben Prozesse die gleiche Prioritätsebene, so wird zumeist das RoundRobin-Verfahren angewandt. Die Priorität der Prozesse kann nach verschiedenen Kriterien festgelegt werden. Das Prioritätenbasierte Scheduling kann statisch oder dynamisch

verfahren. Beim statischen Verfahren wird die Priorität jedes Prozesses zum Erzeugungszeitpunkt festgelegt. Dieser Wert kann im weiteren Verlauf nicht mehr geändert werden, wodurch eine deterministische Ordnung zwischen den Prozessen erreicht wird. Beim dynamischen Verfahren kann die zugewiesene Priorität vom Betriebssystem verändert werden. Ist das Prioritätenbasierte Scheduling nichtpreemptiv konstruiert, so bleibt ein Prozess solange über dem Prozessor, bis er eine blockierende Systemfunktion aufruft oder den Prozessor freiwilig abgibt. Ist es hingegen preemptiv, so wird die aktuelle Prozessorzuordnung unmittelbar unterbrochen, sobald ein neuer Prozess mit einer höheren Priorität bereitsteht.

9.5 First Come First Served

Prozesse werden in ihrer Ankunftsreihenfolge abgearbeitet. Der Algorithmus ist nicht preemptiv. Man kann den Algorithmus als Prozess-Queue (First in First out) ansehen.

9.6 Last Come First Served

Prozesse werden in entgegen ihrer Ankunftsreihenfolge abgearbeitet. Der Algorithmus ist nicht preemptiv. Man kann LCFS als Prozess-Stack (First in Last Out) betrachten.

9.7 Round Robin

Prozesse werden in Reihenfolge ihrer Ankunft verarbeitet. Am Prozess wird solange gearbeitet bis er beendet ist oder ein maximales Zeitfenster (timeslice) verstrichen ist.

10 Schlusswort

Schedulingverfahren bilden den Kern moderner Betriebsysteme. Ohne Verständnis von Scheduling wird man kaum in der Lage sein, Abläufe auf modernen Rechenanlagen zu verstehen. Beenden möchten wir mit dem Zitat eines Betriebsysteme-Klassikers:

> The key to multiprogramming is scheduling. [2]

Literatur

1. Tanenbaum, Andrew S. *Moderne Betriebssysteme.* Pearson Studium. 2009
2. Stallings, William. *Operating systems : internals and design principles.* Pearson Education International 2009