Betriebssysteme I – Projektarbeit

Implementierung einer Prozessverwaltung mit First Fit in SimOS

1. Getroffene Annahmen

Wir als Gruppe versuchen, die Prozessverwaltung von SimOs so nah wie möglich an realen Gegebenheiten anzupassen. Zum Beispiel die effektive Nutzung des zur Verfügung stehen Speichers durch Verschmelzung von entstandenen Lücken ,einer Kompaktierung beim Auftreten mehrerer Lücken die in Summe Platz für einen Prozess bieten würden und die Möglichkeit der Einlagerung von mehreren Prozess die in Summe mehr Speicherplatz brauchen als der Arbeitsspeicher zu Verfügung stellt.

Die momentane Version von SimOS und unsere Erweiterung beinhalten keinen Scheduler, weshalb wir annehmen, dass kein Prozess vor seiner regulären Terminierung beendet oder angehalten werden kann

Wir wollen auch versuchen, unsere Implementation so zu entwickeln, dass sie möglich verständlich, modular und austauschbar ist. Ebenso soll unsere Implementierung selbst bei extremen Bedingungen funktionieren. Dies beinhaltet zum Beispiel den Fall, das bis zu 1000 Prozesse eingelagert werden können.

Da wir annehmen, dass ein Prozess nicht spaltbar ist, darf er keine Speichergröße, größer als die zuvor maximal definierte Speichergröße ist, anfordern. Da wir von sinnvollen Prozess Attributen ausgehen, werden unsinnige Prozess Attribute wie z.B. Speichergröße Null nicht abgefangen.

Wir nehmen an, dass kein Prozess zu einem Zeitpunkt startet der größer als 1000 Zeiteinheiten ist. Weiterhin muss der erste Prozess beim Zeitpunkt 00 starten.

Die process.txt darf nicht leer sein und keine Zeichen enthalten, die nicht zu einem Prozess zugeordnet werden können.

Wir reduzieren die Attribute eines Prozess in unsere Prozessverwaltung auf die für uns notwendigen:

- unsigned PID — Process identifier
- unsigned Start — Startposition eines Prozesse
- unsigned Length — Größe eines Prozesses
- Boolean OccupiedByProcesss — Prozess ist ein Prozess und keine Lücke

Wir nehmen an, dass ein Prozess nur aus seinen Attributen besteht.

2. Eigenschaften unsere Prozessverwaltung

Zunächst möchten wir unsere Entscheidung bezüglich der Verwaltung des Speichers erklären.

Datenstruktur	Unsere Meinung
Bitmaps	Suche in Bitmaps sind ineffizient
Einfach Verkettet Liste	Rückschritt nur doch vollen Durchlauf
Doppelt Verkettet Liste	Rückschritt schnell dank Rückverweis

Quelle Foliensatz 03-Speicherverwaltung erstellt von Prof.Dr.Blaurock

Unsere Prozessverwaltung besitzt somit eine doppelt verkettete Liste für Prozesse und Lücken.

Beim Einfügen des aller ersten Prozesses entsteht eine Lücke, die den gesamten verfügbaren Speicher darstellt. Der erste Prozess wird in diese Lücke eingefügt und erstellt eine weitere Lücke, wenn der Prozess nicht den gesamten Speicher braucht.

Ebenfalls möchten wir unsere Entscheidung bezüglich der Auswahlstrategien erklären.

Auswahlstrategie	Unsere Meinung
First Fit	Leichte Implementierung
Next Fit	Effizient wie First Fist, aufwendigere Implementierung
Best Fit	Langsamer als First Fit, erzeugt viele kleine Speicherfragmente
Worst Fit	Weniger effizient als First Fit
Quick Fit	Mehrere Listen, aufwendige Implementierung

Quelle Foliensatz 03-Speicherverwaltung erstellt von Prof.Dr.Blaurock

Die Prozessverwaltung soll folgende Eigenschaften haben:

- Der aller erste Prozess muss keine Lücke suchen
- Jeder weitere Prozess muss mit Hilfe unserer Auswahlstrategie eine Lücke finden
- Wird eine Lücke gefunden die größer ist als der vom Prozess benötigte Speicher entsteht eine neue Lücken mit dem restlichen Speicher
- Für einen Prozess der mangels Speicher nicht eingelagert werden kann entsteht eine Warteschlange, wo dieser und weiter Prozesse mangels Speicher eigereiht werden können
- Ein Prozess in der Warteschlange hat Vorrang vor Prozessen die neu dazu kommen
- Warteschlange wird der Reihe nach abgearbeitet
- Wenn ein Prozess terminiert entsteht an seiner Stelle eine Lücke
- Sobald eine Lücke entsteht wird sie wenn möglich mit ihren direkten Nachbar verschmolzen
- Ein Prozess der eingelagert werden könnte, wenn alle Lücken verschmolzen wären, stößt die Kompaktierung an und wird durch First Fit eingelagert

a. Mechanismen der Prozessverwaltung und unserer Auswahlstrategie

Unsere Prozessverwaltung besteht aus einer doppelt verketteten Liste wobei jeder Prozess und jede Lücke eine Zelle darstellt.

Die Warteschlange besteht aus einem zum Start des Programmes initialisierten Array. Die Array-Kapazität wird durch die maximal definierte Anzahl der Prozesse bestimmt.

Wird ein Prozess in die Warteschlange eingereiht, wird der Index, wo der nächste Prozess eingefügt werden kann, erhöht. Beim Auslesen wird gleich verfahren.

Prozess Start:

Sobald ein Prozess nach dem Regeln des SimOS startet wird er durch First Fit in unsere Prozessverwaltungsliste entsprechend eingereiht. Sollte ein Prozess auf Grund seiner Größe nicht gestartet werden können, wird er in die Warteschlange eingereiht. Ausgenommen die Prozessgröße ist größer als die definierte Speichergröße. So ein Prozess betrachten wir als Ausnahmefall und ermöglichen nicht dessen Ausführung. Nach der Terminierung eines Prozesses wird ein Prozess, falls einer da ist, aus der Warteschlange geholt. Ist in der Prozessverwaltungs-Liste immer noch keine Lücke zu finden die größer oder gleich der Prozessgröße ist, wird eine Kompaktierung der gesamten Liste vorgenommen. Nun kann der Prozess gestartet und mit Hilfe von First Fit eingelagert werden.

Prozess Terminierung:

Sobald ein Prozess nach dem Regeln des SimOS terminiert ist, darf er in der Prozessverwaltungs-Liste als Lücke markiert werden. Sollte dieser Eintrag benachbarte Lücken besitzen werden diese Lücken miteinander verschmolzen.

Erläuterung der Funktionsweise der Warteliste

Ein Prozess, dessen Speichergröße größer als der noch zu Verfügung stehende Speicher ist, kann nicht sofort gestartet werden. Dieser Prozess wird erst in die Warteschlange eingereiht. Sobald ein anderer Prozess terminiert ist, wird in der Warteschlange nach einem Eintrag gesucht. Wird ein Prozess als Eintrag in der Warteschlange gefunden, wird analysiert ob er starten könnte, wenn eine Kompaktierung durchführen wird. Könnte er das nicht, wird gewartet, bis der nächste Prozess terminiert ist und neu geprüft. Könnte er gestartet werden, wir die Kompaktierung durchgeführt der Prozess aus der Warteliste entfernt, gestartet und mit Hilfe von First Fit eingelagert.

3. Beschreibung der Funktionen, ihren Aufgaben und Strukturen

Kurze Beschreibung aller Funktionen die nicht zur Ausgabe dienen

Funktionen	Aufgaben
void firstfit(unsigned pid, unsigned size)	Findet die Stelle wo ein Prozess in die Prozessverwaltungs-Liste eingelagert werden darf
void insertTheProcess(unsigned pid, unsigned size)	Lagert ein Prozess in die Prozessverwaltungs-Liste ein
void removeTheProcess(unsigned pid, unsigned size)	Markiert einen Eintrag in der Prozessverwaltungs-Liste als Lücke
void inserTheFirstEmptyFeld()	Einen Eintrag in der Prozessverwaltungs-Liste erstellen der den Restlichen Speicherplatz abbildet
void mergeEmptyField(unsigned pid, unsigned size)	Verschmilzt wenn möglich benachbarte Lücken
void compaction()	Kompaktiert die Prozessverwaltungs-Liste
Boolean isGapAvailable(unsigned pid, unsigned size)	Gibt zurück ob eine Prozess in irgendeine Lücke passt
void processOutput()	Gibt die Prozessverwaltungs-Liste in die Standart-Ausgabe aus
Boolean enQueue(BlockedListElement_t blockedProcess)	Fügt einen Prozess in die Warteschlange ein
Boolean deQueue()	Holt den ersten Prozess aus der Warteschlange
unsigned returnTotalGapSize ()	Liefert die Größe aller Lücken

Kurze Beschreibung aller im Log hinzugefügten Funktionen

Funktionen	Aufgaben
void logPidInList(unsigned pid, unsigned start, unsigned length, unsigned currentMemorySize, char *message)	Ausgabe welcher Prozess mit seinen Attributen zu welcher Systemzeit hinzugefügt wird und die Größe des noch zu Verfügung stehen Speichers
void logPidFromList(unsigned pid, unsigned start, unsigned length, unsigned currentMemorySize, char *message)	Ausgabe welcher Prozess mit seinen Attributen zu welcher Systemzeit terminiert und die Größe des noch zu Verfügung stehen Speichers
void logPidProcessTooLarge(unsigned pid)	Ausgabe das der Prozess zu groß ist
void logMerge(unsigned count, unsigned length, Boolean site)	Ausgabe welche Lücke mit welcher anderen Lücke wohin verschmolzen wird
void logFormat(char *message)	Ausgabe der übergeben Anzahl von Zeilenumbrüchen zur Formatierung
void logHeadline(char *message)	Ausgabe einer Überschrift
void logHMainMemoryRespresentation(Boolean PidOrLenght,	Ausgabe der Prozessverwaltungs-

unsigned size)	Liste
void logFirstMemorySize(unsigned size)	Ausgabe der definierte Gesamtspeicher Größe

Nähere Beschreibung der wichtigsten Funktionen

First Fit:

Unserer Liste wird von vorn nach hinten nach einer passenden oder größeren Lücke durchsucht.

Bei einer:

1. Passende Lücke: Alle Attribute der Lücke werden mit den Attributen des einzulagernden Prozesses ersetzt.

2. Größere Lücken:
   2.a) Alle Attribute außer der Speichergröße, mit den Attributen der Lücke mit den Attributen des einzulagernden Prozesses ersetzt.

   2.b) Danach wird berechnet wie viel Restspeicherplatz übrig bleibt, indem die Speichergröße der Lücke mit der Speichergröße des Prozesses subtrahiert wird.

   2.c) Nun wird die Speichergröße der Lücke wird mit der Speichergröße des Prozesses ersetzt.

   2.d) Danach wird eine neue Lücke mit der Größe des zuvor berechneten Rests erstellt und nach dem Prozess eingereiht.

Bitte nehmen Sie nur Veranschaulichung das Diagramm auf Seite 9 zu Kenntnis.

Verschmelzen benachbarter Lücken

Die Prozessverwaltungs-Liste besteht beim Start eines Prozesses aus einen Eintrag für diesen Prozess und der „Start-Lücke“ die den restlichen Speicherplatz darstellt.

Sobald ein Prozess endet, wird sein Eintrag in der Prozessverwaltungs-Liste geändert, in dem er als Lücke markiert wird. Durch die Terminierung eines Prozesses kann es somit dazu kommen das Lücken benachbarte Lücken haben. Um diesen Fall zu vermeiden, ruft unsere Implementation nach jeder Terminierung die Funktion *mergen* auf. In dieser Funktion wird geprüft, ob die neu entstandene Lücke benachbarte Lücken besitzt. Ist das der Fall, verschmilzt diese Lücken mit ihren Nachbar- oder ihren beiden Nachbar Lücken. Dabei sind 3 Fälle zu beachten.

1. Die Lücke hat einen linken und einen rechten Nachbarn die ebenfalls Lücken sind.
    a. Die Funktion berechnet zunächst, wie groß alle Lücken zusammen sind, und weist diesen Wert der mittleren Lücke zu.
    b. Danach werden alle Verweise auf die Lücken links und rechts der mittleren Lücke so verändert, dass die Zellen, die direkte Nachbarn der Lücke sind, in der Prozessverwaltungs-Liste nur noch auf die mittlere Lücke verweisen. Ebenfalls werden die Verweise der mittleren Lücke geändert, sodass sie nun auf die Zelle vor der linken und nach der rechten Lücke zeigen.
    c. Nun können die linke und rechte Lücke aus der Prozessverwaltungs-Liste entfernt werden.
2. Die Lücke hat nur einen linken Nachbarn, der bereits eine Lücke ist.
    a. Die Funktion berechnet die Größe beider Lücken und weist diesen Wert der linken Lücke zu.
    b. Danach werden alle Verweise, die auf rechte Lücke zeigen, dahingehen verändert, dass Sie nun auf die linke Lücke zeigen.
    c. Nun kann die rechte Lücke aus der Prozessverwaltungs-Liste entfernt werden.
3. Die Lücke hat nur einen rechten Nachbarn, der bereits eine Lücke ist.
    a. Die Funktion berechnet die Größe beider Lücken und weist diesen Wert der rechten Lücke zu.
    b. Danach werden alle Verweise die auf linke Lücke zeigen, dahingehen verändert, dass sie nun auf die rechte Lücke zeigen.
    c. Nun kann die linke Lücke aus der Prozessverwaltungs-Liste entfernt werden.

Für den Fall, dass vor oder nach den nun verschmelzenden Lücken keine Zellen, sind werden die, Verweise der verschmelzenden Lücken entsprechend angepasst.

Bitte nehmen Sie nur Veranschaulichung das Diagramm auf Seite 12 zu Kenntnis.

Kompaktierung

Szenario: Ein Prozess ist größer als alle zu Verfügung stehenden Lücken. Aber nicht als alle Lücken zusammen. Deshalb führt die Implementierung die Funktion *compaction* aus.

Die Funktion geht die Prozessverwaltungs-Liste von vorne nach hinten durch. Findet Sie dabei eine Lücke wird diese solange mit ihren rechten Nachbarn, sollte der Nachbar ein Prozess sein, vertauscht. Sollte die nun vertauschte Lücke als Nachbar keinen Prozess, sondern eine Lücke haben, wird die Funktion *mergen* aufgerufen.

Durch diesen Vorgang verändert sich die Prozessverwaltungs-Liste dahin, dass es nun nur noch eine Lücke am Ende der Liste gibt.

4. Nötige Änderungen der bereits vorhanden Implementation

a. Core

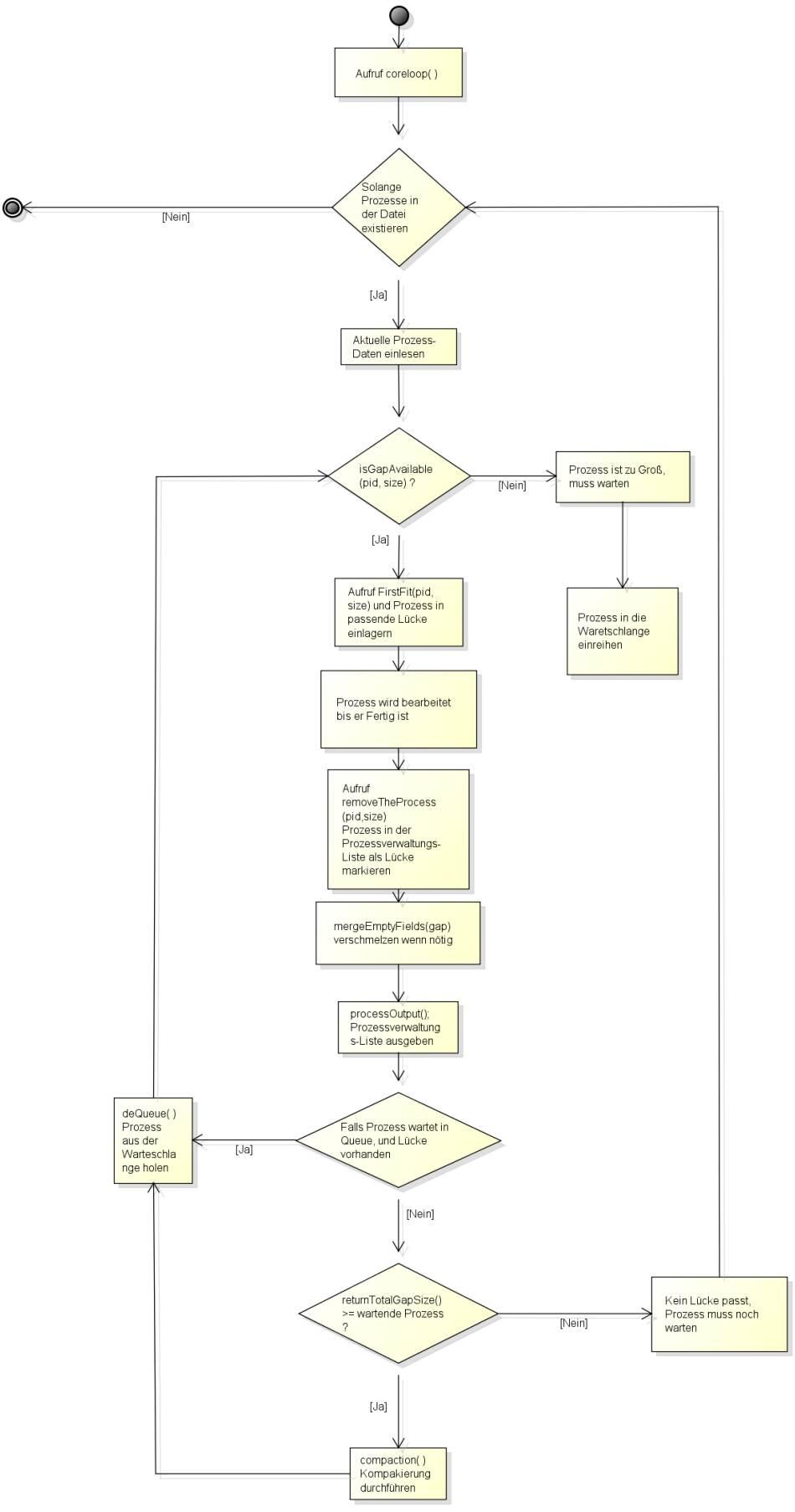
Der Core übernimmt das starten eines Prozesses. Daher muss der Core geändert werden, dass er beim Starten eines Prozesses diesen in unsere Prozessverwaltungs-Liste, durch die neue Funktion *firstfit*, einfügt. Danach möchten wir durch den aufruft von der Funktion *processOut*, zur Verbesserung der Nachvollziehbarkeit, die Prozessverwaltungs-Liste ausgeben.

Ebenfalls verändern wir den Core so, dass wir Prozesse, die wegen ihrer Größe nicht gestartet werden können, in die Warteschlange einreihen. Wenn ein Prozess terminiert ist muss der Core so erweitert werden, dass der Prozess als Lücke markiert wird. Danach muss in die Warteschlange geschaut werden, ob ein Prozess wartet. Wartet einer, wird dieser aus der Warteschlange entfernt und darf beim nächsten durchlauf gestartet werden.

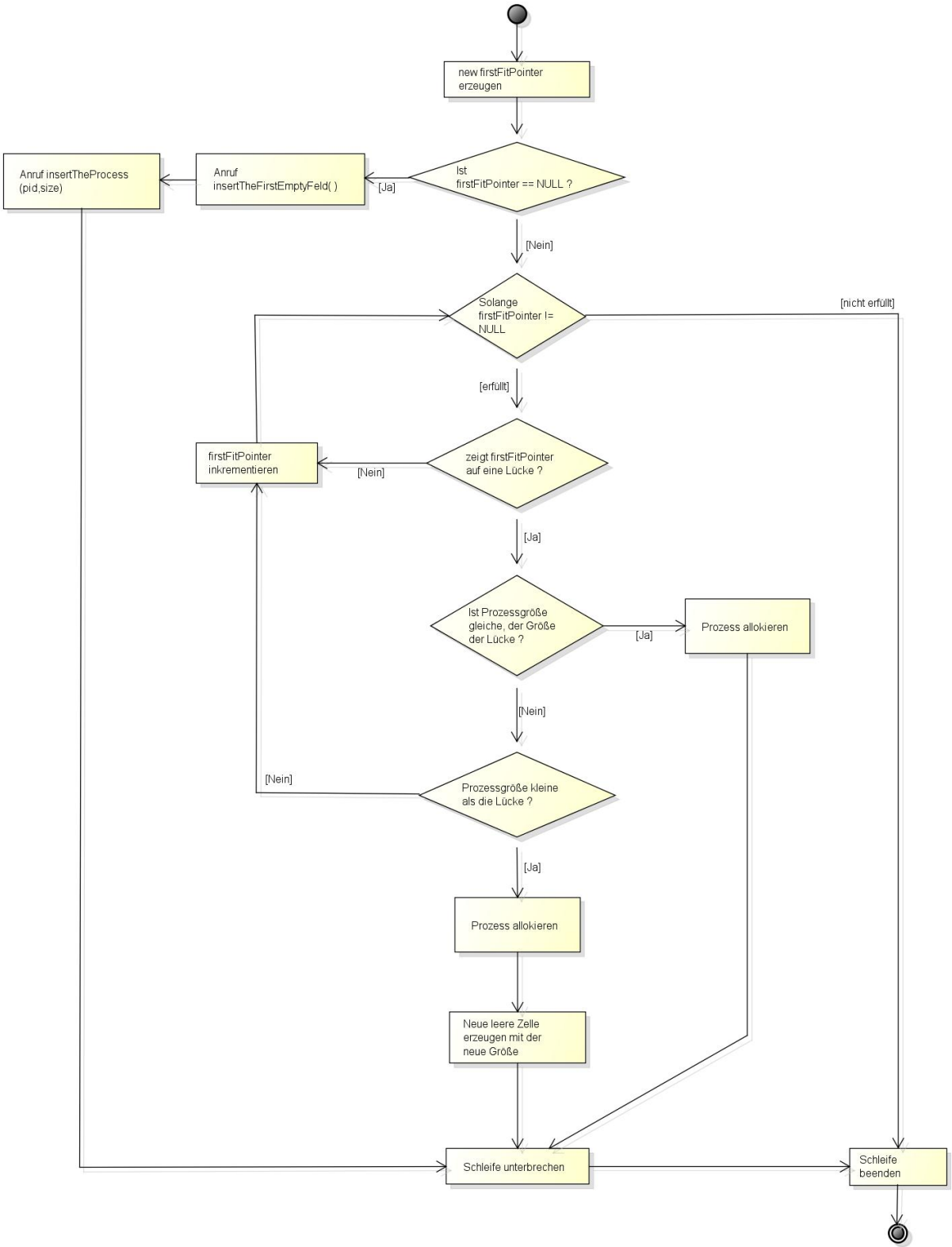
b. Log

Wir erweitern den Log dahingehen, das mehrere Funktionen zur Verfügung gestellt werden, die eine leichtere Nachvollziehbarkeit ermöglichen.

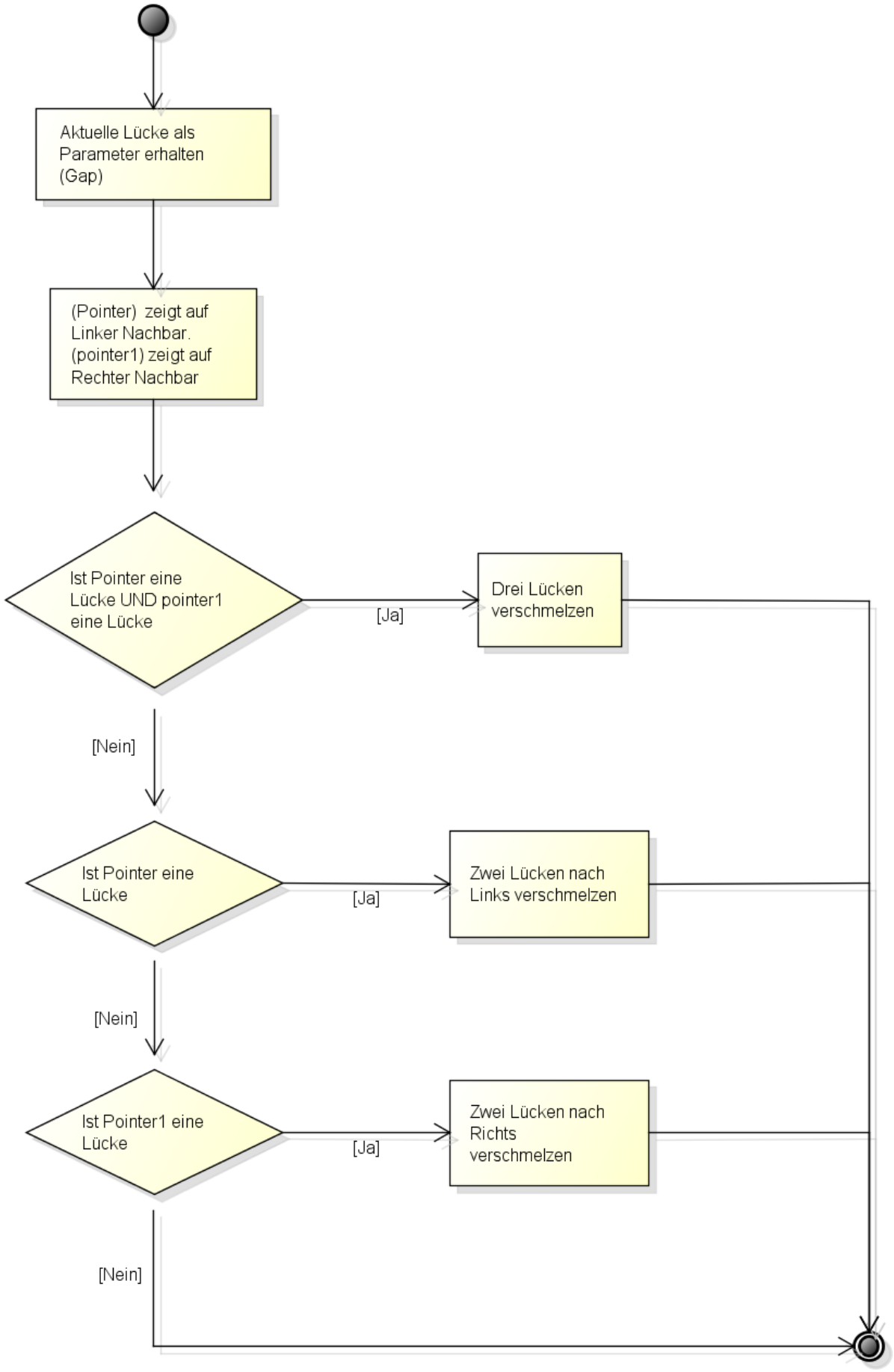
Gesamtsystem mit Veranschaulichung unserer Einstiegspunkte



FirstFit:


new firstFitPointer erzeugen
Ist firstFitPointer == NULL ?
[Ja]
Anruf insertTheFirstEmptyFeld( )
Anruf insertTheProcess (pid,size)
[Nein]
Solange firstFitPointer != NULL
[nicht erfüllt]
[erfüllt]
zeigt firstFitPointer auf eine Lücke ?
[Nein]
firstFitPointer inkrementieren
[Ja]
Ist Prozessgröße gleiche, der Größe der Lücke ?
[Ja]
Prozess allokieren
[Nein]
Prozessgröße kleine als die Lücke ?
[Nein]
[Ja]
Prozess allokieren
Neue leere Zelle erzeugen mit der neue Größe
Schleife unterbrechen
Schleife beenden

Mergen:


Aktuelle Lücke als Parameter erhalten (Gap)
(Pointer) zeigt auf Linker Nachbar. (pointer1) zeigt auf Rechter Nachbar
Ist Pointer eine Lücke UND pointer1 eine Lücke
[Ja]
Drei Lücken verschmelzen
[Nein]
Ist Pointer eine Lücke
[Ja]
Zwei Lücken nach Links verschmelzen
[Nein]
Ist Pointer1 eine Lücke
[Ja]
Zwei Lücken nach Richts verschmelzen
[Nein]

Autorenschaft:

Dieser Design-Report wurde in Zusammenarbeit von Wail Solaiman und Alexander Luhmann mit Hilfestellung von Prof.Dr.Blaurock erstellte.

Der Design-Report ist ein möglicher Erweiterungsvorschlag, zu dem uns zu Verfügung gestellten Programmes SimOs.

Quellen:

Exakte Zitate ober Kopien ganzer Absätze wurden nicht Verwendet.

Jedoch wurden einige Informationen vom Foliensatz 03-Speicherverwaltung, erstellt von Prof.Dr.Blaurock, entnommen.