Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τομέας Ηλεκτρονικής και Υπολογιστών

7ο Εξάμηνο
Χειμώνας 2016

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Εργαστήριο - Άσκηση 5 Διαχείριση Μνήμης

Εισαγωγή

Στις σύγχρονες αρχιτεκτονικές υπολογιστών η μνήμη είναι οργανωμένη ιεραρχικά. Αρχίζοντας από την ταχύτερη: καταχωρητές, κρυφή μνήμη, κύρια μνήμη και δευτερεύουσα μνήμη (π. χ. σκληροί δίσκοι). Το τμήμα εκείνο του ΛΣ που καλείται διαχειριστής μνήμης συντονίζει τη χρήση των διαφόρων τύπων μνήμης, καταγράφοντας ποια τμήματά τους είναι διαθέσιμα, ποια είναι δεσμευμένα και, αναλόγως με τις απαιτήσεις των διεργασιών, εκχωρεί ή απελευθερώνει τμήματα για να τα χρησιμοποιήσουν οι τελευταίες. Αυτή η δραστηριότητα λέγεται διαχείριση εικονικής μνήμης, αφού η συνολική μνήμη που είναι σε θέση να αξιοποιήσουν οι διεργασίες κατά την εκτέλεσή τους μπορεί να υπερβαίνει το μέγεθος της κύριας μνήμης (της φυσικής μνήμης RAM), μέσω της δέσμευσης ενός τμήματος του σκληρού δίσκου από τον πυρήνα το οποίο χρησιμοποιείται από τον τελευταίο σαν επέκταση της κύριας μνήμης. Τα προηγμένα Λ.Σ. αποφεύγουν, όπου και όταν είναι δυνατό, τη χρήση αυτής της τεχνικής, επειδή η χρήση δευτερεύουσας μνήμης ως κύριας μειώνει την ταχύτητα του συστήματος.

Στόχος του εργαστηρίου είναι η μελέτη της κατανομής της μνήμης μιας διεργασίας. Δεδομένης της πολύ-προγραμματιστικής / πολύ-χρηστικής φύσης των Unix/Linux ΛΣ, η ύπαρξη εκατοντάδων ταυτόχρονων διεργασιών είναι πολύ πιθανή, κάθε μια από τις οποίες απαιτεί πόρους σε μνήμη. Το ΛΣ πρέπει να κατανείμει τις σελίδες της μνήμης με δίκαιο και ομαλό τρόπο. Μέθοδοι όπως η απαίτηση σελίδας (demand paging), ο διαμοιρασμός σελίδας (page sharing) και η ελάχιστα χρησιμοποιούμενη σελίδα – θύμα (Least Recently Used victim page) συχνά χρησιμοποιούνται στη διαχείριση της μνήμης.

Χρήσιμες αναφορές

Δυναμική διάθεση μνήμης - malloc

Η συνάρτηση malloc() μας επιτρέπει να ζητήσουμε έναν ελεύθερο χώρο μνήμης για να χρησιμοποιήσουμε στο πρόγραμμά μας.Η malloc έχει ως όρισμα το μέγεθος της μνήμης που απαιτούμε και επιστρέφει ένα δείκτη στην αρχή του αντίστοιχου χώρου. Ο τύπος της τιμής που επιστρέφει δεν είναι αντίστοιχος με τον τύπο που χρειαζόμαστε και γι' αυτό πρέπει να χρησιμοποιήσουμε την αντίστοιχη μετατροπή (π.χ. (double *)malloc...).Σε περίπτωση που το σύστημα δεν έχει άλλη διαθέσιμη μνήμη η malloc επιστρέφει τη σταθερά NULL.

Παράδειγμα:

```
char *c;

        c = (char *)malloc(1000 * sizeof(char));
        if (c == NULL) {
                printf("Out of memory!\n");
```

```
                return (1);
        }
```

Με τη χρήση της malloc μπορούμε να δημιουργήσουμε δυναμικά χώρο για πίνακες τους οποίους δηλώνουμε ως δείκτες. Το παρακάτω παράδειγμα ζητάει από το χρήστη να ορίσει το μέγεθος ενός πίνακα και στη συνέχεια να δώσει τα στοιχεία του.

```
#include <stdlib.h>
#include <stdio.h>

main()
{
        int *int_table;         /* Table of integers */
        int nelem;              /* Number of elements */
        int i;

        printf("Enter number of elements:");
        scanf("%d", &nelem);
        int_table = (int *)malloc(nelem * sizeof(int));
        for (i = 0; i < nelem; i++) {
                printf("Element %d=", i);
                scanf("%d", &int_table[i]);
        }
}
```

Η συνάρτηση free - Η συνάρτηση realloc

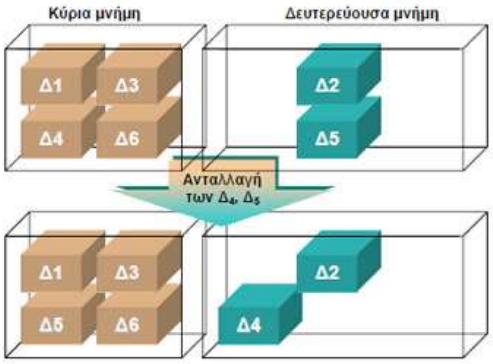
Η συνάρτηση free(δείκτης) ελευθερώνει το χώρο που έχει επιστραφεί από τη malloc για ανακύκλωση. Η συνάρτηση realloc(δείκτης, νέο μέγεθος) λαμβάνει ως όρισμα ένα δείκτη που έχει επιστραφεί από τη malloc και επιστρέφει έναν δείκτη σε χώρο με το νέο μέγεθος που έχει προσδιοριστεί.

Παράδειγμα:

```
 int *iptr;

        iptr = (int *)malloc(10 * sizeof(int));
        /* iptr[0] to iptr[9] can now be used */

        iptr = (int *)realloc(iptr, 20 * sizeof(int));
        /*
         * iptr[0] to iptr[9] retain old values
         * iptr[0] to iptr[19] can now be used */
        */

        free(iptr);
        /* Do not access iptr[0] to iptr[19] from this point onwards */
```

Ανταλλαγή

Η δευτερεύουσα μνήμη είναι πολύ πιο αργή από την κύρια (π.χ. ένας σκληρός δίσκος είναι εκατό χιλιάδες φορές πιο αργός από την κύρια μνήμη)· συγχρόνως όμως είναι και αρκετά πιο φθηνή. Έτσι, οι περισσότεροι υπολογιστές διαθέτουν πολύ περισσότερο χώρο αποθήκευσης σε δευτερεύουσα μνήμη παρά σε κύρια (π.χ. εκατό φορές περισσότερο).



Όταν ένας υπολογιστής εξυπηρετεί ένα μεγάλο αριθμό διεργασιών, τα προγράμματα και τα δεδομένα όλων δε χωρούν στην κύρια μνήμη. Τότε επιστρατεύεται η δευτερεύουσα μνήμη για να βοηθήσει: τα προγράμματα και τα δεδομένα ορισμένων διεργασιών κρατούνται στη δευτερεύουσα μνήμη, και κάθε φορά που είναι η σειρά μιας τέτοιας διεργασίας να εκτελεστεί, τότε μόνο φορτώνονται στην κύρια μνήμη. Για να απελευθερωθεί όμως χώρος στην κύρια μνήμη για αυτά, πρέπει κάποια άλλη διεργασία να μεταφερθεί με τη σειρά της στη δευτερεύουσα μνήμη.

Η διαδικασία αυτή ονομάζεται ανταλλαγή (swapping) και δημιουργεί σοβαρές χρονικές καθυστερήσεις στην εκτέλεση των διεργασιών, εξαιτίας της συνεχούς λειτουργίας της δευτερεύουσας μνήμης. Ο χρόνος που απαιτεί μια διεργασία για να εκτελεστεί είναι το άθροισμα του πραγματικού χρόνου εκτέλεσής της στην ΚΜΕ, του χρόνου που απαιτήθηκε για τις μεταφορές της μεταξύ κύριας και βοηθητικής μνήμης και του χρόνου για Ε/Ε. Για να είναι η λειτουργία του υπολογιστή αποδοτική απαιτείται πιο πολύπλοκη οργάνωση της ανταλλαγής δεδομένων μεταξύ κύριας και δευτερεύουσας μνήμης.

Εικονική μνήμη

Η κεντρική μνήμη του υπολογιστή αποτελείται από διαδοχικές θέσεις, με μέγεθος η κάθε μια π.χ. 16 ή 32 bits, που ονομάζονται λέξεις (words). Κάθε λέξη της μνήμης έχει τη δική της διεύθυνση, με την οποία αναφέρονται τα προγράμματα σε αυτή. Αν η μνήμη έχει μέγεθος N λέξεων, τότε η πρώτη λέξη έχει διεύθυνση 0, η δεύτερη έχει διεύθυνση 1 κ.ο.κ., ενώ η τελευταία λέξη έχει διεύθυνση N-1. Το σύνολο αυτών των N διευθύνσεων είναι σταθερό και ονομάζεται χώρος φυσικών διευθύνσεων (physical address space) ή χώρος απολύτων διευθύνσεων (absolute address space) ή χώρος πραγματικών διευθύνσεων (real address space).

Κάθε πρόγραμμα πρέπει να χρησιμοποιήσει διευθύνσεις για να αναφερθεί στα δεδομένα του, τα οποία βρίσκονται στη μνήμη. Τι γίνεται όμως αν οι διευθύνσεις των δεδομένων για δυο διεργασίες συμπίπτουν;

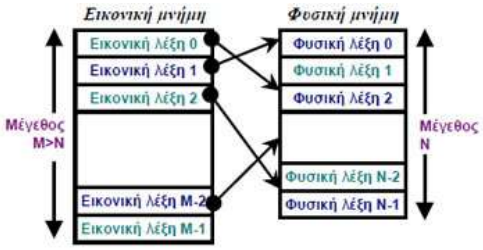
Εικονικές διευθύνσεις

Η πρώτη και πιο απλή σκέψη που μπορεί να γίνει για να λυθεί το πρόβλημα της επικάλυψης των διευθύνσεων είναι να γράφονται έτσι τα προγράμματα ώστε καθένα να χρησιμοποιεί διαφορετικές διευθύνσεις μνήμης. Με τον τρόπο αυτό δεν υπάρχει περίπτωση η μια διεργασία να επηρεάσει την άλλη. Αυτή η λύση όμως δεν είναι στην πράξη και πολύ αποτελεσματική για δυο λόγους:

- Τα προγράμματα γράφονται συνήθως ανεξάρτητα από διάφορους ανθρώπους, και δεν είναι δυνατό να βρεθεί ένας τρόπος συνεννόησης και συντονισμού τους.
- Το πλήθος των διαθέσιμων διευθύνσεων κύριας μνήμης είναι πεπερασμένο και μάλλον μικρό, οπότε αρκετά γρήγορα θα εξαντληθούν.

Θέλουμε λοιπόν μια λύση που να επιτρέπει στα προγράμματα να χρησιμοποιούν ελεύθερα οποιεσδήποτε διευθύνσεις, και να εξασφαλίζεται από το ΛΣ ότι δε θα υπάρχει επικάλυψη μεταξύ τους. Η λύση αυτή είναι εκείνη των *εικονικών διευθύνσεων* (virtual addresses): κάθε πρόγραμμα μεταφράζεται σε γλώσσα μηχανής σαν να έχει όλο το χώρο διευθύνσεων στη διάθεσή του, και αναφέρεται στα δεδομένα του χρησιμοποιώντας εικονικές -δηλαδή όχι πραγματικές- διευθύνσεις· όταν το πρόγραμμα φορτώνεται για εκτέλεση, το ΛΣ επιλέγει θέσεις μνήμης που είναι ελεύθερες και τις αντιστοιχίζει στις εικονικές διευθύνσεις.

Κατά την εκτέλεση του προγράμματος, κάθε αναφορά σε μια εικονική διεύθυνση μεταφράζεται από το ΛΣ στην αντίστοιχη φυσική διεύθυνση, η οποία τελικά προσπελάζεται. Η διαδικασία αυτή της μετάφρασης εικονικής διεύθυνσης σε φυσική συνήθως γίνεται πολύ γρήγορα, γιατί οι ΚΜΕ περιέχουν ειδικά κυκλώματα για αυτή.



Ο *χώρος εικονικών διευθύνσεων* (virtual address space), που πολλές φορές αναφέρεται και ως *εικονική μνήμη* (virtual memory), έχει συνήθως μεγαλύτερο μέγεθος από το χώρο φυσικών διευθύνσεων για να έχουν οι διεργασίες περισσότερο «χώρο μνήμης» στη διάθεσή τους. Οι εικονικές διευθύνσεις που δεν έχουν αντίστοιχες στη φυσική μνήμη, συνήθως αντιστοιχίζονται σε κάποια διεύθυνση της δευτερεύουσας μνήμης. Με τον τρόπο αυτό η δευτερεύουσα μνήμη συμβάλλει στην αύξηση της διαθέσιμης κύριας μνήμης του συστήματος.

Ο τρόπος αντιστοίχισης των διευθύνσεων για διαφορετικά προγράμματα μπορεί να γίνει με διάφορους τρόπους, δηλαδή με διάφορες στρατηγικές διαχείρισης εικονικής μνήμης. Οι βασικές στρατηγικές είναι δυο, η *σελιδοποίηση* (paging) και η *κατάτμηση* (segmentation), και υπάρχει ένας συνδυασμός των δύο, η *κατατμημένη σελιδοποίηση* (segmented paging). Αυτές οι στρατηγικές έχουν την ιδιότητα να επιτρέπουν πολλές διεργασίες να χρησιμοποιούν από κοινού μνήμη, όπως π.χ. διεργασίες που εκτελούν ταυτόχρονα το ίδιο πρόγραμμα χρησιμοποιούν από κοινού τη μνήμη στην οποία είναι αυτό αποθηκευμένο.

Διαχείριση εικονικής μνήμης με σελιδοποίηση

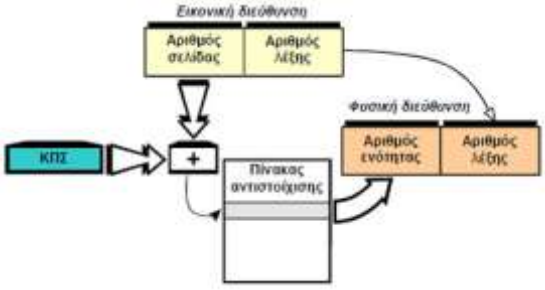
Στη μέθοδο της σελιδοποίησης, η εικονική μνήμη διαιρείται σε ίσα και συνεχόμενα μέρη, τα οποία ονομάζονται *σελίδες* (pages). Με τον ίδιο τρόπο διαιρείται και η φυσική μνήμη σε *ενότητες* (blocks, frames). Το μέγεθος της ενότητας είναι ίδιο με αυτό της σελίδας· έτσι μια σελίδα της εικονικής μνήμης και όλες οι διευθύνσεις που αυτή περιέχει αντιστοιχούν ακριβώς σε μια ενότητα της φυσικής μνήμης και τις διευθύνσεις της.

Το ΛΣ κρατά ένα πίνακα αντιστοίχισης σελίδων και ενοτήτων. Για κάθε σελίδα της εικονικής μνήμης φαίνεται στον πίνακα αυτό η ενότητα της φυσικής μνήμης στην οποία αντιστοιχεί ή, αν δεν αντιστοιχεί σε κάποια ενότητα, σημειώνεται ότι η σελίδα βρίσκεται στη δευτερεύουσα μνήμη.

Για να μεταφραστεί μια εικονική διεύθυνση (ΕΔ), πρώτα βρίσκεται η σελίδα (Σ) στην οποία ανήκει. Το ΛΣ αναζητά τη σελίδα αυτή στον πίνακα αντιστοίχισης, και βρίσκει ότι αντιστοιχεί στην ενότητα Ε. Η φυσική διεύθυνση (ΦΔ) θα ανήκει στην ενότητα αυτή, και θα βρίσκεται σε αντίστοιχη θέση με αυτή της ΕΔ μέσα στη σελίδα Σ.

Κάθε εικονική διεύθυνση σε έναν υπολογιστή χωρίζεται σε δύο τμήματα: το πρώτο είναι ο αριθμός της σελίδας και το δεύτερο ο αριθμός της λέξης μέσα στη σελίδα. Στο παράδειγμά μας, κάθε τέτοιο τμήμα ήταν ένα δεκαδικό ψηφίο· στους υπολογιστές κάθε τμήμα της εικονικής διεύθυνσης αποτελείται από πολλά ψηφία, αφενός γιατί χρησιμο- ποιείται το δυαδικό σύστημα, αφετέρου γιατί το μέγεθος της εικονικής μνήμης είναι μεγάλο. Αντίστοιχα και κάθε φυσική διεύθυνση χωρίζεται στον αριθμό ενότητας και τον αριθμό λέξης. Όταν μεταφράζεται μια εικονική διεύθυνση σε φυσική, ο αριθμός λέξης μένει ο ίδιος.

Ας δούμε πώς μεταφράζεται μια εικονική διεύθυνση σε φυσική:



Ο πίνακας αντιστοίχισης της διεργασίας βρίσκεται στην κύρια μνήμη, και η διεύθυνση όπου αρχίζει κρατείται σε μια μονάδα αποθήκευσης της ΚΜΕ που ονομάζεται *καταχωρητής πίνακα σελίδων* (ΚΠΣ, page table register). Κάθε «θέση» του πίνακα σελίδων κρατά τον αριθμό ενότητας για την αντίστοιχη σελίδα. Έτσι, αν στην αρχική διεύθυνση του πίνακα σελίδων προσθέσουμε ένα αριθμό σελίδας, θα πάρουμε τη θέση του πίνακα που αντιστοιχεί στη σελίδα αυτή, και θα βρούμε τον αντίστοιχο αριθμό ενότητας για τη σελίδα.

Για να βρούμε λοιπόν τον αριθμό ενότητας για μια εικονική διεύθυνση προσθέτουμε τον αριθμό σελίδας της με το περιεχόμενο του καταχωρητή πίνακα σελίδων. Το αποτέλεσμα είναι μια διεύθυνση μνήμης, και στη διεύθυνση αυτή θα βρούμε τον αριθμό ενότητας στη φυσική μνήμη. Αυτός ο αριθμός ενότητας συνδυάζεται με τον αριθμό λέξης για να δώσουν την τελική φυσική διεύθυνση.

Αν για μια σελίδα δεν υπάρχει αντίστοιχη ενότητα στην κύρια μνήμη, τότε πρέπει να ενεργοποιηθεί μια διαδικασία μεταφοράς της ενότητας αυτής από τη δευτερεύουσα στην κύρια μνήμη, ενώ συγχρόνως κάποια άλλη σελίδα μεταφέρεται από την κύρια στη δευτερεύουσα μνήμη για να απελευθερώσει την ενότητα που χρειάζεται. Η θέση στη δευτερεύουσα μνήμη όπου βρίσκονται όσες σελίδες είναι εκτός κύριας μνήμης μπορεί να καταγράφεται στον πίνακα σελίδων ή σε κάποιον άλλο πίνακα.

Ο πίνακας σελίδων είναι και αυτός αποθηκευμένος στη μνήμη. Έτσι κάθε μετάφραση εικονικής διεύθυνσης σε φυσική απαιτεί μια επιπλέον προσπέλαση μνήμης για την ανάγνωση του πίνακα2. Αυτή η προσπέλαση εισάγει μια ανεπιθύμητη καθυστέρηση στην εκτέλεση των εντολών. Μια λύση στο πρόβλημα αυτό είναι η χρήση ειδικής, γρήγορης μνήμης, για την αποθήκευση του πίνακα σελίδων.

Το κυριότερο μειονέκτημα της σελιδοποίησης είναι ότι κάθε διεργασία καταλαμβάνει περισσότερο χώρο από ό,τι χρειάζεται. Συνήθως η μνήμη που απαιτεί μια διεργασία δεν είναι πολλαπλάσιο του μεγέθους της σελίδας, αλλά χρειάζεται π.χ. 10 σελίδες ολόκληρες και 20 λέξεις επιπλέον. Επειδή ο χώρος εικονικών διευθύνσεων που της δίνεται έχει μέγεθος που είναι πολλαπλάσιο της σελίδας, θα της δοθούν 11 σελίδες· από την 11η θα χρησιμοποιηθούν μόνο οι 20 πρώτες λέξεις. Η τελευταία σελίδα κάθε διεργασίας έτσι έχει ένα τμήμα, μικρότερο ή μεγαλύτερο, το οποίο μένει αχρησιμοποίητο. Αυτό το φαινόμενο ονομάζεται *εσωτερικός κατακερματισμός* (internal fragmentation).

Διαχείριση εικονικής μνήμης με κατάτμηση και κατατμημένη σελιδοποίηση

Η σελιδοποίηση είναι η πιο κοινά χρησιμοποιούμενη μέθοδος για τη διαχείριση της εικονικής μνήμης, γιατί συνήθως υποστηρίζεται από το υλικό των υπολογιστών. Δυο άλλες μέθοδοι που χρησιμοποιούνται, αλλά όχι τόσο ευρέως, είναι η κατάτμηση και η κατατμημένη σελιδοποίηση· η δεύτερη είναι συνδυασμός κατάτμησης και σελιδοποίησης.

Η μέθοδος της κατάτμησης προσπαθεί να αποφύγει τον εσωτερικό κατακερματισμό δίνοντας σε κάθε διεργασία ακριβώς όση μνήμη της χρειάζεται. Χωρίζει τη μνήμη σε τμήματα διαφορετικών μεγεθών, και για κάθε ένα από αυτά κρατά την αρχική διεύθυνση και το μέγεθός του. Για να μεταφράσει μια εικονική διεύθυνση σε φυσική, αναζητεί τις πληροφορίες που έχει κρατήσει για το αντίστοιχο τμήμα, ελέγχει αν η δεδομένη διεύθυνση είναι μέσα στα όρια του τμήματος, βρίσκει την αρχική του διεύθυνση στη φυσική μνήμη, και τέλος τη συνδυάζει με τη θέση μνήμης μέσα στο τμήμα για να υπολογίσει την τελική φυσική διεύθυνση.

Όταν δημιουργείται μια νέα διεργασία, το ΛΣ αναζητά ένα τμήμα εικονικών διευθύνσεων αρκετά μεγάλο για να χωρέσει τη μνήμη που χρειάζεται η διεργασία, και της αποδίδει τις διευθύνσεις αυτές.

Η κατατμημένη σελιδοποίηση συνδυάζει τις δυο τεχνικές, εργαζόμενη πάλι με τμήματα, τα οποία όμως αποτελούνται από σελίδες σταθερού μεγέθους. Η κατάτμηση έτσι είναι η ειδική περίπτωση της κατατμημένης σελιδοποίησης όπου κάθε σελίδα αποτελείται μόνο από μία λέξη.

Επίδειξη της Εναλλαγής με τη χρήση της df

Σε ένα σύστημα εικονικής μνήμης με σελιδοποίηση που χρησιμοποιεί την **LRU** (least recently used), οι λιγότερο χρησιμοποιούμενες σελίδες μεταφέρονται στον δίσκο μέχρι (αν ποτέ) ζητηθούν ξανά. Στο Solaris αυτός ο χώρος εναλλαγής χρησιμοποιείται και για την διατήρηση προσωρινών αρχείων στον φάκελο **/tmp.**

Για να δείτε τον χρησιμοποιούμενο χώρο αναμονής, πληκτρολογήστε:

```
df  -h
```

Να σημειωθεί πως αν ο χώρος εναλλαγής είναι πολύ μικρός, τότε δεν υπάρχει αρκετός χώρος να διατηρηθεί μητρώο των μη χρησιμοποιούμενες σελίδες, οπότε είναι πιθανός ο κατακερματισμός (thrashing). Από την άλλη, αν ο χώρος εναλλαγής είναι πολύ μεγάλος, χάνεται πολύτιμος χώρος αποθήκευσης αρχείων.

Παρακολουθώντας τη διαδικασία Σελιδοποίησης

Πληκτρολογώντας:

```
vmstat 2 5
```

θα εκκινήσετε 5 αναφορές **vmstat**, μια κάθε δυο δευτερόλεπτα, με την πρώτη αναφορά να είναι ο μέσος όρος από την ώρα που εκκίνησε το σύστημα. Διαβάστε το manual της **vmstat** για να δείτε τι πληροφορίες παρέχει.. Τα στατιστικά της κύριας μνήμης είναι:

swap: Μέγεθος χώρου εναλλαγής διαθέσιμος σε Kbytes.

free: Το μέγεθος της λίστας ελεύθερων σελίδων σε Kbytes.

pi: Σελίδες που δεσμεύονται σε Kilobytes ανά δευτερόλεπτο. Οι σελίδες που χρειάζεται ο ΛΣ για τις διεργασίες που συνεχίζουν να εκτελούνται.

po: Σελίδες που αποδεσμεύονται σε Kilobytes ανά δευτερόλεπτο.

fr: Kilobytes που αποδεσμεύονται λόγω αποδέσμευσης σελίδων ή τερματισμού διεργασιών.

Συνήθως η τιμή του po είναι μηδέν.

Τμήματα μιας διεργασίας

Μια Unix/Linux διεργασία δεσμεύει πολλά τμήματα στη μνήμη:

- Ένα τμήμα κειμένου, που περιέχει τον κώδικα μηχανής της διεργασίας.
- Ένα τμήμα δεδομένων, που περιέχει τις καθολικές μεταβλητές της διεργασίας. Αρχικά, κάποιες από τις μεταβλητές αυτές έχουν τιμές και κάποιες όχι. Οι δεύτερες διατηρούνται σε ένα τμήμα γνωστό ως τμήμα **bss**.
- Ένα τμήμα σωρού, όπου διατηρούνται οι καινούριες καθολικές μεταβλητές που παράγονται.
- Ένα τμήμα στοίβας, όπου διατηρούνται οι καινούριες καθολικές μεταβλητές που παράγονται, καθώς και οι παράμετροι των συναρτήσεων και οι τιμές που επιστρέφουν.

Με τη χρήση της **size** μπορεί κάποιος να δει το μέγεθος του τμήματος κειμένου, δεδομένων και bss στην εικόνα μιας διεργασίας στον δίσκο. Για παράδειγμα, πληκτρολογήστε:

```
> which ls                          # Πού στον δίσκο κρατιέται η ls;
      /bin/ls
> size /bin/ls
      15678 + 1241 + 1963 = 18882      # κώδικας + δεδομένα + bss == σύνολο
```

Διαμοιρασμός Μνήμης

Καθώς το Unix/Linux τρέχει σε αρχιτεκτονικές σελίδων, μπορεί να χρησιμοποιήσει προστασία σελίδων για να διαμοιράσει τμήματα μνήμης μόνο για ανάγνωση **(read-only)** ανάμεσα σε διεργασίες. Άλλη μια χρήση του διαμοιρασμού σελίδων είναι στις διαμοιραζόμενες βιβλιοθήκες **(shared libraries).** Αυτές είναι υπο-ρουτίνες που κοινές σε όλα τα προγράμματα.

Με τη χρήση της ldd μπορεί κάποιος να δει τις διαμοιραζόμενες βιβλιοθήκες που χρησιμοποιούνται από κάθε πρόγραμμα:

```
> which ls                  # Πού στον δίσκο κρατιέται η ls;
         /bin/ls


> ldd /bin/ls               # Δείξε τις διαμοιρασμένες βιβλιοθήκες που χρησιμοποιούνται
        libc.so.1 =>      /usr/lib/libc.so.1
        libdl.so.1 =>     /usr/lib/libdl.so.1


> /usr/lib/libc.so.1   # Μέγεθος της διαμοιραζόμενης βιβλιοθήκης;
         670256 + 25284 + 6500 = 702040


> ldd a.out
        libpthread.so.1 =>        /usr/lib/libpthread.so.1
        libc.so.1 =>      /usr/lib/libc.so.1
        libdl.so.1 =>     /usr/lib/libdl.so.1
        libthread.so.1 =>         /usr/lib/libthread.so.1
```

Δομή Μνήμης

Έστω ότι η μνήμη είναι ένας τεράστιος πίνακας με **0xffffffff** στοιχεία. Ένας pointer που δείχνει στη μνήμη θα μπορούσε να χρησιμοποιηθεί για πρόσβαση στα στοιχεία του πίνακα αυτού. Έτσι, όταν ο pointer έχει τιμή **0xefffe034**, δείχνει στο **0xefffe035**-στο στοιχείο στον πίνακα μνήμης. Δυστυχώς, δεν μπορείς να προσπελάσει κανείς όλα τα στοιχεία στη μνήμη. Παράδειγμα είναι το στοιχείο **0**. Αν

προσπαθήσει κανείς να αναφερθεί στο στοιχείο με δείκτη 0, οδηγεί σε παράβαση κατάτμησης **(segmentation violation).** Αυτός είναι ο τρόπος του ΛΣ για να ενημερώσει ότι η προσπέλαση της θέσης αυτής είναι παράνομη.

Για παράδειγμα ο παρακάτω κώδικας θα δημιουργήσει **segmentation violation**:

```
main( )
{
char *s;
  char c;
  s = (char *) 0;
  c = *s;
}
```

Σελιδοποίηση (Paging)

Το ΛΣ αναθέτει έναν αριθμό σελίδων για κάθε χρήστη. Κάθε φορά που επιχειρεί ο χρήστης να διαβάσει ή να γράψει σε μια διεύθυνση στη μνήμη, γίνεται έλεγχος αν η διεύθυνση βρίσκεται σε σελίδα που ανήκει στον χρήστη. Αν όχι, τότε παράγεται **segmentation violation.**

Αυτό θα συμβεί αν γράψετε:

```
s = (char *) 0;
c = *s;
```

Αποδεικνύεται ότι οι πρώτες 8 σελίδες σε κάθε μηχάνημα με Unix/Linux ΛΣ είναι void. Δηλαδή προσπάθεια προσπέλασης των θέσεων **0** έως **0xffff** θα οδηγήσει σε **segmentation violation.**

Η επόμενη σελίδα (αρχική διεύθυνση **0x10000**) είναι η πρώτη του τμήματος κώδικα. Το τμήμα αυτό τελειώνει στη μεταβλητή **&etext**. Το τμήμα καθολικών μεταβλητών αρχίζει στη σελίδα με αρχική διεύθυνση **0x20000**. Το τμήμα αυτό τελειώνει στη μεταβλητή **&end**. Ο σωρός αρχίζει αμέσως μετά την **&end**, και φτάνει μέχρι την **sbrk(0)**. Η στοίβα τελειώνει στη διεύθυνση **0xffffffff**. Η αρχή της εξαρτάται από τις διαδικασίες που καλούνται. Κάθε σελίδα ανάμεσα στο τέλος του σωρού και την αρχή της στοίβας είναι void.

Οι μεταβλητές &etext and &end

Είναι δυο εξωτερικές μεταβλητές που ορίζονται ως εξής:

```
extern   etext;
extern   end;
```

Προσέξτε ότι δεν έχουν τύπο. Ποτέ δε χρησιμοποιούνται μόνο ως "etext" και "end". Αντιθέτως, καλούνται με τις διευθύνσεις τους, οι οποίες δείχνουν απευθείας στο τέλος των τμημάτων κειμένου και καθολικών μεταβλητών αντίστοιχα.

Εκτελέστε το πρόγραμμα ***testaddr1.c***, το οποίο τυπώνει τις διευθύνσεις των **etext** και **end** και στη συνέχεια 6 τιμές:

- main ένας δείκτης στην πρώτη εντολή της main()

- Η I είναι καθολική μεταβλητή. Κατά συνέπεια, η &I πρέπει να είναι μια διεύθυνση στο τμήμα των καθολικών μεταβλητών
- Η i είναι τοπική μεταβλητή. Κατά συνέπεια η &i πρέπει να είναι μια διεύθυνση στη στοίβα
- Η argc είναι όρισμα στη main().Κατά συνέπεια, η &argc πρέπει να είναι μια διεύθυνση στη στοίβα
- Η ii είναι τοπική μεταβλητή. Κατά συνέπεια η &ii πρέπει να είναι μια διεύθυνση στη στοίβα. Παρόλα αυτά, ο ii είναι pointer στη μνήμη που έχει οριστεί με malloc. Άρα, η ii πρέπει να είναι μια διεύθυνση στο σωρό.

Παράδειγμα

```
/* testaddr1.c*/
#include <stdio.h>

extern  end;
extern  etext;

extern int  I;
extern int  J;
int  I;

main(int argc, char **argv)
{

     int i;
     int *ii;

     printf("&etext = 0x%lx\n", &etext);
     printf("&end = 0x%lx\n", &end);
     printf("\n");

     ii = (int *) malloc(sizeof(int));

     printf("main = 0x%lx\n", main);
     printf("&I = 0x%lx\n", &I);
     printf("&i = 0x%lx\n", &i);
     printf("&argc = 0x%lx\n", &argc);
     printf("&ii = 0x%lx\n", &ii);
     printf("ii = 0x%lx\n", ii);
}
```

Πιθανό αποτέλεσμα είναι κάτι τέτοιο:

```
> testaddr1
&etext = 0x10b64
&end = 0x20cf0

main = 0x1095c
&I = 0x20ce8
&i = 0xffbefbac
&argc = 0xffbefc04
&ii = 0xffbefba8
ii = 0x20d00
```

Κατά συνέπεια, το τμήμα κώδικα βρίσκεται από **0x10000** σε **0x10b64**. Το τμήμα καθολικών μεταβλητών από **0x20000** έως **0x20cf0**. Ο σωρός από **0x20cf0** μέχρι κάποια διεύθυνση μεγαλύτερη από **0x20d00**. Η στοίβα ξεκινάαπό μια διεύθυνση μικρότερη από **0xefffe8f8** έως **0xeffffff**.

Τώρα, δείτε και εκτελέστε το πρόγραμμα ***testaddr2.c***.

```
/* testaddr2.c*/
#include <stdio.h>
```

```
extern end;
extern etext;

main( )
{
   char *s;
   char c;

   printf("&etext = 0x%lx\n", &etext);
   printf("&end = 0x%lx\n", &end);

   printf("\n");

   printf("Enter memory location in hex (start with 0x): ");
   fflush(stdout);

   scanf("0x%x", &s);

   printf("Reading 0x%x: ", s);
   fflush(stdout);
   c = *s;
   printf("%d\n", c);
   printf("Writing %d back to 0x%x: ", c, s);
   fflush(stdout);
   *s = c;
   printf("ok\n");
}
```

Εκτυπώνει τις **&etext** και **&end**, και στη συνέχεια ζητά από τον χρήστη να εισάγει μια διεύθυνση σε δεκαεξαδικό. Αποθηκεύει τη διεύθυνση στη μεταβλητή s.

Δώστε την (μη επιτρεπτή) τιμή μηδέν:

```
> testaddr2
&etext = 0x10c0c
&end = 0x20ee8

Enter memory location in hex (start with 0x): 0x0
Reading 0x0: Segmentation Fault
```

Προφανώς επιστρέφεται **Segmentation fault**, καθώς η θέση μνήμης αυτή βρίσκεται στο void τμήμα. Οποιαδήποτε θέση μνήμης στο διάστημα **0x0** έως **0xffff** είναι μη νόμιμο και θα επιστρέψει **segmentation violation**

Η μεταβλητή Sbrk(0)

Η **sbrk( )** είναι κλήση συστήματος. Επιστρέφει στον χρήστη το τρέχον τέλος του σωρού. Λόγω της δυνατότητας κλήσης της malloc(), σε διάφορα σημεία, η τιμή της sbrk(0) μπορεί να αλλάζει.

Η testaddr3.c επιστρέφει την τιμή της sbrk(0) – προσέξτε ότι είναι στη σελίδα 16 (0x20000 - 0x21fff). Καθώς το υλικό κάνει έλεγχο ανά διαστήματα των 8192-bytes, μπορεί να επιστραφεί οποιοδήποτε byte στην σελίδα 16, παρότι η sbrk(0) επιστρέφει 0x20c78:

```
/* testaddr3.c */

#include <stdio.h>
```

```
extern end;
extern etext;

main( )
{
  char *s;
  char c;

  printf("&etext = 0x%lx\n", &etext);
  printf("&end = 0x%lx\n", &end);
  printf("sbrk(0)= 0x%lx\n", sbrk(0));
  printf("&c = 0x%lx\n", &c);

  printf("\n");

  printf("Enter memory location in hex (start with 0x): ");
  fflush(stdout);

  scanf("0x%x", &s);

  printf("Reading 0x%x: ", s);
  fflush(stdout);
   c = *s;
  printf("%d\n", c);
  printf("Writing %d back to 0x%x: ", c, s);
  fflush(stdout);
  *s = c;
  printf("ok\n");
}
```

Ενδεικτική έξοδος κατά την εκτέλεση:

```
> testaddr3
&etext = 0x10c84
&end = 0x20f68
sbrk(0)= 0x20f68
&c = 0xffbefbab

Enter memory location in hex (start with 0x): 0x21fff
Reading 0x21fff: 0
Writing 0 back to 0x21fff: ok
```

Οι τιμές των **&end** και **sbrk(0)** ίδιες, καθώς δεν έχουμε καλέσει καμία **malloc()**.
Στην ***testaddr3a.c*** γίνεται κλήση της malloc() και γι' αυτό τον λογο οι τιμές των &end και sbrk(0) είναι διαφορετικές:

```
/* testaddr3a.c */
#include <stdio.h>

extern end;
extern etext;

main( )
{
char *s;
char c;
char *buf;

buf = (char *) malloc(1000);

printf("&etext = 0x%lx\n", &etext);
printf("&end = 0x%lx\n", &end);
printf("sbrk(0)= 0x%lx\n", sbrk(0));
printf("&c = 0x%lx\n", &c);
```

```
printf("\n");

printf("Enter memory location in hex (start with 0x): ");
fflush(stdout);

scanf("0x%x", &s);

printf("Reading 0x%x: ", s);
fflush(stdout);
c = *s;
printf("%d\n", c);
printf("Writing %d back to 0x%x: ", c, s);
fflush(stdout);
*s = c;
printf("ok\n");
}
```

Κατά την εκτέλεση της testaddr3a:

```
> testaddr3a
&etext = 0x10cc4
&end = 0x20fb8
sbrk(0)= 0x22fb8
&c = 0xffbefbab

Enter memory location in hex (start with 0x): 0x23fff
Reading 0x23fff: 0
Writing 0 back to 0x23fff: ok

> testaddr3a
&etext = 0x10cc4
&end = 0x20fb8
sbrk(0)= 0x22fb8
&c = 0xffbefbab

Enter memory location in hex (start with 0x): 0x24000
Reading 0x24000: Segmentation Fault
```