HY-225: Οργάνωση Υπολογιστών
Άνοιξη 2026

Τμ. Επ. Υπολογιστών
© Πανεπιστήμιο Κρήτης

Άσκηση 7:
Άσκηση Διαδικασιών και Συνδεδεμένης Λίστας

Από την 4η για το τέλος της 6ης εβδομάδας του Εξαμήνου
Υπενθύμιση: η Εξέταση Προόδου πλησιάζει!

7.1 Δομές Δεδομένων (Data Structures):

Σε αυτή την άσκηση θα χρησιμοποιήσουμε μία δομή δεδομένων (structure) που θα αποτελεί ένα κόμβο μιάς συνδεδεμένης λίστας (linked list). Κάθε κόμβος (δομή δεδομένων) μας θα αποτελείται από δύο λέξεις των 32 bits καθεμία (αφού ο RARS προσομοιώνει τον 32-μπιτο RISC-V, σε αντίθεση με το (Αγγλικό) βιβλίο που μιλά γιά τον 64-μπιτο): έναν ακέραιο **data** που θα περιέχει την "πληροφορία χρήστη", κι έναν δείκτη σύνδεσης (pointer) **nxtPtr** που θα περιέχει τη διεύθυνση του επόμενου κόμβου στη λίστα (στον τελευταίο κόμβο της λίστας, nxtPtr==0). Τα δύο στοιχεία (λέξεις) της δομής μας θα βρίσκονται σε διαδοχικές θέσεις (λέξεις) της μνήμης. Επομένως, κάθε δομή (κόμβος) μας θα έχει μέγεθος 2×4=8 Bytes. Διεύθυνση μιάς δομής είναι η διεύθυνση του πρώτου στοιχείου της, δηλαδή του στοιχείου με "μηδενικό offset", που γιά μας είναι το "data". Άρα, το δεύτερο στοιχείο της δομής μας, ο "nxtPtr", βρίσκεται στη διεύθυνση που προκύπτει προσθέτοντας 4×1=4 στη διεύθυνση της δομής (κόμβου).

7.2 Δυναμική Εκχώρηση Μνήμης:

Το πρόγραμμά σας θα ζητάει και θα παίρνει δομές (κόμβους) από το "περιβάλλον" (λειτουργικό σύστημα) "δυναμικά", την ώρα που τρέχει (σε run-time). Γιά το σκοπό αυτό θα χρησιμοποιήστε το κάλεσμα περιβάλλοντος (environment call - ecall) **Sbrk** (Set Break). Το κάλεσμα αυτό "σπρώχνει" πιό πέρα (προς αύξουσες διευθύνσεις μνήμης) το σημείο "Break", το όριο δηλαδή πριν από το οποίο οι διευθύνσεις μνήμης που γεννά το πρόγραμμα είναι νόμιμες, ενώ μετά από το οποίο (και μέχρι την αρχή της στοίβας) οι διευθύνσεις είναι παράνομες και ενδεχόμενη χρήση τους προκαλεί το γνωστό από την C "segmentation violation - core dumped". Το κάλεσμα περιβάλλοντος "Sbrk" έχει κωδικό 9, και περιγράφεται στην καρτέλα Help→RISCV→Syscalls του RARS, και λειτουργεί κατ' αναλογία με τα άλλα καλέσματα περιβάλλοντος (εκτύπωσης και ανάγνωσης) που χρησιμοποιήσατε σε προηγούμενες ασκήσεις: Πριν το καλέσετε, βάλτε τον κωδικό της "υπηρεσίας", 9, στον καταχωρητή a7 (x17), και βάλτε στον

γνωστό καταχωρητή a0 (x10) το όρισμα του καλέσματος, που εδώ είναι το πλήθος των νέων bytes που επιθυμείτε (ακέραιος αριθμός). Μετά την επιστροφή του, ο γνωστός καταχωρητής επιστρεφόμενης τιμής, a0 (x10) περιέχει τη διεύθυνση του νέου block μνήμης, του ζητηθέντος μεγέθους, που το σύστημα δίνει στο πρόγραμμά σας (έναν pointer). Η επιστρεφόμενη διεύθυνση μνήμης είναι πάντα διάφορη του μηδενός (εκτός –πιθανότατα– όταν γεμίσει όλη η μνήμη, αλλά δεν χρειάζεται εσείς εδώ να ελέγχετε κάτι τέτοιο), και είναι πάντα ευθυγραμμισμένη σε όρια λέξεων (πολλαπλάσιο του 4) (τουλάχιστο στη δική μας περίπτωση, που ζητάμε πάντα blocks μεγέθους πολλαπλάσιου του 4, αλλά –πιστεύω– και σε κάθε περίπτωση).

Άσκηση
7.3 Κατασκευή και Σάρωση Συνδεδεμένης Λίστας

Γράψτε και τρέξτε στον RARS, σε Assembly του RISC-V, ένα πρόγραμμα που πρώτα θα κατασκευάζει και θα γεμίζει με θετικούς ακέραιους αριθμούς μία συνδεδεμένη λίστα (linked list), και στη συνέχεια θα την σαρώνει επαναληπτικά, τυπώνοντας κάθε φορά ένα διαφορετικό υποσύνολο των στοιχείων της –συγκεκριμένα: όσα στοιχεία της είναι μεγαλύτερα από δοθείσα τιμή. Το πρόγραμμά σας θα κρατάει στον καταχωρητή **s0** (x8) τον pointer στην αρχή (στον πρώτο κόμβο) της λίστας, και θα αποτελείται από δύο κομμάτια, §7.4 και §7.5 –βλ. αμέσως παρακάτω. Στη συνέχεια, βασικά κομμάτια του προγράμματός σας θα τα κάνετε διαδικασίες (procedures), όπως περιγράφει η §7.6. Μερικές από τις διαδικασίες θα είναι υπερβολικά μικρές, αλλά αυτό γίνεται γιά λόγους εξάσκησης, ούτως ώστε το βάθος καλεσμάτων να φτάνει 2 επίπεδα κάτω από την main. (Σημείωση γιά την επιστροφή από διαδικασία: την ψευδοεντολή **jr**, που γιά επιστροφή από διαδικασία κανονικά είναι "`jr ra`", ο RARS δεν την δέχεται έτσι, αλλά απαιτεί να εμφανίζεται και το Immediate Offset, οπότε πρέπει να την δίνετε ως: "**`jr ra, 0`**").

7.4 Κατασκευή της Λίστας

Χρησιμοποιήστε τον καταχωρητή **s1** (x9) σαν pointer στην ουρά (στον τελευταίο κόμβο) της λίστας. Γιά διευκόλυνση του βρόχου κατασκευής της λίστας (επειδή η εισαγωγή σε κενή λίστα διαφέρει από την εισαγωγή σε μη κενή λίστα), αρχικοποιήστε τη λίστα να περιέχει ένα αδιάφορο ("dummy") κόμβο: ένα κόμβο με data=0. Η αρχικοποίηση γίνεται ζητώντας και παίρνοντας ένα κόμβο από το περιβάλλον (λειτουργικό σύστημα), γράφοντας data=0 και nxtPtr=0 (τελευταίος κόμβος) σε αυτόν, και κάνοντας τους s0 και s1 να δείχνουν σε αυτόν τον κόμβο (να περιέχουν τη διεύθυνσή του). Μετά, μπείτε στο βρόχο ανάγνωσης στοιχείων και κατασκευής της λίστας. Σε κάθε ανακύκλωση αυτού του βρόχου:

1. Διαβάζουμε έναν ακέραιο αριθμό από την κονσόλα.
2. Εάν ο αριθμός αυτός είναι αρνητικός ή μηδέν, βγαίνουμε από το βρόχο, αλλιώς:
3. Ζητάμε έναν νέο κόμβο από το περιβάλλον (Sbrk - memory allocation).
4. Τοποθετούμε τον αριθμό που διαβάσαμε στο πεδίο "data" του κόμβου.
5. Συνδέουμε το νέο κόμβο στην ουρά της λίστας.

7.5 Σάρωση της Λίστας

Το δεύτερο μέρος του προγράμματος θα διαβάζει έναν μη αρνητικό αριθμό, και θα τυπώνει, με τη σειρά από την αρχή μέχρι το τέλος, όσα στοιχεία της λίστας είναι **μεγαλύτερα** από αυτόν τον αριθμό. (Αφού ο αριθμός είναι ≥0, και τυπώνουμε τα στοιχεία που είναι μεγαλύτερα από αυτόν, προκύπτει ότι ο κόμβος "dummy", που έχει data=0, δεν θα τυπώνεται ποτέ· παρατηρήστε ότι όταν δίδεται το 0 ως αριθμός σύγκρισης, θα τυπώνονται όλα τα στοιχεία της λίστας (που είναι πάντα όλα θετικά), εκτός του "dummy"). Μην χρησιμοποιήστε τον pointer στον τελευταίο κόμβο της λίστας (από την παλαιά τιμή του καταχωρητή s1 (x9)) γιά να βρίσκετε πού τελειώνει η λίστα – χρησιμοποιήστε τον nxtPtr κάθε κόμβου γιά να ξέρετε αν υπάρχει ή όχι επόμενος κόμβος στη λίστα. Το μέρος αυτό του προγράμματος κάνει τα εξής:

1. Διαβάζει έναν ακέραιο αριθμό από την κονσόλα και τον φυλάει στον καταχωρητή s1 (x9). Εάν ο αριθμός αυτός είναι αρνητικός, το πρόγραμμα τερματίζει μέσω της κλήσης περιβάλλοντος (ecall) "Exit" που έχει κωδικό 10 (δηλαδή βάζετε τον αριθμό 10 στον καταχωρητή a7 πριν το ecall).
2. Αρχικοποιεί τον καταχωρητή s2 (x18) σαν δείκτη (pointer) σάρωσης, να δείχνει στον πρώτο κόμβο της λίστας (τον ξέρουμε από τον s0 (x8)).
3. Μπαίνει σ' ένα βρόχο, σε κάθε ανακύκλωση του οποίου:
    i. ελέγχει αν τα "data" του κόμβου όπου δείχνει ο s2 (x18) είναι ή όχι μεγαλύτερα από τον s1 (x9),
    ii. αν είναι μεγαλύτερα τα τυπώνει,
    iii. ελέγχει αν υπάρχει ή όχι επόμενος κόμβος στη λίστα,
    iv. αν δεν υπάρχει βγαίνει από το βρόγχο,
    v. αν υπάρχει, προχωρεί τον s2 (x18) να δείξει σε αυτόν τον επόμενο κόμβο και επιστρέφει στην αρχή του βρόχου.
4. Μετά την έξοδο του βρόχου, επιστρέφει (πάντα) στην αρχή του δεύτερου μέρους του προγράμματος, γιά να ζητήσει μιά νέα τιμή και να ξανατυπώσει τα μεγαλύτερα από αυτήν στοιχεία (εάν η τιμή δεν είναι αρνητική).

7.6 Χρήση Διαδικασιών

- Μετατρέψτε το βήμα 7.4(1) σε μια υπορουτίνα **`read_int()`** η οποία δεν παίρνει καμία παράμετρο (όρισμα) εισόδου, διαβάζει έναν ακέραιο αριθμό, και επιστρέφει τον αριθμό που διάβασε.
- Μετατρέψτε το βήμα 7.4(3) σε μια υπορουτίνα **`node_alloc()`** η οποία δεν παίρνει καμία παράμετρο εισόδου, ζητάει από το περιβάλλον να δεσμεύσει ένα κόμβο για τη λίστα, και επιστρέφει τη διεύθυνση της μνήμης που έχει δεσμευθεί, ώστε το πρόγραμμα που καλεί τη node_alloc() να εισάγει τον κόμβο στη λίστα.
- Αλλάξτε το πρόγραμμα του 7.4 ώστε να χρησιμοποιεί τις ρουτίνες read_int() και node_alloc(). Στη χρήση καταχωρητών από τις διαδικασίες που γράφετε –εδώ από τις read_int() και node_alloc()– όπως και από το πρόγραμμα σας που τις καλεί, πρέπει να τηρήσετε τις συμβάσεις χρήσης καταχωτητών (αν και, ειδικά οι read_int() και node_alloc() είναι αρκετά απλές και δεν είναι απαραίτητο να έχουν τοπικές μεταβλητές).
- Γράψτε μια υπορουτίνα **`print_node()`** που κάνει ό,τι περιγράφουν τα βήματα 7.5(3i) και 7.5(3ii). Η υπορουτίνα θα παίρνει ως εισόδους (α) τη διεύθυνση ενός κόμβου, και (β) τον ακέραιο προς τον οποίο συγκρίνουμε, και δεν θα επιστρέφει τίποτε.
- Γράψτε μια υπορουτίνα **`search_list()`** που υλοποιεί όλο το βήμα 7.5(3). Η ρουτίνα θα παίρνει ως πρώτη είσοδο τη διεύθυνση του πρώτου κόμβου τις λίστας (δείκτης σάρωσης) και ως δεύτερη είσοδο την τιμή για την οποία πρέπει να ψάξει. Στη συνέχεια θα υλοποιεί τα βήματα (i,ii,iii,iv,v), καλώντας τη συνάρτηση print_node() για τα βήματα (i,ii). Όπως είπαμε, πρέπει να τηρείτε τις συμβάσεις χρήσης καταχωτητών· κάθε υπορουτίνα που χρειάζεται αποθήκευση κάποιου καταχωρητή ή/και της διεύθυνσης επιστροφής, θα πρέπει να το κάνει στη στοίβα, με τον τρόπο που ορίζουν οι συμβάσεις.
- Αλλάξτε το πρόγραμμα σας του μέρους 7.5 ώστε να καλεί τις read_int() και search_list(), όπου η τελευταία θα καλεί την print_node(). Και πάλι πρέπει να χρησιμοποιήσετε τις συμβάσεις χρήσης των καταχωρητών μεταξύ καλούσας και καλούμενης διαδικασίας. Επίσης η κάθε ρουτίνα που χρειάζεται τη στοίβα θα πρέπει να το κάνει με τον τρόπο που ορίζουν οι συμβάσεις.
- Τελικά, το συνολικό πρόγραμμα σας θα πρέπει να έχει μια **`main()`** που αποτελείται απο δύο μέρη. Το πρώτο μέρος της θα υλοποιεί το 7.4 με χρήση των node_alloc() και read_int(), ενώ το δεύτερο θα υλοποιεί το 7.5 με χρήση των read_int(), search_list(), και print_node().

Τρόπος Παράδοσης: Παραδώστε μέσω του λογαριασμού σας στο **`elearn`**, επιλέγοντας HY-225 – Ασκήσεις 7, τα εξής: τον κώδικά σας, "**`ex07.asm`**", κι ένα στιγμιότυπο της εκτέλεσής του στον RARS, "**`ex07.jpg`**".
Θα εξεταστείτε ***και προφορικά*** για την Άσκηση 7, από βοηθούς του μαθήματος, με διαδικασία γιά την οποία θα ενημερωθείτε μέσω ηλτά (email) στη λίστα του μαθήματος.