ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Προσαρμοστική διαχείριση μνήμης σε πολυπύρηνα ενσωματωμένα συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΗΛΙΑΣ Β. ΠΛΙΩΤΑΣ

Επιβλέπων : Δημήτριος Σούντρης
Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος, 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Προσαρμοστική διαχείριση μνήμης σε πολυπύρηνα ενσωματωμένα συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΗΛΙΑΣ Β. ΠΛΙΩΤΑΣ

Επιβλέπων : Δημήτριος Σούντρης
Επίκουρος Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την .....

.....	.....	.....
Δημήτριος Σούντρης	Κιαμάλ Πεκμετζή	Γεώργιος Οικονομάκος
Επίκουρος Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2013

.....
ΗΛΙΑΣ Β. ΠΛΙΩΤΑΣ
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στα πλαίσια της παρούσας διπλωματικής εργασίας αναπτύχθηκε ένας προσαρμοστικός διαχειριστής δυναμικής μνήμης για πολυνηματικές εφαρμογές και εκτιμήθηκε η επίδοση του σε σχέση με άλλους σύγχρονους διαχειριστές, τόσο ερευνητικούς (hoard) όσο και βιομηχανικούς (ptmalloc/glibc, jemalloc/facebook) χρησιμοποιώντας την γλώσσα προγραμματισμού C. Η παρούσα διπλωματική εστιάζει στην βελτιστοποίηση του χρόνου εκτέλεσης και την ελαχιστοποίηση των προσβάσεων στην μνήμη. Οι μετρήσεις λήφθηκαν με το εργαλείο ανάλυσης perf της σουίτας perf tools.

Abstract

The scope of the current diploma thesis is to develop a runtime adaptive multithreaded dynamic memory manager in C and compare its performance against state-of-art allocators, build either in academia (hoard) or in industry (ptmalloc2/glibc , jemalloc/facebook). The purpose of this manager is minimize execution time as well as memory accesses, by means of runtime adaptive characteristics. Meaurements were taken using the perf tool (linux perf tools).

Πίνακας Περιεχομένων

1 Εισαγωγή

Γενική Περιγραφή

Η υπηρεσία σωρού για τη γλώσσα C προσφέρεται μέσω των συναρτήσεων δυναμικής διαχείρισης μνήμης (malloc/realloc/calloc/free) της GNU libc, το σύνολο των οποίων αναφέρεται ως διαχειριστής δυναμικής μνήμης (allocator ή dynamic memory manager). Μία εφαρμογή κατά τη λειτουργία της μπορεί να αποκτά πρόσβαση σε χώρο μνήμης το μέγεθος του οποίου καθορίζεται κατά τη διάρκεια του χρόνου εκτέλεσης, ζητώντας από το διαχειριστή μνήμης να της επιστρέψει δείκτη σε μπλοκ μνήμης και στη συνέχεια να δηλώνει ότι δε χρειάζεται το εν λόγω μπλοκ καλώντας την συνάρτηση free με όρισμα τον δείκτη που επέστρεψε η malloc.

Κατά τις προηγούμενες δεκαετίες , με την παράλληλη αύξηση της υπολογιστικής ισχύος των επεξεργαστών και της πολυπλοκότητας των εφαρμογών (web servers, database managers) που εκτελούνται σε αυτούς και είναι γραμμένες σε γλώσσες που χρησιμοποιούν τέτοιο μοντέλο διαχείρισης μνήμης (C / C++) , οι δυναμικοί διαχειριστές μνήμης αποτελούν εδώ και αρκετό καιρό ένα κρίσιμο , για την επίδοση των εφαρμογών , στοιχείο , ενώ σε αρκετές περιπτώσεις σχηματίζουν το bottleneck της επίδοσης της εκάστοτε εφαρμογής.

Ειδικά την τελευταία δεκαετία τόσο η εισαγωγή σε ευρεία χρήση (desktop market) πολυπύρηνων επεξεργαστικών στοιχείων όσο και η εξέλιξη της τεχνολογίας και την μείωση του κόστους παραγωγής VLSI κυκλωμάτων με ολοένα και αυξανόμενες δυνατότητες (embedded systems market) , οδηγούν σε πλατφόρμες εκτέλεσης MPSoC όπου θα εκτελούνται εφαρμογές με αυξανόμενες απαιτήσεις (ελαχιστοποίηση κατανάλωσης μνήμης και χρόνου εκτέλεσης, αξιοποίηση thread-level παραλληλισμού κ.ο.κ.) Οι εν λόγω απαιτήσεις διαμορφώνουν αντίστοιχα και νέες προκλήσεις για τον σχεδιασμό δυναμικών διαχειριστών μνήμης.

Αντικείμενο διπλωματικής

Μέχρι στιγμής, πέρα από την υλοποίηση του allocator της glibc, dlmalloc και μετεξελίξεις του ptmalloc έχει αναπτυχθεί πλήθος διαφορετικών allocators, ενδεικτικά αναφέρουμε mtmalloc, phkmalloc , hoard , tcmalloc, jemalloc η μελέτη των οποίων παρουσιάζει διαφορετικές προσεγγίσεις για την επίλυση του προβλήματος της αποδοτικής δυναμικής διαχείρισης μνήμης. Σκοπός της διπλωματικής είναι η ανάπτυξη ενός νέου διαχειριστή μνήμης με διαφορετική προσέγγιση και, στη συνέχεια, η σύγκριση του με άλλους state-of-art διαχειριστές.

2 Θεωρητικό Υπόβαθρο

Σωρός

Η σωρός (heap) αποτελεί μία global δομή δεδομένων η οποία μαζί με ένα σύνολο συναρτήσεων παρέχει στον προγραμματιστή την δυνατότητα δημιουργίας και καταστροφής μεταβλητών με χρόνο ζωής που καθορίζει ο ίδιος. Οι διάφορες υλοποιήσεις της γλώσσας προγραμματισμού C (compilers) παρέχουν περιορισμένες αντίστοιχες δυνατότητες, με χρόνο ζωής μεταβλητών είτε ίσο με το χρόνο ζωής του προγράμματος, είτε ίσο με το χρόνο ζωής μίας συνάρτησης. (static/auto storage classes) και μέγεθος το οποίο καθορίζεται στατικά.

Πληροφορίες χαμηλού επιπέδου

Header fields: Σε κάθε μπλοκ που αναπαριστά ελεύθερη μνήμη , έχει προστεθεί πεδίο header στο οποίο αποθηκεύεται χρήσιμη πληροφορία πχ το μέγεθος ενός μπλοκ.

Link fields: Αφορά την περίπτωση , όπου για την αναπαράσταση κομματιών ελεύθερου χώρου χρησιμοποιούνται free-lists από τέτοια κομμάτια,οπότε είναι απαραίτητη η ύπαρξη ενός ή δύο δεικτών, για απλή και διπλή λίστα αντίστοιχα.

Boundary Tags: Αν η αναπαράσταση του ελεύθερου χώρου γίνεται με χρήση μπλοκ που αποτελούν στοιχεία λίστας και ο διαχειριστής μνήμης υποστηρίζει ενοποίηση ελεύθερων μπλοκ (coalescing) , ένα boundary tag είναι συμπληρωματική πληροφορία που τοποθετείται στο τέλος κάθε μπλοκ. Κατά την απελευθέρωση ενός μπλοκ από την εφαρμογή, ο διαχειριστής μπορεί να διαπιστώσει εάν το χωρικά προηγούμενό του στην μνήμη είναι δεσμευμένο ή όχι προκειμένου να τα ενοποίησει.

Ταξινόμηση allocators

Ταξινόμηση ως προς τους εσωτερικούς μηχανισμούς

(1) Sequential list fits : Κατά την αναζήτηση ελεύθερου χώρου διατρέχεται, με χρήση των link fields, απλή ή διπλά συνδεδεμένη λίστα από ελεύθερα μπλοκς έως ότου βρεθεί κάποιο μπλοκ με μέγεθος αρκετά μεγάλο ώστε να ικανοποιήσει το αίτημα. Ανάλογα με το κριτήριο τερματισμού της αναζήτησης διακρίνουμε τις περιπτώσεις :

best-fit: Η αναζήτηση τερματίζει όταν βρεθεί το ελάχιστο μέγεθος μπλοκ, που αρκεί για να εξυπηρετήσει το αίτημα. Καθώς η μέθοδος εγγυάται την εύρεση ελάχιστου δυνατού μεγέθους, ένα μειονέκτημα της είναι το ότι η αναζήτηση γίνεται με εξαντλητικό τρόπο κατά μήκος ενός χώρου κατά συνέπεια επιβαρύνεται ο χρόνος εκτέλεσης της.

first-fit : Η αναζήτηση τερματίζει όταν βρεθεί το πρώτο μπλοκ, του οποίου το μέγεθος είναι αρκετά μεγάλο ώστε να ικανοποιήσει το αίτημα. Στη συνέχεια ο διαχειριστής μνήμης μπορεί να

επιλέξει ή όχι να διασπάσει (split) το μπλοκ ώστε να ελαχιστοποιήσει το internal fragmentation.

worst-fit : Η αναζήτηση τερματίζει με την εύρεση του μεγαλύτερου μπλοκ της λίστας, το οποίο διασπάται σε μπλοκ μεγέθους αιτήματος και υπόλοιπου, με στόχο την ελαχιστοποίηση του external fragmentation. Όπως και στην περίπτωση best-fit η αναζήτηση πρέπει να είναι εξαντλητική.

next-fit: Πρόκειται για τροποποίηση της first-fit , όπου σε κάθε επιτυχή αναζήτηση αποθηκεύεται η θέση εντός της λίστας του ευρεθέντος μπλοκ. Η επόμενη αναζήτηση δεν εκκινεί από την αρχή της λίστας αλλά από την προαναφερθείσα θέση.

(2) Segregated fits:

Στην περίπτωση των seggregated free lists χρησιμοποιούνται πίνακες των οποίων τα στοιχεία (slots) είναι δείκτες σε λίστες ελεύθερων μπλοκ. Στην διαδικασία αναζήτηση κάποιου μπλοκ , προστίθεται αρχικά και το βήμα της επιλογής του κατάλληλου index. Ο διαχωρισμός των λιστών μπορεί ,λοιπόν, να γίνεται θεωρώντας ότι κάθε slot περιέχει μπλοκς συγκεκριμένου μεγέθους ή μπλοκς συγκεκριμένου εύρους μεγεθών. Στο survey του Wilson[*] ορίζονται τριών ειδών seggregated fits.

Exact lists : Χρησιμοποιείται λίστα για κάθε πιθανό μέγεθος μπλοκ που μπορεί να αιτηθεί μια εφαρμογή.Καθώς το εύρος μεγεθών είναι συνήθως αρκετά μεγάλο, εάν τα slots δεν είναι διατεταγμένα ως προς το μέγεθος αναπαράστασης, χρειάζεται να εισαχθεί και συμπληρωματική δομή(πχ δέντρο) που θα επιταχύνει την εύρεση του κατάλληλου slot. Επειδή το μέγεθος των μπλοκ της λίστας κάθε slot είναι προκαθορισμένο, η αναζήτηση, μετά τον καθορισμό του slot, γίνεται σε σταθερό χρόνο.

Strict size classes with rounding : Τα μεγέθη που αναπαριστούν τα slots, οργανώνονται σε ομάδες. Μπορούν να χρησιμοπιηθούν πχ δυνάμεις του δύο, αριθμοί της ακολουθίας fib ή όποια αύξουσα ακολουθία επιλέξει ο σχεδιαστής του allocator. Επισημαίνεται ότι, λόγω της στρογγυλοποίησης προς τα πάνω του μεγέθους αιτήματος ,η ακολουθία πρέπει να είναι προσεκτικά επιλεγμένη ώστε να μην προκύπτει μεγάλη επιβάρυνση στην κατανάλωση μνήμης λόγω internal fragmentation , ενώ η αναζήτηση επίσης γίνεται σε σταθερό χρόνο.

Size classes with range lists : Στην περίπτωση αυτή, τα slots αντιστοιχούν σε λίστες που περιέχουν μπλοκς το μέγεθος των οποίων βρίσκεται εντός κάποιου ορισμένου ανά slot εύρους. Εφόσον πλέον

δεν παρέχεται εγγύηση για το ελάχιστο δυνατό μέγεθος κάθε μπλοκ μίας λίστας, η αναζήτηση δεν γίνεται σε σταθερό χρόνο, αλλά σε γραμμικό με χρήση κάποιου από τους αλγορίθμους sequential fits.

(3) Buddy systems: Ο αναπαριστόμενος χώρος μνήμης διαχωρίζεται σε δύο κομμάτια. Στη συνέχεια κάθε υποκομμάτι επίσης διαχωρίζεται στα δύο και η περιγραφόμενη διαδικασία συνεχίζεται αναδρομικά μέχρι κάποιο ελάχιστο μέγεθος. Μέσω της περιγραφόμενης ιεραρχικής διάσπασης, δημιουργούνται αναπαραστάσεις ελεύθερου χώρου μεγέθους που προκύπτει από την διαδικασία διαχωρισμού, με περιορισμούς όμως όσον αφορά την χωρική τοποθέτηση των ζητούμενων μπλοκ στη μνήμη, αλλά και την δυνατότητα ενοποίησης ελεύθερων προς χρήση μπλοκς, αφού κάθε υποκομμάτι μπορεί να ενωθεί μόνο με το γειτονικό του. Η στατική τοποθέτηση των κομματιών εξασφαλίζει το ότι μπορεί γρήγορα να εξεταστεί η δυνατότητα δημιουργίας μεγαλύτερου διαθέσιμου χώρου με ενοποίηση. Ανάλογα με τον λόγο των μεγεθών που προκύπτουν από την ιεραρχική διάσπαση διακρίνουμε τις περιπτώσεις:

Binary buddies: Τα υποκομμάτια της διάσπασης έχουν ίσο μέγεθος και , κατά συνέπεια , κάθε υποχώρος της ιεραρχίας έχει μέγεθος δύναμηςτου δύο. Το γεγονός αυτό εισάγει αρκετά μεγάλο internal fragmentation.

Fibonacci buddies: Κατά την διάσπαση ο αρχικός χώρος αντιστοιχίζεται σε κάποια τιμής της ακολουθίας fibonacci και τα μεγέθη των υποκομματιών είναι ίσα με με τους δύο προηγούμενους όρους της ακολουθίας. Το σκεπτικό χρησιμοποίησης του σχήματος έναντι των binary buddies , είναι ότι δημιουργεί κλάσεις μεγεθών που βρίσκονται πιο κοντά μεταξύ τους οπότε μειώνει το internal fragmentation.

Weighted buddies: Η ακολουθία διάσπασης αποτελείται από την ένωση της ακολουθίας των δυνάμεων του δύο με την ακολουθία των δυνάμεων του δύο , πολλαπλασιασμένων κατά τρία. (2,4,6,8,12,18 ... κοκ). Έτσι , υποχώρος ακριβής δύναμης (2,4,8,16...) μπορεί να διασπαστεί σε δύο ίσα κομμάτια, ενώ κάθε υποχώρος πολλαπλασίου μπορεί είτε να διασπαστεί σε δύο ίσα κομμάτια μέγεθους πολλαπλασίου είτε σε δύο άνισα κομμάτια με λόγο μεγέθους 1/3 και 2/3.

Double buddies: Για την διάσπαση χρησιμοποιούνται δύο διαφορετικά binary buddies υποσυστήματα , με πρώτη ακολουθία ακριβή δύναμη του δύο (2,4,8,16,32 ...) και δεύτερη ακολουθία δύναμη του δύο με διαφορετική βάση (3, 6 , 12 , 24 , 48). Ο κανόνας διάσπασης διαφέρει, καθώς κάθε υποχώρος, διασπάται αποκλειστικά σε ίσα κομμάτια κατά συνέπεια προκύπτουν μεγέθη υποχώρων που εμπίμπτουν σε μία από τις δύο ακολουθίες μόνο.

(4) Indexed fits: Όπως και στην κατηγορία των sequential fits, ο χώρος μνήμης αναπαρίσταται από

μπλοκς μεταβλητού μεγέθους. Για την κατηγορία των indexed fits, χρησιμοποιούνται δομές δεδομένων που δεικτοδοτούν τα μπλοκ ως προς κάποια χαρακτηριστικά τους ώστε η αναζήτηση να καθίσταται όσο το δυνατόν πιο αποδοτική. Τέτοιου είδους χαρακτηριστικά μπορεί να είναι το μέγεθος , με αντίστοιχη δομή δεικτοδότησης κάποιο ισορροπημένο δέντρο με κλειδιά (key values) κόμβων τα μεγέθη των ελεύθερων μπλοκ , ή η διεύθυνση όπου οι κόμβοι του δέντρου αντιστοιχούν σε διευθύνσεις. Ένα προφανές πλεονέκτημα των παραδειγμάτων αυτών είναι ο χαμηλότερος εγγυημένος ή αναμενόμενος , ανάλογα με την δομή , χρόνος αναζήτησης , λογαριθμικός για δέντρα έναντι του γραμμικού των λιστών. Αξίζει να σημειωθεί ότι με χρήση δομής δεδομένων ακόμα μεγαλύτερης πολυπλοκότητας (πχ Cartesian Trees) η δεικτοδότηση μπορεί να πραγματοποιείται για πολλαπλά χαρακτηριστικά.

(5)Bitmapped fits: Οι Wilson et al περιγράφει τον τρόπο αναπαράστασης στα bitmapped fits, ως μία σειρά από bits , η τιμή του καθενός από τα οποία αναπαριστά την κατάσταση κάθε λέξης του αναπαριστόμενου χώρου (1 - ελεύθερη, 0 - κατειλλημένη), με προφανές πλεονέκτημα την ελαχιστοποίηση του μεγέθους των μεταδεδομένων αλλά με μειονέκτημα γραμμικό χρόνο αναζήτησης. Στα πλαίσια της παρούσας διπλωματικής ο εν λόγω μηχανισμός χρησιμοποιήθηκε κατά κόρον και επεκτάθηκε, ώστε κάθε bit να αναπαριστά την κατάσταση μπλοκ μνήμης με μέγεθος όχι μόνο μεγαλύτερου από μίας λέξης αλλά και μεταβαλλόμενου κατά το χρόνο εκτέλεσης, όπως επίσης αναζητήθηκαν λύσεις στην κατεύθυνση της δραστικής μείωσης του χρόνου αναζήτησης.

Ταξινόμηση ως προς την πολυνηματικότητα

Οι Berger et.al [] υποδεικνύουν μία διαφορετική ταξινόμηση των δυναμικών διαχειριστών μνήμης, που αγνοεί τον ακριβή αλγόριθμο ικανοποίησης αιτημάτων μνήμης αλλά γίνεται με βάση τον τρόπο με τον οποίο εξυπηρετούνται τέτοιου είδους αιτήματα όταν προέρχονται από πολυνηματικές εφαρμογές. Η εν λόγω ταξινόμηση ορίζει πέντε κλάσεις πολυνηματικών διαχειριστών μνήμης : Single serial heap, concurrent single heap, pure private heaps, pure private heaps with ownership, private heaps with thresholds.

Single serial heap:

Πρακτικά, κάθε διαχειριστής μνήμης που ανήκει σε οποιαδήποτε ομάδα της ταξινόμησης ως προς τους εσωτερικούς μηχανισμούς μπορεί, με την προσθήκη καθολικού κλειδώματος της συνολικής του δομής, να μετασχηματιστεί σε διαχειριστή της συγκεκριμένης κλάσης. Στην περίπτωση αλληλεπίδρασης του διαχειριστή μνήμης με πολυνηματικές εφαρμογές, τέτοιου είδους σχεδίαση έχει δύο βασικά μειονεκτήματα. Πρώτον, επιβάλλει τη σειριοποίηση αιτημάτων εξυπηρέτησης που προέρχονται από διαφορετικά νήματα με συνέπεια την επιβάρυνση στο χρόνο εκτέλεσης τους λόγω

της μη αξιοποίησης δυνατοτήτων παραλληλισμού (lock contetion). Η επιβάρυνση αυξάνεται, όσο μεγαλώνει και ο αριθμός των νημάτων, γεγονός που καθιστά τη σχεδίαση μη κλιμακώσιμη. Δεύτερον, λόγω του ότι αιτήματα από διαφορετικά νήματα, τα οποία σε πολυπήρυνες αρχιτεκτονικές εκτελούνται σε διαφορετικούς πυρήνες, εξυπηρετούνται από κοινό χώρο μνήμης εισάγεται πρόσθετη επιβάρυνση στον χρόνο εκτέλεσης λόγω του φαινομένου του false sharing.

Concurrent single heap:

Ο διαχειριστής μνήμης επιλέγει να χειρίζεται τον χώρο μνήμης με χρήση μοναδικής , παράλληλης δομής δεδομένων. Και σε αυτήν την περίπτωση, ανεξάρτητα από την κατηγοριοποίηση ως προς τους εσωτερικούς μηχανισμούς, ο μετασχηματισμός σε διαχειριστή της εν λόγω κλάσης μπορεί να επιτευχθεί, προσθέτοντας κατάλληλα κλειδώματα και σχήματα συγχρονισμού ώστε να εξασφαλίζεται η ατομικότητα των προσβάσεων κατά την εξυπηρέτηση αιτημάτων από διαφορετικά νήματα σε κάθε υποδομή του διαχειριστή μνήμης. Πχ αν χρησιμοποιούνται seggregated freelists θα μπορούσε να τοποθετηθεί διαφορετικό κλείδωμα ανά slot.

Όπως ισχύει και για την κλάση single serial heap, ο βαθμός εμφάνισης φαινομένων internal/external fragmentation εξαρτάται από τους μηχανισμούς αναπαράστασης. Όσον αφορά την αλληλεπίδραση με πολυνηματικές εφαρμογές, η παρουσία τοπικών κλειδωμάτων έναντι ενός καθολικού εξασφαλίζει καλύτερη απόδοση λόγω μείωσης του ανταγωνισμού κλειδώματος (lock contetion) και καλύτερη κλιμακωσιμότητα. Επειδή, όμως, τα αιτήματα που προέρχονται από διαφορετικά νήματα εξακολουθούν να ικανοποιούνται από κοινό χώρο μνήμης, διαχειριστές που ανήκουν στην κλάση concurrent single heap δεν αντιμετωπίζουν αποτελεσματικά το false sharing.

Pure private heaps:

Στην κλάση αυτή, ο διαχειριστής μνήμης διαθέτει για κάθε νήμα έναν υποχώρο μνήμης και, στη συνέχεια, περιορίζει εντός αυτού του χώρου την εξυπηρέτηση των αιτημάτων δέσμευσης και αποδέσμευσης μνήμης του νήματος, εξασφαλίζοντας παράλληλα ότι ο αριθμός των υποχώρων θα είναι ίσος με τον αριθμό των νημάτων. Ενώ τίθεται ένα άνω όριο στο εύρος των κλειδωμάτων που απαιτούνται (granularity), καθώς κάθε νήμα δεν χρειάζεται να "κλειδώσει" παραπάνω από έναν υποχώρος, και κατά συνέπεια μειώνεται η επιβάρυνση του χρόνου εκτέλεσης λόγω ανταγωνισμού κλειδωμάτων. Η εν λόγω μείωση είναι ανάλογη της ικανότητας της εσωτερικής δομής αναπαράστασης του υποχώρου, να εξυπηρετεί παράλληλα αιτήματα.

Η απόδοση των διαχειριστών μνήμης της κλάσης pure private heaps ως προς την έκταση της

εμφάνισης του φαινομένου του false sharing εξαρτάται από τη συμπεριφορά της ίδιας της εφαρμογής. Εάν κάθε νήμα απελευθερώνει κομμάτια μνήμης τα οποία έχουν δεσμευτεί αποκλειστικά από το ίδιο, το φαινόμενο εξαλείφεται εντελώς. Καθώς όμως αναφερόμαστε σε μοντέλο εκτέλεσης πολυνηματισμού όπου τα νήματα μοιράζονται έναν κοινό χώρο διευθύνσεων , υπάρχει η περίπτωση ένα νήμα να δεσμεύει μπλοκ μνήμης και ακολούθως να το παραδίδει σε άλλο νήμα για αποδέσμευση. Σε τέτοια μοτίβα εκτέλεσης , είναι δυνατό μπλοκς που ανήκουν στο ίδιο cache line να τοποθετούνται σε διαφορετικούς υποχώρους και άρα να δημιουργούν false sharing όταν θα χρησιμοποιηθούν για εξυπηρετήσουν μελλοντικά αιτήματα διαφορετικών νημάτων.

Private heaps with ownership:

Αποτελεί σχεδιαστική παραλλαγή της κλάσης pure private heaps. Διαχειριστές μνήμης που ανήκουν σε αυτήν κατά την εξυπηρέτηση αιτημάτων αποδέσμευσης, δεδομένης της ύπαρξης πολλών υποχώρων μνήμης, τοποθετούν το μπλοκ στον υποχώρο από τον οποίο προήλθε η δέσμευσή του. Με τον τρόπο αυτό εξαλείφονται φαινόμενα false sharing που εμφανίζονται στην περίπτωση των pure private heaps για συγκεκριμένα μοτίβα συνεργατικού πολυνηματισμού. Όσον αφορά την επίδοση, αναμένεται να εμφανίζεται μεγαλύτερη σε σχέση με εκείνη των διαχειριστών των κλάσεων single serial και concurrent , ενώ το αποτέλεσμα της σύγκρισης της με διαχειριστές της κλάσης pure private heaps εξαρτάται από την αποδοτικότητα την υλοποίησης μηχανισμών ιδιοκτησίας.

Private heaps with thresholds:

Αποτελεί , επίσης , σχεδιαστική παραλλαγή της κλάσης pure private heaps , με δύο βασικές διαφορές. Πρώτον , την προσθήκη ενός καθολικού υποχώρου μνήμης στον οποίο έχουν πρόσβαση όλα τα νήματα , και , δεύτερον , την ύπαρξη στατικής ή μεταβαλλόμενης τιμής κατωφλίου ανά τοπικό υποχώρο νήματος. Όταν , το μέγεθος ενός τοπικού χώρου που αντιστοιχεί σε ελεύθερη μνήμη ξεπερνάει την τιμή κατωφλίου , πυροδοτείται μεταφορά μπλοκς ελεύθερης μνήμης από τον τοπικό στον καθολικό υποχώρο. Η ύπαρξη τέτοιου μηχανισμού περιορίζει το φαινόμενο blowup , καθώς διασφαλίζεται το ότι η ποσότητα ελεύθερης μνήμης υποχώρου δεν είναι απεριόριστη αφού περιορίζεται από την τιμή κατωφλίου. Αν η εν λόγω τιμή παραμένει στατική , τότε η εμφάνιση του blowup είναι ασυμπτωτικά σταθερή.

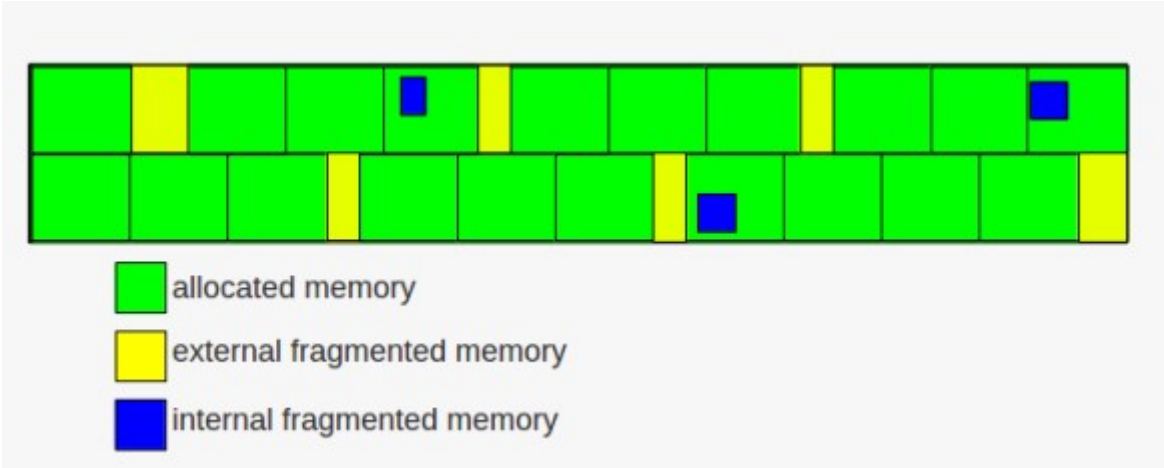
3 Φαινόμενα που αντιμετωπίζουν οι διαχειριστές δυναμικής μνήμης

External fragmentation

Προκύπτει όταν ένας ελεύθερος χώρος μνήμης είναι μοιρασμένος σε μικρά ελεύθερα κομμάτια τα οποία παρεμβάλλονται από δεσμευμένα κομμάτια μνήμης. Στην περίπτωση αυτή , είναι δυνατόν ένα αίτημα από την εφαρμογή να μην μπορεί να εξυπηρετηθεί από τον συγκεκριμένο χώρο μνήμης , παρότι το άθροισμα των μεγεθών των ελεύθερων κομματιών υπερβαίνει το μέγεθος του αιτήματος.

Internal fragmentation

Προκύπτει όταν ο allocator δεσμεύει εσωτερικά και επιστρέφει περισσότερη μνήμη από όση αιτείται η εφαρμογή. Για την αντιμετώπιση του internal fragmentation κάποιοι allocators επιλέγουν υπό συνθήκες να διαχωρίζουν το ευρεθέν μπλοκ μνήμης σε δύο κομμάτια μικρότερου μεγέθους , το ένα εκ των οποίες παραδίδεται στην εφαρμογή και το δεύτερο παραμένει ως ελεύθερο στις εσωτερικές δομές του allocator.



Cache Coherence και False sharing

cache coherence :

Ένα σύστημα μνήμης είναι συναφές (coherent) όταν:

(1) Μία ανάγνωση από έναν πυρήνα P σε μία τοποθεσία μνήμης X , που ακολουθεί μία εγγραφή από τον P στην X , χωρίς να παρεμβάλλονται άλλες εγγραφές στην X ανάμεσα στις δύο λειουργίες , επιστρέφει πάντα την τιμή που εγγράφεται από τον P. (program order)

(2) Μία ανάγνωση από έναν πυρήνα P1 σε μία τοποθεσία μνήμης X , που ακολουθεί μία εγγραφή από έναν άλλο πυρήνα P2 , επιστρέφει την εγγραμένη τιμή εάν οι λειτουργίες χωρίζονται αρκετά χρονικά και δεν προκύπτουν άλλες εγγραφές στην X ανάμεσα στις δύο λειτουργίες.

(3) Εγγραφές στην ίδια τοποθεσία μνήμης σειριοποιούνται. Δηλαδή, δύο εγγραφές στην ίδια τοποθεσία μνήμης από δύο διαφορετικύς πυρήνες γίνονται ορατές με την ίδια σειρά από όλους του πυρήνες.

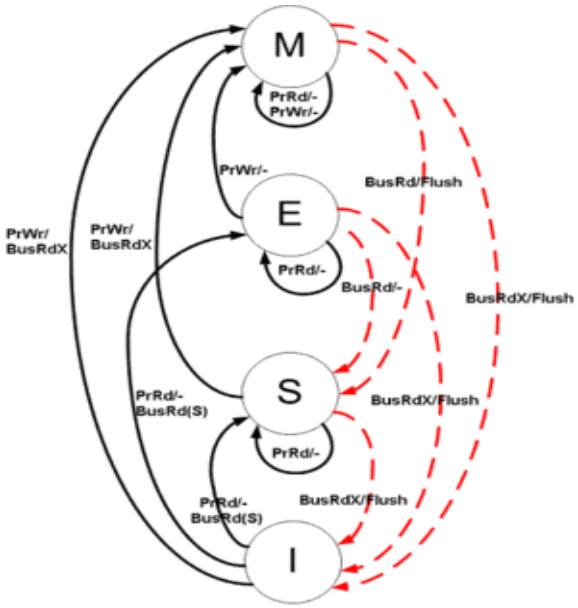
Χονδρικά , συστήματα πολλαπλών επεξεργαστών εγγυόνται συνάφεια μνήμης χρησιμοποιώντας δύο κλάσεις πρωτοκόλλων :

(1) directory based - centralized : η κατάσταση κάθε μπλοκ μνήμης αποθηκεύεται σε κεντρικό σημείο.

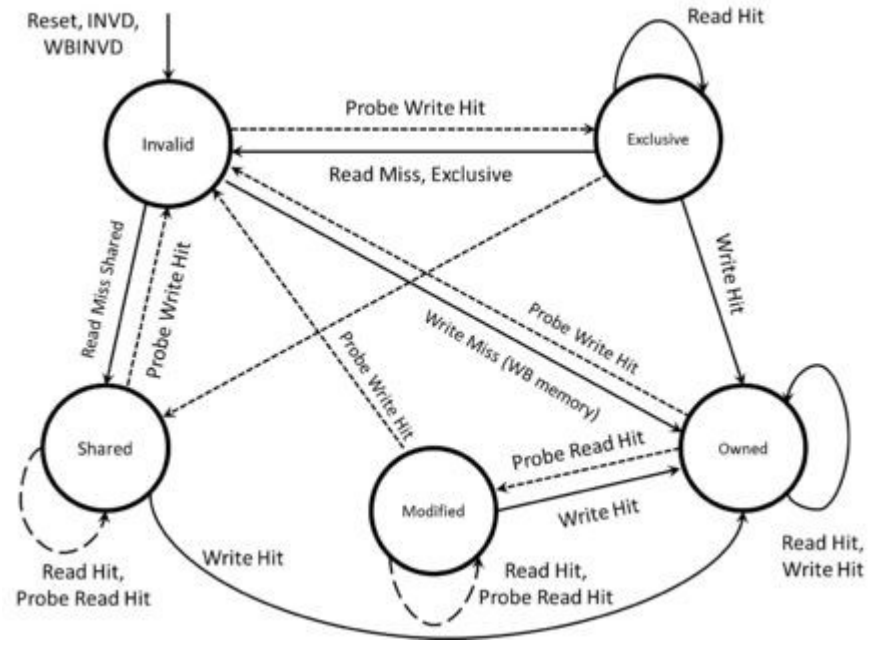
(2) snooping - decentralized : Κάθε αντίγραφο μπλοκ μνήμης που βρίσκεται σε κάποια από τις κατανεμημένες caches , συνοδεύεται από bits που αναπαριστούν την κατάσταση του μπλοκ ως προς το πρωτόκολλο συνάφειας μνήμης.

Καθώς η δεύτερη κλάση είναι η πιο συνηθισμένη και οι αρχιτεκτονικές στις οποίες θα δοκιμαστεί ο allocator υλοποιούν πρωτόκολλα κλάσης snooping, με συνάφεια ανά cache line, θα επικεντρωθούμε σε αυτήν.Η snooping κλάση πρωτοκόλλων χωρίζεται σε δύο υποκλάσεις : write update και write invalidation. Στην πρώτη περίπτωση , κάθε εγγραφή σε cache line που ανήκει σε τοπική cache ενός επεξεργαστή γίνεται broadcast, ώστε να ενημερώνονται τα τοπικά αντίγραφα που βρίσκονται στις υπόλοιπες caches. Πρακτικά, η χρήση τέτοιου πρωτοκόλλου συνεπάγεται μεγάλη απαίτηση για bandwidth του bus που συνδέει caches και κεντρική μνήμη και για το λόγο αυτό στην πράξη συνήθως δεν προτιμάται. Στην δεύτερη περίπτωση, κάθε εγγραφή δεν ενημερώνει, αλλά θέτει ως invalid τα υπόλοιπα τοπικά αντίγραφα. Ο μηχανισμός τήρησης της συνάφειας μνήμης μπορεί να περιγραφεί πλήρως ως πεπερασμένο αυτόματο, με καταστάσεις που αντιστοιχούν στην εγκυρότητα ή μη του τοπικού αντιγράφου.Παραδείγματα MESI και MOESI πρωτοκόλλων συνάφειας μνήμης:

MESI


M
E
S
I
PrRd/-
PrWr/-
PrWr/-
PrWr/
BusRdX
PrWr/
BusRdX
PrRd/-
BusRd/-
BusRd/Flush
BusRdX/Flush
PrRd/-
BusRd(S)
BusRdX/Flush
PrRd/-
PrRd/-
BusRd(S)
BusRdX/Flush

MOESI

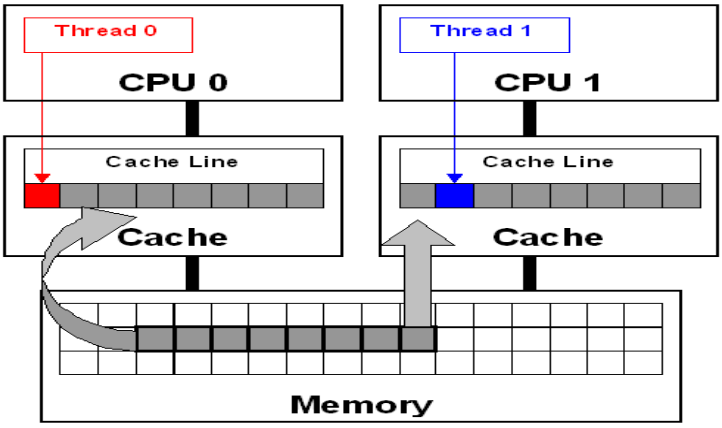

Reset, INVD,
WBINVD
Read Hit
Invalid
Probe Write Hit
Exclusive
Read Miss, Exclusive
Read Miss Shared
Probe Write Hit
Write Hit
Probe Write Hit
Write Miss (WB memory)
Probe Write Hit
Shared
Probe Read Hit
Owned
Modified
Write Hit
Read Hit,
Probe Read Hit
Write Hit
Read Hit,
Probe Read Hit
Read Hit,
Write Hit

Καθίσταται προφανές, ότι όσον αφορά τις προσβάσεις στη μνήμη, πέρα από τα κλασικά compulsory-capacity-confict misses, που αποτελούν πρόβλημα για την επίδοση μονοπύρηνων συστημάτων με ιεραρχία μνήμης, στα πολυπήρυνα συστήματα που διασφαλίζουν συνάφεια μνήμης με write invalidation protocols εμφανίζεται και το φαινόμενο των coherence misses , τα οποία αποδίδονται σε δύο διαφορετικά αίτια : true sharing και false sharing.

True sharing coherence misses εμφανίζονται όταν εντολές εκτελούμενες σε διαφορετικούς πυρήνες με μη κοινές caches προσπελαύνουν το ίδιο αντικείμενο στην μνήμη , με συνέπεια ένας να κατέχει έγκυρο αντίγραφο στην cache του και οι υπόλοιποι μη έγκυρα - invalidated , οπότε υποχρεώνονται σε αντιγραφή του από την πρώτη cache ή από κάποιο κατώτερο κοινό επίπεδο της ιεραρχίας μνήμης με μεγαλύτερο κόστος , ανάλογα με την υλοποίηση της αρχιτεκτονικής.

False sharing coherence misses εμφανίζονται με αντίστοιχο τρόπο, με τη διαφορά ότι από τις προαναφερθείσες εντολές προσπελαύνονται διαφορετικά αντικείμενα που όμως ανήκουν στο ίδιο μπλοκ για το οποίο η αρχιτεκτονική διατηρεί συνάφεια, συνήθως μεγέθους ενός cache line. Στην περίπτωση αυτή , οι πολιτικές εξυπηρέτησης requests από έναν πολυνηματικό διαχειριστή μνήμης επηρεάζουν σε μεγάλο βαθμό την εμφάνιση ή μη false sharing , με μεγάλη επίπτωση στην επίδοση πολυνηματικών εφαρμογών. Ένας διαχειριστής μνήμης θα μπορούσε να εξαλείψει εντελώς την εμφάνιση του false sharing εξυπηρετώντας κάθε αίτημα με το μικρότερο δυνατό αλλά μεγαλύτερο του αιτήματος πολλαπλάσιο του μπλοκ συνάφειας μνήμης. Κάτι τέτοιο όμως θα προκαλούσε αύξηση στην κατανάλωση μνήμης λόγω όξυνσης του internal fragmentation.

False sharing ενδέχεται να εισάγεται από τον ίδιο τον διαχειριστή με δύο διαφορετικούς τρόπους, ενεργητικό και παθητικό (actively/passively induced). Ενεργητική εισαγωγή προκύπτει όταν ο διαχειριστής εξυπηρετεί αιτήματα δέσμευσης, με μπλοκ μνήμης τα οποία επικαλύπτουν ένα μπλοκ τήρησης συνάφειας. Παθητική εισαγωγή, προκύπτει όταν κατά την εξυπηρέτηση αιτήματος αποδέσμευσης, ο διαχειριστής τοποθετεί το αποδεσμευμένο μπλοκ, αρκετά κοντά χωρικά ώστε μαζί με ένα άλλο να επικαλύπτουν, πάλι , μπλοκ συνάφειας μνήμης. Με τον τρόπο αυτό, false sharing δεν προκύπτει άμεσα αλλά κατά την εξυπηρέτηση μελλοντικών αιτημάτων δέσμευσης.



Συνοψίζοντας, το πρόβλημα του false sharing προκύπτει κατά την εκτέλεση πολυνηματικών εφαρμογών, με κοινό χώρο διευθύνσεων για όλα τα νήματα, οι οποίες εκτελούνται σε πολυπύρηνες αρχιτεκτονικές με κατανεμημένες caches. Στην συνήθη περίπτωση τέτοιες αρχιτεκτονικές , ικανοποιούν την απαίτηση συνάφειας μνήμης χρησιμοποιώντας write invalidation protocols, τα οποία ανάλογα με την συμπεριφορά του διαχειριστή μνήμης και της εφαρμογής.

Blowup

Φαινόμενο που εμφανίζεται σε πολυνηματικούς διαχειριστές μνήμης με πολλαπλούς υποχώρους, χωρίς μηχανισμό ισορρόπησης της διαχειριζόμενης ελεύθερης μνήμης μεταξύ των υποχώρων. Συνέπεια του αποτελεί η μεγαλύτερη κατανάλωση μνήμης λόγω προώθησης αιτημάτων δέσμευσης σε υποχώρους που δεν έχουν πρόσβαση στην ελεύθερη μνήμη άλλων, με συνέπεια την εκ νέου απαίτηση παραχώρησης μνήμης από τον πυρήνα του λειτουργικού συστήματος.

4 Χώρος σχεδιασμού δυναμικών διαχειριστών μνήμης

Η κατασκευή ενός αποδοτικού διαχειριστή δυναμικής μνήμης , απαιτεί την λήψη μιας σειράς σχεδιαστικών αποφάσεων και , στη συνέχεια, την συγγραφή του κώδικα υλοποίησης. Το σύνολο των δυνατών αποφάσεων σχηματίζει έναν κοινό χώρο , στοιχεία του οποίου μπορούν να χρησιμοποιηθούν για τον χαρακτηρισμό οποιουδήποτε διαχειριστή. Μέχρι στιγμής έχουν προταθεί ερευνητικά αναπαραστάσεις τέτοιων χώρων που καλύπτουν τις σχεδιαστικές αποφάσεις τόσο μονονηματικών όσο και πολυνηματικών διαχειριστών μνήμης. Η απαρίθμηση των αποφάσεων , που δημιουργεί τους εν λόγω χώρους , γίνεται με τέτοιο τρόπο ώστε η μεταξύ τους αλληλεπίδραση να είναι όσο το δυνατόν ελάχιστη ή να εξαλείφεται εντελώς.

Κατηγορίες ενδονηματικών παραμέτρων

Δομής μπλοκ μνήμης :

Σε αυτήν την κατηγορία , εισάγονται τεσσάρων ειδών ορθογώνια δέντρα που διαχειρίζονται τις δομές αναπαράστασης ελεύθερου χώρου : δέντρα δομής , μεγεθών , ετικετών , και καταγραφόμενης πληροφορίας. Το δέντρο δομής ορίζει τους διαφορετικούς τύπους μπλοκ μνήμης , που απαιτούνται για την αναπαράσταση και κατασκευή αναπαράστασης δυναμικών δεδομένων. Το δέντρο μεγεθών συμπεριλαμβάνει τα διαφορετικά μεγέθη των βασικών μπλοκ που χρησιμοποιούνται σε αλγορίθμους segregated fits , buddy systems , bitmap (bytes per cell) κοκ. Τα δέντρα ετικετών και καταγραφόμενης πληροφορίας ορίζουν τα συμπληρωματικά μεταδεδομένα , τα οποία είναι διαφανή για την εφαρμογή αλλά χρησιμοποιούνται από τον διαχειριστή είτε για λειτουργίες profiling είτε για αναπροσαρμογή της συμπεριφοράς του κατά τον χρόνο εκτέλεσης.

Οργάνωσης pools :

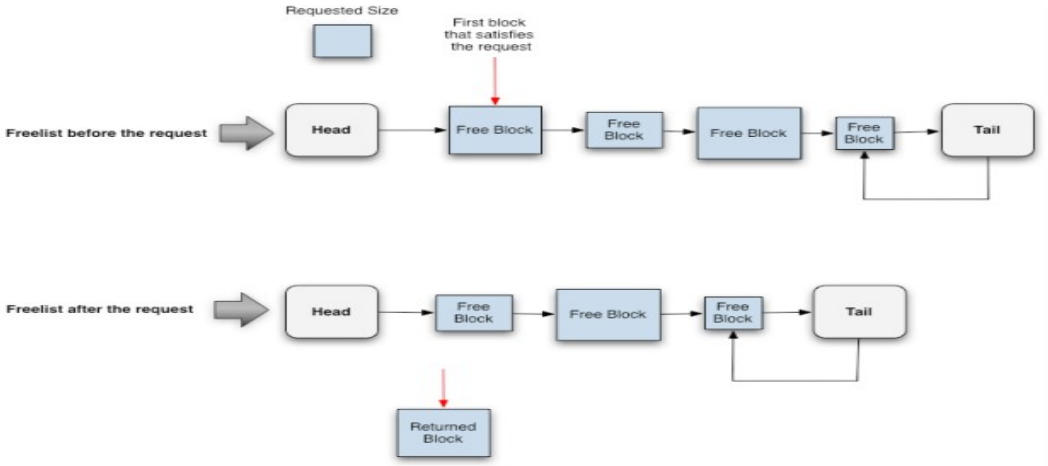
Εισάγονται τριών ειδών δέντρα, που καθορίζουν τα κριτήρια διαίρεσης του διαθέσιμου χώρου μνήμης σε υποχώρους : μέγεθος, διάταξη και μπλοκς. Κάτι τέτοιο συνεπάγεται ή την ύπαρξη ενός μοναδικού χώρου ή την ύπαρξη πολλαπλών για κάθε κριτήριο.

Δέσμευσης μπλοκ μνήμης:

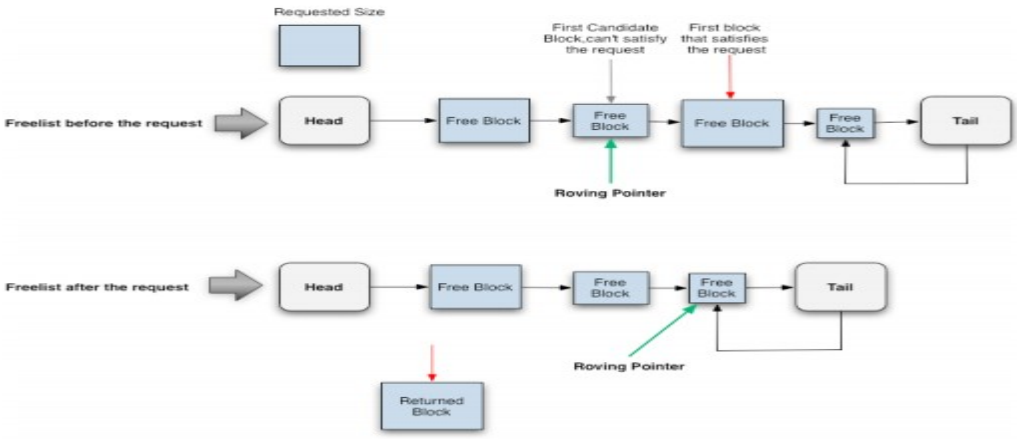
Εισάγονται δύο εντελώς ορθογώνια δέντρα , στα οποία εντάσσονται οι αποφάσεις του διαχειριστή , για την εξυπηρέτηση αιτημάτων δέσμευσης μνήμης.

Το δέντρο αναζήτησης διάταξης , περικλείει τις αποφάσεις μηχανισμών αναζήτησης , ανάλογα και με την ύπαρξη ή μη δεικτοδότησης.. Το δέντρο πολιτικών αναζήτησης αναφέρεται στους διαθέσιμους αλγορίθμους. εύρεσης ελεύθερου μπλοκ : firstFit, nextFit και bestFit.

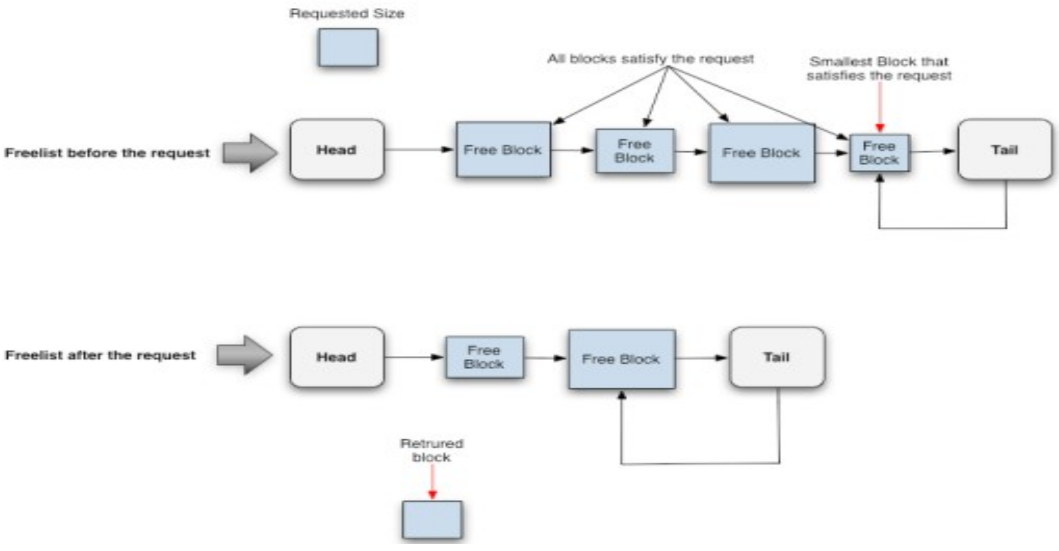
First fit



Next fit



Best fit

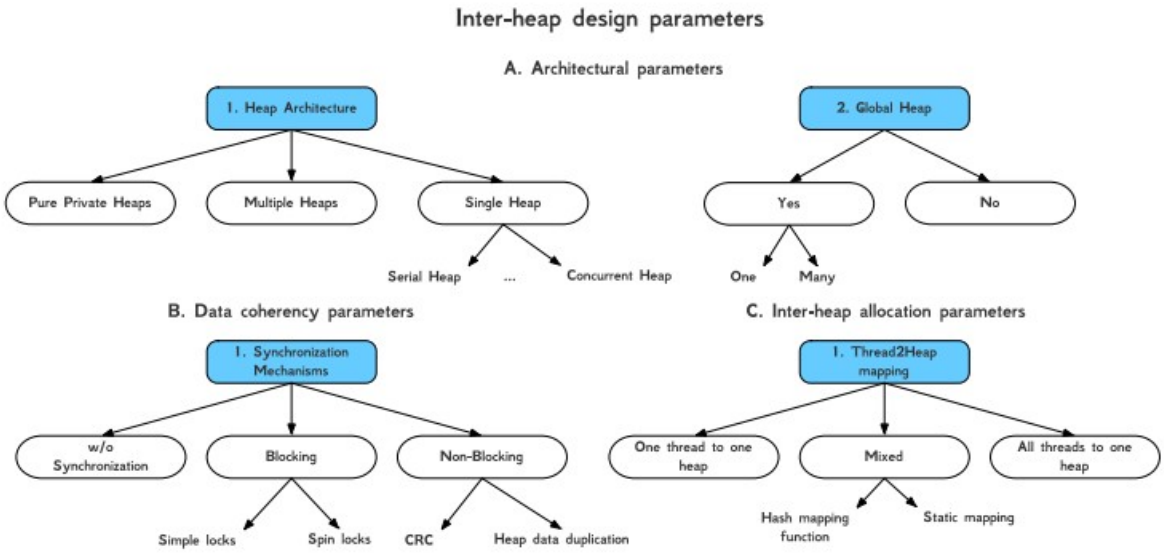


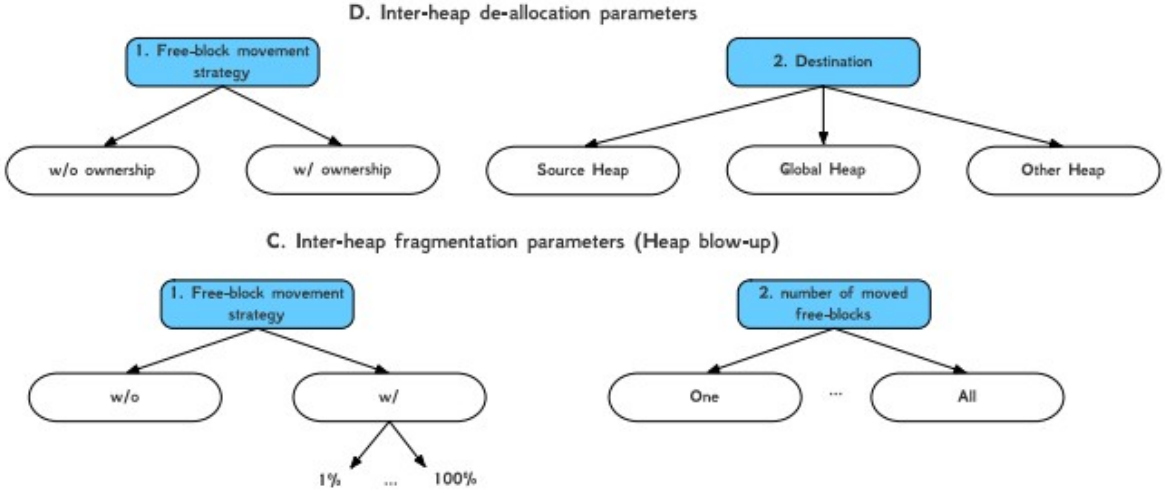
Αποδέσμευσης μπλοκ μνήμης:

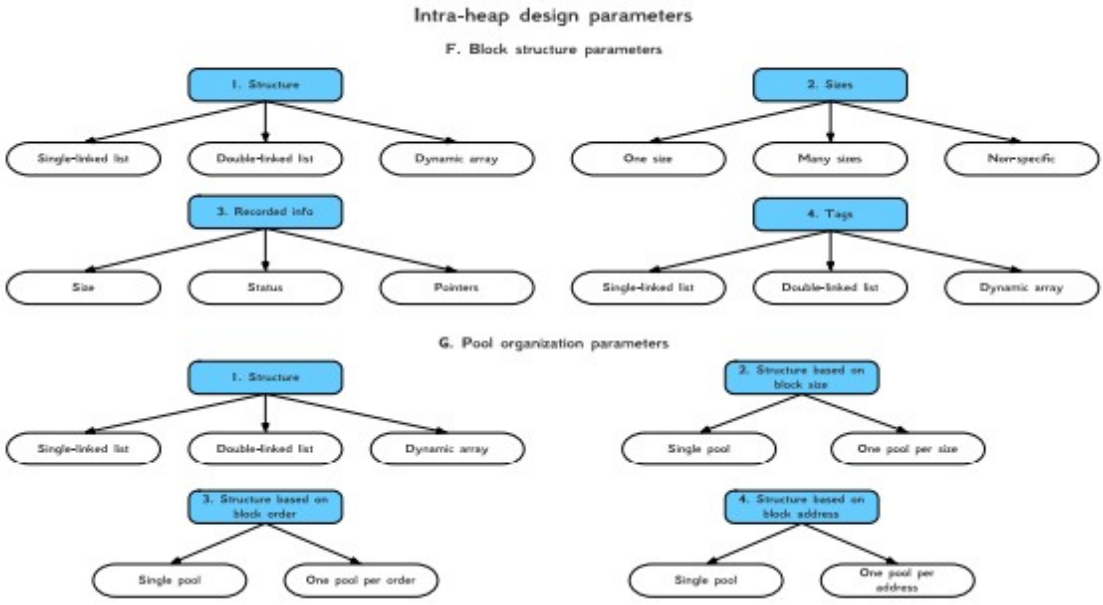
Ορίζονται δύο ορθογώνια δέντρα που καθορίζουν τις αποφάσεις που απαιτούνται για την αποδέσμευση ενός κατειλειμμένου μπλοκ στο εσωτερικό ενός χώρου μνήμης. Το δέντρο αναζήτησης αποδέσμευσης , ορίζεται κατά αντιστοιχία του δέντρου δέσμευσης. Το δέντρο κατεύθυνσης pool αφορά τις αποφάσεις που λαμβάνονται σχετικά με την τοποθέτηση του μπλοκ , μετά της αποδέσμευση του από την εφαρμογή.

Ενοποίησης/Διαχωρισμού :

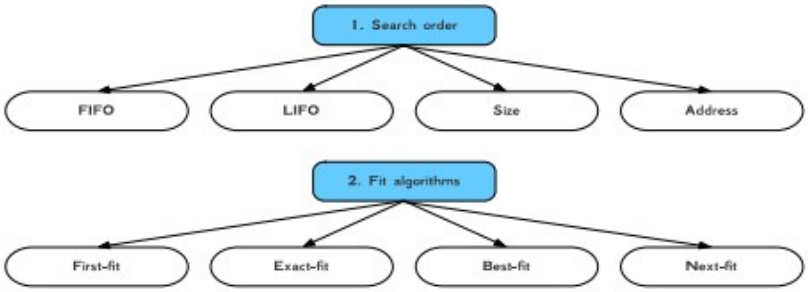
Συμπεριλαμβάνει τα δέντρα στα οποία κατηγοριοποιούνται σχεδιαστικές αποφάσεις που αφορούν την ενοποίηση ή τον διαχωρισμό ελεύθερων μπλοκ κατά την αποδέσμευση ή την δέσμευση ενός. Το δέντρο μεγεθών μπλοκ, ορίζει τα μεγέθη που χρησιμοποιεί ο διαχειριστής μνήμης ώς ελάχιστα ή μέγιστα αντίστοιχα κριτήρια για τις συγκεκριμένες λειτουργίες. Τα δέντρα συχνότητας ενοποίησης/διαχωρισμού αναφέρονται στον χρονική συχνότητα που , ανάλογα με την εσωτερική σχεδίαση (τεχνικές instant/deferred split-coalescse), πραγματοποιούνται τέτοιες λειτουργίες , ενώ τα δέντρα πυροδότητης (triggering) στα γεγονότα που προκαλούν την εκτέλεση των παραπάνω λειτουργιών , ενώ το δέντρο κατεύθυνσης ορίζει την δυνατότητα τοποθέτησης μπλοκ μνήμης που δημιουργούνται από ενοποίηση/διαχωρισμό άλλων , σε διαφορετικούς υποχώρους.



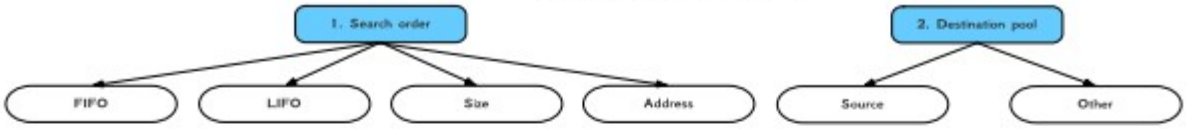

D. Inter-heap de-allocation parameters
1. Free-block movement strategy
w/o ownership
w/ ownership
2. Destination
Source Heap
Global Heap
Other Heap
C. Inter-heap fragmentation parameters (Heap blow-up)
1. Free-block movement strategy
w/o
w/
1%
...
100%
2. number of moved free-blocks
One
...
All


Intra-heap design parameters
F. Block structure parameters
1. Structure
Single-linked list
Double-linked list
Dynamic array
2. Sizes
One size
Many sizes
Non-specific
3. Recorded info
Size
Status
Pointers
4. Tags
Single-linked list
Double-linked list
Dynamic array
G. Pool organization parameters
1. Structure
Single-linked list
Double-linked list
Dynamic array
2. Structure based on block size
Single pool
One pool per size
3. Structure based on block order
Single pool
One pool per order
4. Structure based on block address
Single pool
One pool per address

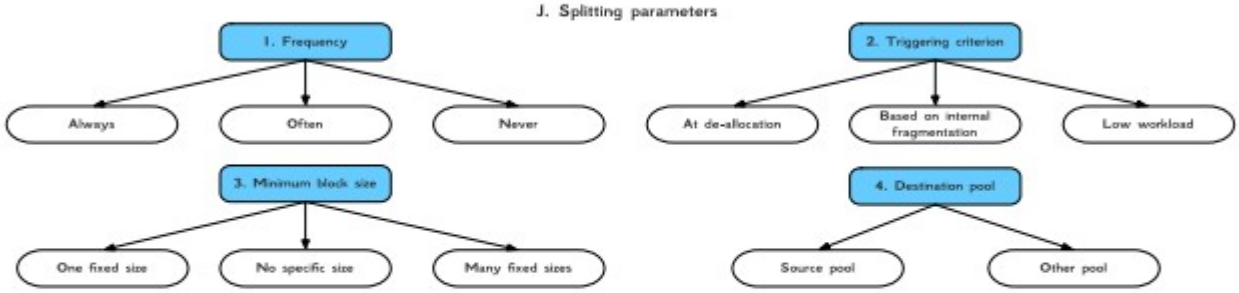
H. Block allocation parameters



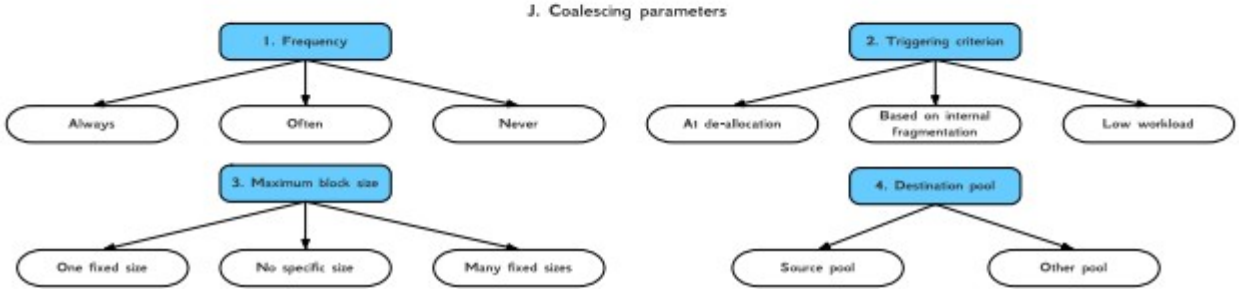
I. Block de-allocation parameters



J. Splitting parameters



J. Coalescing parameters



Κατηγορίες διανηματικών παραμέτρων

Αρχιτεκτονικές Παράμετροι:

Οι σχεδιαστικές αποφάσεις που αφορούν την οργάνωση των υποχώρων μνήμης ενός πολυνηματικού διαχειριστή ομαδοποιούνται σε δύο ορθογώνια δέντρα : αρχτεκτονικής heap και καθολικού heap. Στην πρώτη περίπτωση, το δέντρο αρχιτεκτονικής καθορίζει την εσωτερικη οργάνωση του heap. Σε αντιστοίχιση με την ταξινόμηση διαχειριστών ως προς πολυνηματικές εφαρμογές, η απόφαση single heap ορίζει ότι ο διαχειριστής θα ανήκει στην πρώτη κλάση της ταξινόμησης (single heap), ενώ η απόφαση private heaps ορίζει τουλάχιστον τη ύπαρξη ισάριθμων διαθέσιμων υποχώρων με τον αριθμό των νημάτων.Σημειώνεται ότι αποφάσεις του εν λόγω δέντρου, ενδέχεται να οδηγούν σε εσωτερικές οργανώσεις διαχειριστών που δεν εμπίμπτουν σε κάποια κλάση ταξινόμησης. Ως τέτοιο παράδειγμα οργάνωσης, μπορούμε να αναφέρουμε την ύπαρξη περισσότερων του ενός υποχώρων, αλλά όχι ισάριθμων με τον αριθμό των νημάτων με παροχή δυνατοτήτων παραλληλισμού επιπέδου νήματος (thread-level parallelism) μικρότερης από την θεωρητικά μέγιστη.Το δέντρο καθολικού χώρου ορίζει την ύπαρξη ή μη, μοναδικού καθολικού χώρου για την εξυπηρέτηση αιτημάτων από όλα τα νήματα, τακτική στην οποία ενδέχεται να καταφύγει ο διαχειριστής όταν οι τοπικοί υποχώροι αδυνατούν να ικανοποιήσουν τα αιτήματα των νημάτων.

Παράμετροι συνάφειας/ακεραιότητας δεδομένων:

Οι αποφάσεις ομαδοποιούνται σε δέντρα απόφασης συνάφειας και μηχανισμών συγχρονισμού. Για την πρώτη περίπτωση, εφόσον αναφερόμαστε σε πολυπύρηνες πλατφόρμες με κατανεμημένες caches που εκτελούν πολυνηματικές εφαρμογές , πρέπει να διασφαλίζεται η επιθυμητή ιδιότητα της συνάφειας μνήμης, χωρίς να επηρεάζονται οι σημασιολογικές ιδιότητες της εφαρμογής. Σημείωνεται, ότι οι συγκεκριμένες αποφάσεις λαμβάνονται συνήθως από τον κατασκευαστή της φυσικής αρχτεκτονικής υλικού και όχι από τον σχεδιαστή του δυναμικού διαχειριστή μνήμης. Το δέντρο μηχανισμών συγχρονισμού χειρίζεται τις αποφάσεις που αφορούν την ατομικότητα των προσβάσεων στα μεταδεδομένα ενός πολυνηματικού διαχειριστή μνήμης , ώστε να διασφαλίζεται η ακεραιότητα τους. Το πρόβλημα της ατομικότητας των προσβάσεων, αντιμετωπίζεται με λύσεις που εκτίνονται από την εκμετάλλευση της παρεχόμενης από την αρχιτεκτονική , ατομικότητας (platform dependent locking) έως την αξιοποίηση αντίστοιχων δυνατοτήτων που παρέχονται από στοίβες λογισμικού (glibc futexes - pthread mutex locks / pthread spinlocks) , με σκοπό την διευκόλυνση συγγραφής μεταφέρσιμου (portable) και , ανεξάρτητου από την αρχιτεκτονική (platform independent) κώδικα διαχειριστή μνήμης. Ως παραδείγματα τέτοιων στοιβών , αναφέρονται οι περιπτώσεις των pthread mutexes , με ορισμό κρίσιμων περιοχών (critical sections) στο εσωτερικό του κώδικα , και μη χρησιμοποίηση της CPU κατά την αναμονή του

νήματος για πρόσβαση στην περιοχή ή pthread spinlocks , οπότε η CPU χρησιμοποιείται εξαντλητικά μέχρι την απόκτηση του κλειδώματος περιοχής και , συνεπώς , χρησιμοποιείται όταν ο σχεδιαστής κρίνει ότι η αναμονή θα είναι αρκούντως μικρή (minimal lock contention).

Παράμετροι διανηματικής εξυπηρέτης αιτημάτων δέσμευσης:

Το σύνολο των αποφάσεων που αφορούν όλα τα δυνατά σχήματα αντιστοίχισης αιτημάτων δέσμευσης νημάτων σε υποχώρους μνήμης. Στα εν λόγω σχήματα, ανεξάρτητα από την αλγοριθμική υλοποίηση τους (hashing) διακρίνονται δύο οριακές περιπτώσεις :

ένας υποχώρος ανά νήμα: Για κάθε νήμα, ο διαχειριστής κατευθύνει τα αιτήματα εξυπηρέτησης σε μοναδικό υποχώρο , εξασφαλίζοντας καθολική ένα προς ένα αντιστοίχιση νημάτων-υποχώρων. Για την υλοποίηση τέτοιου τύπου αντιστοίχισης ο σχεδιαστής του δυναμικού διαχειριστή μνήμης μπορεί να καταφύγει, ομοίως με την κατηγορία παραμέτρων συνάφειας/ακεραιότητας, είτε σε λύσεις υποστηριζόμενες από την αρχιτεκτονική (hardware implementated TLS local storage) ή είτε σε λύσεις που παρέχονται από software βιβλιοθήκες πολυνηματισμού και χρησιμοποιούν τον αναγνωριστή (ID) του νήματος .

ένας κοινός χώρος για όλα τα νήματα: Δεν χρησιμοποιείται κανενός είδους σχήμα αντιστοίχισης και όλα τα αιτήματα δέσμευσης εξυπηρετούνται από μοναδικό χώρο.

mixed : Στην περίπτωση αυτή, υλοποιείται σχήμα αντιστοίχισης νημάτων σε υποχώρων , με τον αριθμό των τελευταίων να είναι μικρότερος από τον αριθμό των πρώτων. Τόσο ο αριθμός των υποχώρων και ο μηχανισμός αντιστοίχισης μπορεί να γίνεται δυναμικά κατά τον χρόνο εκτέλεσης, είτε στατικά κατά τον χρόνο μεταγλώττισης.

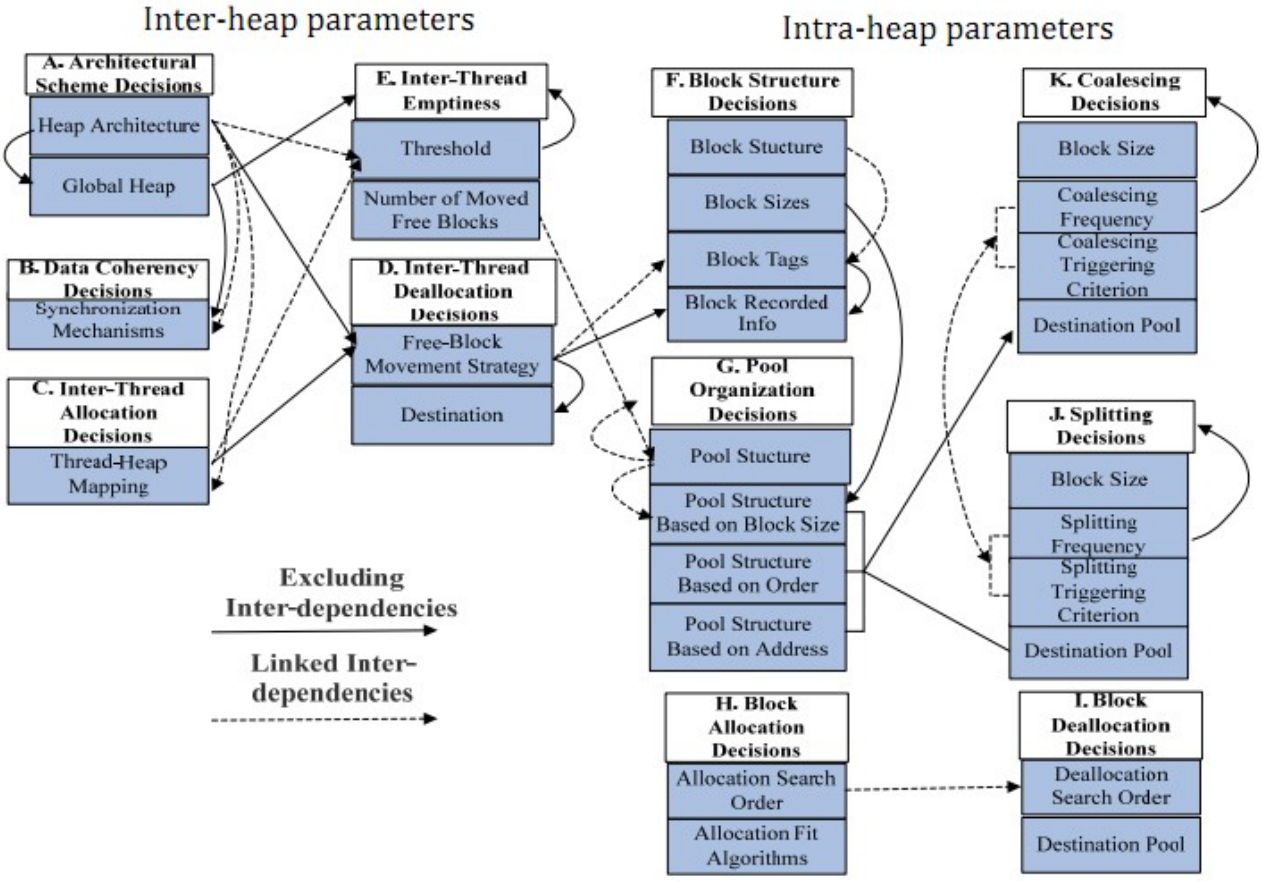
Παράμετροι διανηματικής εξυπηρέτησης αιτημάτων αποδέσμευσης:

Στην κατηγορία εντάσσονται και οργανώνονται σε δύο ορθογώνια δέντρα, οι μηχανισμοί που απαιτούνται για την εξυπηρέτηση αιτημάτων αποδέσμευσης μνήμης. Η στρατηγική μετακίνησης ελεύθερων μπλοκ μεταξύ υποχώρων μνήμης διαφορετικών νημάτων είτε μετά την απελευθέρωση τους από την εφαρμογή είτε μετά από απόφαση του διαχειριστή μνήμης ώστε να βελτιώνεται κάποια εσωτερική μετρική επίδοσης, επιβάλλει την αναγκαιότητα ύπαρξης μεταδεδομένων αναπαράστασης της ιδιοκτησίας, δηλαδή αντιστοίχισης ελεύθερου μπλοκ σε υποχώρο. Τέτοιου είδους μηχανισμοί χρησιμοποιούνται, αν ο διαχειριστής εξασφαλίζει δυναμικά τον περιορισμό των φαινομένων blowup και false sharing. Ανάλογα με τη φύση των δομών δεδομένων που αναπαριστούν τα μπλοκ ελεύθερης μνήμης, ο μηχανισμός ιδιοκτησίας μπορεί να επιφέρει ή όχι επιβάρυνση στο μέγεθος των μεταδεδομένων καθώς κόστος συγχρονισμού κατά τον χρόνο εκτέλεσης πχ όταν ο διαχειριστής λαμβάνει αίτημα αποδέσμευσης για μπλοκ μνήμης ελεύθερης λίστας (freelist), χρησιμοποιεί την πληροφορία ιδιοκτησίας για να το τοποθετήσει σε υποχώρο μνήμης, ανάλογα με την κλάση πολυνηματικότητας (private heaps, private heaps with

ownership), στην οποία ανήκει ενώ αν το αίτημα αποδέσμευσης αφορά μπλοκ μνήμης που αναπαρίσταται από bitmaps, δεν προκύπτει κόστος τήρησης μεταδεδομένων.

Παράμετροι διανηματικής αντιμετώπισης κατακερματισμού:

Η κατηγορία αποτελείται από δύο ορθογώνια δέντρα, στα οποία ομαδοποιούνται οι αποφάσεις που αφορούν την αντιμετώπιση του φαινομένου του blowup που εμφανίζεται σε πολυνηματικούς διαχειριστές μνήμης με πολλαπλούς υποχώρους μνήμης. Το δέντρο κατωφλίου καθορίζει την ύπαρξη ή μη, μηχανισμών, που εξασφαλίζουν είτε στατικά είτε κατά τον χρόνο εκτέλεσης την μεταφορά μπλοκ ελεύθερης μνήμης από τοπικούς υποχώρους σε έναν καθολικό , με σκοπό είτε την απευθείας εξυπηρέτηση των αιτημάτων των νημάτων από τον καθολικό χώρο, είτε τον επαναδιαμοιρασμό της στους υπόλοιπους υποχώρους. Σε κάθε τοπικό χώρο μπορεί να αντιστοιχίζονται είτε διαφορετικές στατικές τιμές κατωφλίων είτε διαφορετικές δυναμικές πολιτικές αλλαγής της κατά τον χρόνο εκτέλεσης , ανάλογα με την εσωτερική οργάνωση του υποχώρου και τις απαιτήσεις της εφαρμογής.Το δέντρο αριθμού μετακίνησης ελεύθερων μπλοκ , καθορίζει τις ποσότητες μετακίνησης ελεύθερης μνήμης μεταξύ τοπικών υποχώρων και καθολικού.

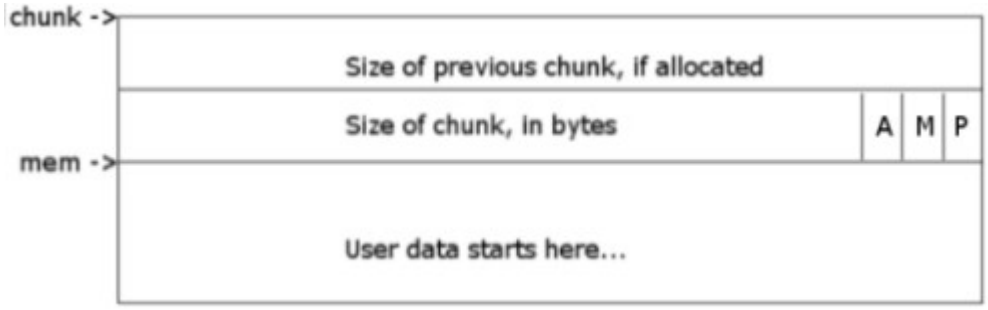


5 Σύγχρονοι διαχειριστές

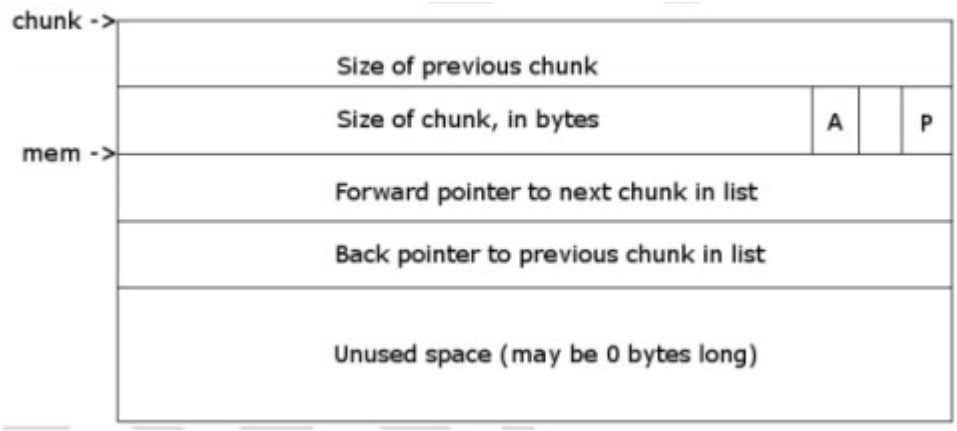
ptmalloc2

Ο διαχειριστής δυναμικής μνήμης ptmalloc2 αποτελεί επέκταση του μονονηματικού διαχειριστή dlmalloc, με βελτιωμένες δυνατότητες ως προς την εξυπηρέτηση αιτημάτων δέσμευσης/αποδέσμευσης μνήμης, και είναι ο βασικός διαχειριστής που παρέχεται με την βιβλιοθήκη GNU libc. Για την αναπαράσταση ελεύθερου χώρου, χρησιμοποιεί κυρίως μπλοκ μνήμης, οργανωμένων σε στοιχεία διπλά συνδεδεμένων λιστών. Για δεσμευμένα από την εφαρμογή μπλοκς οι δείκτες αφαιρούνται, προσθέτοντας έτσι περισσότερο χώρο στο μπλοκ.

Memory layout δεσμευμένου μπλοκ.



Memory layout ελεύθερου μπλοκ.



Τα παραπάνω μπλοκ οργανώνοται σε exact size class λίστες προκαθορισμένου μεγέθους για γρήγορη εξυπηρέτηση σε σταθερό χρόνο, είτε σε διπλές λίστες με κλασικούς αλγορίθμους αναζήτησης.

Για τον χειρισμό πολυνηματικότητας, παρέχονται μεγαλύτεροι υποχώροι μνήμης χωρίς ακριβή αντιστοίχιση υποχώρων-νημάτων, οργανωμένοι σε διπλά συνδεδεμένες λίστες Κάθε νήμα απευθύνεται στον υποχώρο από τον οποίο εξυπηρετήθηκε το τελευταίο αίτημα του.Αν το κλείδωμα στον υποχώρο είναι ανεπιτυχές, τότε διατρέχεται ολόκληρη η διπλή λίστα υποχώρων μέχρις προσπάθεια κλειδώματος να είναι επιτυχής. Αν και αυτή η προσπάθεια είναι ανεπιτυχής, τότε

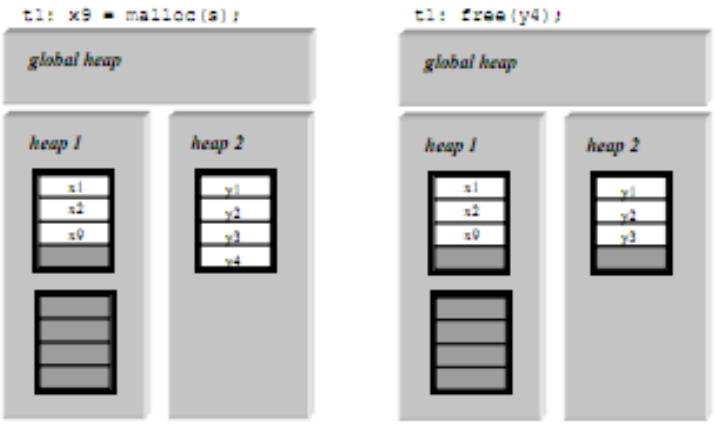
δημιουργείται νέος υποχώρος που εξυπηρετεί το αίτημα και εισάγεται στην λίστα.

Hoard

Ο hoard μπορεί να περιγραφεί σχεδιαστικά ώς ένας διαχειριστής που αποφεύγει την εμφάνιση των φαινομένων false sharing και ανταλλάσει αυξημένη δέσμευση μνήμης προκειμένου να μειώσει κόστη συγχρονισμού λόγω αιτημάτων προερχόμενων από πολυνηματικές εφαρμογές , χρησιμοποιώντας πολλαπλούς υποχώρους μνήμης και έναν καθολικό. Κάθε νήμα μπορεί να ανακατευθήνει αίτημα δέσμευσης στον δικό του υποχώρο ή στον καθολικό. Για την διαμόρφωση των εσωτερικών του δομών, ο hoard δεσμεύει μεγαλύτερα κομμάτια μνήμης απευθείας από το λειτουγικό σύστημα (superblocks) ίδιου μεγέθους , το οποίο είναι κάποιο πολλαπλάσιο σελίδων. Τα μπλοκ ελεύθερης μνήμης που περιέχονται σε ένα superblock είναι ίδιου μεγέθους b, γεγονός που περιορίζει την εμφάνιση του φαινομένου του internal fragmentation κατά μία τάξη μεγέθους b . Αιτήματα για μεγέθη μεγαλύτερα ή ίσα του μισού μεγέθους ενός superblock, εξυπηρετούνται με κλήσεις συστήματος. Σε κάθε υποχώρο μνήμης, που χρησιμοποιείται ανά νήμα, ανατίθεται ένα σύνολο από superblocks. Όταν κάποιο νήμα δεν εντοπίσει ελεύθερο χώρο για αίτημα δέσμευσης, ο διαχειριστής μετακινεί ένα superblock από τον καθολικό χώρο μνήμης, δεσμεύοντας superblock από το λειτουργικό σύστημα αν δεν βρεθεί αντίστοιχα superblock στον καθολικό χώρο, στον τοπικό υποχώρο του νήματος. Ο συγκεκριμένος διαχειριστής δεν υποστηρίζει λειτουργικές επιστροφής μνήμης στο λειτουργικό σύστημα (trimming).

Στον hoard αποτελεί σχεδιαστική επιλογή ο περιορισμός του φαινομένου του blowup με σταθερή χωρική πολυπλοκότητα. Για τον λόγο αυτό, η υλοποίηση του περιλαμβάνει μηχανισμό μεταφοράς superblocks, που πληρούν προυποθέσεις χωρητικότητας ελεύθερης μνήμης, από τους τοπικούς υποχώρους στον καθολικό, όταν η συνολική διαθέσιμη μνήμη ανά υποχώρο υπερβεί ένα προκαθορισμένο κατώφλι. Η εύρεση κατάλληλων τοπικών superblocks προς μεταφορά στον καθολικό χώρο υλοποιείται σε σταθερό χρόνο, με διαχωρισμό των superblocks ανά υποχώρο, σε ομάδες με κριτήριο την ελεύθερη μνήμη που περιέχουν.

Εξυπηρέτηση αιτημάτων χωρίς μεταφορά superblocks



Εξυπηρέτηση αιτημάτων με μεταφορά superblocks

jemalloc

Ο jemalloc αποτελεί έναν διαχειριστή μνήμης που αναπτύχθηκε έχοντας κατά νου συγκεκριμένες ιδέες και πολιτικές δέσμευσης. Οι ιδέες αυτές θα μπορούσαν συνοψιστούν ώς εξής:

Χωρισμός μικρών αντικειμένων σύμφωνα με κλάσεις μεγεθών και προτίμηση χαμηλών διευθύνσεων κατά την επαναχρησιμοποίηση αντικειμένων, πολιτική που χρησιμοποιήθηκε αρχικά στον phkmalloc διαχειριστή μνήμης.

Προσεκτική επιλογή κλάσεων αντικειμένων : Αν το διάστημα ανάμεσα σε δύο κλάσεις είναι αρκετά μεγάλο, τα αιτήματα εξυπηρετούνται με αρκετά μεγαλύτερη μνήμη από την απαιτούμενη προκαλόντας μη επιθυμητό internal fragmenation. Εάν, όμως, ένα χώρος μνήμης διαχωριστεί ώστε ο αριθμός των κλάσεων μεγεθών αιτημάτων που εξυπηρετεί αυξάνεται, θα αυξάνεται και το ποσοστό μνήμης που δεν χρησιμοποιείται για κανένα αίτημα, με συνέπεια την όξυνση του external fragmenation.

Επιβολή αυστηρών ορίων στο μέγεθος των χρησιμοποιούμενων μεταδεδομένων: Προσπερνώντας και τα δύο είδη fragmenation, ο jemalloc περιορίζει το μέγεθος των μεταδεδομένων σε λιγότερο από 2% , για όλες τις κλάσεις μεγεθών.

Ελαχιστοποίηση του ενεργού συνόλου σελίδων : Οι πυρήνες λειτουργικών συστημάτων διαχειρίζονται την εικονική μνήμη σε μεγέθη σελίδων, άρα είναι σημαντικό τα δεδομένα που χρησιμοποιεί η εφαρμογή καθώς και τα μεταδεδομένα του διαχειριστή μνήμης να είναι συγκεντρωμένα σε όσο το δυνατόν μικρότερο αριθμό σελίδων . Η παρατήρηση είναι σημαντική ειδικά για την περίπτωση εφαρμογών που χρησιμοποιούν μεγάλο σύνολο δεδομένων κατά τη λειτουργία τους, οπότε και οι αποφάσεις του διαχειριστή ενδέχεται να επηρεάσουν ή όχι την εμφάνιση φαινομένων swapping μεταξύ κεντρικής μνήμης και δίσκου.

Ελαχιστοποίηση του πιθανού ανταγωνισμού κλειδωμάτων υποχώρων για πολυνηματικές εφαρμογές : Οι πρώτες εκδόσεις του διαχειριστή jemalloc χρησιμοποίησαν πολλαπλούς υποχώρους για την μείωση του ανταγωνισμού στην περίπτωση πολυνηματικών εφαρμογών, δίχως να υπάρχει ένα προς ένα αντιστοίχιση νήματος-υποχώρου.

Με την πάροδο του χρόνου, έγινε προφανές ότι η σχεδιαστική επιλογή των πολλαπλών , ασύνδετων με νήματα , υποχώρων υστερούσε σε επίδοση έναντι της επιλογής ύπαρξης ισάριθμων με τα νήματα υποχώρων με στατική αντιστοίχιση (thread-specific caching), οπότε στις μετέπειτα εκδόσεις ακολουθείται συνδυασμός αυτών του μοντέλου εξυπηρέτησης αιτημάτων από πολυνηματικές εφαρμογές.

Ο jemalloc διαχωρίζει τα αντικείμενα σε 3 κύριες , μη πλήρεις , κλάσεις κατηγοριών μεγεθών :

Small : [8] , [16,32,48,...,128] , [192, 256, 320, ..., 512], [768, 1024, 1280, ..., 3840]

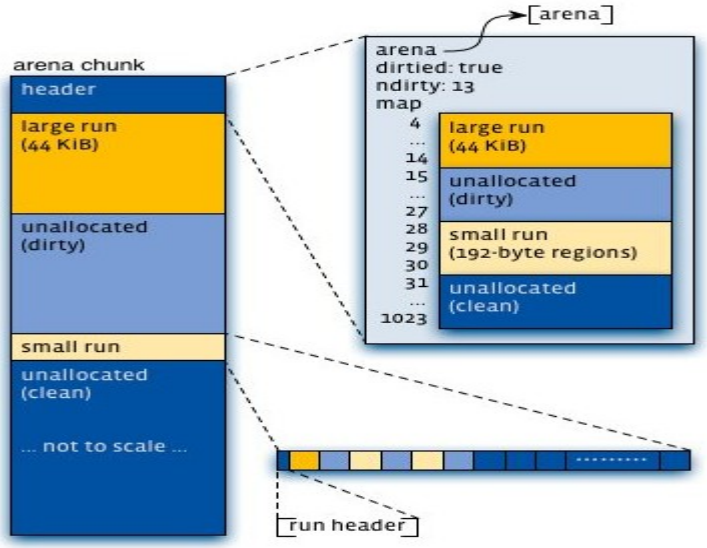
Large : [4 KiB, 8 KiB, 12 KiB, ..., 4072 KiB]

Huge : [4 MiB, 8 MiB, 12 MiB, ...]

Η εικονική μνήμη θεωρείται νοητικά διαχωρισμένη σε κομμάτια μεγέθους δύναμης του δύο, κατά συνέπεια ο εντοπισμός των μεταδεδομένων για τις δύο πρώτες κατηγορίες γίνεται σε σταθερό χρόνο με λογικές (and) λειτουργίες σε δείκτες, ενώ για την τρίτη κατηγορία τα μεταδεδομένα εντοπίζονται με λειτουργίες αναζήτησης σε red-black δέντρο.

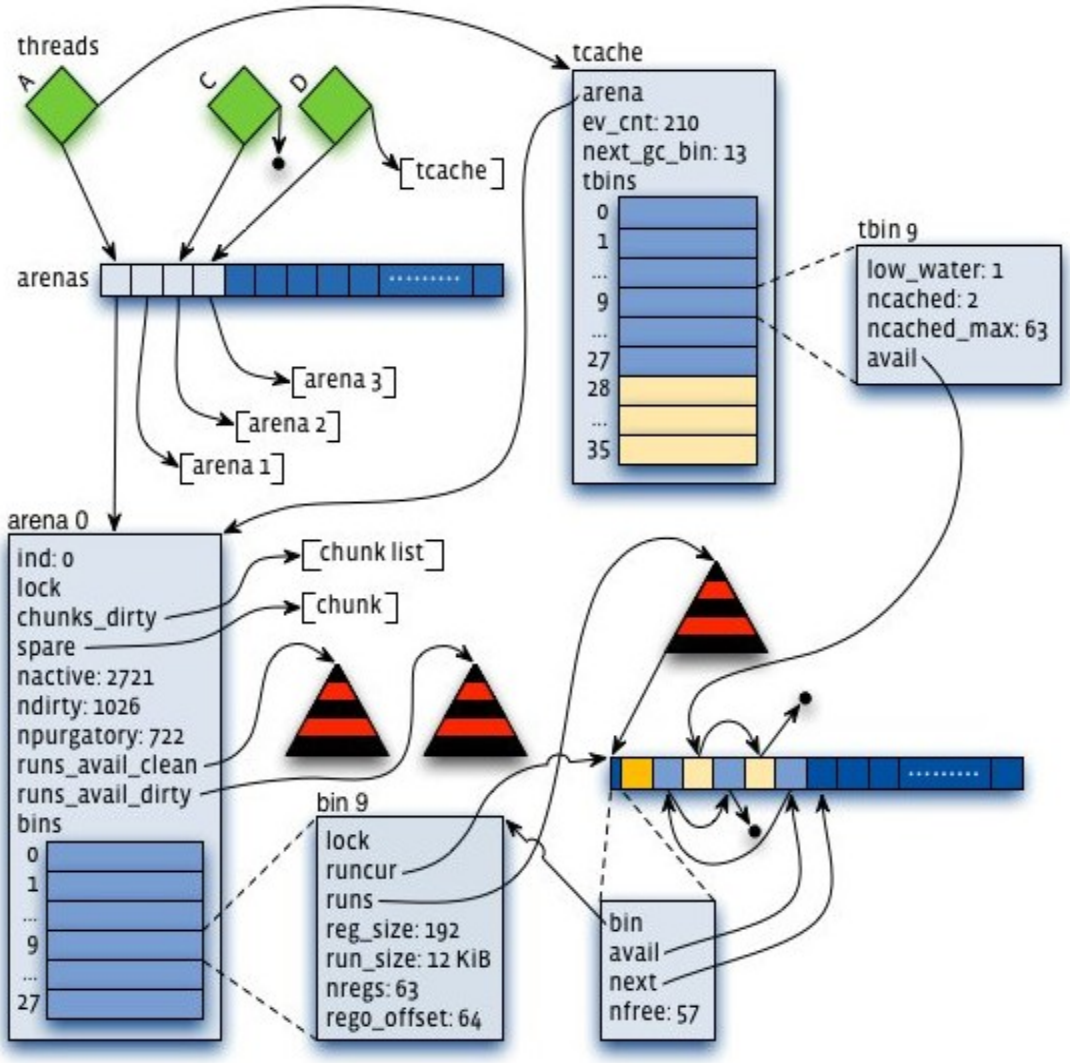
Οι υποχώροι μνήμης ανατίθονται σε νήματα μέσω round-robin αλγορίθμου, ενώ είναι ανεξάρτητοι μεταξύ τους. Κατά την αποδέσμευση μπλοκ μνήμης από κάποιο νήμα, το μπλοκ επιστρέφεται στον υποχώρο στον οποίο ανήκει.

Εσωτερικά arenas



Πέρα από τους υποχώρους-αρένες, για κάθε νήμα διατίθεται, μέσω μηχανισμών thread caching, ένας υποχώρος-cache , που περιέχει μπλοκ ελεύθερης μνήμης, μέχρι κάποιο όριο. Με τον τρόπο αυτό , εξασφαλίζεται ότι ένα μεγάλο μέρος των αιτημάτων θα κατευθυνθεί σε υποχώρους-cache χωρίς απαιτούμενο συγχρονισμό , πριν ανακατευθυνθεί σε υποχώρους-αρένες , όπου θα χρειαστούν κλειδώματα. Ο μέγιστος αριθμός μπλοκ ελεύθερης μνήμης στους υποχώρους-caches περιορίζεται στατικά.

Πολυνηματική διαχείριση αιτημάτων στον jemalloc :



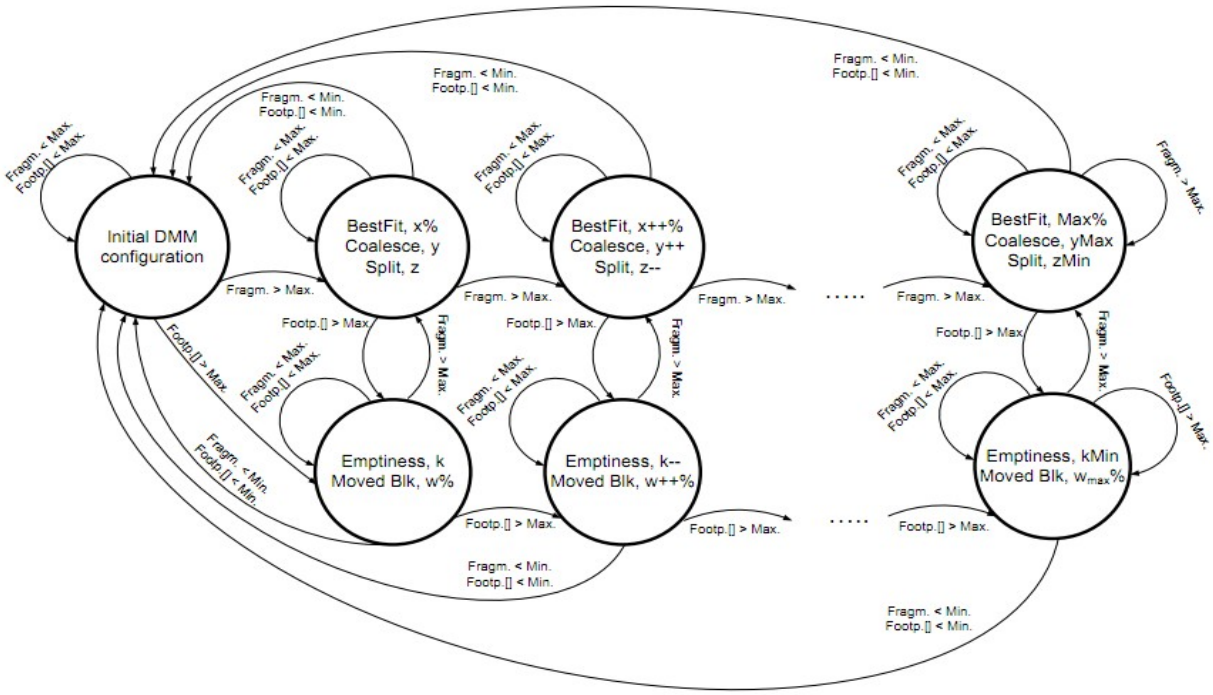
Όπως φαίνεται στο σχήμα, για την βελτίωση του ρυθμού εξυπηρέτησης αιτημάτων πολυνηματικών εφαρμογών έχουν χρησιμοποιηθεί τόσο thread local caches όσο και πολλαπλά arenas.

6 Περιπτώσεις runtime adaptivity

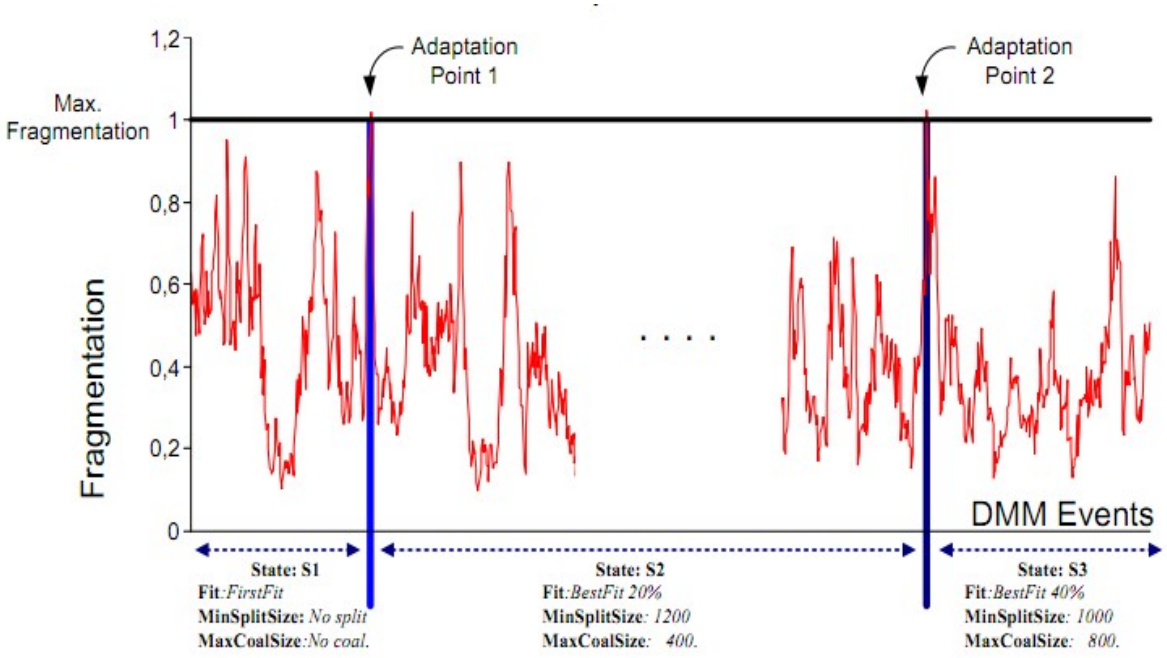
Αναπροσαρμογή παραμέτρων δομών freelists με βάση το external fragmenation

Εάν ένας διαχειριστής μνήμης χρησιμοποιεί freelists μπλοκ μνήμης, χρησιμοποιεί ως μετρική μία ποσοτική εκτίμηση του external fragmentation προκειμένου να αναπροσαρμόζει χαρακτηριστικά των λειτουργιών αναζήτησης (fit policies) και των μηχανισμών διαχωρισμού/ένωσης (split/coalesce) κατά δέσμευση ή αποδέσμευση ενός μπλοκ μνήμης. Καθώς η παρουσία ή μή external fragmenation επηρεάζει τον χρόνο εκτέλεσης, λόγω των πολλαπλών βημάτων αναζήτησης σε λίστα (hops), τον αριθμό προσβάσεων στη μνήμη, αφού κάθε αποτυχημένο βήμα οδηγεί σε επόμενη προσπέλαση, και την πίεση στην ιεραρχία κρυφών μνημών που συνεπάγεται η ανάγνωση επικεφαλίδων μπλοκ που δεν αντιστοιχούν σε τερματισμό της αναζήτησης. Καθώς η ανάγνωση περαιτέρω μεταδεδομένων και η εσωτερική τους επεξεργασία επιβαρύνει τον χρόνο εξυπηρέτησης, αξιολογείται η ισορροπία κέρδους λόγω αναπροσαρμογής με την απώλεια λόγω της λειτουργίας ελεγκτή.

Ένα παράδειγμα FSM προσαρμοστικού ελεγκτή είναι το ακόλουθο:



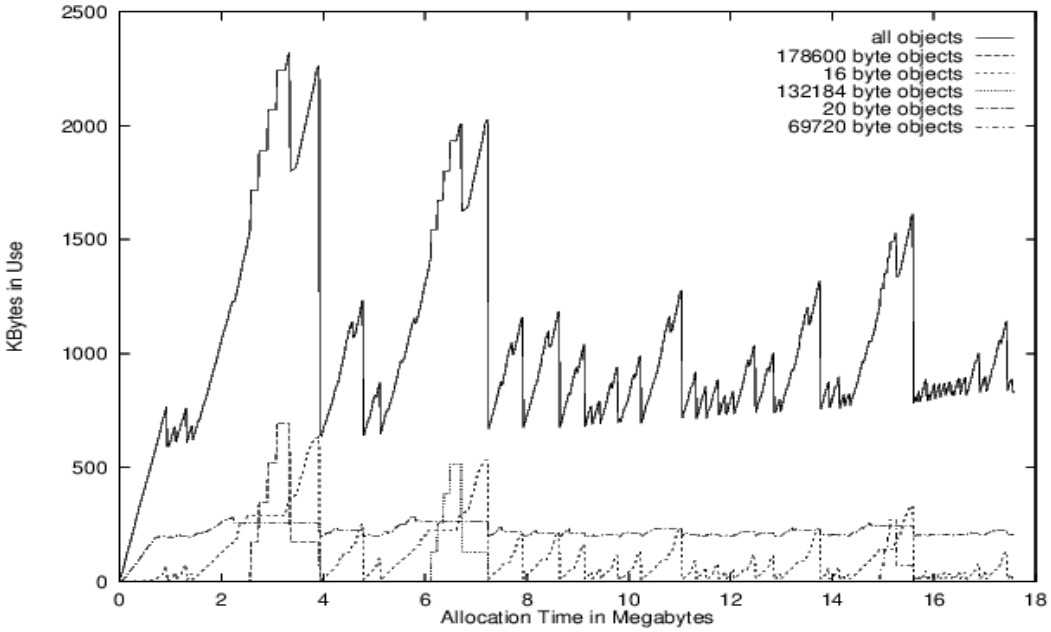
Η εκτέλεση μιας εφαρμογής, με διαχειριστή μνήμης που ενσωματώνει μπορεί να διαχωριστεί σε φάσεις ανάλογα με την τρέχουσα κατάσταση στην οποία βρίσκεται το FSM. Όταν η τιμή μιας εσωτερικής μετρικής αυξάνεται ή μειώνεται πέρα από στατικά καθορισμένες τιμές, προκαλείται μετάβαση σε διαφορετική φάση.



Διαχειριστές που ακολουθούν το μοτίβο δεσμεύσεων

Στην περίπτωση αυτή, οι διαχειριστές δημιουργούν δυναμικά δομές που ανταποκρίνονται πιο αποδοτικά έναντι της χρησιμοποίησης γενικών δομών δεδομένων, στα μεγέθη των αιτημάτων που δέχονται.

Για ένα πιθανό προφίλ δεσμεύσεων :



ένας προσαρμοστικός διαχειριστής, θα μπορούσε να αντικαταστάσει γενικά freelist που καλύπτουν εύρη μεγεθών, με exact free lists μεγεθών που ζητούνται, εξασφαλίζοντας ιδανικά σταθερό χρόνο εξυπηρέτησης (worst case ή amortized).

7 Σχεδιασμός και υλοποίηση προτεινόμενου διαχειριστή μνήμης

Αναπαράσταση ελεύθερου χώρου

Κατά την σχεδίαση του διαχειριστή μνήμης της παρούσας διπλωματικής , για την αναπαράσταση του χώρου τον οποίο διαχειρίζεται, χρησιμοποιούμε απλές ακολουθίες bits (bitmaps). Θεωρούμε, ότι ο χώρος μνήμης είναι διαμερισμένος σε κελιά (cells) προκαθορισμένου μεγέθους ,τα οποία σημειώνονται ως κατειλλειμένα ή ελεύθερα με τιμές 0 και 1 , αντίστοιχα. Έτσι , η συνολική κατάσταση του χώρου μνήμης μπορεί να περιγραφεί πλήρως από την κατάσταση των κελιών που τον συνθέτουν, πληροφορία που αποθηκεύεται ως πίνακας. Κάθε στοιχείο του πίνακα, αντιστοιχεί σε μία υποακολουθία μεγέθους ίσου με τον αριθμό των bits του στοιχείου (πχ 32 για uint32_t , 64 για uint64_t κοκ).

Αν διαθέτουμε την διεύθυνση του πρώτου byte ενός χώρου, για την αντιστοίχιση ενός bit στοιχείου πίνακα με ένα κελί μνήμης θεωρώντας :

base mem : διεύθυνση αναπαριστόμενου χώρου

bmap : πίνακας

index: δείκτης στοιχείου πίνακα

j : θέση bit στο εσωτερικό στοιχείο , θεωρώντας το λιγότερο σημαντικό bit αντιστοιχεί στην θέση 0

resolution : αριθμός αναπαριστόμενων bytes ανά κελί

BMAP_EL_TYPE : τύπος στοιχείου πίνακα

sizeof : τελεστής της C , που επιστρέφει το μέγεθος του ορίσματός του σε bytes

BMAP_EL_SIZE_BITS : αριθμός των bits στοιχείου πίνακα , που προκύπτει ως sizeof(BMAP_EL_TYPE)

προκύπτει το σχήμα

bmap[index] , #bit j , bytes_per_cell

base mem + (index * BMAP_EL_SIZE_BITS + j) * resolution.

Αντίστροφα, για την αντιστοίχιση της διεύθυνσης του πρώτου byte ενός κελιού μνήμης, έστω mem, σε bit στοιχείου πίνακα, χρησιμοποιείται το σχήμα

mem

cell_no = (mem - base_mem) / resolution

index = cell_no DIV BMAP_EL_SIZE_BITS

bit_pos = cell_no MOD BMAP_EL_SIZE_BITS

Τα δύο βασικά στοιχεία της αναπάραστασης, δηλαδή ο πίνακας bitmap και κάποιος αναπαριστόμενος χώρος, ομαδοποιούνται σε υποχώρους μνήμης, οι οποίοι στην υλοποίηση ονομάζονται raw blocks. Ανάλογα με την απαιτούμενη λειτουργικότητα τους, και συγκεκριμένα, με την δυνατότητα τους να απαντούν σε αιτήματα δέσμευσης/αποδέσμευσης μνήμης με ένα ή πολλαπλά κελιά μνήμης , χωρίζονται σε δύο κατηγορίες: BITMAP και BITMAP_SINGLE.

BITMAP :

Για κάθε αίτημα μνήμης το μέγεθος του αιτήματος διαιρείται με το resolution του raw block και αντιστοιχίζεται σε αριθμό από cells. Λόγω του ότι η συγκεκριμένη κατηγορία raw block μπορεί να εξυπηρετήσει requests για πολλαπλούς αριθμούς από cell, είναι απαραίτητο να αποθηκεύεται ο αριθμός των cells που εξυπηρέτησαν το request. Για τον λόγο αυτό, το μέγεθος του αιτήματος έχει αυξηθεί κατά μία ποσότητα που επαρκεί για την αποθήκευση του εν λόγω αριθμού (στην παρούσα υλοποίηση φέρει το όνομα chunk_header).

malloc

(1) Εύρεση διεύθυνσης του πίνακα bitmap

(2) Εύρεση των bytes_per_cell

(3) Εύρεση του base_mem

(4) Αύξηση του μεγέθους κατά chunk header

(5) Διαίρεση (ceil) του μεγέθους με το resolution , ώστε να προκύψει ο συνολικός απαιτούμενος αριθμός από cells

(6) Αναζήτηση ακολουθίας συνεχόμενων 1 bits στον πίνακα bitmap (το LSbit του στοιχείου bitmap[index+1] θεωρείται το επόμενο του MSBit του bitmap[index].

(7) Εάν δεν ευρεθεί τέτοια ακολουθία , επιστροφή τιμής NULL.

Διαφορετικά , τα bits της ευρεθείσας ακολουθίας τίθονται ίσα με 0 στο bitmap.

(8) Εύρεση της διεύθυνσης του πρώτου byte με τον παραπάνω αλγόριθμο αντιστοίχισης.

(9) Εγγραφή του αριθμού των cells σε sizeof(chunk_header_t) bytes

(10) Επιστροφή της διεύθυνσης του πρώτου byte αντιστοίχισης , προσαυξημένης κατά sizeof(chunk_header_t).

free

(1) Εύρεση διεύθυνσης του πίνακα bitmap

(2) Εύρεση του resolution

(3) Εύρεση του base_mem

(4) Μείωση του ptr κατά sizeof(chunk_header_t)

(5)Ανάγνωση του αριθμού των cells στον οποίο αντιστοιχεί το συγκεκριμένο κομμάτι μνήμης.

(6) Εύρεση cell_no : Έστω mem η διεύθυνση που προέκυψε από τη αφαίρεση του βήματος 3 .

cell_no = (base_mem - mem) / resolution

(7) Εύρεση του index του πίνακα bitmap στο οποίο βρίσκεται το πρώτο bit της ακολουθίας από 0 , τα οποία απαρτίζουν το chunk

index = cell_no DIV BMAP_EL_SIZE_BITS

(8) Εύρεση του πρώτου bit της ακολουθίας (βρίσκεται στο bmap[index])

first_bit_pos = cell_no MOD BMAP_EL_SIZE_BITS

(9) Τα 0 bits , αριθμού που προέκυψε από το βήμα (3) που ξεκινούν από τη θέση του βήματος (6) τίθονται ίσα με 1.

BITMAP_SINGLE:

Στην περίπτωση αυτή , κάθε αίτημα μνήμης εξυπηρετείται από ένα cell μνήμης του αναπαριστόμενου χώρου , το οποίο είναι αρκετά μεγάλο ώστε να καλύψει το μέγεθος του αιτήματος. Καθώς γνωρίζουμε ότι αναφερόμαστε σε ένα μοναδικό cell ανά bitmap_malloc ή bitmap_free , η κατάσταση του bitmap αναπαριστά πλήρως ότι πληροφορία χρειάζεται να γνωρίζουμε για την κατάσταση του χώρου και , κατά συνέπεια , δεν εγγράφουμε header μεταδεδομένα. Οι προαναφερόμενες λειτουργίες απλοποιούνται και γίνονται ώς εξής :

malloc

(1) Εύρεση διεύθυνσης του πίνακα bitmap

(2) Εύρεση resolution

(3) Εύρεση base_mem

(4) Αναζήτηση ενός bit με τιμή 1 στον πίνακα bitmap.

(5) Εάν δεν ευρεθεί τέτοιο bit , επιστροφή τιμής NULL.

Διαφορετικά , τα ευρεθέν bit τίθεται ίσα με 0 στο bitmap.

(6) Εύρεση της διεύθυνσης του πρώτου byte με τον αλγόριθμο αντιστοίχισης.

(7) Επιστροφή της διεύθυνσης του πρώτου byte αντιστοίχισης .

free

(1) Εύρευση διεύθυνσης του πίνακα bitmap

(2) Εύρεση resolution

(3) Εύρεση base mem

(4) Εύρεση του cell_no : (ptr - base_mem) / resolution

(5) Εύρεση index : cell_no DIV BMAP_EL_SIZE_BITS

(6) Εύρεση του θέσης του bit αντιστοίχισης : cell_no MOD BMAP_EL_SIZE_BITS

(7) Ανάθεση τιμής 1 στο bit αντιστοίχισης

Αναζήτηση ελεύθερου χώρου

Για τα παραπάνω είδη raw blocks και χρησιμοποιώντας bitmaps για την αναπαράσταση χώρων μνήμης το πρόβλημα της αναζήτησης ελεύθερου χώρου προκειμένου κάποιο request να εξυπηρετηθεί , έχει αναχθεί στο πρόβλημα αναζήτησης ακολουθίας από m συνεχόμενα 1 σε μια σειρά bits μήκους n > m.

Στην περίπτωση των BITMAP raw blocks προκειμένου να εκτελείται η αναζήτηση χρειάζεται να υλοποιηθεί ένα FSM το οποίο θα δέχεται σύμβολα 0,1 και θα τερματίζει όταν θα δεχθεί m 1 bits , με τις εξής ιδιομορφίες : (1) τα σύμβολα δίδονται διαδοχικά ως bits στοιχείων πίνακα (2) κατά τον τερματισμό του πρέπει να επιστρέφει τις αρχικές θέσεις της m-ακολουθίας (index πίνακα και θέση bit εντός του στοιχείου). Ο κώδικας για ένα τέτοιο FSM θα είχε τη μορφή :

```
static ind_pos search(BMAP_EL_TYPE *array ,size_t count ,int array_size)
{
        ind_pos ret;
        int i;
        unsigned int j;

        ret.pos = 0;
        ret.index = -1;

        int on_ace_streak = 0 ;
        size_t runner_count = 0;

        int count_from_i;
        unsigned int count_from_j;


        int single_pos = 0 ;

        if (count == 1) {
                for (i = 0; i < array_size; i++) {
                        single_pos = __builtin_ffsll(array[i]);
                        if (single_pos) {
                                ret.pos = single_pos - 1;
                                ret.index = i;
                                break;
                        }
                }
                return ret;
        }

        // count > 1
        BMAP_EL_TYPE current_bit;
        BMAP_EL_TYPE bmap_elem;
```

```
	for (i = 0; i < array_size; i++) {
		bmap_elem = array[i];
		for (j = 0;j < (sizeof(BMAP_EL_TYPE) * 8); j++) {
			current_bit = (bmap_elem >> j) & 0x1;
			if (current_bit) {
				if (on_ace_streak) {
					++runner_count;
					if (runner_count == count) {
						ret.index = count_from_i;
						ret.pos = count_from_j;
						return ret;
					}
				} else {
					on_ace_streak  = 1;
					runner_count = 1;
					count_from_i = i;
					count_from_j = j;
				}
			} else {
				on_ace_streak = 0;
				runner_count = 0 ;
			}
		}
	}

	return ret;
}
```

Σε BITMAP_SINGLE raw blocks , το ζητούμενο είναι η εύρεση της θέσης ενός (του πρώτου) set bit σε μία σειρά στοιχείων από bits. Λόγω του ότι το πρόβλημα είναι αρκετά συνηθισμένο σε πλήθος εφαρμογών , έχουν αναπτυχθεί για την επίλυση του πολύ πιο αποδοτικές λύσεις από τη σειριακή αναζήτηση με διαδοχικά shifts, ενώ αρκετές αρχιτεκτονικές παρέχουν ως μέρος του instruction set τους εντολή που εκτελεί την εν λόγω λειτουργία. Καθώς το συγκεκριμένο κομμάτι της υλοποίησης είναι εξαιρετικά κρίσιμο για την επίδοση του allocator εξετάζουμε μια σειρά από λύσεις που εμπλέκουν ή όχι το hardware.

Βελτίωση της επίδοσης της αναζήτησης

Roving index

Μετά την αναγωγή του προβλήματος της αναζήτησης ελεύθερου χώρου στην αναζήτηση ενός set bit σε κάποιο στοιχείο πίνακα, το ζήτημα που προκύπτει είναι από ποιο στοιχείο του πίνακα θα εκκινεί η αναζήτηση.Στην απλή περίπτωση , όπου εκτελούμε διαδοχικά ffs σε στοιχεία πίνακα από index 0 μέχρι N-1 (N ο αριθμός των στοιχείων του πίνακα) , ο χρόνος εκτέλεσης ενδέχεται να επιβαρύνεται εάν το allocation pattern έχει αφήσει αρκετά συνεχόμενα στοιχεία χωρίς κανένα "ελεύθερο" bit. Ένα τέτοιο allocation pattern θα μπορούσε να αντιστοιχεί σε κάποιο growing phase ακολουθούμενο από "ταλαντώσεις" malloc/free. Μία λύση στο προαναφερθέν πρόβλημα , αποτελεί η προσθήκη και χρησιμοποίηση του roving index, μιας ποσότητας που υποδεικνύει στη αναζήτηση από ποιο στοιχείο να ξεκινήσει , και στη συνέχεια να συνεχίσει σειρικά και κυκλικά μέχρι το στοιχείο roving_index - 1 (roving index > 0).

Όταν η αναζήτηση είναι επιτυχής το roving index ανανεώνεται στην τιμή του index , στο οποίο βρέθηκε set bit.

Recommendation stack

Όταν τα μοτίβα δέσμευσης καθίστανται ακόμα πιο τυχαία, μία δομή που ενδεχομένως θα μπορούσε να βοηθήσει στην μείωση του χρόνου αναζήτησης είναι το recommendation stack. Κάθε recommendation αποτελεί ένα tuple (index, position) που "δείχνει" ένα set bit εντός στοιχείου του bitmap πίνακα και το recommendation stack μία στοίβα από τέτοια tuples, με συνολικό μέγεθος που καθορίζεται είτε στατικά είτε κατά την δημιουργία του raw block.Η δομή συμπληρώνεται με κάποιον ακέραιο, που δείχνει την κορυφή της στοίβας (stack top index) ενώ αν χρησιμοποιηθεί ο ψευδοκώδικας

malloc

(1) Εύρεση διεύθυνσης του πίνακα bitmap.

(2) Εύρεση resolution.

(3) Εύρεση base_mem.

(4) Αν η στοίβα δεν είναι άδεια , βρες τα index / position από την κορυφή της , μείωσε τον top stack index κατά και πήγαινε στο (7).

(5) Αναζήτηση ενός bit με τιμή 1 στον πίνακα bitmap.

(6) Εάν δεν ευρεθεί τέτοιο bit , επιστροφή τιμής NULL.

Διαφορετικά , τα ευρεθέν bit τίθεται ίσα με 0 στο bitmap.

(7) Εύρεση της διεύθυνσης του πρώτου byte με τον αλγόριθμο αντιστοίχισης.

(8) Επιστροφή της διεύθυνσης του πρώτου byte αντιστοίχισης .

free

(1) Εύρευση διεύθυνσης του πίνακα bitmap.

(2) Εύρεση resolution.

(3) Εύρεση base mem.

(4) Εύρεση του cell_no : (ptr - base_mem) / resolution.

(5) Εύρεση index : cell_no DIV BMAP_EL_SIZE_BITS.

(6) Εύρεση του θέσης του bit αντιστοίχισης : cell_no MOD BMAP_EL_SIZE_BITS.

(7) Ανάθεση τιμής 1 στο bit αντιστοίχισης.

(8) Εάν η στοίβα δεν είναι γεμάτη , προσέθεσε στην κορυφή της το ζεύγος (index , position) και αύξησε τον top stack index κατά 1.

Παράδειγματα δομής στατικής στοίβας recommendation και επικεφαλίδας BITMAP_SINGLE raw

block , το οποίο θα μπορούσε να ικανοποιήσει τις παραπάνω προδιαγραφές είναι το εξής :

```
typedef struct {
        unsigned int rec_i;
        unsigned int rec_p;
} rec_el;

typedef struct {
        size_t bytes_per_cell;
        void *base_mem;
        int  rov_index;
        BMAP_EL_TYPE *bmap_ptr;
        size_t elements;
        int rec_index; // -1 for empty stack
        rec_el rec_arr[MAX_REC_INDEX];
} bitmap_rb_t;
```

Αγνοώντας την εισαγωγή κενών bytes που εισάγει ο μεταγλωττιστής στο εσωτερικό των structs και ο διαχειριστής μνήμης μεταξύ των σταδίων της στοίβας δεδομένων raw block header - bitmap single raw block header - bitmap array - mem (padding) , η συνολική τοποθέτηση των μεταδεδομένων του διαχειριστή καθώς και ο διαχειριζόμενος χώρος στην μνήμη (memory layout) σε BITMAP_SINGLE raw block,, απεικονίζεται ως εξής :

rb_type type;	τύπος raw block	γενική επικεφαλίδα raw block
size_t size;	μέγεθος raw block	
lock_t lock;	mutex/spinlock αν χρησιμοποιείται κλείδωμα σε επίπεδο raw block	
struct raw_block_header_s *next;	δείκτης σε επόμενο raw block (freelist δομή)	
size_t bytes_per_cell;	μέγεθος κελιού χώρου σε bytes	επικεφαλίδα BITMAP_SINGLE raw block
void *base_mem	δείκτης στην αρχική διεύθυνση	
size_t rov_index;	roving index	
BMAP_EL_TYPE *bmap_ptr;	δείκτης σε πίνακα bitmap	
size_t elements;	αριθμός στοιχείων πίνακα bitmap	
rec_el rec_arr[MAX_REC_INDEX]	στατικός πίνακας από recommendations	
bmap_ptr[0]	πρώτο στοιχείο bitmap	πίνακας bitmap
bmap_ptr[1]	δεύτερο στοιχείο	
bmap_ptr[2]	…	
bmap_ptr[3]	…	
bmap_ptr[4]		
…		
…		
…		
…		
bmap_ptr[elements-1];		
cells		διαχειριζόμενος χώρος μνήμης

Συμπιεσμένες επικεφαλίδες

Κατά την εξυπηρέτηση αιτημάτων από τον διαχειριστή μνήμης, η ροή εκτέλεσης του κώδικα διέρχεται από δύο στάδια στα οποία πραγματοποιούνται αναγνώσεις στοιχείων τόσο των γενικών όσο και των ειδικών (BITMAP, BITMAP_SINGLE) επικεφαλίδων, οι οποίες μεταφράζονται σε προσβάσεις μνήμης.Μία σχεδιαστική επιλογή θα ήταν , εφόσον τα όρια των αριθμητικών τιμών των στοιχείων που προσπελαύνονται είναι γνωστά στον χρόνο μεταγλώττισης,

διαφορετικά πεδία να πολυπλέκονται μεταξύ τους με σκοπό την μείωση των προσβάσεων στην μνήμη, με μετακύλιση του κόστους σε κύκλους CPU. Κατά την κατασκευή του διαχειριστή μνήμης της διπλωματικής , υλοποιήθηκαν

(1) δυνατότητες πολύπλεξης/αποπολύπλεξης στο πεδίο type του raw block των πεδίων bytes per cell, και της απόστασης μέχρι την ελεύθερη μνήμη με κλήσεις αντίστοιχα σε απλοποιημένες εκδόσεις των βασικών συναρτήσεων (exact_bitmap_free_demuxed έναντι exact_bitmap_free) οι οποίες δεν διαβάζουν τις απαιτούμενες τιμές από τις επικεφαλίδες.

```
#define rb_type         size_t
#define BITMAP_SINGLE   0
#define BITMAP          1
#define BIGBLOCK        2

#define TYPE_BITS       2
#define TYPE_MASK       ((((size_t) 1) << TYPE_BITS) - 1)

#define RES_BITS        20
#define RES_MASK        ((((size_t) 1) << RES_BITS) - 1)

#define demux_type(val)     (val & TYPE_MASK)
#define demux_res(val)      ((val >> TYPE_BITS) & RES_MASK)
#define demux_offset(val)   (val >> (TYPE_BITS + RES_BITS))

static size_t multiplex(rb_type type, size_t res, size_t offset)
{
        size_t val = offset;
        val <<= RES_BITS;
        val |= res;
        val <<= TYPE_BITS;
        val |= type;
        return val;
}
```

όπου η πολύπλεξη γίνεται την στιγμή δημιουργίας ενός raw block με την συνάρτησης, ενώ η αποπολύπλεξη κατά την εξυπηρέτηση κάθε αιτήματος(free) με τα macros.

(2) δυνατότητες πολύπλεξης/αποπολύπλεξης των στοιχείων bytes per cell (πολλαπλάσιο του 32),roving index , elements σε ένα κοινό στοιχείο bre που μπορεί να προστεθεί στην επικεφαλίδα ενός BITMAP_SINGLE raw block.

Παράδειγμα τέτοιας λειτουργικότητας, με τύπο bre uint64_t και περιορισμούς στα 14, 25 και 25 στα απαιτούμενα bits αναπαράστασης:

```
#define EBITS           25
```

```
#define EBITS_MASK		((1 << EBITS) - 1)

#define RBITS			25
#define RBITS_MASK		((1 << RBITS) - 1)

#define BBITS			14
#define BBITS_MASK		((1 << BBITS) - 1)

#define bre_getb(bre) (bre & BBITS_MASK)
#define bre_getr(bre)  ((bre >> BBITS) & RBITS_MASK)
#define bre_gete(bre) ((bre >> (BBITS + RBITS)) & EBITS_MASK)

#define SETBZERO_MASK	(~((uint64_t) BBITS_MASK))
#define SETRZERO_MASK	(~(((uint64_t) RBITS_MASK) << BBITS))
#define SETEZERO_MASK	(~(((uint64_t) BBITS_MASK) << (RBITS + BBITS)))

#define bre_setb(new_b, old_bre)		((old_bre & SETBZERO_MASK) | \
						((uint64_t) new_b))

#define bre_setr(new_r, old_bre)		((old_bre & SETRZERO_MASK) | \
						(((uint64_t) new_r) << BBITS))

#define bre_sete(new_e, old_bre)		((old_bre & SETEZERO_MASK) | \
						((uint64_t) new_e) << (RBITS + BBITS))

static uint64_t _bre_make(size_t elements, size_t rov_index, size_t bytes_per_cell_div32)
{
		uint64_t bre = 0;
		bre = bre_setb(bytes_per_cell_div32, bre);
		bre = bre_setr(rov_index, bre);
		bre = bre_sete(elements, bre);
		return bre;
}
static uint64_t bre_make(size_t elem, size_t rov, size_t bpc)
{
		return _bre_make(elem, rov, bpc / 32);
}
```

Υλοποίηση αφηρημένης δομής συσχετιστικού πίνακα

Στο εσωτερικό του διαχειριστή δυναμικής μνήμης, μεγέθη αιτημάτων αντιστοιχίζονται σε raw blocks είτε κατηγορίας BITMAP είτε κατηγορίας BITMAP_SINGLE. Το πρόβλημα της σχεδίασης ενός ακριβούς αλγορίθμου αντιστοίχισης, αποτελεί υποπερίπτωση του γενικότερου ζητήματος κατασκευής της αφηρημένης δομής δεδομένων λεξικού ή συσχετιστικού πίνακα. Στην γενική του μορφή, το πρόβλημα συμπεριλαμβάνει λειτουργίες πρόσθεσης, αφαίρεσης, τροποποίησης και αναζήτησης ενώ, ανάλογα με την εφαρμογή της δομής, κάποιες από αυτές μπορεί να αγνοηθούν. Στο πλαίσιο της υλοποίησης του προτεινόμενου διαχειριστή, αναζητούμε τρόπους αναπαράστασης συνόλων ζευγών (key,value), όπου keys είναι μοναδικές τιμές μεγεθών και values δείκτες σε raw blocks, στα οποία κατευθύνονται αιτήματα δέσμευσης.Οι λειτουργίες που είναι απαραίτητο να υλοποιηθούν συμπεριλαμβάνουν πρόσθεση ζεύγους, αφαίρεση και αναζήτησης.Καθώς ο τύπος δεδομένων των keys είναι ακέραιοι και, ανάλογα με το καλυπτόμενο εύρος, μπορούν να χρησιμοποιούνται είτε στατικοί πίνακες είτε reb black δέντρα.

Για την περίπτωση των στατικών πινάκων, καθώς οι ποσότητες bytes_per_cell και τα μεγέθη των αιτημάτων, μετατρέπονται σε πολλαπλάσια προκαθορισμένης ποσότητας (16/32/64 κοκ) για λόγους ευθυγράμμισης, οι λειτουργίες γίνονται σε σταθερό χρόνο μέσω ακέραια διαίρεσης με την εν λόγω ποσότητα.

Ένα παράδειγμα τέτοιας δομής με βασική ποσότητα 32 και διάστημα κάλυψης αιτημάτων με BITMAP_SINGLE raw blocks [0, 32x10 = 320),

0	0x7f1b54200000
1	0x7f1b54000000
2	0x7f1b53e00000
3	0x7f1b53c00000
4	NULL
5	NULL
6	0x7f1b53600000
7	0x7f1b53400000
8	NULL
9	NULL

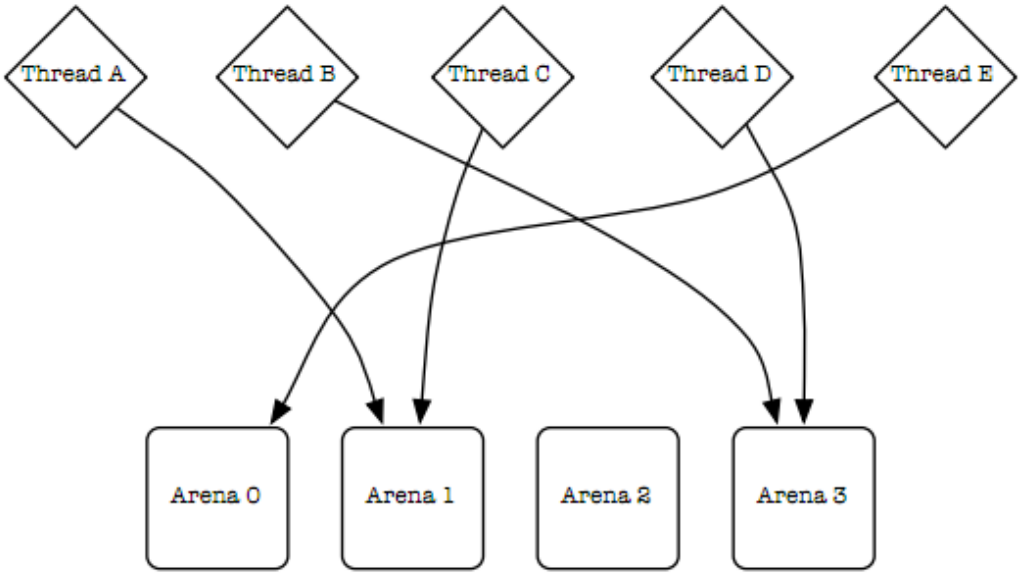
Ένα αίτημα malloc(64) αντιστοιχίζεται στο slot 64 div 32 = 2 και προωθείται στο αντίστοιχο raw block , ενώ αίτημα malloc(128) με τιμή slot 128 div 32 = 4, οδηγεί τον διαχειριστή μνήμη στο να ζητήσει μνήμη από το λειτουργικό σύστημα, προκειμένου να διαμορφώσει raw block , να το τοποθετήσει στο συγκεκριμένο slot και , στη συνέχεια, να προωθήσει το αίτημα στο νέο raw block.

Μία τέτοια προσέγγιση είναι αποδοτική, όταν το εύρος μεγεθών είναι σχετικά μικρό αλλά καθίσταται μη πρακτική καθώς το απαιτούμενο εύρος αυξάνεται. Για τέτοιου είδους εύρη, η δομή συσχετιστικού πίνακα υλοποιείται με χρήση red-black δέντρου.Η ιδιομορφία που παρουσιάζεται στην χρήση δυναμικών δομών στο εσωτερικό του κώδικα διαχειριστών μνήμης, είναι ότι για την δημιουργία των υποστοιχείων μιας δυναμικής δομής η μνήμη δεσμεύεται και απελευθερώνεται με κατώτερου επίπεδου λειτουργίες από τη βασική διεπαφή malloc/free. Το γεγονός αυτό επιβάλει την εισαγωγή ενός επιπέδου εσωτερικής διαχείρισης μνήμης, το οποίο παρεμβάλλεται ανάμεσα στις λειτουργίες εισαγωγής/αναζήτησης και στην διεπαφή του δέντρου. Το σύνολο των απαιτούμενων συναρτήσεων για την κατασκευή ολόκληρης της δομής παρουσιάζεται στον παρακάτω πίνακα:

Βασική διεπαφή	
rb_new	αρχικοποίηση δέντρου
rb_insert	εισαγωγή στοιχείου
rb_search	αναζήτηση στοιχείου
rb_remove	αφαίρεση στοιχείου
rb_nsearch	αναζήτηση στοιχείου/επιστροφή του αμέσως μεγαλύτερου αν δεν προκύψει αποτέλεσμα
rb_psearch	Αναζήτηση στοιχείου.Επιστροφή του αμέσως μικρότερου αν δεν βρεθεί.
Εσωτερική διαχείριση μνήμης	
init_tree_block	Αρχικοποίηση δέντρου με δέσμευση μνήμης για τους κόμβους του από το λειτουργικό σύστημα.
tree_block_alloc_node	Εσωτερική δέσμευση χώρου για κόμβο δέντρου.
nodeCmpIdent	Σύγκριση κλειδιών κόμβου.
Εισαγωγή/Αναζήτηση σε red black δέντρο	
insert_prof_key	Εισαγωγή στοιχείου με μετρητή γεγονότων.Χρησιμοποιείται εάν είναι ενεργοποιημένος ο εσωτερικός profiler.
insert_key_bitmap_single	Εισαγωγή στοιχείου με δείκτη σε BITMAP_SINGLE raw block.
insert_key_bitmap	Εισαγωγή στοιχείου με δείκτη σε BITMAP raw block.
key_exists	Αναζήτηση στοιχείου/επιστροφή δείκτη σε raw block για επιτυχή αναζήτηση.
print_nodes	Διάσχιση του profiler δέντρου και συλλογή στατιστικών. Αν ο profiler είναι ενεργοποιημένος τα στατιστικά στοιχεία που αφορούν αιτήματα δέσμευσης τυπώνονται στο stdout μετά το τέλος της εφαρμογής.

Πολυνηματικότητα / per thread caching

Για την μείωση της επιβάρυνσης λόγω κλειδωμάτων αλλά και για την βελτίωση της κλιμακωσιμότητας, ο διαχειριστής μνήμης μπορεί να οργανώνει τη μνήμη που έχει δεσμεύσει από το λειτουργικό σύστημα, σε πολλαπλούς υποχώρους και να τους αντιστοιχίζει στατικά ή δυναμικά σε νήματα. Ένα τέτοιο σχήμα που χρησιμοποιήθηκε στις πρώτες εκδόσεις του διαχειριστή jemalloc :



Καθώς τέτοια σχήματα μειώνουν την επιβάρυνση στο χρόνο εκτέλεσης σε σχέση με υλοποιήσεις κεντρικού κλειδώματος, αλλά διατηρούν το κόστος συγχρονισμού κλειδώματος (pthread_mutex_lock / pthread_mutex_unlock) και επίσης είναι πιθανό να εισάγουν false sharing για μεγέθη αιτημάτων συγκρίσιμα με το cache line, για την κατασκευή του διαχειριστή της παρούσας διπλωματικής υλοποιήθηκαν thread local caches. Η αξιοποίηση TLS (thread local storage), παρέχει δύο βασικά πλεονεκτήματα έναντι της σχεδίασης με πολλαπλούς υποχώρους και δυναμική αντιστοίχιση. Εξασφαλίζει την ύπαρξη ισάριθμων με τα νήματα υποχώρων (caches) ελάχιστες απαιτήσεις στον χρόνο εκτέλεσης έναντι δυναμικών σχημάτων για την τήρηση της προαναφερθείσας ιδιότητες και , επίσης, προσφέρει δυνατότητες γρήγορης αντιστοίχισης.

Από κάποιο, προκαθορισμένο κατά το χρόνο μεταγλώττισης (TLS_TRESHOLD) όριο, μεγέθους, αιτήματα προερχόμενα από όλα τα νήματα εξυπηρετούνται από κοινό χώρο. Δεδομένης της συσχέτισης που παρουσιάζεται μεταξύ των τιμών του μεγέθους και της συχνότητας εμφάνισης αιτήματος , ο ανταγωνισμός κλειδωμάτων δεν αναμένεται να είναι μετρήσιμος από κάποια τιμής μεγέθους και άνω. Για λόγους πληρότητας, στον τομέα των πειραματικών αποτελεσμάτων, μετρήθηκε υ ύπαρξη τέτοιου κατωφλίου μεταξύ μεγεθών αιτημάτων με μεγάλη ένταση.

8 Μετρικές επίδοσης

Για τη μέτρηση της επίδοσης ενός διαχειριστή δυναμικής μνήμης, τυπικά αξιολογείται ο συνδυασμός χρόνου εκτέλεσης και μέσης ή μέσης κατανάλωσης μνήμης κατά τον χρόνο εκτέλεσης. Για την εκτίμηση και τη βελτίωση της αποδοτικότητας της εσωτερικής υλοποίησης του διαχειριστή , μπορούν να χρησιμοποιηθούν είτε θεωρητικές μέθοδοι ασυμπτωτικής ανάλυσης των δομών δεδομένων και των αλγορίθμων του διαχειριστή είτε, πρακτικές μέθοδοι, όπως μέτρηση του χρόνου εκτέλεσης που δαπανά ο διαχειριστής για την εξυπηρέτηση μεμονωμένων αιτημάτων ή προγράμματα που επαναληπτικά δεσμεύουν και αποδεσμεύουν μεταβαλλόμενου μεγέθους κομμάτια μνήμης (malloc/free tight loops), με παράλληλη χρήση profiler και αναπροσαρμογή του κώδικα ανάλογα με τα αποτελέσματα του profiler. Όμως τέτοιες μέθοδοι ενώ μπορούν να παράσχουν πρόχειρη εκτίμηση της αποδοτικότητας , είναι ανεπαρκείς.

Στην προαναφερθείσα περίπτωση, δεν λαμβάνεται υπόψιν η επίδραση που έχουν στην επίδοση μιας εφαρμογής οι αποφάσεις του διαχειριστή σχετικά με την τοποθέτηση στη μνήμη των μπλοκ που εξυπηρετούν αιτήματα δέσμευσης/αποδέσμευσης, που μπορούν να καθορίζουν σε μεγάλο βαθμό την μετέπειτα αλληλεπίδραση της εφαρμογής με το λειτουργικό σύστημα αλλά και την αρχιτεκτονική υλικού. Αν τα αιτήματα έχουν δέσμευσης έχουν μέγεθος ίδιας τάξης με το μέγεθος σελίδας , τότε οι διαδοχικές αποφάσεις τοποθέτησης για μοτίβα τέτοιου τύπου αιτημάτων, επηρεάζουν τον βαθμό εμφάνισης ανεπιθύμητων φαινόμενων του υποσυστήματος διαχείρισης εικονικής μνήμης του λειτουργικού συστήματος , όπως σφάλματα σελίδων (page faults , swapping) . Όσο το μέγεθος των αιτημάτων μειώνεται και γίνεται συγκρίσιμο με εκείνο των μονάδων μεταφοράς μιας πολυεπίπεδης ιεραρχίας μνήμης (cache lines), οι αποφάσεις τοποθέτησης του διαχειριστή γίνονται κρίσιμες για το κατά πόσο προσβάσεις στη μνήμη που προέρχονται από την εφαρμογή και έπονται των αιτημάτων , εμφανίζουν την επιθυμητή ιδιότητα της χωρικής τοπικότητας (data locality) και μειωμένο ρυθμό cache misses. Βεβαίως , αν μία εφαρμογή δεν πραγματοποιεί μεγάλο αριθμό αιτημάτων προς τον διαχειριστή, αν πχ δεσμεύει την μνήμη που χρησιμοποιεί ως ένα ενιαίο κομμάτι, ο τρόπος υλοποίησης του διαχειριστή δεν επηρεάζει την συμπεριφορά της εφαρμογής.

Για την ασυμπτωτική ανάλυση η μνήμη θεωρείται ενιαίος χώρος με σταθερό χρόνο πρόσβασης , κάτι που δεν ισχύει δεδομένης της ύπαρξης πολυεπίπεδων ιεραρχιών μνήμης , συνεπώς δομές υλικού τύπου caches αγνοούνται εντελώς. Ειδικά , όσον αφορά την ανάλυσης χειρότερης περίπτωσης (worst case complexity) είναι χαρακτηριστικό το γεγονός ότι οι δομές δεδομένων που χρησιμοποιούνται, ακόμα σε ευρέως διαδεδομένους και διαπιστωμένα αποδοτικούς διαχειριστές μνήμης, είναι σχετικά απλές και υπολείπονται σε χρονική πολυπλοκότητα από πλήθος άλλων δομών θεωρητικά καλύτερης χρονικής επίδοσης αλλά χειρότερης αλληλεπίδρασης με την

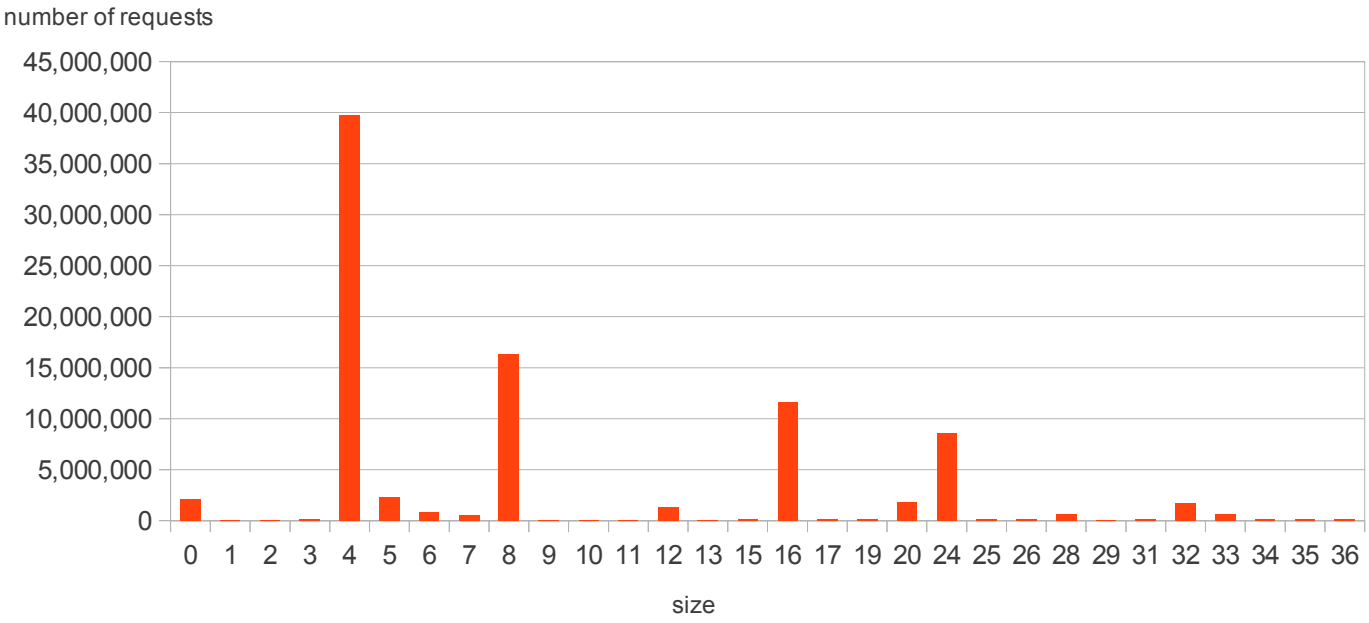
ιεραρχία μνήμης. Επίσης, κατά την ασυμπτωτική ανάλυση, αγνοούνται οι σταθερές αλγοριθμικών χρόνων που προκύπτουν, οι οποίες ενδέχεται να διαφοροποιήσουν την επίδοση ενός διαχειριστή μνήμης κατά τάξεις μεγέθους.Από τα παραπάνω προκύπτει ότι η επίδοση ενός διαχειριστή μνήμης μπορεί να εκτιμηθεί με ακριβή τρόπο μόνο όταν οι βασικές μετρικές (χρόνος, προσβάσεις στη μνήμη, συνολική κατανάλωση μνήμης) .

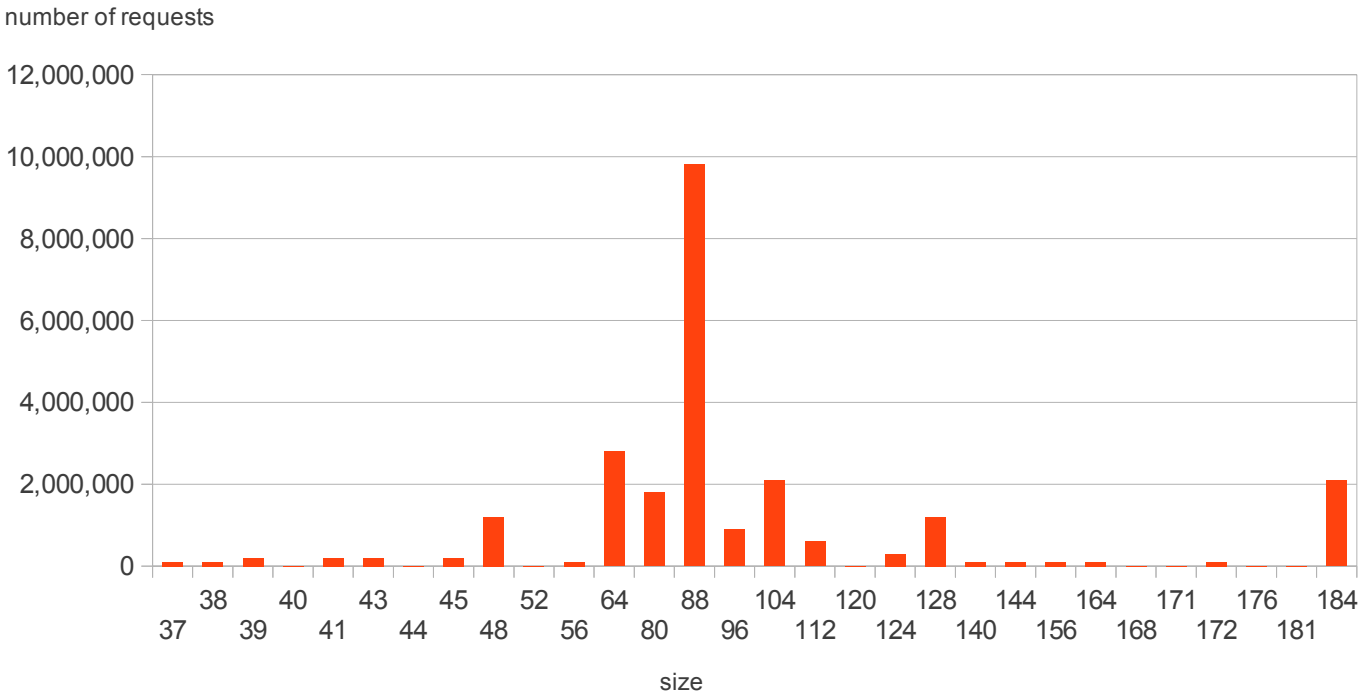
Όσον αφορά πολυνηματικούς διαχειριστές ενσωματωμένων συστημάτων, η προσέγγιση εκτίμησης της επίδοσης δεν αλλάζει, αλλά εισέρχονται οι παράμετροι της δυνατότητας του διαχειριστή να αυξάνει την απόδοση του με την προσθήκη περισσότερων επεξεργαστικών μονάδων υλικού και την αύξηση των νημάτων της εφαρμογής, καθώς και η ικονότητά του να διατηρεί χαμηλή την κατανάλωση ενέργειας με όσο το δυνατόν λιγότερες προσβάσεις στη μνήμη.

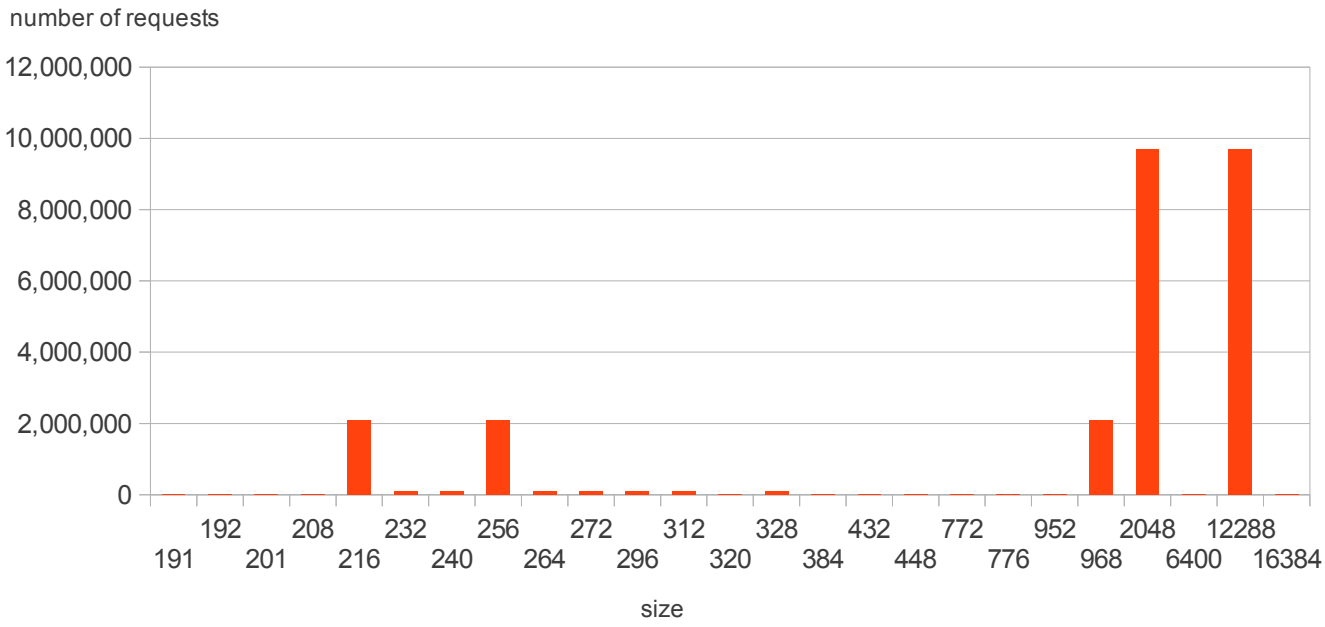
9 Πειραματική αξιολόγηση

gawk

Αποτελεί μονονηματική εφαρμογή που έχει χρησιμοποιηθεί ευρέως στην επιστημονική βιβλιογραφία για την εκτίμηση επίδοσης διαχειριστών δυναμικής μνήμης.Αρχικά, μέσω του εσωτερικού profiler του διαχειριστή της διπλωματικής, λαμβάνουμε το προφίλ εκτέλεσης της εφαρμογής (αριθμός αιτημάτων ανά μέγεθος) το οποίο, για την διευκόλυνση κατανόησης της μεθολογίας μέτρησης χωρίζεται σε τρία κομμάτια :




number of requests
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
37 38 39 40 41 43 44 45 48 52 56 64 80 88 96 104 112 120 124 128 140 144 156 164 168 171 172 176 181 184
size


number of requests
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
191 192 201 208 216 232 240 256 264 272 296 312 320 328 384 432 448 772 776 952 968 2048 6400 12288 16384
size

Στη συνέχεια εκτελούμε το gawk με χρήση τριών διαφορετικών διαχειριστών δυναμικής μνήμης και τεσσάρων διαφορετικών εκδόσεων της βιβλιοθήκης που αναπτύχθηκε στα πλαίσια της διπλωματικής και μετράμε με χρήση της υποεντολής perf stat : cycles, instructions, minor-faults, L1-dcache-loads, L1-dcache-misses, L1-dcache-stores, L1-dcache-store-misses, time elapsed.

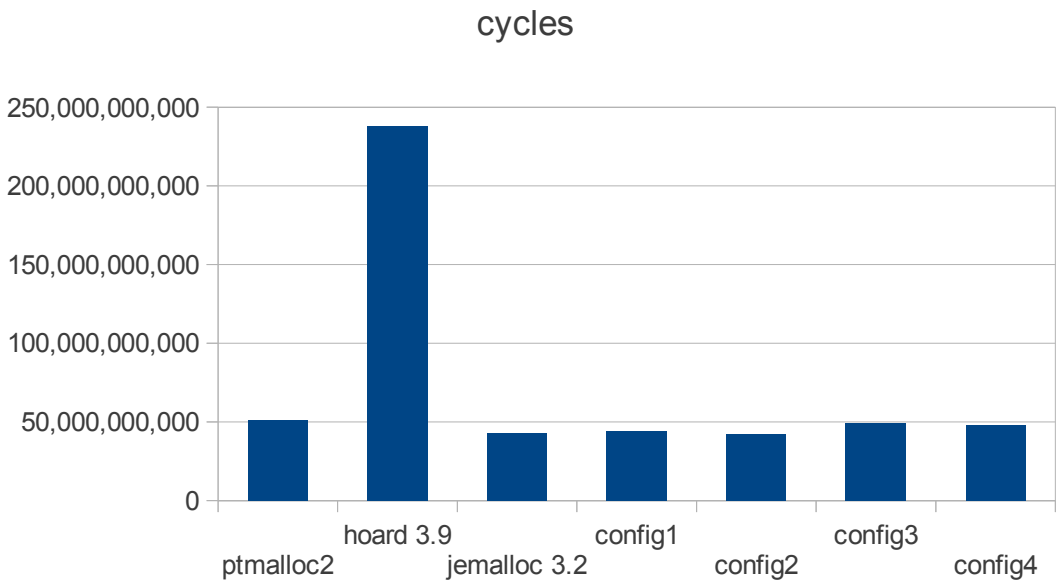
config1: static array thread caching / compressed headers / no global heap

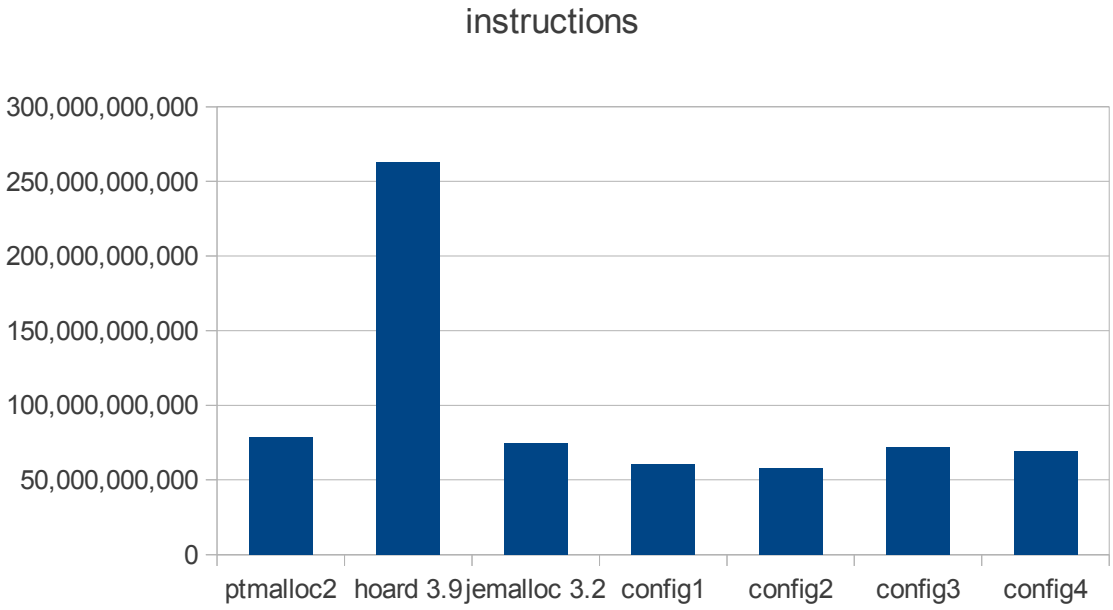
config2 : static array thread caching / uncompressed headers / no global heap

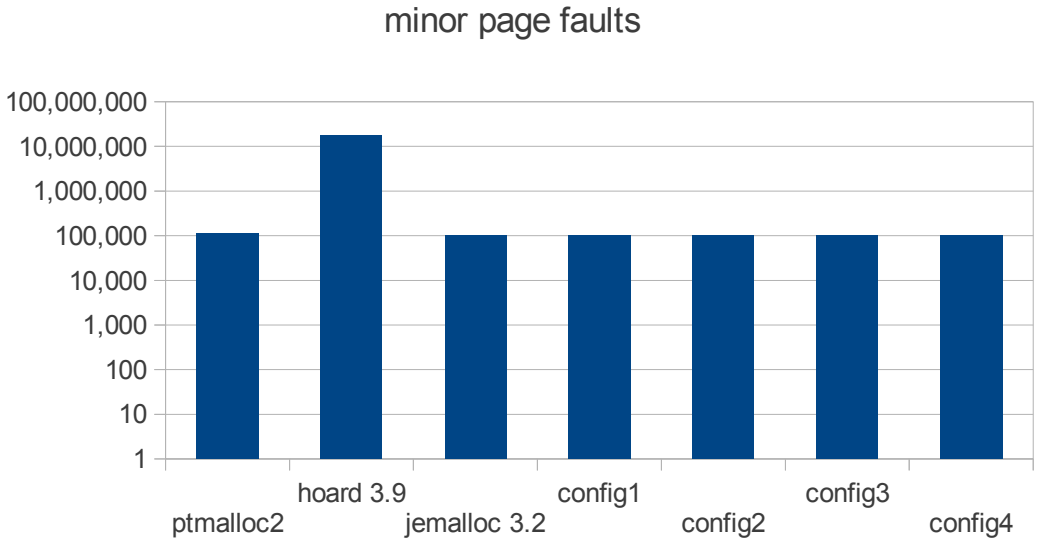
config3 : red black tree thread caching / compressed headers / no global heap

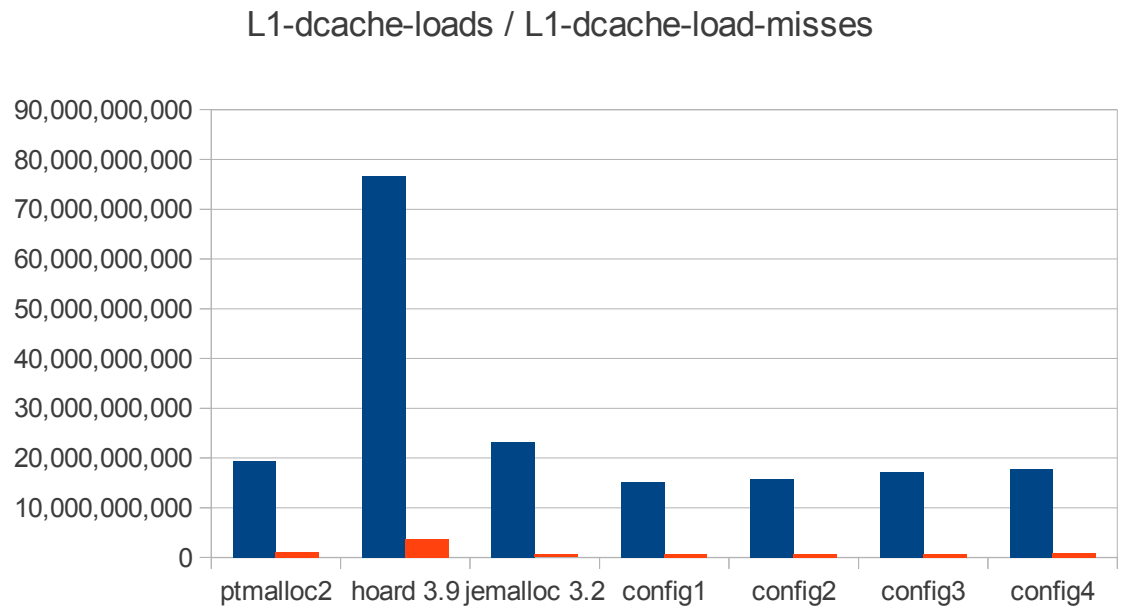
config4 : red black tree thread caching / compressed headers / no global heap

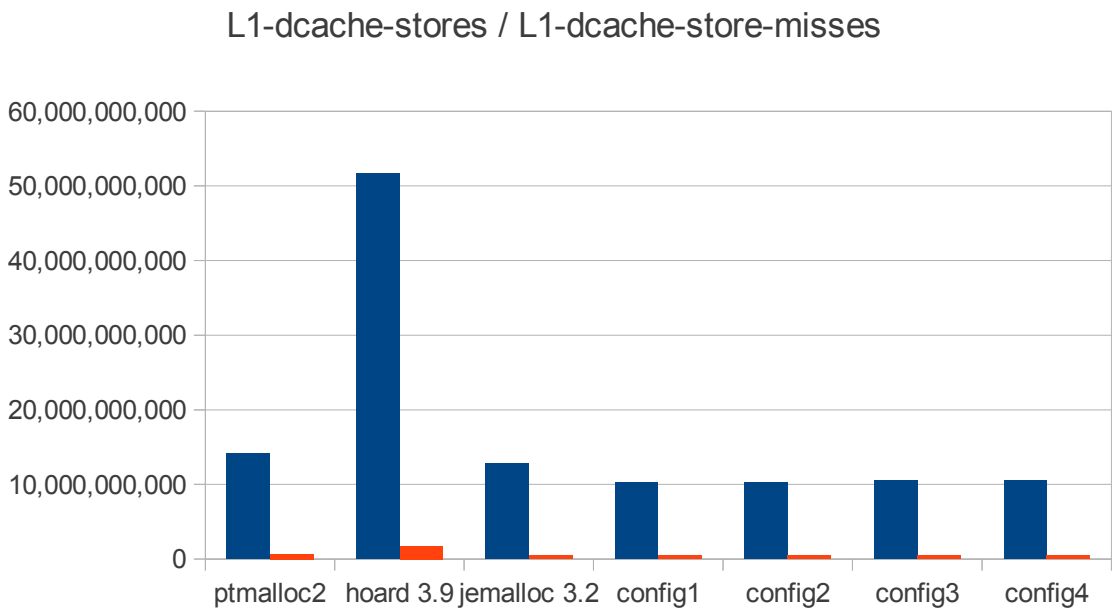
Ο διαχωρισμός επιλέγεται για εκτιμηθούν οι διαφορές στον χρόνο εκτέλεσης με αξιοποίηση δομών static arrays και red black trees και λειτουργιών συμπίεσης. Ειδικά για τα configs compressed/uncompressed header μας ενδιαφέρουν οι μετρικές L1-dcache-loads / L1-dcache-load-misses.

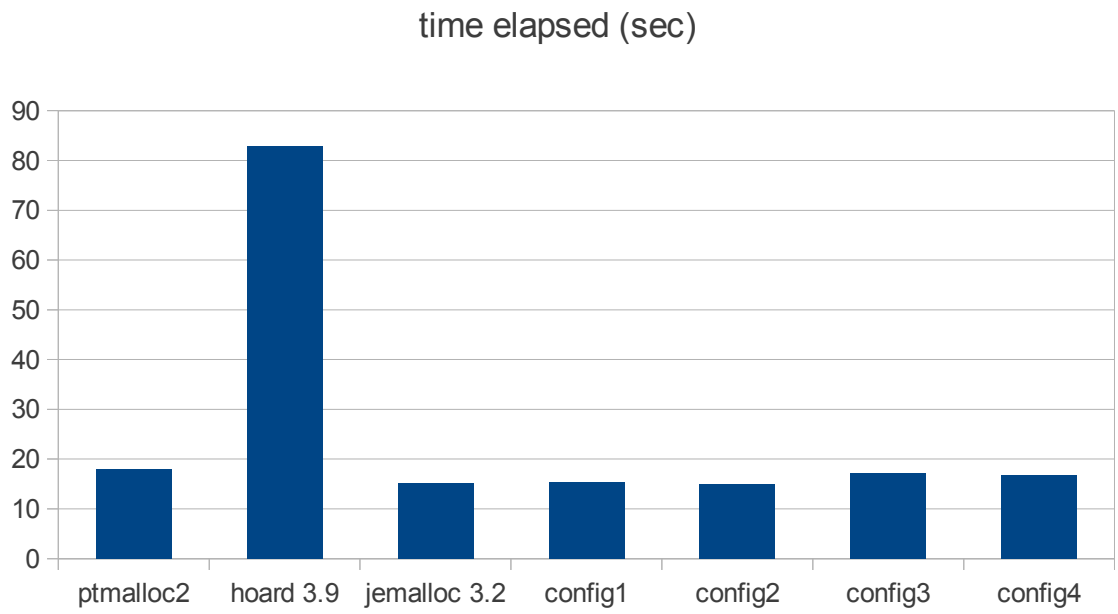



instructions
300,000,000,000
250,000,000,000
200,000,000,000
150,000,000,000
100,000,000,000
50,000,000,000
0
ptmalloc2
hoard 3.9
jemalloc 3.2
config1
config2
config3
config4


minor page faults
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
ptmalloc2
hoard 3.9
jemalloc 3.2
config1
config2
config3
config4


L1-dcache-loads / L1-dcache-load-misses
90,000,000,000
80,000,000,000
70,000,000,000
60,000,000,000
50,000,000,000
40,000,000,000
30,000,000,000
20,000,000,000
10,000,000,000
0
ptmalloc2
hoard 3.9
jemalloc 3.2
config1
config2
config3
config4


L1-dcache-stores / L1-dcache-store-misses
60,000,000,000
50,000,000,000
40,000,000,000
30,000,000,000
20,000,000,000
10,000,000,000
0
ptmalloc2
hoard 3.9
jemalloc 3.2
config1
config2
config3
config4



Ειδικότερα, συγκρίνοντας τον jemalloc με το config2, παρατηρούμε ότι ενώ το config2 αποδίδει καλύτερα και ως προς τον αριθμό των εντολών και ως προς την πίεση προς την L1 cache, παραμένει οριακά καλύτερο ω ς προς τον συνολικό χρόνο εκτέλεσης. Παραθέτοντας τα τελευταία στατιστικά

config2

16,237,440,017 dTLB-loads

105,488,543 dTLB-load-misses # 0.65% of all dTLB cache hits

10,889,000,922 dTLB-stores

9,781,389 dTLB-store-misses

jemalloc 3.2

23,256,009,306 dTLB-loads

6,547,561 dTLB-load-misses # 0.03% of all dTLB cache hits

12,799,644,486 dTLB-stores

533,975 dTLB-store-misses

συμπεραίνουμε ότι η επιθετική προσέγγιση του εικονικού address space του config2 δημιουργεί περισσότερα TLB misses.

Η λήψη των συγκεκριμένων μετρήσεων έγινε με μεγάλο cache threshold, έτσι να ελαχιστοποιείται το κόστος συγχρονισμού. Για την εκτίμηση της επιβάρυνσης που θα προέκυπτε, αν τα αιτήματα

περνούσαν κατά την εκτέλεση τους καθώς για την εκτίμηση της ρυθμού αιτημάτων για την οποία η εν λόγω επιβάρυνση θα γινόταν μετρήσιμη, επιλέγουμε το config2 και τροποποιούμε τον κώδικα του, προσθέτοντας global heap με δέντρο. Παρατηρώντας το προφίλ εκτέλεσης του gawk, βλέπουμε ότι ο κύριος όγκος αιτημάτων είναι συγκεντρωμένος γύρω από 4 περιοχές.Έτσι, για την λήψη μετρήσεων χρησιμοποιούμε 5 διαφορετικές υποεκδόσεις της config2 με μεταβαλλόμενο κατώφλι διαχωρισμού thread-caching και global heap.

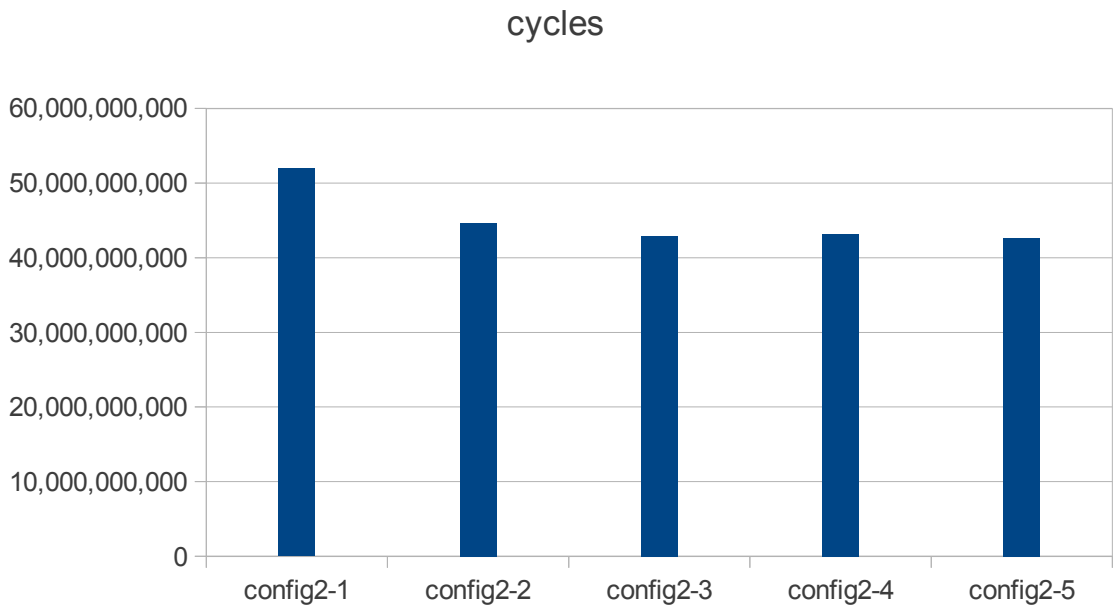
config2-1 : TLS threshold 0

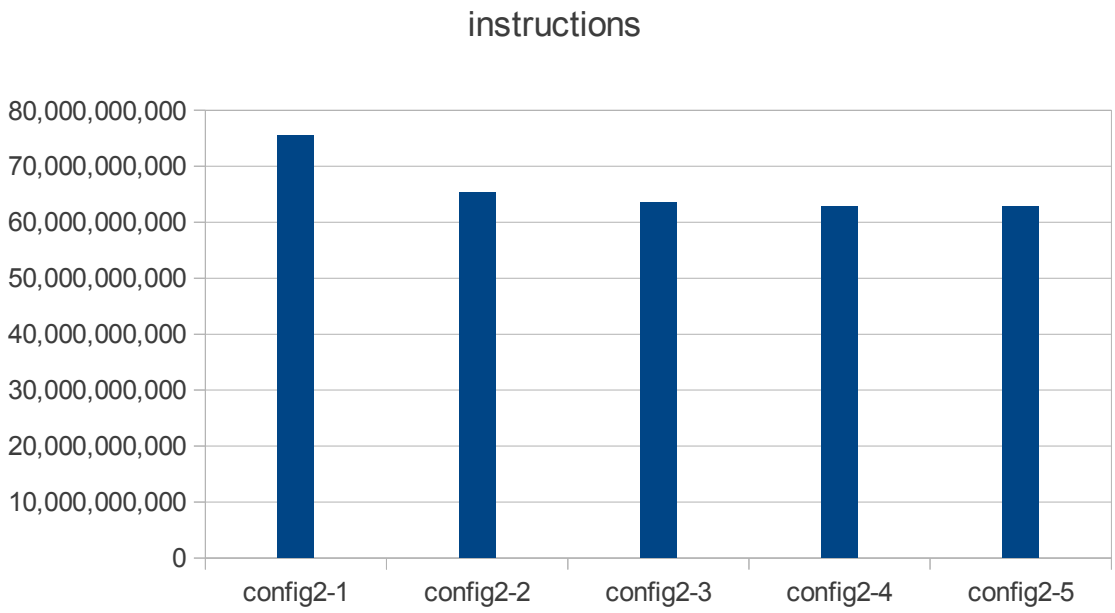
config2-2 : TLS threshold 32

config2-3 : TLS threshold 128

config2-4 : TLS threshold 512

config2-5 : TLS threshold 16384




instructions
80,000,000,000
70,000,000,000
60,000,000,000
50,000,000,000
40,000,000,000
30,000,000,000
20,000,000,000
10,000,000,000
0
config2-1
config2-2
config2-3
config2-4
config2-5

larson

Το μοτίβο αιτημάτων που δημιουργεί το larson, προσομοιώνει εξυπηρετητή που δέχεται αιτήματα από νήματα - πελάτες. Κατά τη στιγμή δημιουργίας ενός νήματος, έχει προηγηθεί δέσμευση ενός συνόλου από μπλοκ μνήμης. Ένα νήμα επιλέγει με τυχαίο τρόπο ένα από τα υπάρχοντα μπλοκ και το αποδεσμεύει ενώ στη συνέχεια δεσμεύει ένα καινούργιο. Όταν το νήμα ολοκληρώσει έναν προκαθορισμένο αριθμό επαναλήψεων αυτής της διαδικασίας, προωθεί σε ένα νέο νήμα το σύνολο των δεσμευμένων μπλοκ του σε ένα νέο νήμα και τερματίζει την εκτέλεση του. Το larson έχει περιγραφεί ως stress test για τις εσωτερικές δομές του διαχειριστή δυναμικής μνήμης. Εξετάζοντας τον κώδικα του, διαπιστώθηκε ότι κάθε νήμα απλά εγγράφει το πρώτο byte και διαβάζει το δεύτερο byte κάθε μπλοκ. Καθώς, η πειραματική αξιολόγηση που πραγματοποιήθηκε έχει ως σκοπό την εκτίμηση της επίδοσης ενός διαχειριστή ως προς την συμπεριφορά πραγματικών εφαρμογών, η λειτουργία των νημάτων τροποποιήθηκε ώστε σε κάθε μπλοκ να εκτελείται η συνάρτηση memset. Πραγματοποιήθηκαν μετρήσεις μεταβαλλόμενου εύρους μεγεθών, προκειμένου να εκτιμηθεί η επίδοση του διαχειριστή υπό διαφορετικές συνθήκες έντασης φορτίου, ενώ επίσης εκτιμήθηκε η επιβάρυνση που προκύπτει με το πέρασμα από στατικό thread caching σε εξυπηρέτηση από καθολικό υποχώρο με red black δέντρο. Η τελική αριθμητική τιμή operations που αντιστοιχεί σε malloc/frees μετράται από το ίδιο το benchmark και εκτυπώνεται μετά το πέρας της εκτέλεσής του.

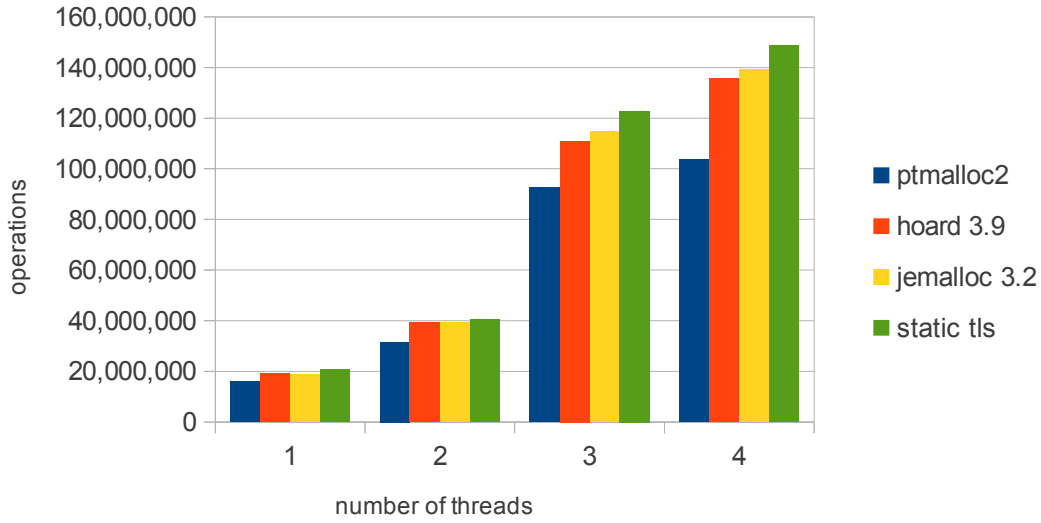
larson config:

chunks per thread: 9000

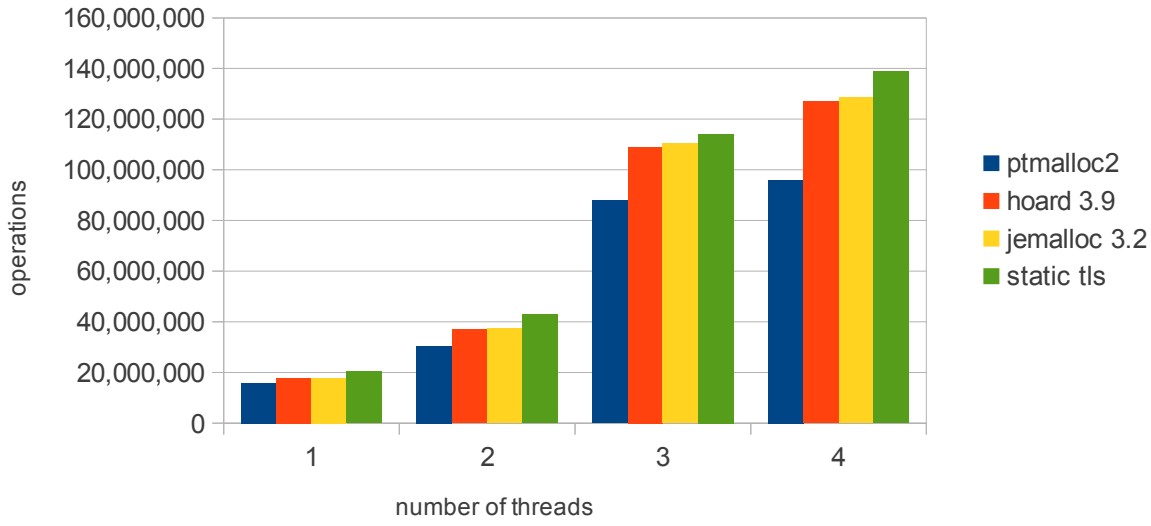
time : 6 sec

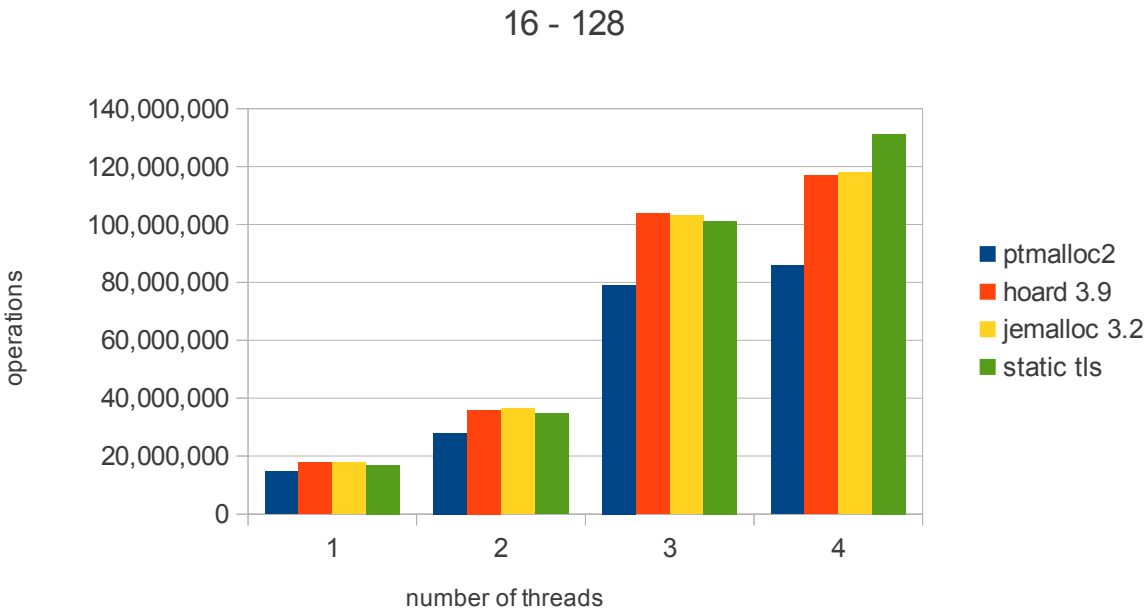
rounds: 1

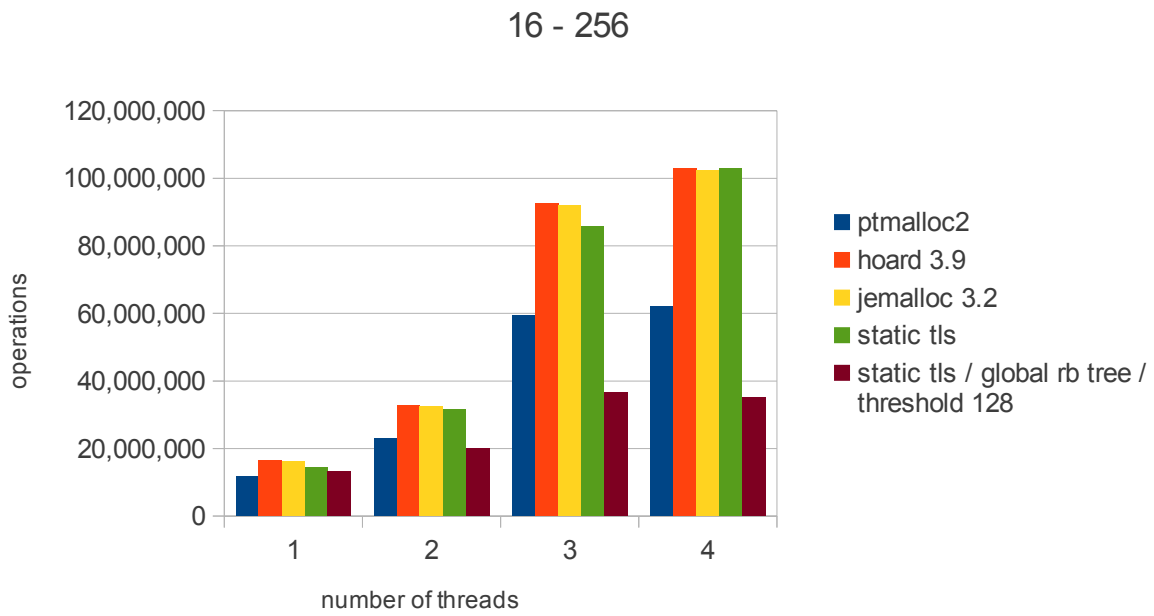
16 - 32


160,000,000
140,000,000
120,000,000
100,000,000
80,000,000
60,000,000
40,000,000
20,000,000
0
operations
1
2
3
4
number of threads
ptmalloc2
hoard 3.9
jemalloc 3.2
static tls

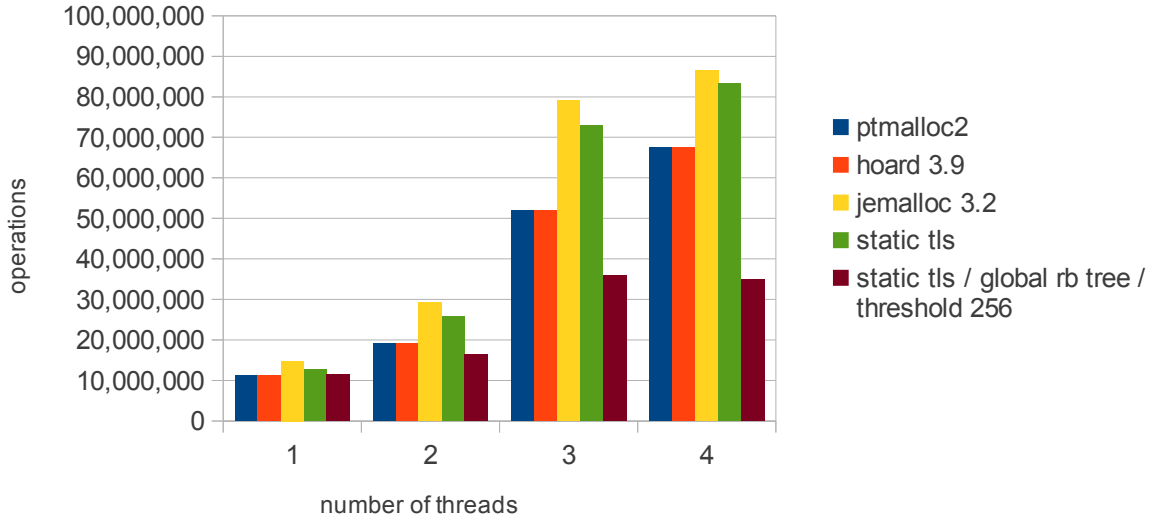
16 - 64


160,000,000
140,000,000
120,000,000
100,000,000
80,000,000
60,000,000
40,000,000
20,000,000
0
operations
1
2
3
4
number of threads
ptmalloc2
hoard 3.9
jemalloc 3.2
static tls

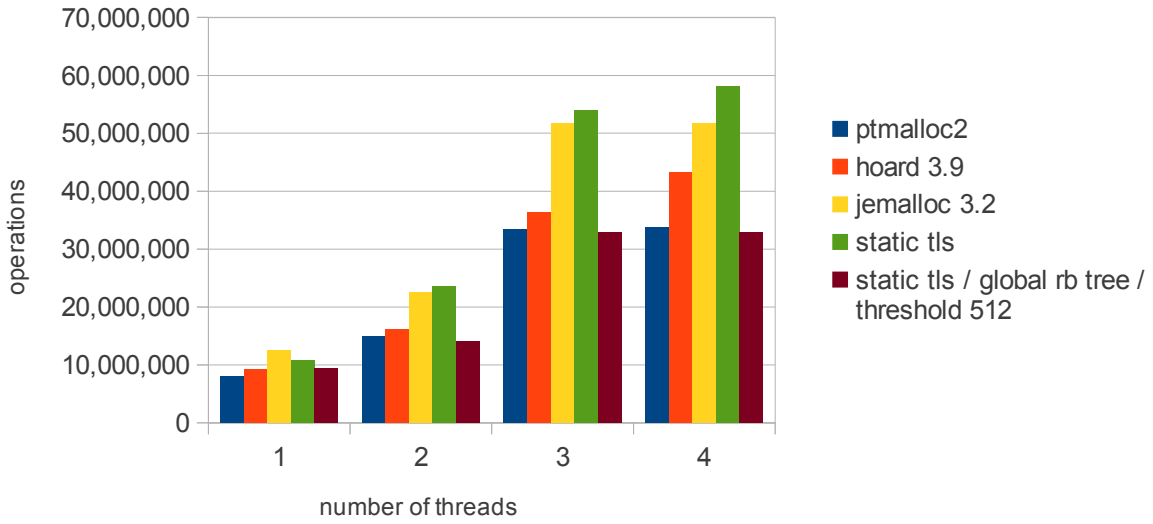

16 - 128
140,000,000
120,000,000
100,000,000
80,000,000
60,000,000
40,000,000
20,000,000
0
operations
1
2
3
4
number of threads
ptmalloc2
hoard 3.9
jemalloc 3.2
static tls


16 - 256
120,000,000
100,000,000
80,000,000
60,000,000
40,000,000
20,000,000
0
operations
1
2
3
4
number of threads
ptmalloc2
hoard 3.9
jemalloc 3.2
static tls
static tls / global rb tree / threshold 128

16 - 512



16 - 1024



10 Συμπεράσματα

Στο πλαίσιο της παρούσας διπλωματικής, διευρενήθηκε η αποδοτικότητα υλοποίησεων πολυνηματικών διαχειριστών δυναμικής μνήμης με δυνατότητες προσαρμοστικότητας στον χρόνο εκτέλεσης, με προσαρμοστικότητα του ακολουθεί το μοτίβο μεγέθους δεσμεύσεων μνήμης. Τα πειραματικά αποτελέσματα, έδειξαν την δυνατότητα βελτίωσης της επίδοσης μιας εφαρμογής, υπό την προυπόθεση της ύπαρξης προηγμένης υλοποίησης του υποσυστήματος εικονικής μνήμης του πυρήνα του λειτουργικού συστήματος και ,συγκεκριμένα, την ύπαρξη μηχανισμών demand paging και διαχωρισμού virtual/resident μνήμης, οι οποίοι επιτρέπουν επιθετικότητα όσον αφορά την δέσμευση περιοχών εικονικής μνήμης (virtual address space mappings) χωρίς πραγματική δέσμευση σελίδων.

11 Βιβλιογραφικές αναφορές

Doug Lea. A memory allocator,
http://g.oswego.edu/dl/html/malloc.html

W. Gloger. Dynamic memory allocator implementations in linux system libraries.

A. K. Iyengar. Dynamic Storage Allocation on a Multiprocessor. PhD thesis, MIT, 1992. MIT Laboratory for Computer Science Technical Report MIT/LCS/TR–56

M. S. Johnstone and P. R. Wilson. The memory fragmentation problem: Solved? In ISMM, Vancouver, B.C., Canada, 1998

P. Larson and M. Krishnan. Memory allocation for long-running server applications. In ISMM, Vancouver, B.C., Canada, 1998

V.-Y. Vee and W.-J. Hsu. A scalable and efficient storage allocator on shared-memory multiprocessors. In International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99), pages 230–235, Fremantle, Western Australia, June 1999

E. D. Berger. Reconsidering custom memory allocation. In OOPSLA,2002.

Emery D. et al. Berger. Hoard: a scalable memory allocator for multithreaded applications. ASPLOS, 2000

Aniruddha Bohra and Eran Gabber. Are mallocs free of fragmentation? In USENIX Annual Technical Conference, 2001

Yi Feng and Emery D. Berger. A locality-improving dynamic memory allocator. In Workshop on Memory System Performance, 2005

Simon Kahan and Petr Konecny. MAMA!: a memory allocator for multithreaded architectures. In PPoPP, 2006

C. Lattner and V. Adve. Automatic Pool Allocation: Improving Performance by Controlling Data Structure Layout in the Heap. In PLDI, 2005

T. Jeremiassen and S. Eggers. Reducing false sharing on shared memory multiprocessors through compile time data transformations. In ACM Symposium on Principles and Practice of Parallel Programming, pages 179–188, July 1995