3η Εργασία Λειτουργικών Συστημάτων
Φοιτητές: Γρηγόρης Ηλιάδης, Δημήτρης Κεφαλάς, Φοίβος Πουρναρόπουλος
ΑΕΜ: 2522, 2533, 2614
Διδάσκων: Εμμανουήλ Κουτσουμπέλιας
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

Στην 3η εργασία καλούμαστε να αλλάξουμε την υλοποίηση του SLOB allocator του πυρήνα του λειτουργικού συστήματος Linux. Πιο συγκεκριμένα έγινε τροποποίηση του διαχειριστή SLOB όσον αφορά τον αλγόριθμο δέσμευσης μνήμης από First Fit σε Best Fit, ενώ πραγματοποιήθηκε και τροποποίηση όσον αφορά τον αλγόριθμο επιλογής σελίδας για δέσμευση μνήμης από Next Fit σε Best Fit. Ο κύριος στόχος αυτών των τροποποιήσεων ήταν η μείωση του εξωτερικού κατακερματισμού κατά τη δέσμευση μνήμης.

Πιο συγκεκριμένα, έγιναν τροποποιήσεις στις συναρτήσεις slob_alloc και slob_page_alloc, όπου η πρώτη υλοποιεί την επιλογή σελίδας στην οποία θα γίνει η δέσμευση μνήμης, ενώ η δεύτερη την επιλογή του κατάλληλου slob της εκάστοτε σελίδας που θα δεσμευθεί.

Περιγραφή Υλοποίησης της slob_alloc:

Η ιδέα της υλοποίησης είναι να ταξινομήσουμε την λίστα των pages και στη συνέχεια να διατρέχουμε την ταξινομημένη λίστα. Κατά τη διαπέραση της λίστας, για όσα pages έχουν ελεύθερο χώρο μεγαλύτερο ή ίσο της αίτησης πραγματοποιούμε μία απόπειρα δέσμευσης ενός slob μέχρις ότου επιτύχει η δέσμευση ή έχουμε διαπεράσει όλη τη λίστα. Η ταξινόμηση γίνεται χρησιμοποιώντας την συνάρτηση list_sort που είναι υλοποιημένη στον πυρήνα του ΛΣ και πραγματοποιεί ταξινόμηση στη λίστα με βάση τον αλγόριθμο merge sort (πολυπλοκότητας: O(nlogn)). Για να πετύχουμε καλύτερους χρόνους εύρεσης των υποψήφιων pages της λίστας, επειδή είναι ταξινομημένη, δεν διαπερνάμε όλα τα στοιχεία της παρά μόνο εκείνα που έχουν ελεύθερο χώρο μεγαλύτερο ή ίσο της αίτησης. Έτσι θα έπρεπε να καθορίσουμε ποιο είναι το αριστερότερο στοιχείο της λίστας που μπορεί να ικανοποιήσει την αίτηση. Επειδή κατά την ταξινόμηση της λίστας με την list_sort καλείται η συνάρτηση cmp *(δίνεται ως παράμετρος στην list_sort)* κάθε φορά που γίνεται μία σύγκριση των δύο στοιχείων της λίστας, έχουμε ενσωματώσει ένα μέρος κώδικα στο σώμα της cmp προκειμένου να δίνουμε την τιμή σε ένα δείκτη limit, ο οποίος θα δείχνει σε μία σελίδα που έχει ελεύθερο χώρο πιο κοντά στην αίτηση μας. Επειδή μπορεί να υπάρχουν πολλές σελίδες με τον ίδιο ελεύθερο χώρο, μετακινούμε τον δείκτη limit στην αριστερότερη από αυτές. *Για παράδειγμα έστω η ταξινομημένη λίστα και request 3, (1 2 3 3 3 4 5 10 20), ο δείκτης limit μετά το τέλος της list_sort μπορεί να δείχνει σε ένα από τα χρωματισμένα στοιχεία της λίστας, εμείς όμως το μετακινούμε στο αριστερότερο δηλαδή στο κόκκινο.* Σε περίπτωση όπου ο δείκτης limit είναι NULL, γνωρίζουμε ότι δεν υπάρχει κάποια σελίδα στη λίστα μας με ελεύθερο μεγαλύτερο ή ίσο της αίτησης και έτσι αποφεύγουμε την άσκοπη διαπέραση της και δεσμεύουμε κατευθείαν μια καινούργια σελίδα. Σε κάθε άλλη περίπτωση διατρέχουμε τη λίστα από το limit έως το tail μέχρι να βρεθεί μία σελίδα που να γίνει η δέσμευση, εάν δεν βρεθεί μια καινούργια σελίδα δεσμεύεται.

Ο αλγόριθμος υλοποίησης της slob_alloc:
Global variables: limit, request_diff, request_size.

```
slob_alloc ( size ){
    page sp;
    list slob_list;
    slob b;

    slob_list = choose_free_list_according (size);
    request_diff = -1;
    request_size = size;
    limit = NULL;

    //Ταξινόμηση λίστας με βάση τον merge sort O(nlogn)
    //κατα αύξουσα σειρά.
    //Ο δείκτης limit δείχνει στο page το οποίο απέχει
    //λιγότερο απο το μέγεθος της αίτησης και έχει χώρο >=
    //της αίτησης. Ουσιαστικά δείχνει στο αριστερότερο
    //στοιχείο της λίστας που μπορεί να ικανοποιήσει την
    //αίτηση.
    list_sort(slob_list);

    if (limit != NULL){
        traverse_list_from(sp, limit, head){
            b = slob_page_alloc(sp, size);
            if (b != null){ //Successful allocation
                break;
            }
        }
    }

    if (b == NULL){
        allocate_new_page (size, slob_list);
    }
}
```

Οι καθολικές μεταβλητές request_diff και request_size χρησιμοποιούνται στην cmp.

Περιγραφή Υλοποίησης της slob_page_alloc:

Στην slob_page_alloc υλοποιείται επίσης ο αλγόριθμος best_fit. Ο αλγόριθμος αυτός ψάχνει ολόκληρη τη λίστα και επιλέγει το μικρότερο slob του page που μπορεί να ικανοποιήσει την αίτηση εφόσον αυτό υπάρχει. Σε περίπτωση όπου η αίτηση δεν μπορεί να ικανοποιηθεί από κανένα slob της λίστας, δηλαδή όταν όλα τα slob έχουν μέγεθος μικρότερο της αίτησης επιστρέφουμε null. Για βελτίωση της απόδοσης του συστήματος, σταματάμε την διαπέραση των slobs της σελίδας όταν ο εναπομείναντας ελεύθερος χώρος της είναι μικρότερος της αίτησης, καθώς δεν επιφέρει κάποιο αποτέλεσμα η διαπέραση των υπολοίπων slobs.

Αλγόριθμος υλοποίησης slob_page_alloc:

```
slob_page_alloc(sp, size){
    slob_t *best_fit = NULL, *best_fit_prev;
    int delta = 0, units = SLOB_UNITS(size), slob_free_space = sp->units;
    slobidx_t avail;

    traverse_list(sp, head){
        avail = slob_units(curr);
        if(avail >= units + delta){ //room enough for current node
            if(best_fit == NULL || free_space_units(curr) < free_space_units(best_fit)){
                best_fit = curr; //new best fit slob found
                best_fit_prev = prev_entry(best_fit);
                if(avail == units + delta){ //exact fit
                    break; //found best fit, exit
                }
            }
        }
        slob_free_space = slob_free_space - avail;
        if(slob_free_space < units || curr == last_node){
            break;
        }
    }

    if(best_fit != NULL){
        if(check_for_exact_fit() == true){
            unlink();
        }
        else{
            fragment();
        }
        sp->units -= units;
        if(!sp->units){
            clear_slob_page_free(sp);
        }
        return(curr);
    }
    else{
        return(NULL); //none
    }
```

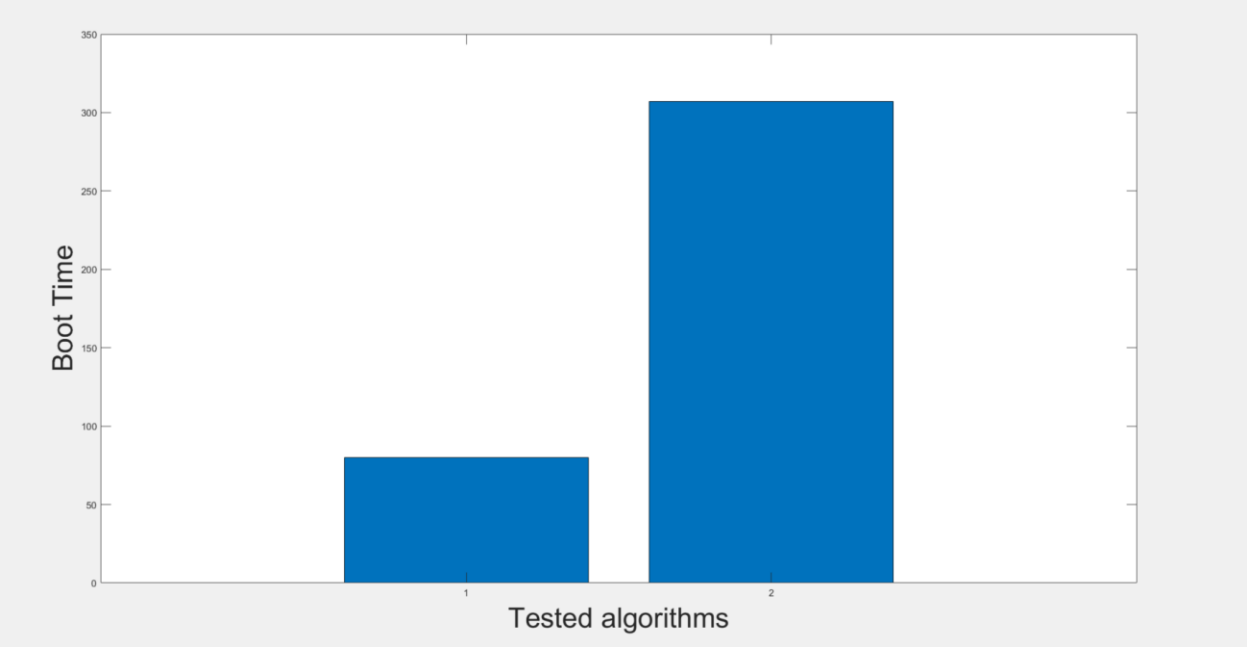
Παρατηρήσεις ως προς τον χρόνο εκκίνησης του συστήματος:

Από την επισκόπηση και των δύο αλγορίθμων περιμένουμε πιο αργή εκκίνηση του συστήματος που χρησιμοποιεί τον αλγόριθμο Best Fit. Αυτό συμβαίνει διότι γίνεται διαπέραση όλων των στοιχείων της λίστας τόσο για την επιλογή του “καλύτερου” page, όσο και για επιλογή του “καλύτερου” slob μέσα σε ένα page. Αντίθετα, στην περίπτωση που χρησιμοποιούμε τους αλγορίθμους Next Fit και First Fit, για την επιλογή του page όπου θα γίνει η δέσμευση και για την επιλογή του slob αντίστοιχα, περιμένουμε πιο γρήγορη εκκίνηση εφόσον και οι δύο αλγόριθμοι έχουν μικρότερο κόστος υλοποίησης.

Επιχειρώντας να εκκινήσουμε και τα δύο συστήματα παρατηρήθηκαν οι εξής χρόνοι:

#1: Next Fit, First Fit: 80sec.
https://youtu.be/GcWCJGkra1w

#2: Best Fit, Best Fit: 307sec.
https://www.youtube.com/watch?v=kHxBzgib2pE&feature=youtu.be



Ο χρόνος εκκίνησης του συστήματος είναι πολύ μεγαλύτερος για τον αλγόριθμο Best Fit. Παρατηρούμε αύξηση κατά 73,9%, διότι κατά την εκκίνηση του συστήματος όπου γίνονται πολλά memory allocations ο best fit επιβαρύνεται σημαντικά με αποτέλεσμα να είναι πιο αργός. Αυτό συμβαίνει επειδή διατρέχουμε ολόκληρη την λίστα των pages, σε αντίθεση με τον next fit όπου επιλέγουμε το επόμενο διαθέσιμο page. Έτσι, η απόδοση ως προς το χρόνο του best fit, ειδικά όταν υπάρχει μεγάλος αριθμός free pages, είναι κατά πολύ μικρότερη σε σχέση με εκείνη του Next Fit. Επιπρόσθετα κατά την δέσμευση ενός block σε ένα page ο first fit είναι πιο αποδοτικός καθώς δεν διατρέχει όλη την λίστα, εκτός εάν δεν υπάρχει κάποιο block που να μας εξυπηρετεί, περίπτωση στην οποία έχουν παρόμοια συμπεριφορά.

Εκτύπωση στοιχείων εκτέλεσης αλγορίθμων:

Προκειμένου να δείξουμε τη σωστή λειτουργία της slob_page_alloc παραθέτουμε τα εξής στιγμιότυπα του log file.

α) Διακοπή αναζήτησης εξαιτίας εύρεσης block με το ακριβές μέγεθος της αίτησης. -Επιτυχής εύρεση best_fit slob.

```
slob_alloc: Request: 28
slob_alloc: Candidate blocks size:  20 4 3 6 2 4 2 14 2 2 2 4 3 4 28
slob_alloc: Best Fit: 28
```

β)Διαπέραση της λίστας έως ότου ο εναπομείναντας ελεύθερος χώρος του page (λίστας των slob) είναι μεγαλύτερος της αίτησης. -Επιτυχής εύρεση best_fit slob.

```
slob_alloc: Request: 96
slob_alloc: Candidate blocks size:  16 2 16 14 6 10 12 8 28 116 8 6 14 16 58 8 24 8
slob_alloc: Best Fit: 116
```

γ) Διαπέραση της λίστας έως ότου ο εναπομείναντας ελεύθερος χώρος του page (λίστας των slob) είναι μεγαλύτερος της αίτησης. -Ανεπιτυχής εύρεση best_fit slob.

```
slob_alloc: Request: 64
slob_alloc: Candidate blocks size:  48 2 2 2 2 2 2 50 2 16 2 2 50 2 2 50 50 2 2 50 2 48 50 50 2 2 2 66 2 48 50 2 48
slob_alloc: Best Fit: N0NE
```

Λήψη στατιστικών στοιχείων:

Για την λήψη των στατιστικών δημιουργήσαμε ένα scriptaki προκειμένου να τρέχουμε έναν ικανό αριθμό επαναλήψεων (100 επαναλήψεις), και να υπολογίζουμε το μέσο όρο της συνολικής δεσμευμένης και ελεύθερης μνήμης.

Script_code:

```
#Executing app 100 times.
for i in {1..100}
        do ./app
done > out_$1

mem_alloc=0
free_mem=0
compare="Memory allocated"

#Traversing through file and computing sum of
#allocated and free memory.
while IFS=:  read -r string value
do
        if  [[ "$string" == "$compare" ]]
        then
                #echo "$value"
                mem_alloc=$((mem_alloc+value))
        else

                free_mem=$((free_mem+value))

        fi
done < out_$1

#Calculate average allocated and free memory.
mem_alloc=$((mem_alloc/100))
free_mem=$((free_mem/100))

echo "mem_alloc = $mem_alloc"
echo "free_mem = $free_mem"
```

Best_fit results:

```
user@os:~$ ./script.sh
mem_alloc = 1800512
free_mem = 890189
```

Next_fit results:

```
user@os:~$ ./script.sh
mem_alloc = 1781928
free_mem = 1150095
```

Από τα αποτελέσματα των πειραμάτων, παρατηρούμε ότι για παρόμοιες τιμές δεσμευμένης μνήμης η ελεύθερη μνήμη του συστήματος είναι σημαντικά μεγαλύτερη στο Next Fit από ότι στο Best Fit. Πιο συγκεκριμένα, παρατηρούμε αύξηση περίπου 22% της ελεύθερης μνήμης του συστήματος που υλοποιείται με βάση το First Fit έναντι του Best Fit. Αυτό συμβαίνει επειδή ο Best Fit διαπερνάει όλες τις διαθέσιμες σελίδες με σκοπό να ελαχιστοποιήσει τον κατακερματισμό του συστήματος επιλέγοντας να δεσμεύσει μνήμη από την σελίδα ώστε να δημιουργήσει το μικρότερο υπόλοιπο μεταξύ των ελεύθερων σελίδων και στη συνέχεια δεσμεύοντας το block που θα δημιουργήσει το μικρότερο υπόλοιπο μεταξύ των υποψήφιων block της επιλεγμένης σελίδας. Με αυτό τον τρόπο αξιοποιούμε καλύτερα τον ελεύθερο χώρο της κάθε σελίδας, όμως δημιουργούνται μικρότερες οπές σε σχέση με τον αλγόριθμο First Fit, οι οποίες είναι δύσκολο να δεσμευτούν στο μέλλον «πριονίδι». Αντιθέτως ο First Fit δεν αξιοποιεί το ίδιο καλά τον ελεύθερο χώρο της κάθε σελίδας, γεγονός που οδηγεί στην δέσμευση περισσότερων σελίδων κάτι όμως που βελτιώνει την ταχύτητα του. Παρόλα αυτά, ο συνδυασμός του Next Fit για την επιλογή σελίδας και του First Fit για την επιλογή του κατάλληλου block στην σελίδα, έχει σημαντικά πιο γρήγορη ταχύτητα αφού δεν διατρέχουν κάθε φόρα όλη τη λίστα των διαθέσιμων σελίδων και block αντίστοιχα.

Η σημαντική διαφορά της ελεύθερης μνήμης του συστήματος για παρόμοιες τιμές δεσμευμένης μνήμης είναι εμφανής και στο παρακάτω διάγραμμα, όπου το πράσινο γράφημα αντιστοιχεί στον αλγόριθμο First Fit, ενώ το καφέ στον Best Fit.

Για την κατασκευή του διαγράμματος πραγματοποιήθηκε δειγματοληψία των τιμών της δεσμευμένης και της ελεύθερης μνήμης του συστήματος κατά το τρέξιμο του script για την λήψη του μέσου όρου τους.