Λειτουργικά Συστήματα

Τμήμα Πληροφορικής
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

2η Εργασία
Χειμερινό Εξάμηνο 2019

Εισαγωγή

Στόχος της εργασίας είναι η εξοικείωσή σας με αλγόριθμους δρομολόγησης διεργασιών, αλγόριθμους τοποθέτησης δυναμικής τμηματοποίησης κύριας μνήμης καθώς και αλγόριθμους αντικατάστασης σελίδων μνήμης.

Για το σκοπό αυτό, σε κάθε επιμέρους τμήμα της εργασίας, σας ζητείται να γράψετε κώδικα σε γλώσσα *Java*, ο οποίος θα υλοποιεί τους αντίστοιχους αλγόριθμους. Τα διεκπεραιωτικά κομμάτια του κώδικα έχουν υλοποιηθεί και σας παραδίδονται έτοιμα, ώστε η έξοδος των προγραμμάτων να είναι τυποποιημένη. Αυτά δεν θα πρέπει να τροποποιηθούν με κανένα τρόπο. Εσείς, θα πρέπει να συμπληρώσετε κώδικα μόνο στις μεθόδους αυτές που σημειώνονται με το σχόλιο *«Put your code here!»*. Μπορείτε ελεύθερα να προσθέσετε όσες επιπλέον (στατικές) μεθόδους ή μεταβλητές κρίνετε απαραίτητο. Φροντίστε να σχολιάσετε επαρκώς τον κώδικά σας.

Μπορείτε να βρείτε τα αρχεία πηγαίου κώδικα Java τα οποία θα πρέπει να συμπληρώσετε, για καθεμία από τις επιμέρους ασκήσεις, στην τοποθεσία:

`http://users.auth.gr/gvlahavas/opsys2019/`

Για την παράδοση της εργασίας, θα δημιουργήσετε ένα συμπιεσμένο αρχείο με όνομα *opsys2018-assignment2-1234.zip*, αντικαθιστώντας το 1234 με τον αριθμό μητρώου σας, το οποίο θα ανεβάσετε στην πλατφόρμα elearning. Το αρχείο αυτό θα πρέπει να περιέχει μόνο τα αρχεία πηγαίου κώδικα *.java*, τα οποία κατεβάσατε από την προηγούμενη τοποθεσία, με συμπληρωμένα τα κομμάτια κώδικα που σας ζητούνται σε κάθε περίπτωση. *Δεν* θα πρέπει να ανεβάσετε μαζί οποιαδήποτε άλλα αρχεία, όπως project files του IDE που χρησιμοποιήσατε κλπ.

Η εργασία είναι **ομαδική** και θα οργανωθείτε σε ομάδες των **τεσσάρων ατόμων**. Στο .zip αρχείο που θα καταθέσετε, θα πρέπει να συμπεριλάβετε κι ένα αρχείο κειμένου, στο οποίο θα αναγράφονται τα ονοματεπώνυμα και ΑΕΜ όλων των μελών της ομάδας.

Αν έχετε απορίες σχετικά με την εργασία, μπορείτε να τις υποβάλετε μέσω της πλατφόρμας elearning στο αντίστοιχο forum συζητήσεων.

1 Δρομολόγηση Διεργασιών

Στην άσκηση αυτή, θα χρειαστεί να συμπληρώσετε κομμάτια κώδικα τα οποία αναφέρονται στους αλγόριθμο δρομολόγησης διεργασιών FCFS, SJF και RoundRobin. Θα πρέπει δηλαδή να συμπληρώσετε τον κώδικα που λείπει από τα αρχεία *FCFS.java*, *SJF.java* και *RoundRobin.java*.

Έτσι, σε κάθε ένα από αυτά, θα πρέπει να συμπληρώσετε τον κώδικα που υλοποιεί δύο μεθόδους. Η πρώτη από αυτές, με όνομα *calcWaitingTime*, έχει την υπογραφή:

```
int[] calcWaitingTime(int burstTime[], int quantum)
```

Δέχεται δηλαδή ως όρισμα, ένα πίνακα, ο οποίος περιλαμβάνει τις τιμές των χρόνων καταιγισμού (*burstTime*) των διεργασιών που πρόκειται να δρομολογηθούν. Επίσης δέχεται ως όρισμα, το μέγεθος του κβάντου χρόνου (*quantum*), που χρησιμοποιείται από κάποιους αλγορίθμους. Αν ο αλγόριθμος δεν απαιτεί τη χρήση κάπους κβάντου χρόνου, αγνοήστε το στον κώδικά σας. Η μέθοδος θα πρέπει να κάνει τους απαραίτητους υπολογισμούς (στην ουσία να υλοποιεί τους αντίστοιχους αλγόριθμους), ώστε να υπολογίζει τους αντίστοιχους χρόνους αναμονής για κάθε διεργασία. Αυτούς, θα πρέπει να τους επιστρέφει ως ένα πίνακα τιμών (ο οποίος προφανώς θα πρέπει να έχει το ίδιο μέγεθος με τον πίνακα burstTime).

Η δεύτερη μέθοδος της οποίας τον κώδικα πρέπει να συμπληρώσετε, η *calcTurnAroundTime*, έχει την υπογραφή:

```
static int[] calcTurnAroundTime(int burstTime[], int waitingTime[])
```

Δέχεται δηλαδή ως όρισμα, δύο πίνακες. Ο πρώτος από αυτούς (burstTime) περιέχει όπως και στην προηγούμενη μέθοδο τους χρόνους καταιγισμού των διεργασιών. Ο δεύτερος (waitingTime) περιέχει τους χρόνους αναμονής των διεργασιών, όπως αυτοί υπολογίστηκαν από την μέθοδο *calcWaitingTime*. Η μέθοδος θα πρέπει να υπολογίζει για κάθε διεργασία τους αντίστοιχους χρόνους επιστροφής (turnaround times), τους οποίους θα πρέπει να επιστρέφει με τη μορφή πίνακα. Θεωρήστε ότι όλες οι διεργασίες καταφθάνουν ταυτόχρονα. Στην περίπτωση του αλγορίθμου FCFS θεωρήστε ότι καταφθάνουν με την αναφερόμενη σειρά.

Σε οποιαδήποτε περίπτωση, οι δείκτες (indeces) των πινάκων αναφέρονται στο ID της κάθε διεργασίας και οι οποίοι θα πρέπει να οι ίδιοι για όλους τους πίνακες. Π.χ. στη διεργασία με ID 5, θα αντιστοιχεί το στοιχείο *burstTime[5]*, το *waitingTime[5]* κλπ. Τα μεγέθη των πινάκων, άρα και ο αριθμός των διεργασιών που πρέπει να δρομολογηθούν, ενδέχεται να είναι οποιαδήποτε.

Παρέχεται έτοιμος ο κώδικας της μεθόδου *printAvgTimes*. Αυτή δέχεται ως είσοδο πίνακα με τους χρόνους καταιγισμού των διεργασιών και το μέγεθος του κβάντου χρόνου του αλγορίθμου RoundRobin. Στη συνέχεια εκτελεί τις μεθόδους *calcWaitingTime* και *calcTurnAround-*

Time, τυπώνει τα αποτελέσματά τους, ενώ επίσης υπολογίζει και τυπώνει τις μέσες τιμές των χρόνων καταιγισμού και αναμονής που έχουν υπολογιστεί. Δεν πρέπει να τροποποιήσετε με κανένα τρόπο τον κώδικα της μεθόδου.

Στη μέθοδο *main* περιλαμβάνεται ένα απλό παράδειγμα εισόδου για τις παραπάνω μεθόδους. Έτσι, αν ως είσοδος στις παραπάνω μεθόδους, χρησιμοποιηθούν 4 διεργασίες με αντίστοιχους χρόνους καταιγισμού ίσους με 5, 15, 4, 3 sec, και μέγεθος κβάντου χρόνου ίσο με 3 sec, τότε η έξοδος που θα δημιουργηθεί με τη χρήση του αλγορίθμου RoundRobin θα είναι η παρακάτω:

```
Process  Burst Time  Waiting Time  Turnaround Time
=======  ==========  ============  ===============
0            5            9             14
1            15           12            27
2            4            14            18
3            3            9             12

Average waiting time = 11.0
Average turnaround time = 17.75
```

Μπορείτε να τροποποιήσετε τη μέθοδο *main* ώστε να δοκιμάσετε διαφορετικά παραδείγματα εισόδου (χρόνοι καταιγισμού και κβάντο χρόνου). Σε οποιαδήποτε περίπτωση, επιβεβαιώστε τα αποτελέσματα που προκύπτουν από την εκτέλεση του κώδικα.

2 Αλγόριθμοι τοποθέτησης δυναμικής τμηματοποίησης

Στην άσκηση αυτή, καλείσθε να υλοποιήσετε τον κώδικα τους αλγορίθμους δυναμικής τμηματοποίησης κύριας μνήμης Best-Fit, First-Fit, Next-Fit και Worst-Fit. Έτσι, για παράδειγμα, θα πρέπει να συμπληρώσετε τον κώδικα που αφορά στη μέθοδο *worstFit* του αρχείου *WostFit.java*. Η μέθοδος, έχει την ακόλουθη υπογραφή:

```
static ArrayList<Integer> worstFit(int sizeOfBlocks[],
                                   int sizeOfProcesses[])
```

Δηλαδή ως ορίσματα, δέχεται δύο πίνακες. Ο πρώτος από αυτούς (*sizeOfBlocks*) αναφέρεται στα μεγέθη των διαθέσιμων block μνήμης σε KB. Ο δεύτερος, αναφέρεται στα μεγέθη των διεργασιών που πρέπει να τοποθετηθούν στα προηγούμενα block μνήμης, πάλι σε KB. Τόσο η αρίθμηση των blocks, όσο και των διεργασιών, είναι zero-indexed, οπότε το 1ο διαθέσιμο block είναι το block 0, το 2ο είναι το block 1 κ.ο.κ..

Αντίστοιχα για τους υπόλοιπους αλγορίθμους, θα πρέπει να συμπληρώσετε τον κώδικα στις μεθόδους bestFit, firstFit και nextFit.

Όπως φαίνεται από την υπογραφή της μεθόδου, αυτή θα πρέπει να επιστρέφει ένα ArrayList. Αυτό θα πρέπει να έχει ως δείκτη τον αριθμό της διεργασίας και να περιλαμβάνει ως τιμές, τους αριθμούς των (αρχικών) block στα οποία τοποθετήθηκε η κάθε διεργασία. Όταν δεν είναι δυνατή η τοποθέτηση μιας διεργασίας σε κάποιο block μνήμης, λόγω του ότι η διεργασία είναι μεγαλύτερη από οποιοδήποτε διαθέσιμο block, αντί για τον αριθμό του block, θα πρέπει να τοποθετείται ως τιμή στο ArrayList η τιμή -255.

Για παράδειγμα, η σωστή έξοδος, όταν υπάρχουν 5 διαθέσιμα block μνήμης με μεγέθη 200, 500, 100, 300 και 600 KB, ενώ υπάρχουν 4 διεργασίες με μεγέθη 214, 415, 112 και 425 KB, και για τον αλγόριθμο Worst-Fit είναι η παρακάτω:

```
Process No.   Block No.
===========   =========
0             4
1             1
2             4
3             Not Allocated
```

Δίνεται έτοιμος ο κώδικας της μεθόδου *printMemoryAllocation*, η οποία δέχεται ως όρισμα τον πίνακα που επιστρέφει η μέθοδος *worstFit* (όπως και οι *bestFit*, *firstFit* και *extFit*) και τυπώνει την κατανομή των διεργασιών στα block μνήμης όπως φαίνεται στο προηγούμενο παράδειγμα. Όπως βλέπετε, για όσες διεργασίες έχει λάβει την τιμή -255, αυτή τυπώνει το μήνυμα «Not Allocated». Την μέθοδο *printMemoryAllocation* δεν πρέπει να την τροποποιήσετε με κανένα τρόπο.

Στην μέθοδο *main* δίνεται επίσης ένα απλό παράδειγμα εισόδου για την μέθοδο *worstFit*. Μπορείτε να δοκιμάσετε διαφορετικά παραδείγματα ώστε να επιβεβαιώσετε την ορθή λειτουργία του κώδικά σας.

3 Αλγόριθμοι αντικατάστασης σελίδων

Στην άσκηση αυτή, καλείσθε να υλοποιήσετε κώδικα, ο οποίος θα υπολογίζει τον αριθμό των σφαλμάτων σελίδας (page faults) που παράγονται χρησιμοποιώντας τους αλγόριθμους αντικατάστασης σελίδων LRU (Least Recently Used) και FIFO (First In First Out). Στην ουσία πρέπει να υλοποιήσετε τη λειτουργία των αλγορίθμων αυτών.

Θα πρέπει λοιπόν, να συμπληρώσετε τον κώδικα της μεθόδου *pageFaults* στα αρχεία *LRU.java* και *FIFO.java*. Αυτή έχει την υπογραφή:

```
static int pageFaults(int pages[], int capacity)
```

Δηλαδή, δέχεται ως όρισμα ένα πίνακα, ο οποίος περιλαμβάνει τις τιμές του αλφαριθμητικού αναφοράς (reference string). Η αρίθμηση των πλαισίων μνήμης ξεκινά από το 0, ενώ το μέγεθος του πίνακα μπορεί να είναι οποιοδήποτε. Η μέθοδος δέχεται επίσης ως όρισμα, την παράμετρο *capacity*, η οποία αναφέρεται στον αριθμό των διαθέσιμων πλαισίων μνήμης, ο οποίος μπορεί να είναι επίσης οποιοσδήποτε. Η μέθοδος πρέπει να επιστρέφει ένα μοναδικό ακέραιο, ο οποίος θα αντιστοιχεί στον αριθμό των σφαλμάτων σελίδας που έχουν παραχθεί.

Για παράδειγμα, ο αριθμός των σφαλμάτων σελίδας που παράγονται με τη χρήση του αλγορίθμου LRU όταν ως είσοδος χρησιμοποιείται το reference string

$$5, 1, 0, 3, 2, 3, 0, 4, 2, 3, 0, 3, 5, 2$$

και υπάρχουν 3 διαθέσιμα πλαίσια μνήμης, είναι ίσος με 11.

Στη μέθοδο *main* δίνεται ένα απλό παράδειγμα εισόδου για τους δύο αλγορίθμους, το οποίο μπορείτε να τροποποιήσετε αν επιθυμείτε ώστε να ελέγξετε την ορθότητα των αλγορίθμων σας. Επιπρόσθετα ορίζεται ο αριθμός των διαθέσιμων πλαισίων μνήμης, τον οποίο μπορείτε επίσης να αλλάξετε. Η μέθοδος *main*, όπως μπορείτε εύκολα να δείτε, απλά εκτελεί την ζητούμενη μέθοδο *pageFaults* και τυπώνει το αποτέλεσμα που αυτή επιστρέφει.