

3η Άσκηση - Readme
Απόστολος Κουκουβίνης
1115202000098

Περιγραφή Αλγορίθμου

Πριν προχωρήσουμε σε μια ακριβή ανάλυση των προγραμμάτων που υλοποιούν τον αλγόριθμο που έχει επιλεγεί, καλό θα ήταν να παρουσιαστεί ο αλγόριθμος. Η βασική απαίτηση είναι να οριστούν με κάποιο τρόπο κάποιες δομές, των οποίων το πλήθος είναι δυναμικό, οι οποίες θα μπορούν να κλειδώνουν ένα μεταβλητό πλήθος από εγγραφές και όσο εργάζονται πάνω σε αυτές δεν θα πρέπει να υπάρχουν συνθήκες ανταγωνισμού. Για να επιτευχθεί αυτό ξεκινάμε με την διαπίστωση ότι σε οποιαδήποτε χρονική στιγμή, όταν μια διεργασία υποβάλλει το αίτημα της για να εργαστεί πάνω μια εγγραφή, υπάρχει ένα αυστηρώς καθορισμένο πλήθος *locks*—εμποδίων τα οποία πρέπει να περάσει, ή καλύτερα να περιμένει να "πέσουν". Οπότε όταν ένας *reader* υποβάλλει το αίτημα του για ανάγνωση από κάποιες εγγραφές *U*, τότε πρέπει να περιμένει όλους τους ενεργούς *writers* που γράφουν σε κάποια από τις εγγραφές που ανήκουν στο *U* να τελειώσουν την δουλειά τους. Αντίστοιχα όταν ένας *writer* υποβάλλει το αίτημα του για γράψιμο πάνω σε κάποια εγγραφή, τότε πρέπει να περιμένει όλους τους ενεργούς *writers* που γράφουν στην εν λόγω εγγραφή και όλους τους *readers* που διαβάζουν την εγγραφή αυτή. Όταν κάποια διεργασία p_1 μπλοκαριστεί από κάποιο *lock* θα πρέπει όλες οι διεργασίες, οι οποίες υποβάλλουν το αίτημα τους μετά την διεργασία p_1 και έχουν έστω μια κοινή εγγραφή με την p_1 πάνω στην οποία η εργασία τους πρέπει να μπλοκάρει την εργασία της p_1 , να μπλοκαριστούν από την p_1 . Για παράδειγμα αν η διεργασία p_1 διαβάζει από τις εγγραφές 10-20, και έρθει η διεργασία p_2 η οποία προσπαθεί να γράψει στην εγγραφή 15, τότε θα μπλοκαριστεί από την p_1 . Έπειτα αν έρθει η διεργασία p_2 η οποία θέλει να διαβάσει τις εγγραφές 13-25 τότε, παρόλο που p_1, p_3 μπορούν να εργαστούν μαζί, η διεργασία p_3 θα πρέπει να μπλοκαριστεί από την p_2 , ώστε να ακολουθήσουμε την λογική *FIFO*. Αντίστοιχα αν πρώτη έρθει η p_2 , τότε οι p_1, p_3 θα πρέπει να περιμένουν την p_2 να τελειώσει την δουλειά της και μετά να συνεχίσουν την εκτέλεση τους παράλληλα, αν δεν τις εμποδίζει κάτι άλλο. Τέλος αν p_1, p_3 έρθουν πρώτες, με οποιαδήποτε σειρά, τότε η p_2 πρέπει να περιμένει και τις 2 να τελειώσουν πριν συνεχίσει. Οπότε, μπορεί να ειπωθεί, ότι όταν μια διεργασία υποβάλει το αίτημα της, υπάρχουν καθορισμένα αιτήματα που την μπλοκάρουν, αυτά είναι τα αιτήματα με τα οποία έχει κοινές εγγραφές που επηρεάζει, είτε είναι σε αναμονή προς εξυπηρέτηση είτε εξυπηρετούνται. Έτσι, αυτό που έχει να κάνει μια διεργασία με το που υποβάλει το αίτημα της είναι να δει ποια αιτήματα την μπλοκάρουν. Για τον αλγόριθμο μας θα θεωρήσουμε ότι ένα *lock* είναι ένα αίτημα. Θεωρούμε ότι μια διεργασία έχει υποβάλει το αίτημα της αν έχει δημιουργήσει κάποιο *lock*. Είναι σαφές ότι αφού οριστεί το σύνολο των *locks* τα οποία πρέπει να ξεπεράσει η διεργασία, τότε είναι αδιάφορο από ποιο ακριβώς θα μπλοκαριστεί αρκεί να ελέγξει μετά όλα τα υπόλοιπα.

Υλοποίηση Αλγορίθμου

Για λόγους απλότητας του προγράμματος, θα θεωρήσουμε μόνο έναν τύπο *locks*. Το κάθε *lock* θέλουμε σίγουρα να περιλαμβάνει σίγουρα τα εξής :

1. Πληροφορίες για το ποιες/ποια εγγραφές/εγγραφή επεξεργάζεται
2. Αν πρόκειται για *reader* ή *writer*
3. Έναν *semaphore* στον οποίο θα γίνονται *blocked* τυχόν διεργασίες που περιμένουν στο *lock*

Η βασική ιδέα για την υλοποίηση είναι να υπάρχει ένας πίνακας σταθερού μεγέθους από τέτοιου τύπου *locks* στο *shared memory* και κάθε *process* να αρχικοποιεί ένα μόνο από αυτά τα *locks*. Αφού αρχικοποιήσει με κατάλληλο τρόπο το *lock* που του αναλογεί έπειτα θα ελέγξει ποια από τα υπόλοιπα ενεργά *locks* την εμποδίζουν, και θα κρατήσει τα κατάλληλα *indexes* στον πίνακα των *locks*. Έπειτα θα πρέπει να περάσει-*iterate* από όλα τα *locks* που το εμποδίζουν και να περιμένει μέχρι να πέσουν. Για να γίνει αυτό, απλά περιμένει στο πρώτο (βάσει του *indexing* σε αύξουσα σειρά), μετά στο δεύτερο κ.ο.κ. μέχρι να περάσει απ' όλα. Θα πρέπει να διασφαλίσουμε ότι τα *locks* στα οποία περιμένει ένα *process* είναι έγκυρα, για αυτό χρησιμοποιούμε ένα *valid bit*, και απλά προσπερνάμε *locks* με *valid bit* ίσο με *false*. Κάτι ακόμη που είναι άξιο προσοχής εδώ είναι το εξής φαινόμενο : έστω ότι ένα *process* p_1 πρέπει να περιμένει στα *locks* με *index* 3 και 7. Όσο περιμένει στο *lock* με *index* 3, το *process* με *index* 7 τερματίζει και απελευθερώνει το *lock*, τότε είναι πιθανό ένα άλλα *lock* να καταλάβει το *lock* με *index* 7. Αν αυτό το νέο *process* επηρεάζει διαφορετικά *records* από αυτά του p_1 , τότε όλα καλά. Αν όμως επηρεάζει κοινά *records* τότε το *lock* που θα δημιουργηθεί στο *index* 7, αφού το p_1 προχωρήσει από το *lock* 3 θα μπλοκαριστεί και στο *lock* 7, το οποίο θα είναι λάθος αφού σίγουρα δεν ικανοποιείται η *FIFO* σειρά και πιθανότατα θα έχουμε και *deadlock*. Γι' αυτό πρέπει να προσθέσουμε ένα επιπλέον χαρακτηριστικό στο *lock* το οποίο θα κρατάει το πόσες φορές έχει αλλαχτεί το κάθε *lock*. Είναι σαφές ότι ο πίνακας με τα *locks* που εμποδίζουν την κάθε διεργασία θα πρέπει να κρατά με κάποιο τρόπο αυτή την πληροφορία και να ελέγχει πριν περιμένει σε κάποιο *lock* αν είναι ίσες οι δύο τιμές, αν όχι τότε αυτό σημαίνει πως το *lock* δημιουργήθηκε μεταγενέστερα και δεν πρέπει να περιμένει σε αυτό. Επίσης, όταν μια *process* θέλει να άρει το *lock* που της αντιστοιχεί, πρέπει να καλέσει ένα πλήθος από σήματα $V()$ για να αφυπνίσει πιθανές διεργασίες που είναι μπλοκαρισμένες. Έτσι θα πρέπει το κάθε *lock* να κρατά το πλήθος των διεργασιών που είναι μπλοκαρισμένες σε αυτό. Τέλος, πρέπει να δοθεί σημασία στο εξής φαινόμενο : για να θεωρηθεί ένα *lock* ως *valid* πρέπει να έχουν επιτυχώς προχωρήσει όλες οι διεργασίες που περίμεναν στο *lock* αυτό, αλλιώς θα μπορούσαν να ανακύψουν διάφορα προβλήματα με τις διεργασίες που περιμένουν στο *lock* και τα σήματα $V()$ που λαμβάνουν. Δηλαδή, μέχρι να ξεκινήσουν επιτυχώς όλα τα *processes* που περιμένουν σε κάποιο *lock*, το *lock* αυτό αφενός να μην χρησιμοποιηθεί από κάποια άλλο *process* και αφετέρου να μην το θεωρήσουν ως ενεργό *lock*, και συνεπώς να περιμένουν σε αυτό, άλλα *processes*. Έτσι, χρειαζόμαστε ένα ακόμη *valid bit* που όταν είναι *true* θα σημαίνει πως η διεργασία που χρησιμοποιούσε αυτό το *process* έχει φύγει. Όταν τελικά όλα τα *processes* που περίμεναν στο *lock* έχουν ξεκινήσει, και τα δυο *bits* γίνονται *false*. Τελικά η δομή που χρησιμοποιείται είναι η εξής :

```
typedef struct{
    int startingId;
    int endingId;
    int WriterReader;
    bool valid;
    sem_t semaphore;
    int NumberOfBlocked;
    bool ProcessLockingLeft;
    int timesChanged;
}RangeLock;
```

Σχήμα 1: *Range Lock Struct*

Λεπτομέρειες προγραμμάτων

Όλο το *project* αποτελείται από 3 πηγαία προγράμματα, ένα αρχείο κεφαλίδας και ένα αρχείο *Makefile*. Τα 3 πηγαία προγράμματα είναι τα εξής :

1. *writer.cpp* : πηγαίος κώδικας για τους *writers*
2. *reader.cpp* : πηγαίος κώδικας για τους *readers*
3. *Init.cpp* : πρόγραμμα που αρχικοποιεί την κοινή μνήμη και με την χρήση κλήσεων *fork()* – *exec()* δημιουργεί ένα πλήθος από *writers* και *readers*

Η εκτέλεση των προγραμμάτων θα μπορούσε να γίνει από διαφορετικά *ttys* ωστόσο έχει επιλεγεί ένας συνδυασμός από *fork()* – *exec()*. Η *main* συνάρτηση του πηγαίου αρχείου είναι υπεύθυνη για την δημιουργία του *shared memory* και των *reader – writers* διεργασιών. Η *main* αυτή δέχεται ένα σύνολο ορισμάτων με μόνο ένα από αυτά να είναι υποχρεωτικό :

- *-f < filename >* : υποχρεωτικό όρισμα που αρχικοποιεί το όνομα του αρχείου
- *-r < #readers >* : προαιρετικό όρισμα που αρχικοποιεί το πλήθος των διεργασιών τύπου *reader* (*default = 10*)

- $-w < \#writers >$: προαιρετικό όρισμα που αρχικοποιεί το πλήθος των διεργασιών τύπου *writer* (*default* = 10)
- $-v < MaxValue >$: προαιρετικό όρισμα που αρχικοποιεί την μέγιστη τιμή αύξησης ή μείωση του *balance* από τους *writers* (*default* = 100)
- $-d < MaxValue >$: προαιρετικό όρισμα που αρχικοποιεί το μέγιστο χρονικό διάστημα που θα κάνει *sleep* μια διαδικασία σε *microseconds* (*default* = 20000000)
- $-print$: όταν αυτό το όρισμα δοθεί από την γραμμή εντολών, τότε το πρόγραμμα θα τυπώσει τις εγγραφές του αρχείου

Η *main* συνάρτηση αρχικοποιεί το *POSIX shared memory* με όνομα */SharedMemory* και κάνει *Truncate* και *map* το *Shared Memory*. Αρχικοποιεί κατάλληλα κάποιες από τις μεταβλητές του *shared memory*, τυπώνει, αν έχει δοθεί το κατάλληλο όρισμα στο *command line*, τις εγγραφές και έπειτα δημιουργεί τις υπόλοιπες διεργασίες. Για να το κάνει αυτό, χρησιμοποιεί την *rand()* της *stdlib* και περνάει τα ορίσματα στις διεργασίες παιδιά μέσω της *execl*. Έπειτα περιμένει τα παιδιά να τερματίσουν, τυπώνει τα στατιστικά τους προγράμματος, διαγράφει με κατάλληλο τρόπο το *shared memory* και τερματίζει.

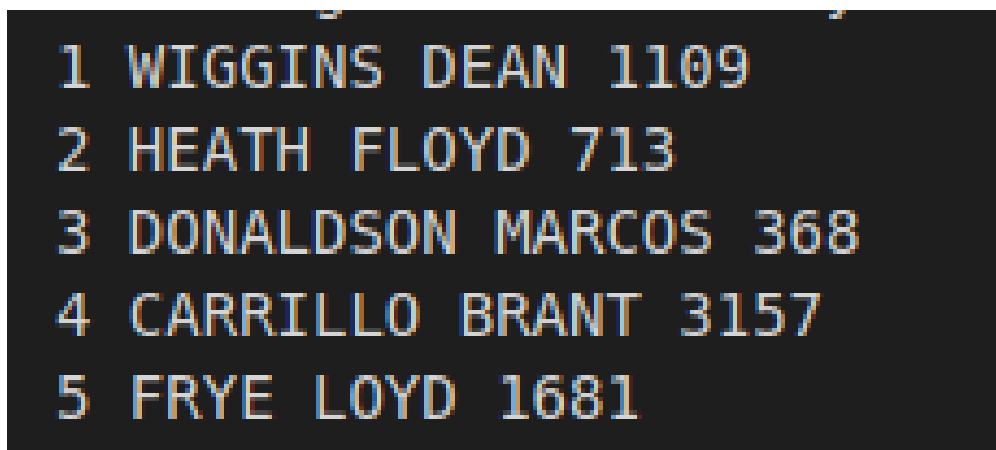
Εφόσον τα αρχεία *writer.cpp* και *reader.cpp* είναι παρόμοια (στην πραγματικότητα οι μόνες διαφορές είναι ο τρόπος που ελέγχεται αν πρέπει να περιμένει η εκάστοτε διαδικασία σε κάποιο *lock* και ο τρόπος με τον οποίο διαβάζονται οι προς επεξεργασία εγγραφές), θα αναλύσουμε μόνο τους *readers*. Αρχικά διαβάζονται τα ορίσματα όπως ακριβώς υποδεικνύει η εκφώνηση της εργασίας, κάνει *map* το *shared memory* και περιμένει να πάρει πρόσβαση σε αυτό μέσω του *semaphore* για το *shared memory*. Όταν λάβει πρόσβαση πρέπει να βρει το *lock* το οποίο θα καταλάβει και τα υποψήφια προς έλεγχο *locks*, αυτό το κάνει εντός μιας *for loop* με τις κατάλληλες συνθήκες ελέγχου. Έπειτα προχωρά ελέγχοντας ένα προς ένα τα υποψήφια *locks* κάνοντας *sem_wait* στα *locks* τα οποία είναι *valid* και επηρεάζουν κοινές εγγραφές. Αφού ελέγξει και περάσει από όλα αυτά τα *blocks* μπορεί να περάσει στην επεξεργασία των εγγραφών αφού το *lock* που έχει δημιουργήσει αποτρέπει τα *race conditions* για τις εγγραφές που επηρεάζονται. Αφού διαβάσει τις εγγραφές και υπολογίσει τον μέσο όρο με τον τετριμμένο πλέον τρόπο τυπώνει τα κατάλληλα μηνύματα. Σημειώνεται εδώ πως έχουν προστεθεί μηνύματα-εκτυπώσεις πέραν αυτών που ζητούνται από την εκφώνηση για να είναι κατανοητό το *runtime* του προγράμματος. Αφού ολοκληρώσει την επεξεργασία του, προχωρά σε ένα σύνολο από *sem_post* για να αφυπνήσει τυχόν διεργασίες που περίμεναν στο *lock* του. Τέλος ζητά πάλι πρόσβαση στο *shared memory*, ανανεώνει τα στατιστικά και τερματίζει.

Το αρχείο κεφαλίδας *shared.h* περιλαμβάνει όλες τις δηλώσεις των *structs* που χρησιμοποιούνται από τα προγράμματα καθώς και υπερπαραμέτρους με την χρήση της *#define*. Σημειώνεται εδώ ότι πρέπει να είναι εκ των προτέρων γνωστό, ένα άνω φράγμα του πλήθους των διεργασιών που πρόκειται να εκτελεστούν παράλληλα. Το πλήθος των *range locks* στην κοινή μνήμη θα είναι δύο φορές αυτό το άνω φράγμα. Αυτό γιατί σίγουρα η κάθε διεργασία χρειάζεται ένα *lock*, όμως αφού τερματίσει, το *lock* δεν γίνεται κατευθείαν διαθέσιμο σε άλλες διεργασίες, αλλά πρέπει πρώτα να αφυπνηστούν όλες οι διεργασίες που ήταν μπλοκαρισμένες σε αυτό το *lock*, δύο φορές το πλήθος των διεργασιών είναι μια αρκετά καλή τιμή που θα διασφαλίσει ότι κάθε

διεργασία θα βρει ένα *lock* για να καταλάβει.

Ορθότητα Προγράμματος

Ο τρόπος με τον οποίο έχει αναπτυχθεί το πρόγραμμα, δεν επιτρέπει σε καμία διεργασία να προχωρήσει την εκτέλεση της αν δεν τερματίσουν όλες οι διεργασίες που έχουν έρθει προγενέστερα από αυτήν και επηρεάζουν κοινές εγγραφές. Έτσι διασφαλίζεται η σειρά *FIFO* και εξαλείφονται τα *race conditions*. Σε σχόλια στο αρχείο *init.cpp* έχει δημιουργηθεί μια ειδική *for loop* η οποία δημιουργεί 5 *readers* και 5 *writers*. Η σειρά που θα δημιουργήσουν τα *locks* δεν είναι δεδομένη, αλλά συνήθως δημιουργείται μια αρκετά πεπλεγμένη σειρά. Οι *writers* προσπαθούν να γράψουν στην εγγραφή 3 (*0 – based indexing*) και οι *readers* διαβάζουν από τις εγγραφές 0 έως 4, δηλαδή όλοι οι *writers* μπλοκάρουν όλους τους *writers* και όλους τους *readers*, ενώ όλοι οι *readers* μπλοκάρουν όλους τους *writers*. Για μια εκτέλεση θα δούμε ότι το πρόγραμμα όντως δουλεύει. Έχουν προστεθεί επιπλέον εκτυπώσεις μηνυμάτων από τους *readers – writers* για καλύτερη κατανόηση. Αρχικά οι πρώτες 5 εγγραφές είναι :



```
1 WIGGINS DEAN 1109
2 HEATH FLOYD 713
3 DONALDSON MARCOS 368
4 CARRILLO BRANT 3157
5 FRYE LOYD 1681
```

Σχήμα 2: *First 5 records*

Με μέσο όρο 1407.6, έπειτα παρουσιάζεται η σειρά που δημιουργούνται τα *locks* και σε ποιο *lock* μπλοκάρεται η διεργασία.

```
Writer 17712 is trying to write to 3 and created lock 0
Writer 17712 has passed all locks
Number of active writers is 1
Writer 17710 is trying to write to 3 and created lock 1
Writer 17714 is trying to write to 3 and created lock 2
Reader 17717 is trying to read from 0 to 4 and created lock 3
Reader 17717 is waiting at lock 0
Writer 17718 is trying to write to 3 and created lock 4
Reader 17711 is trying to read from 0 to 4 and created lock 5
Reader 17711 is waiting at lock 0
Reader 17713 is trying to read from 0 to 4 and created lock 6
Reader 17713 is waiting at lock 0
Reader 17715 is trying to read from 0 to 4 and created lock 7
Reader 17715 is waiting at lock 0
Writer 17716 is trying to write to 3 and created lock 8
Reader 17719 is trying to read from 0 to 4 and created lock 9
Reader 17719 is waiting at lock 0
```

Σχήμα 3: *Locks creation*

Δηλαδή, πρώτα 3 *writers*, μετά 1 *reader*, 1 *writers*, 3 *readers*, 1 *writer* και τέλος 1 *reader*. Όπως περιμέναμε, όλοι οι *readers* περιμένουν στο *lock* που δημιούργησε ο πρώτος *writer* που ήρθε. Ο πρώτος *writer*, περνάει ανενόχλητος στο *critical section* του. Οι υπόλοιποι *writers* θα μπλοκαριστούν από τον πρώτο *writer* όπως φαίνεται στην συνέχεια. Η συνέχεια του προγράμματος είναι η εξής :

```
New Record is 4 CARRILLO BRANT 3234
Writer 17712 has finished
Writer 17714 is waiting at lock 0
Writer 17718 is waiting at lock 0
Writer 17710 is waiting at lock 0
Reader 17717 is waiting at lock 1
Reader 17715 is waiting at lock 1
Writer 17716 is waiting at lock 0
Writer 17710 has passed all locks
Number of active writers is 1
Reader 17711 is waiting at lock 1
Reader 17719 is waiting at lock 1
Reader 17713 is waiting at lock 1
New Record is 4 CARRILLO BRANT 3317
Writer 17718 is waiting at lock 1
Writer 17710 has finished
Writer 17714 is waiting at lock 1
Writer 17716 is waiting at lock 1
Writer 17714 has passed all locks
Number of active writers is 1
Reader 17717 is waiting at lock 2
Reader 17711 is waiting at lock 2
Reader 17719 is waiting at lock 2
Reader 17713 is waiting at lock 2
Reader 17715 is waiting at lock 2
New Record is 4 CARRILLO BRANT 3410
Writer 17714 has finished
Writer 17718 is waiting at lock 2
Writer 17716 is waiting at lock 2
Reader 17713 is waiting at lock 4
Reader 17715 is waiting at lock 4
Reader 17719 is waiting at lock 4
Reader 17711 is waiting at lock 4
Reader 17717 has passed all locks
Number of active readers is 1
Average balance for records 0 to 4 is 1456.2
Reader 17717 has finished
Writer17718 is waiting at lock 3
Writer17716 is waiting at lock 3
Writer 17718 has passed all locks
Number of active writers is 1
New Record is 4 CARRILLO BRANT 3359
Writer 17718 has finished
Writer 17716 is waiting at lock 4
Reader 17711 has passed all locks
Reader 17719 is waiting at lock 8
Number of active readers is 1
Reader 17715 has passed all locks
Number of active readers is 2
Reader 17713 has passed all locks
Number of active readers is 3
Average balance for records 0 to 4 is 1446
Average balance for records 0 to 4 is 1446
Reader 17711 has finished
Writer17716 is waiting at lock 5
Average balance for records 0 to 4 is 1446
Reader 17715 has finished
Reader 17713 has finished
Writer 17716 has passed all locks
Number of active writers is 1
New Record is 4 CARRILLO BRANT 3445
Writer 17716 has finished
Reader 17719 has passed all locks
Number of active readers is 1
```

Σημειώνουμε ότι η σειρά εμφάνισης των μηνυμάτων είναι μπερδεμένη, αλλά η ουσία είναι ότι και οι 4 επόμενοι *writers* μπλοκάρονται στο *lock* 0 που ανήκει στον πρώτο *writer*. Όταν τελειώσει ο πρώτος *writer* συνεχίζει ο δεύτερος, με τους υπόλοιπους *reders* και *writers* να περιμένουν στο *lock* του δεύτερου *writer*. Αν συνεχίσει κανείς να ακολουθεί τον σκελετό των μηνυμάτων εύκολα καταλαβαίνει ότι διατηρείται η σειρά *FIFO*. Επίσης μέσω των μηνυμάτων που εκτυπώνονται καταλαβαίνουμε ότι μετά τους 3 *writers* το *balance* της εγγραφής 3 είναι ίσο με 3410 και ο νέος μέσος όρος που πρέπει να υπολογιστεί είναι 1456.2, όπως και σωστά εκτυπώνεται. Αν ακολουθήσουμε και τα υπόλοιπα μηνύματα εύκολα διαπιστώνουμε ότι οι εκτέλεση είναι σωστή. Τα στατιστικά που εκτυπώνονται είναι :

```
Number of readers: 5
Mean reader time: 4
Number of writers: 5
Mean writer time: 4
Max waiting time: 30
Number of records processed: 30
```

Σχήμα 5: *Changes on records*

Πράγματι το πλήθος των *readers* και *writers* είναι ίσο με 5 όπως και τυπώνεται, και ο μέσος χρόνος που υπολογίζεται είναι 4 *seconds*. Για το *max waiting time* μπορούμε να κάνουμε μια εκτίμηση για να δούμε αν αυτό που υπολογίζεται είναι σωστό. Γνωρίζουμε ότι την μεγαλύτερη καθυστέρηση θα την έχει ο τελευταίος *reader*, και θα είναι περίπου ίση με $4*3$ (3 πρώτοι συγγραφείς) + $4*1$ (πρώτος αναγνώστης) + $4*1$ (τέταρτος συγγραφέας) $4*1$ (3 συγγραφείς που διαβάζουν ταυτόχρονα) + $4*1$ (τελευταίος συγγραφέας) = $12 + 4 + 4 + 4 + 4 = 28$, που είναι αρκετά κοντά στο 30 και άρα βγάζει νόημα. Τέλος το πλήθος των *records* που επηρεάζονται είναι $5 \cdot 5 + 5 \cdot 1 = 25 + 5 = 30$ όπως και εκτυπώνεται, αναδεικνύεται λοιπόν η ορθότητα του προγράμματος.

Μεταγλώττιση και Εκτέλεση Προγράμματος

Για την μεταγλώττιση του προγράμματος έχει δημιουργηθεί ένα *Makefile*, αρκεί η εντολή *make* στο *tty* για την δημιουργία των εκτελέσιμων από τα 3 αρχεία : δημιουργούνται τα εκτελέσιμα *main* (*Init.cpp*), *reader* (*reader.cpp*), *writer* (*writer.cpp*). Σημειώνεται, ότι στις κλήσεις των *execl* από το *Init.cpp* τα ονόματα των δύο άλλων εκτελέσιμων δίνονται με *hard – coded* τρόπο. Για την εκτέλεση του προγράμματος, μπορούν να δοθούν ένα σύνολο παραμέτρων, όπως αναφέρεται και παραπάνω. Για την πιο απλή εκτέλεση, με τις *default* τιμές για το αρχείο με τις 50 εγγραφές, η εντολή θα είναι :

`./main – f./accounts50.bin`

Το πρόγραμμα έχει δοκιμαστεί για διάφορες τιμές και για όλα τα αρχεία εγγραφών και λειτουργεί φυσιολογικά, ελπίζω τουλάχιστον.

Μια τελευταία σημείωση

Ο αλγόριθμος που έχει επιλεχτεί ορίζει μια προτεραιότητα βάσει των αιτημάτων και όχι βάσει απλά των ενεργών εγγραφών και διαβασμάτων. Δηλαδή, αν προσπαθήσει ένας *reader* να διαβάσει τις εγγραφές από 5-20 και μπλοκάρει στην εγγραφή 15 στην οποία γράφει κάποιος *writer*, τότε αν έρθει ένας *writer* ο οποίος θέλει να γράψει στην εγγραφή 10 τότε θα μπλοκαριστεί από τον *reader*. Έτσι ακολουθείται μια αυστηρή σειρά *FIFO* που είναι δίκαιη ως προς τον χρόνο άφιξης των αιτημάτων.