Χρήστος-Παναγιώτης Παλαμίδας
115201800140

- ***int ref_count[PHYSTOP/PGSIZE]*** (~>kalloc.c)
  - Ξεκινώντας το CoW optimization θα ήθελα να αναφερθώ στον ορισμό ενός array μεγέθους PHYSTOP/PGSIZE, όπως είναι ορισμένα riscv. Κάθε θέση του πίνακα υποδεικνύει πόσα processes αναφέρονται σε μια συγκεκριμένη σελίδα. Η χρήση του θα επεξηγηθεί στις παρακάτω συναρτήσεις.
  - *Σημείωση:* Για τη μοντελοποίηση του reference counter αποφάσισα να είναι απλά ένας πίνακας και όχι ένα struct με δικό του 'spinlock' μιας και στο φροντιστήριο αναφέρθηκε πως δεν είναι αναγκαίο. Αν έπρεπε όμως να το υλοποιήσω έτσι για τη σωστή πρακτική θα το έκανα ως εξης:

```
struct {
    struct spinlock lock;
    int ref_count[PHYSTOP/PGSIZE];
} reference_counter;
```

    Και θα έκανα στην kinit() initialize το spinlock.
- ***freerange()*** (~>kalloc.c)
  - Η συνάρτηση freerange() χρησιμοποιείται για το initialize της μνήμης. Λόγω μιας τροποποίησης στην *kfree()* που θα αναλυθεί παρακάτω, μέσα στο for loop της freerange() μηδενίζω το reference counter κάθε σελίδας και στη συνέχεια καλώ την kfree() (βλ. kfree() για τα περαιτέρω.)
- ***kfree()*** (~>kalloc.c)
  - Η kfree() είναι μια συνάρτηση που αποδεσμεύει σελίδες και έχει υποστεί μια τροποποίηση. Αντι να αποδεσμεύει αμέσως μια σελίδα, πρώτα μειώνει τον reference counter κατα 1 μονάδα (αφού πρώτα κάνει check ότι είναι μεγαλύτερος του 0✱) και στη συνέχεια ελέγχει αν οι αναφορές προς τη σελίδα είναι 0. Αν όχι η συνάρτηση τερματίζει και συνεχίζεται η εκτέλεση του προγράμματος. Αν είναι όμως, τότε και μόνο τότε, η kfree αποδεσμεύει τη σελίδα.
  - ✱Αυτός ο έλεγχος είναι και ο λόγος που μηδενίζω τον reference counter στη freerange. Όπως μας λέει και το documentation, κατά το initialization της μνήμης του συστήματος καλείται η kfree (μεσω της freerange) ακόμα και σε σελίδες που δεν έχουν αρχικοποιηθεί με την *kalloc()* (όπως θα δούμε παρακάτω). Θέτοντας λοιπόν στην freerange() τον reference counter κάθε σελίδας σε '0' και καλώντας αμέσως μετά την kfree(), καταφέρνουμε έτσι να αποδεσμεύσουμε όλες τις σελίδες.
  - *Σημείωση:* Στην αρχή, προσπάθησα να φτιάξω μια εξωτερική συνάρτηση που θα μείωνε τον reference counter και αυτή θα καλούσε την kfree() στην περίπτωση που γινόταν '0'. Το θέμα είναι όμως ότι θα έπρεπε σε κάποια σημεία να αντικαταστήσω την κλήση της kfree() με αυτή την εξωτερική συνάρτηση και σε κάποια άλλα σημεία όχι. Για την αποφυγή τυχόν conflicts αποφάσισα να τροποποιήσω την kfree() κάνοντας κατάλληλο error checking.

- ***kalloc()*** (~>kalloc.c)
    - Η συνάρτηση αυτή κάνει allocate μια σελίδα στην φυσική μνήμη και θα δεσμευτεί από κάποιο process. Για αυτό το λόγο έκανα την εξης προσθήκη. Μόλις βρεθεί και γίνει allocate η σελίδα, θέτω τον reference counter της σε '1' για να δηλώσω ότι κάποιο process αναφέρεται σε αυτή.
- ***increase_ref_counter()*** (~>kalloc.c)
    - Είναι μια εξωτερική συνάρτηση που υλοποίησα, η οποία παίρνει ως όρισμα μια σελίδα και αυξάνει το reference counter της κατά μια μονάδα, αφού κάνει το κατάλληλο error checking.
- ***usertrap()*** (~>trap.c)
    - Στη συνάρτηση αυτή έχω προσθέσει την περίπτωση το r_scause() να επιστρέψει την τιμή '15' η οποία υποδηλώνει ότι έχει υπάρξει ένα page fault πάνω σε read-only σελίδα. Αν ισχύει αυτή η συνθήκη τότε καλώ την συνάρτηση *cowfault()* που θα αναλύσω παρακάτω και η οποία ελέγχει αν το page fault οφείλεται σε Cow Fault. Ο λόγος που κάνω αυτό το έλεγχο είναι γιατί υπάρχουν διάφορα faults για read-only σελίδες, όπως πχ ότι ένα process πάει να γράψει σε κάποια guard σελίδα που δεν πρέπει. Αν προκύψει κάποιο error κατά τη διαδικασία αντιμετώπισης, τότε "σκοτώνουμε" το process.
- ***cowfault()*** (~>vm.c)
    - Η συνάρτηση αυτή όπως προαναφέρθηκε, ελέγχει το read-only πρόβλημα και κάνει τις κατάλληλες διαδικασίες αντιμετώπισης του οι οποίες είναι οι παρακάτω. Αφού γίνει κατάλληλο error checking για την επάρκεια της μνήμης για την εγκυρότητα του pte αλλά και για το αν είναι User & Valid, τότε παίρνουμε την σελίδα που μας ενδιαφέρει, δημιουργούμε μια νέα μέσω της kalloc() και αντιγράφουμε τα στοιχεία της παλιάς στην καινούργια, δίνοντας τα κατάλληλα flags (User, Valid, Read, Write, Execute). Στη συνέχεια μειώνουμε το reference counter της παλιάς σελίδας κατά μια μονάδα και τερματίζει η συνάρτηση με τιμή '0'. Αν υπήρχε το οποιοδήποτε error θα επέστρεφε '-1'.
- ***uvmcopy()*** (~>vm.c)
    - Η συνάρτηση αυτή, έπαιρνε το page table ενός parent process και το έκανε copy στο child process (έκανε copy τόσο το page table όσο και το physical memory όπως αναφέρεται στο documentation). Για το CoW optimization όμως, αυτό που θέλουμε να πετύχουμε είναι να μην δημιουργούνται νέες σελίδες για ένα child process, αλλά να "κοιτάνε" αυτές του parent process. Για να την υλοποίηση αυτού, αφαιρώ το κομμάτι που κάνει allocate και copy και το αντικαθιστώ με την εξης λειτουργία: κάνω τη σελίδα read-only ώστε να μη μπορεί να γράψει το νέο process πάνω της και αυξάνω το reference counter της. Στη συνέχεια το νέο process θα "κοιτάει" τις σελίδες του parent process χωρίς όμως τη δυνατότητα να μπορεί να τις τροποποιήσει.
- ***copyout()*** (~>vm.c)
    - Η λειτουργία που προσθέτουμε σε αυτή τη συνάρτηση είναι ότι ελέγχει αν είναι Cow σελίδα και αν ναί ακολουθείται η ίδια διαδικασία που προαναφέρθηκε στην *cowfault()*. Αυτό συμβαίνει για να αποφευχθούν τυχόν overwrites η conflicts στις σελίδες του parent και των children processes.