#uni/semester3/Betriebssysteme/chapter17/homework

1. First run with the flags -n 10 -H 0 -p BEST -s 0 to generate a few random allocations and frees. Can you predict what alloc()/free() will return? Can you guess the state of the free list after each request? What do you notice about the free list over time?
   ➜ What does alignment mean? malloc - What is aligned memory allocation? - Stack Overflow

```
./malloc.py -n 10 -H 0 -p BEST -s 0 -c
```

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [Size 1]: [addr:1003 sz:97]

Free(ptr[0]) returned 0
Free List [Size 2]: [addr:1000 sz:3] [addr:1003 sz:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [Size 2]: [addr:1000 sz:3] [addr:1008 sz:92]

Free(ptr[1]) returned 0
Free List [Size 3]: [addr:1000 sz:3] [addr:1003 sz:5] [addr:1008 sz:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [Size 3]: [addr:1000 sz:3] [addr:1003 sz:5] [addr:1016 sz:84]

Free(ptr[2]) returned 0
Free List [Size 4]: [addr:1000 sz:3] [addr:1003 sz:5] [addr:1008 sz:8] [addr:1016 sz:84]

ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [Size 3]: [addr:1000 sz:3] [addr:1003 sz:5] [addr:1016 sz:84]

Free(ptr[3]) returned 0
Free List [Size 4]: [addr:1000 sz:3] [addr:1003 sz:5] [addr:1008 sz:8] [addr:1016 sz:84]

ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [Size 4]: [addr:1002 sz:1] [addr:1003 sz:5] [addr:1008 sz:8] [addr:1016 sz:84]

ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [Size 4]: [addr:1002 sz:1] [addr:1003 sz:5] [addr:1015 sz:1] [addr:1016 sz:84]

➜ The free list grows bigger and more fragmented, because no merging happens.

2. How are the results different when using a WORST fit policy to search the free list (-p WORST)? What changes?

```
./malloc.py -n 10 -H 0 -p WORST -s 0 -c
```

```
ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:101
6 sz:84 ]

ptr[3] = Alloc(8)  returned 1016 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:102
4 sz:76 ]

Free(ptr[3]) returned 0
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:101
6 sz:8 ] [ addr:1024 sz:76 ]

ptr[4] = Alloc(2)  returned 1024 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:101
6 sz:8 ] [ addr:1026 sz:74 ]

ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:101
6 sz:8 ] [ addr:1033 sz:67 ]
```

➜ Es wird halt nach dem grössten chunk gesucht, wo der request noch rein passt. Das führt zur längeren Listen, da freigegebener Speicher nicht wieder verwendet werden kann, bis der grösste chunk ganz aufgeteilt ist.

3. What about when using FIRST fit (-p FIRST)? What speeds up when you use first fit?

```
./malloc.py -n 10 -H 0 -p FIRST -s 0 -c
```

➜ Das Suchen in der free list wird beschleunigt, da das erste Treffen genommen wird.

4. For the above questions, how the list is kept ordered can affect the time it takes to find a free location for some of the policies. Use the different free list orderings (~~-l ADDRSORT, -l SIZESORT+, -l SIZESORT~~) to see how the policies and the list orderings interact.

   FRAGEN warum wird nicht nach größe sortiert? Ich glaube dass er erst nach free Aufruf sortiert wird, d.h. nach alloc zeigt nur was zu was geworden ist. Und erst beim free es sortiert wird.

   ```
   ./malloc.py -n 10 -H 0 -p BEST -s 0 -l SIZESORT+ -c
   ```

   BEST: SIZESORT+ ist am besten, weil dann wird das erste entry ausgewählt, der den request erfüllen kann. Weil nach diesem entry gibts auf jeden Fall kein entry der besser passt.

   WORST: SIZESORT- ist am besten, weil dann wird das erste entry der Liste genommen, falls er den request erfüllen kann. O(1)

   FIRST: SIZESORT+ oder SIZESORT-
   Es kommt auf den workload an. Wenn request gross ist, dann ist SIZESORT- schneller. Wenn request klein ist, dann ist SIZESORT+ schneller. (Wobei SIZESORT- dann der Worst Effekt macht)

5. Coalescing of a free list can be quite important. Increase the number of random allocations (say to -n 1000). What happens to larger allocation requests over time? Run with and without coalescing (i.e., without and with the -C flag). What differences in outcome do you see? How big is the free list over time in each case? Does the ordering of the list matter in this case?

   ```
   ./malloc.py -n 1000 -H 0 -p FIRST -s 23 -c -C
   ```

   What happens to larger allocation requests over time?
   ➜ alloc() returns -1, weil kein entry in der free list den request erfüllen kann. Die free list hat eine sehr grosse Anzahl an entries, aber keins kann es erfüllen. Die free list ist somit fragmentiert.

What differences in outcome do you see?
➜ Mit -C werden wenigere pointer allokiert. Wahrscheinlich liegt es daran, dass wenn -1 returned wird, wird das nicht auf die 1000 gezählt.
➜ Free list wird kleiner mit -C

How big is the free list over time in each case?
➜ ohne -C, hat BESTFIT am Ende die kleinste free list. Macht auch Sinn da Best fit wenig fragmentierung produziert.
➜ mit -C, haben alle am Ende eine Liste der size 5

Does the ordering of the list matter in this case?
Ohne -C
➜ Bei Best fit, macht es nicht wirklich einen Unterschied
➜ Bei First fit, SIZESORT+ hat kleinste Liste
➜ Bei worst fit, macht es nicht einen Unterschied, schlecht überall

Mit -C
➜ Bei Best fit, ADDRSORT deutlich eine kleinere Liste am Schluss
➜ Bei first fit, ADDRSORT deutlich eine kleinere Liste
➜ Bei worst fit, ADDRSORT deutlich eine kleinere Liste

➜ für coallesce gibts verschiedene Implementierungen wie buddy und so. Oder man muss die freie Bereiche nach Adresse sortiert

6. What happens when you change the percent allocated fraction -P to higher than 50? What happens to allocations as it nears 100? What about as the percent nears 0?
   ➜ Größer 50 heisst, es wird mehr alloc als free geben
   ➜ In Richtung 100, wird die free list immer kleiner. Bei 100 wird die liste immer nur ein entry haben.
   ➜ In Richtung 0, wird versichert dass am Schluss jeder Pointer freigegeben wird. Free kann nur auftauchen, wenn ein alloc passiert. Das heisst, -P gegen 0 heisst nicht, gar kein alloc, sondern kein alloc ohne free gleich danach.
   Ausserdem werden die free list Größer, da es beinhaltet den Ganzen free Speicher, aber halt fragmentiert.

7. What kind of specific requests can you make to generate a highly fragmented free space?

Use the -A flag to create fragmented free lists, and see how different policies and options change the organization of the free list.

```
./malloc.py -c -A +20,+20,+20,+20,+20,-0,-1,-2,-3,-4
./malloc.py -c -A +20,+20,+20,+20,+20,-0,-1,-2,-3,-4 -C
-> -C korrigiert, und Spicher wird nicht mehr fragmentierrt
```

```
./malloc.py -c -A +10,+20,+30,+40,-0,-1,-2,-3 -l SIZESORT-
./malloc.py -c -A +10,+20,+30,+40,-0,-1,-2,-3 -l SIZESORT- -C

-> -C kann nicht korrigieren
```

```
./malloc.py -c -A +30,+10,+20,+40,-0,-1,-2,-3 -l SIZESORT-
./malloc.py -c -A +30,+10,+20,+40,-0,-1,-2,-3 -l SIZESORT- -C

-> -C kann korrigieren. -C verschmelzt zwei entries, nur wenn in der free list
diese benachbart sind, und die Adressen aufsteigend sortiert sind
-> Erkenntis: Um fragmentierung zu zwingen, müssen chuncks aufsteigen allokiert
werden.
```

```
./malloc.py -c -A +30,+10,+20,+40,-0,-1,-2,-3 -l SIZESORT+
./malloc.py -c -A +30,+10,+20,+40,-0,-1,-2,-3 -l SIZESORT+ -C

-> Mit SIZESORT+ korregiert -C wieder nicht
```

```
./malloc.py -c -A +10,-0,+20,-1,+30,-2,+40,-3 -p FIRST -l SIZESORT- -C
```

```
Es gibt schlimmer:

./malloc.py -c -A +40,-0,+30,-1,+20,-2,+10,-3 -p FIRST -l SIZESORT- -C
```

```
./malloc.py -c -A +40,-0,+30,-1,+20,-2,+10,-3 -p WORST -l SIZESORT-
-> -C korrigiert hier auch nicht
```

```
./malloc.py -c -A +10,-0,+20,-1,+30,-2,+40,-3 -p WORST -l SIZESORT+
$ ./malloc.py -c -A +10,-0,+20,-1,+30,-2,+40,-3 -p WORST -l SIZESORT+ -C
-> -C korrigiert

$ ./malloc.py -c -A +10,-0,+20,-1,+30,-2,+40,-3 -p WORST -l SIZESORT-
$ ./malloc.py -c -A +10,-0,+20,-1,+30,-2,+40,-3 -p WORST -l SIZESORT- -C
-> -C korrigiert nicht!
```

Siehe auch ostep-hw/17 at master · xxyzz/ostep-hw · GitHub