Technische Universität Berlin Fachgebiet Komplexe und Verteilte IT-Systeme Sommersemester 2015	**Aufgabenblatt 1** zu – Systemprogrammierung – Prof. Dr. Odej Kao, Dr. Peter Janacik, Tutoren
Abgabetermin:	[1] – 27.04.2015 23:55 Uhr [2] – 04.05.2015 23:55 Uhr

Aufgabe 1.1: Von-Neumann-Architektur (2 Punkte) (Theorie[1])

Die am weitesten verbreitete Rechnerarchitektur wurde nach John von Neumann benannt.

a) Beschreiben Sie die Funktionen der vier grundlegenden Komponenten der Von-Neumann-Architektur: (1 Punkt)

- CPU
- Speicher
- Ein-/Ausgabegeräte
- Gemeinsamer Bus

b) Wie wird in einem Rechner auf Basis dieser Architektur ein Programm prinzipiell abgearbeitet (Takte)? (0,5 Punkte)

c) Nennen sie zwei Nachteile gegenüber einer parallelen Architektur wie der Harvard-Architektur. Gehen Sie dabei auch auf den *Von-Neumann-Flaschenhals* ein. (0,5 Punkte)

Aufgabe 1.2: Betriebssystem (1 Punkt) (Theorie[1])

Ein Thema in der Vorlesung sind Betriebssysteme.

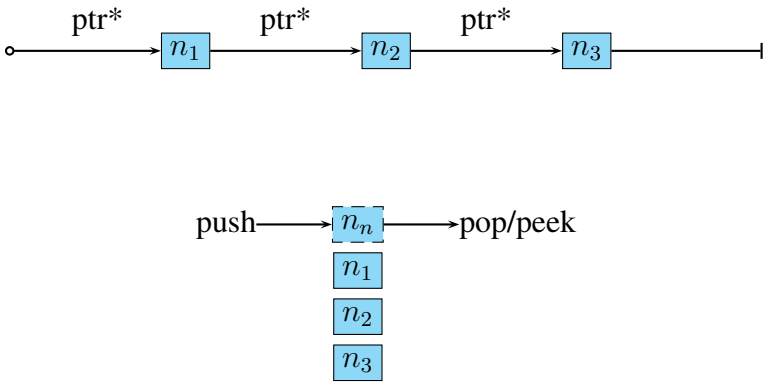
a) Was versteht man unter einem Betriebssystem? (0,5 Punkte)

b) Nennen Sie vier Aufgaben die ein Betriebssystem erfüllt. (0,5 Punkte)

Aufgabe 1.3: Einführung in C (Tafelübung)

Schreiben Sie ein Programm in C, welches die Fakultät einer Zahl n berechnet und ausgibt. Welche Stufen mit welchen Zwischenergebnissen werden durchlaufen, um aus dem Code ein ausführbares Programm zu erzeugen? Wie würde ein entsprechender, beispielhafter GCC-Aufruf lauten?

Aufgabe 1.4: Stack (Tafelübung)



Im folgenden soll die Last-In-First-Out Datenstruktur Stack mit den grundlegenden Funktionen

- **push** legt ein Element auf den Stack
- **pop** liefert und entfernt das oberste Element
- **peek** liefert das oberste Element ohne es zu entfernen

implementiert werden. Desweiteren werden noch die Funktionen **new** (erzeugen eines leeren Stacks) und **free** (löschen des Stacks) benötigt.

Aufgabe 1.5: Priority Queue (2 Punkte) (Praxis^2)

Realisieren Sie eine Priority Queue auf Basis einer einfach verketteten Liste. Die Elemente der Queue sollen hierbei absteigend nach ihrer Priorität innerhalb der Queue gespeichert werden. Die Queue soll dynamisch beim Einfügen von Elementen wachsen bzw. schrumpfen wenn das Element mit der höchsten Priorität entfernt wird.

Implementieren Sie alle in der Headerdatei *PrioQueue.h* vorgegebenen Funktionen in der Sourcedatei *PrioQueue.c*. Die Dateien sind wie folgt aufgebaut:

- *PrioQueue.h* definiert die von Ihnen zu implementierenden Funktionen. Diese Datei soll nicht geändert werden.
- *PrioQueue.c* Implementieren Sie hier die in der Header Datei definierten Funktionen. Verwenden Sie hierzu die structs *PrioQueue* (Priority Queue Container mit Zeiger auf das erste Element) und *q_elem* (Container für die Elemente innerhalb der Queue).

Hinweise:

- **Compiler Flags:** Falls Sie nicht das von uns vorgegebene makefile verwenden, ist es sinnvoll die Compiler flags *-Wall -Wextra -Werror* zu verwenden. Sie weisen den Compiler an mehr Warnungen auszugeben und diese auch als Fehler zu behandeln.

- **Dynamischer Speicher:** Um zu evaluieren ob nach dem Aufruf von *pqueue_free()* der gesamte von der Queue allokierte Speicher wieder freigegeben wurde, empfiehlt sich das Kommandozeilen Werkzeug **valgrind**[1] (unter linux über apt-get install valgrind).
- **apply Funktion:** Die Funktion *pqueue_apply(...)* ist bereits implementiert und muss von Ihnen nicht bearbeitet werden.

[1] http://valgrind.org/