UNIVERSITY OF THESSALY

POLYTECHNICAL SCHOOL

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

«Modification of SLOB algorithm»

3nd Project

Operating Systems

Gianni Ioanna

Kapetanos Georgios

Zafeiri Stamatia Varvara

Supervisor: Christos Antonopoulos

2021-22

1. REQUIREMENTS

The core of the operating system has 3 different memory allocators: SLAB, SLUB and SLOB. The first 2 allocators are complex and used in real systems.

In this project we are going to work with SLOB(Simple List Of Blocks) where we are asked to modify this allocator so that less external fragmentation occurs. Specifically, we are asked to replace the algorithm First-Fit, which is the one SLOB uses by default, with the Best-Fit algorithm. Moreover, the creation of system calls is required to check the external fragmentation at each moment.

2. DESCRIPTION OF SLOB AND THE ALGORITHMS USED BY IT

a. SLOB

SLOB allocator requires less resources than the other allocators that the Linux core has. It has 3 lists containing the free pages that can be allocated to fulfill requests. First list serves requests less than 256 bytes, second list serves requests less than 1024 bytes and third one serves the rest.
For allocating pages Next-Fit algorithm, which is a First-Fit variant, is used while for allocating a block of memory First-Fit algorithm is used.

b. First-Fit

First Fit algorithm passes the list of blocks and selects the first one that has enough space to serve the request

c. Next-Fit

Next Fit algorithm is a variant of First Fit, it passes the list of pages(here) and selects the page that has enough space to serve the request. The difference with the first fit algorithm is that next time the search for a page will continue from where the last search ended in the list and won't start from the head of the list again

d. Best-Fit

Best Fit algorithm examines the list and selects the page/block that serves the request but also creates the least fragmentation

3. EVALUATION

While using the Best-Fit algorithm we notice that it takes longer for the system to boot. That's something we expected since Best-Fit algorithm iterates all lists and free pages to allocate memory which makes the system less efficient.

Best Fit:

```
kaptain@georgios-virtual:~$ systemd-analyze
Startup finished in 1min 27.089s (kernel) + 3min 44.843s (userspace) = 5min 11.932s
graphical.target reached after 3min 19.171s in userspace
```

First Fit:

```
kaptain@georgios-virtual:~$ systemd-analyze
Startup finished in 42.001s (kernel) + 1min 31.756s (userspace) = 2min 13.757s
graphical.target reached after 1min 27.593s in userspace
```

When Booting the system we can see the allocation happening using both algorithms.
Screenshots from the system boot:

Best Fit:
Allocating the block leaving the least fragmentation
Here it's an exact fit

```
    4.998427] slob_alloc: Request: 36
    4.998466] slob_alloc: Candidate blocks size: 16 156 4 128 16 108 16 28 16 1
28 28 24 36 24
    4.998468] slob_alloc: Best Fit: 36
```

First Fit:
Allocating the first block that serves the request
We can see that there are other blocks that could create less fragmentation
First available block is chosen

```
[   26.587155] slob_alloc: Request: 12
[   26.587199] slob_alloc: Candidate blocks size: 116 72 28 16 52 20 20 160 20 3
08
[   26.587201] slob_alloc: Chosen Fit: 116
```

In order to measure the memory allocated and also the unused memory during the allocation we created 2 system calls that print two static variables that calculate these sizes. Allocated memory is calculated in:
slob_new_pages() *by increasing the value* and ***slob_free_pages()*** *by decreasing the value*

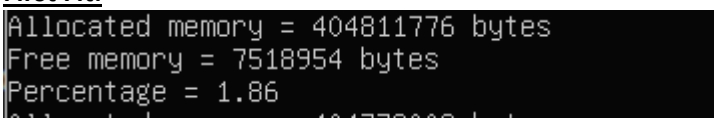
Unused memory is calculated in a helper function **calculate_unused_memory()** that iterates all 3 lists containing the free pages.
The function is later called in slob_page_alloc()

Using a simple user application to measure the allocation and the fragmentation

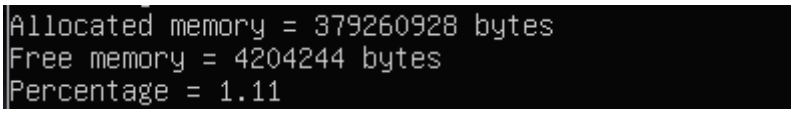
For our user application we chose to iteratively allocate and deallocate random memory chonks. Two extra system calls were used to call be able to call kmalloc and kfree.
By running the application, we notice that the level of fragmentation when using Best Fit is much lower than when we used First Fit

As we can see in the screenshots below, the percentage of fragmentation is 1.86% with First Fit while with Best Fit it goes down to 1.1%.
Generally for all times we run the experiment, the percentages were around those ranges.
Also as expected we can see that free memory with First Fit is higher than with Best Fit.

First Fit:



Best Fit:



4. CONCLUSION

Based on the above we conclude that with the usage of Best-Fit algorithm the external fragmentation is lower and there is higher efficiency when it comes to space management. However, the algorithm is way slower than the First Fit one.