Betriebssysteme Praktikum WS2015/16

Jan Lukas Deichmann, Veith Güntel
Betreuer: Prof. Dr. Ole Blaurock

January 23, 2016

Contents

1 Präambel

In diesem Dokument wird die Lösung zu der gebenen Projektaufgabe im Kurs Betriebssysteme WS2015/16 dargestellt. Bei dieser Aufgabe ging es darum, für eine gegebene Simulierung eines Betriebssystemes, SimOS, eine geeignete Speicherverwaltung zu entwickeln.

2 Getting Started

Die Inbetriebnahme unserer Implementation erfolgt sehr simpel über die entsprechend Funktion von Visual Studio, sollte man sich zum Import in die IDE entscheiden, oder über die beigefügte .exe Datei. Es ist zu beachten, dass SimOS eine *processes.txt* Datei entweder im Projektordner für Visual Studio oder im selben Ordner wie die .exe erwartet. Diese Datei dient als Liste von Prozessen, die von SimOS einzulesen und auszuführen sind. Die processes.txt baut sich folgendermaßen auf: Man legt pro Zeile die Informationen zu einem Prozess fest, wobei die erste Zeile als Kommentar gewertet wird. Die Informationen müssen sich in der Reihenfolge OwnerID, start, duration, size, type in der Datei befinden. Eine beispielhafte Datei würde etwa so aussehen:
INSERT EXAMPLE FILE

Die entsprechende Datei würde folgende Ausgaben generieren:
INSERT EXAMPLE OUTPUT

3 Designreport

3.1 Getroffene Annahmen

In unserer Implementierung wird die Annahme getroffen, dass keinem Prozess jemals die PID 0 zugewiesen wird. Zudem wird angenommen, dass keine Prozesse gestartet werden, deren Speicherbedarf die Größe des Systemspeichers überschreitet.

3.2 Funktionsumfang

Wir erweiterten die gegebene Implementierung von SimOS im wesentlichen um Funktionen zur Verwaltung des Arbeitsspeichers (realisiert in m_list.c) und zur temporären Speicherung von Prozessen, die aktuell zu groß für den Arbeitsspeicher sind (realisiert in process_queue.c). Nähere Informationen zu m_list und process_queue befinden sich in der Erklärung zur Implementierung. Weitere Änderungen waren die Umlagerung von Variablen, um bessere Lokalität zu gewährleisten, und Änderungen, welche die Laufzeit optimieren. LETZTERES GENAUER

3.3 Betriebsbedingungen

Die einzige Betriebsbedingung ist die in "Getting Started" beschriebene, korrekt formatierte *processes.txt* Datei.

3.4 Implementierung

3.4.1 processQueue

irgendwie n Diagramm Die Process Queue die wir in *process_queue.c* realisiert haben ermöglicht es dem Betriebssystem Prozesse die aktuell, mangels ausreichenden Speichers, nicht ausgeführt werden können in einer Warteschlange zu Speichern und später aufzuführen ohne dafür die Ausführung anderer Prozesse blockieren zu müssen. Die Process Queue basiert in ihrer Implementierung auf einer Listenstruktur und stellt die structs processQueue und processQueueNode zur verfügung (siehe *m_list.h*). Desweiteren stellt die Schnittstelle folgende Funtionen zur Verwendung in SimOS zur verfügung:

processQueue* makeProcessQueue() erzeugt eine neue processQueue, alloziert passenden Speicher und gibt einen Pointer auf die Queue zurück.

processQueueNode* makeProcessQueueNode(PCB_t* pcb) erzeugt eine neue processQueueNode, die als Inhalt den übergebenen Pointer auf eine PCB_t enthält, alloziert passenden Speicher und gibt einen Pointer auf die QueueNode zurück.

void enqueue(processQueue* queue, PCB_t* process) ermöglicht das einreihen eines Processes in die Queue. Es wird intern eine processQueueNode erzeugt und in die gegebene queue eingereiht.

PCB_t* dequeue(processQueue* queue) ermöglicht es den ersten Prozess aus der gegebenen Queue zu entfernen und gibt einen Pointer auf den Prozess zurück.

Boolean isEmpty(processQueue* queue) eine Hilfsfunktion die zurückgibt ob gegebene queue leer ist.

3.4.2 mList

auch irgendwie n Diagramm

Die Memory list ermöglicht die tatsächliche Speicherverwaltung. Sie basiert auf einer Listenstruktur und beeinhaltet Blöcke die für die Segmente im Arbeitsspeicher stehen, die jeweils frei oder genutzt sein können. Zur Realisierung dieser Aufgabe stellt die Implementierung die structs mList und MlistNode, sowie einen enum mType für die zustände der Blöcke bereit. Desweiteren stellt die Schnittstelle folgende Funktionen zur Verwendung in SimOS zur verfügung:

mList* makeMList() erzeugt eine neue MemoryList, alloziert passenden Speicher und gibt einen Pointer auf die Liste zurück.

mListNode* makeMListNode(mType type, unsigned pid, int start, int length) erzeugt eine neue mListNode aus gegebenen Parametern, alloziert passenden Speicher und gibt einen Pointer auf die Node zurück.

Boolean addProcess(mList* list, PCB*^ process) fügt den process der gefragten list hinzu.

void removeProcess(mList* list, PCB_t* process) entfernt den process aus der gefragten Liste.

void compact(mList* list) kompaktiert die gegebene Liste um der ihrer Fragmentierung entgegenzuwirken. WANN WIRD GECALLT??

mListNode* swapper(mListNode* a, mListNode* b) eine Hilfsmethode zum Austausch zweier Node innerhalb einer Liste, gibt einen Pointer zur neuen mListNode b zurück. Wird innerhalb von compact genutzt.

mListNode* merge(mListNode* a, mListNode* b) eine Hilfsmethode zum zusammenfügen zweier Nodes, gibt einen Pointer zur zusammengefügten mListNode zurück. Wird innerhalb von compact verwendet.

mListNode* findNextFit(mList* list, int length) eine Hilfsmethode zum finden der nächsten passenden Lücke innerhalb der Liste. Gibt einen Pointer auf die mListNode* VOR der passenden Lücke zurück.