**Aufgabe 3):**

Beim (wiederholten) Ausführen des Codes ohne Eingabeparameter fiel auf:

- Die zugewiesenen Adressen unterscheiden sich stets geringfügig. Ein Unterschied war zu erwarten - das Betriebssystem hat keinen Grund sich zu merken, welcher Prozess früher einmal welche Adresse hatte um diese erneut zu verwenden.

- Die zugewiesene Adresse des statischen Variable ist stets signifikant größer als die Adresse der dynamisch allozierten. Erklären lässt sich dies wohl dadurch, dass `malloc`-Aufrufe speicher auf dem Heap reservieren, während statisch initialisierte Variablen auf dem Stack zu finden sind.

- Der Wert der Pointervariable war stets 0 - dies hat uns geringfügig überrascht, da dieser nicht initialisiert wurde, und C dieses Verhalten nicht garantiert. Wir gehen daher davon aus, dass das Betriebssystem in andernfalls leeren Zyklen Speicher ausnullt, um unsicher geschriebene Programme zu schützen und die Performance von `calloc` zu verbessern.

Beim Ausführen des Codes mit Eingabeparameter fielen uns hauptsächlich Segfaults auf. Diese waren zu erwarten - ein Segfault ist schließlich in den meisten Fällen nichts anderes als eine Warnung, dass Speicher berührt wurde, der diesem Prozess nicht gehört. Daher scheint das schreiben eines vordefinierten oder auch nur eingegebenen Wertes in einen Pointer sehr sinnlos - uns fiel keine Situation ein, in der dies nicht zu einem Problem führen würde.

**Aufgabe 4):**

Die oberste Reihe beschreibt die Quele der aktuellen Handlung - Zahlen entsprechend der in der Aufgabenstellung beschriebenen Programmschritte, XX für eine Handlung des Prozessors - Für gewöhnlich das setzen des geeigneten `Runnable`-Prozesses in `Running`. Die Prozessgrafik zeigt nicht unmittelbar an, dass ein $B_i$ wahr gesetzt wurde - es ist nur indirekt beobachtbar, wenn dadurch ein anderer Prozess, der auf dieses $B_i$ gewartet hat, von `Blocked` in `Runnable` übergeht.

```
     xx xx 31 32 33 xx 11 12 13 xx 21 22 xx 14 xx 34 xx 41 42 xx 23 xx 35 36 xx 43
P1:  N  N  R  R  R  A  A  A  B  B  R  R  A  N  N  N  N  N  N  N  N  N  N  N  N  N
P2:  N  N  N  N  N  N  R  R  R  A  A  B  B  B  B  B  B  R  R  A  N  N  N  N  N  N
P3:  R  A  A  A  B  B  B  B  B  B  R  R  R  R  A  B  B  R  R  R  R  A  A  N  N  N
P4:  N  N  N  R  R  R  R  R  R  R  R  R  R  R  R  A  A  B  B  B  B  R  R  A  N
```

Betriebsysteme Wintersemester 2016 - Sergei Gorlatch
Übungsgruppe 1: Julian Bigge
Hasan Can
Kjeld Schmidt
Robin Schneevogt

**Aufgabe 5):**

a) $A < B$

b) $A < B \land B < A$

Dem geneigten Leser mag auffallen, dass in dieser Situation ein Deadlock unumgänglich ist.

c) $A < B \lor B < A$

In dieser SItuation entsteht kein Problem - zwar können A und B nicht gleichzeitig im Gange sein, jedoch ist die Reihenfolge irrelevant.

d) $A \leq B$

Hier ist zum ersten mal möglich, dass die beide Abschnitte parallel laufen. Welcher zuerst endet ist jedoch nicht zwingend klar - wenn ein äußerer Prozess A blockiert, so kann B zuerst terminen. (Der Prozess $P_A$ endet jedoch in jedem Fall vor $P_B$)

e) $A < B < C \quad \lor \quad A < C < B \quad \lor \quad B < A < C \quad \lor \quad B < C < A$

Hier ist vieles möglich - der Prozessor darf erst selbst wählen, ob er mit A oder B beginnt, und in zweiter Instanz, ob er mit dem Verbliebenen der ersten Runde oder C fortfährt.

f) $A \leq B \quad \lor \quad B \leq A$

Dies ist eine sehr schwache Aussage - eigentlich sogar nichtssagend. Die Abschnitte können in beliebiger Reihenfolge starten.  Wenn einer der beiden durch einen äußeren Prozess blockiert wird können sie auch parallel laufen und sogar in beliebiger Reihenfolge terminieren.