Garbage collection

Τεχνολογία Λογισμικού 2017-18
Παρουσίαση στα πλαίσια της 2ης εργασίας

Μπουγουλιάς Παναγιώτης

Διαχείριση μνήμης

Σε μια γλώσσα όπως η C, ο compiler και το runtime, χωρίζουν το ιδαθέσιμο χώρο διευθύνσεων που τους παραχωρεί το λειτουργικό σε τρία μέρη:

- **Στατική δέσμευση μνήμης:** Η δέσμευσή τους αποφασίζεται κατά το χρόνο της μεταγλώτισσης και αφορά τη δέσμευση global μεταβλητών.
- **Δέσμευση στη στοίβα:** Πρόκειται για δέσμευση μνήμης για τις εγγραφές δραστηριοποίησης στη στοίβα. Αυτές περιλαμβάνουν local variables και formal παραμέτρους.
- **Δέσμευση στο σωρό:** Κατά το χρόνο εκτέλεσης, δεσμεύεται μνήμη για μεταβλητές όταν αυτή ζητείται με malloc.

Δέσμευση μνήμης

- Στη C, η δυναμική δέσμευση δηλώνεται explicitely από τον προγραμματιστή και αντίστοιχα γίνεται και η αποδέσμευση της.
- Η διαχείριση μνήμης είναι καθαρά ευθύνη του προγραμματιστή. Αυτό μπορεί να οδηγήσει σε σοβαρά memory leaks, “κρεμασμένους” pointers, κατακερματισμό του σωρού.

Implicit allocation/deallocation

- Κατακερματισμός του σωρού: Ο garbage collector φροντίζει να κρατήσει συμπαγή το σωρό, είτε σε κάθε αποδέσμευση μνήμης από αυτόν είτε όταν δεν υπάρχει δυνατότητα δέσμευσης στο σωρό μπλοκ μεγάλου μεγέθους.
- Αλγόριθμοι συλλογής σκουπιδιών:
  - Referencing counting
  - Mark-and-sweep
  - Copying
  - Generational

Συλλογή σκουπιδιών

- Τι είναι τα “σκουπίδια”;
  - Αντικέιμενα δεσμευμένα στο σωρό που δεν είναι προσβάσιμα από καμία αλυσίδα από δείκτες από μεταβλητές του προγράμματος.
  - Αυτή η μνήμη πρέπει να ανακυκλώνεται και να επαναχρησιμοποιείται από το πρόγραμμα.
  - Ο GC εκτελείται κατά το χρόνο εκτέλεσης και όχι από το μεταγλωττιστή, αν και πολλές φορές χρειάζεται βοήθεια από αυτόν(escape-analysis).

Escape analysis

- Escape analysis can be used to convert heap allocations to stack allocations, thus reducing the amount of work needed to be done by the garbage collector. This is done using a compile-time analysis to determine whether an object allocated within a function is not accessible outside of it (i.e. escape) to other functions or threads. In such a case the object may be allocated directly on the thread stack and released when the function returns, reducing its potential garbage collection overhead.

Ο γράφος των αντικειμένων στη μνήμη

- Που είναι οι “ρίζες”;
  - Global μεταβλητές
  - Καταχωρητές
  - Τοπικές μεταβλητές και formal παράμετροι των activation records που δεσμεύονται στη στοίβα.

- Τα live αντικείμενα είναι όλα αυτά για τα οποία υπάρχει μονοπάτι που ξεκινά από τις ρίζες και καταλήγει σε αυτά.
- Η πληροφορία έρχεται στον GC από τον compiler σχετικά με το ποιοι καταχωρητές ή/και ποιές μεταβλητές περιέχουν “ρίζες”.

Reference counting

- Για κάθε αντικείμενο στο σωρό κρατάμε έναν μετρητή, ο οποίος έχει την πληροφορία για το πόσα αντικείμενα “δείχνουν” σε αυτό. Έτσι ένα αντικείμενο θεωρείται “ζωντανό”.
  - Κάθε φορά που ένα αντικείμενο σταματάει να δείχνει σε ένα άλλο τότε ο μετρητ του δεύτερου μειώνεται.
  - Κάθε φορά που ένα αντικείμενο δείχνει σε κάπια άλλο, ο μετρητής του δεύτερου αυξάνεται.
- Όταν ο μετρητής μηδενιστεί, το αντικείμενο παύει να θεωρείται ζωντανό και ανακυκλώνεται.
- Η τεχνική αυτή δεν μπορεί να ανιχνεύσει κύκλους από μη προσβάσιμα αντικείμενα και άρα έχουμε memory leaks.
- Επίσης η ανανέωση των μετρητών αναφοράς είναι ακριβή.

Mark-and-sweep

- Πραγματοποιείται σε δυο φάσεις.
  - Mark: Ξεκινώντας από τις ρίζες με DFS τρόπο για κάθε ρίζα μαρκάρουμε κάθε προσβάσιμο αντικέιμενο από αυτή.
  - Sweep: Κάθε unmarked αντικείμενο τοποθετείται σε μια freelist(SLL) και επίσης όλα τα marked αντικείμενα γίνονται unmarked.
  - Το πρόγραμμα συνεχίζει την εκτέλεση, δεσμεύει αντικείμενα από τη freelist και όταν αυτή εξαντληθεί ξανακαλείται ο GC.

Κόστος του mark-and-sweep

- Κατά το mark στάδιο, ο αλγόριθμος έχει γραμμικό κόστος στο μέγεθος των reachable αντικειμένων.
- Κατά το sweep στάδιο έχει γραμμικό κόστος στο μέγεθος στο μέγεθος του σωρού.
- Η “ωφέλιμη” δουλειά του αλγορίθμου είναι να εντοπίσει (αντικείμενα στο σωρό) – (reachable) και να τα ανακυκλώσει.
- Άρα αν τα reachable αντικείμενα είναι περίπου όσα είναι και τα αντικείμενα στο σωρό, τότε το κόστος είναι πολύ μεγάλο.

Copying garbage collector

- Χωρίζει το χώρο των δεσμευμένων αντικειμένων σε δυο ημι-χώρους. Ο GC αντιγράφει τα reachable αντικείμενα από τον έναν(from-space) στον άλλο(to-space), οι ρίζες δείχνουν στον to-space και κάθε φορά που καλείται ο collector οι δυο ημι-χώροι αλλάζουν ρόλους.
- Με την τεχνική αυτή αποφεύγουμε τον κατακερματισμό του σωρού.

Generational GC

- Most objects die young
- Over 90% garbage collected in a GC is newly created post the previous GC cycle
- If an object survives a GC cycle the chances of it becoming garbage in the short term is low and hence the GC wastes time marking it again and again in each cycle.

Generational GC

- The objects can be segregated into age based generations in different ways, e.g. by time of creation. However one common way is to consider a newly created object to be in Generation 0 (Gen0) and then if it is not collected by a cycle of garbage collection then it is promoted to the next higher generation, Gen1. Similarly if an object in Gen1 survives a GC then that gets promoted to Gen2.
- Lower generations are collected more often. This ensures lower system pauses. The higher generation collection is triggered fewer times.
- Long-lived objects are saved repeatedly by a simple copying collector
- Pointers between different generations are rare.

Implicit allocation/deallocation

- Πότε αποδεσμεύουμε τα δεδομένα;
  - Διακόπτουμε τη λειτουργία του προγράμματος όταν εξαντληθεί η μνήμη.
  - Σε πραγματικό χρόνο, όταν για παράδειγμα έχουμε νέα δέσμευση μνήμης στο σωρό.
  - Ταυτόχρονα με τη λειτουργία του προγράμματος, σαν ένα ξεχωριστό νήμα εκτέλεσης.

Stop-the-world vs. incremental vs. concurrent

- Simple stop-the-world garbage collectors completely halt execution of the program to run a collection cycle, thus guaranteeing that new objects are not allocated and objects do not suddenly become unreachable while the collector is running.
- This has the obvious disadvantage that the program can perform no useful work while a collection cycle is running (sometimes called the "embarrassing pause"). Stop-the-world garbage collection is therefore mainly suitable for non-interactive programs. Its advantage is that it is both simpler to implement and faster than incremental garbage collection.
- Incremental and concurrent garbage collectors are designed to reduce this disruption by interleaving their work with activity from the main program. Incremental garbage collectors perform the garbage collection cycle in discrete phases, with program execution permitted between each phase (and sometimes during some phases).
- Concurrent garbage collectors do not stop program execution at all, except perhaps briefly when the program's execution stack is scanned. However, the sum of the incremental phases takes longer to complete than one batch garbage collection pass, so these garbage collectors may yield lower total throughput.
- Careful design is necessary with these techniques to ensure that the main program does not interfere with the garbage collector and vice versa; for example, when the program needs to allocate a new object, the runtime system may either need to suspend it until the collection cycle is complete, or somehow notify the garbage collector that there exists a new, reachable object.