Πληροφορική & Τηλεπικοινωνίες
K25
Ανάπτυξη Λογισμικού
Εαρινό Εξάμηνο 2008
Καθηγητής Γιάννης Ιωαννίδης

Μέρος 1ο: Επίπεδο Αρχείου Μπλοκ

(Ημερομηνία Παράδοσης: Δευτέρα 31/3/2008, 11:59μμ)

Εισαγωγή

Το επίπεδο Αρχείου Μπλοκ υλοποιεί μια Διεπαφή Προγραμματισμού Εφαρμογών που επιτρέπει στα παραπάνω επίπεδα να επικοινωνήσουν με το δίσκο έχοντας σαν μονάδα επικοινωνίας το μπλοκ (τη σελίδα). Η διεπαφή περιέχει συναρτήσεις για τη δημιουργία, το άνοιγμα, και το κλείσιμο αρχείων, για τη σάρωση ενός δεδομένου αρχείου, για το διάβασμα συγκεκριμένου μπλοκ ενός δεδομένου αρχείου, και για την πρόσθεση και αφαίρεση μπλοκ από ένα δεδομένο αρχείο.

Οι ρουτίνες της συγκεκριμένης Διεπαφής περιγράφονται παρακάτω. Τα ονόματα όλων των ρουτινών αρχίζουν με το πρόθεμα BF, ώστε να υποδεικνύουν το γεγονός ότι πρόκειται για υλοποίηση αρχείου σε επίπεδο μπλοκ (BF=Block File). Η γλώσσα υλοποίησης είναι η C.

Οι περισσότερες ρουτίνες επιστρέφουν έναν ακέραιο αριθμό, όπου αρνητική τιμή σημαίνει ότι έχει προκύψει κάποιο σφάλμα. Υπάρχουν διάφοροι κωδικοί σφαλμάτων στο επίπεδο BF, οι οποίοι περιγράφονται στο τέλος της εκφώνησης.

Επιπλέον, η προσπέλαση σε δεδομένα ενός μπλοκ ενός αρχείου περιλαμβάνει το διάβασμα του μπλοκ στην ενδιάμεση μνήμη και μετά την επεξεργασία των περιεχομένων του εκεί (π.χ., διάβασμα ή γράψιμο). Όσο το μπλοκ είναι στη μνήμη διαθέσιμο για επεξεργασία, λέγεται ότι είναι ``καρφωμένο'' στη μνήμη. Το μπλοκ αυτό πρέπει να ``ξεκαρφωθεί'' από την υψηλότερου επιπέδου εφαρμογή που το έφερε στην ενδιάμεση μνήμη όταν αυτή ολοκληρώσει την επεξεργασία του ώστε να μπορεί αυτό να αντικατασταθεί από άλλα μπλοκ που θα χρειαστούν στο μέλλον. Η αντικατάσταση αυτή θα γίνεται σύμφωνα με την πολιτική LRU (Least Recently Used - Λιγότερο Πρόσφατα Χρησιμοποιημένο), ενώ αν το υπό

αντικατάσταση μπλοκ έχει αλλάξει κατά τη διάρκεια της παραμονής του στην ενιδάμεση μνήμη, θα πρέπει πρώτα να γραφτεί πίσω στο δίσκο (ώστε οι αλλαγές να μη χαθούν) και μετά να αντικατασταθεί. Τα μπλοκ αναγνωρίζονται από έναν `αριθμό μπλοκ' ο οποίος έχει άμεση σχέση με τη νοητή σειρά τους στο αρχείο. Ως εκ τούτου δεν είναι απαραίτητο ότι οι αριθμοί αυτοί δίνονται στο παραπάνω επίπεδο λογισμικού κάθε φορά με αύξουσα αριθμητική σειρά, καθώς τα μπλοκ μπορούν νοητά να μπαίνουν και να βγαίνουν από το αρχείο.

Επίσης, το πρώτο μπλοκ ενός αρχείου (π.χ., μπλοκ υπ΄ αριθμόν 0) μπορεί να χρησιμοποιηθεί ως μπλοκ κεφαλίδα από το επίπεδο λογισμικού BF ώστε να αποθηκεύονται εκεί διάφορα δεδομένα του αρχείου που κρίνονται χρήσιμα για το επίπεδο αυτό. Γενικότερα, το μέγεθος της κεφαλίδας ενός αρχείου είναι ακέραιο πολλαπλάσιο του μεγέθους ενός μπλοκ, αλλά για την άσκηση, ένα μόνο μπλοκ είναι αρκετό ως κεφαλίδα. Αν ο σχεδιασμός σας είναι τέτοιος ώστε να μην αρκεί ένα μπλοκ, μπορείτε να χρησιμοποιήσετε δύο (ή περισσότερα). Ανεξαρτήτως μεγέθους, η κεφαλίδα δεν είναι ορατή από τα πιο πάνω επίπεδα.

Τέλος, το μέγεθος της ενδιάμεσης μνήμης προσδιορίζεται από τη σταθερά *BF_BUFFER_SIZE*= 20 μπλοκ, ενώ το μέγεθος ενός μπλοκ προσδιορίζεται από τη σταθερά *BF_BLOCK_SIZE*= 1024 bytes.

Υπενθύμιση: Όλα τα μέρη της εργασίας αυτής θα υλοποιηθούν από ομάδες των δύο ή τριών φοιτητών, οι οποίες πρέπει ήδη να έχουν δηλωθεί στην ιστοσελίδα και να έχουν τοποθετηθεί και σε 2ωρο τμήμα σε μια από τις λίστες έξω από το γραφείο μου.

Ρουτίνες Υλοποίησης Επιπέδου Αρχείου Μπλοκ

BF_Init()

```
void BF_Init( void )
```

Η ρουτίνα αυτή χρησιμοποιείται για να αρχικοποιήσετε τις όποιες καθολικές (global) εσωτερικές δομές δεδομένων αποφασίσετε ότι χρειάζεστε να έχετε για την υλοποιήσετε το αρχείο μπλοκ. Δεν έχει καμμία παράμετρο εισόδου και δεν παράγει καμμία έξοδο.

BF_CreateFile()

```
int BF_CreateFile(
  char   *fileName,      /* όνομα αρχείου */
)
```

Η ρουτίνα αυτή δημιουργεί ένα αρχείο με όνομα *fileName*. Το αρχείο δεν πρέπει να

υπάρχει ήδη. Η ρουτίνα επιστρέφει BFE_OK εάν το καινούργιο αρχείο δημιουργηθεί με επιτυχία, ειδ' αλλοιώς κάποιον κωδικό σφάλματος.

BF_DestroyFile()

```
int BF_DestroyFile(
  char   *fileName       /* όνομα αρχείου */
)
```

Η ρουτίνα αυτή καταστρέφει το αρχείο *fileName*, σβύνοντας όλα τα δεδομένα του. Το αρχείο πρέπει να υπάρχει ήδη αλλά να μην είναι ανοιχτό. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει, ειδ' αλλοιώς κάποιον κωδικό σφάλματος.

BF_OpenFile()

```
int BF_OpenFile (
  char   *fileName       /* όνομα αρχείου */
)
```

Η ρουτίνα αυτή ανοίγει το αρχείο *fileName*. Εάν το αρχείο ανοιχτεί κανονικά, η ρουτίνα επιστρέφει έναν μικρό μη αρνητικό ακέραιο, ο οποίος χρησιμοποιείται για να αναγνωρίζεται το αρχείο (όπως περιγράφουμε παρακάτω). Ειδ' αλλοιώς επιστρέφει κάποιον κωδικό σφάλματος. Ένα αρχείο μπορεί να έχει ανοιχτεί παραπάνω από μία φορές.

Θα πρέπει να κρατάτε στην μνήμη έναν πίνακα για όλα τα ανοιχτά αρχεία. Ο ακέραιος που επιστρέφει η *BF_OpenFile()* είναι ο αριθμός της θέσης του πίνακα που αντιστοιχεί στο αρχείο αυτό. Σ' αυτόν τον πίνακα θα κρατάτε ο,τιδήποτε σχετικό κρίνετε ότι πρέπει να είναι άμεσα διαθέσιμο για κάθε ανοιχτό αρχείο (π.χ., γενικές πληροφορίες για το αρχείο από το μπλοκ-κεφαλίδα του, το αναγνωριστικό του αρχείου έτσι όπως έχει ανοιχθεί από το λειτουργικό σύστημα, κτλ. - χωρίς τα παραπάνω να θεωρούνται απαραίτητα). Μπορείτε να υποθέσετε ότι οποιαδήποτε στιγμή δεν θα υπάρχουν περισσότερα από *MAXOPENFILES* = 25 ανοιχτά αρχεία.

Όπως προαναφέρθηκε, η κεφαλίδα ενός αρχείου είναι ορατή μόνο στο επίπεδο αρχείου μπλοκ που χτίζουμε και όχι στα ανώτερα επίπεδα. Η κεφαλίδα αυτή διαβάζεται στην *BF_OpenFile()* και η πληροφορία που περιέχει αποθηκεύεται στη μνήμη σε κατάλληλες δομές δεδομένων που εσείς θα επιλέξετε. Τα μπλοκ της κεφαλίδας (συνήθως είναι ένα μόνο) δεν τοποθετούνται μέσα στην ενδιάμεση μνήμη που χρησιμοποιείται για τη μεταφορά των μπλοκ που ζητούν τα ανώτερα επίπεδα. Επίσης τα μπλοκ κεφαλίδας δεν καρφώνονται στη μνήμη και για αυτά δεν ισχύει η πολιτική αντικατάστασης που ακολουθείται.

Το ίδιο αρχείο μπορεί να ανοιχθεί περισσότερες από μία φορές. Με κάθε άνοιγμα, η πληροφορία που το αφορά καταλαμβάνει μία επιπλέον θέση στον πίνακα ανοιχτών αρχείων. Στην περίπτωση που ένα αρχείο είναι ήδη ανοιχτό, η *BF_OpenFile()* επιστρέφει κανονικά ένα μικρό θετικό ακέραιο που δείχνει στη νέα θέση του πίνακα ανοιχτών αρχείων που χρησιμοποιήθηκε. Στη μεταβλητή *BF_errno* δίνεται η τιμή BFE_FILEOPEN και με αυτόν τον τρόπο μπορεί το ανώτερο επίπεδο να γνωρίζει ότι το αρχείο είναι ήδη ανοιχτό και αν θέλει να λάβει μέτρα, χωρίς

αυτό να σημαίνει ότι υπάρχει κάποιο σφάλμα.

BF_CloseFile()

```
int BF_CloseFile (
  int    fileDesc        /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
)
```

Η ρουτίνα αυτή κλείνει το αρχείο με αναγνωριστικό αριθμό *fileDesc*. Επίσης σβύνει την όλη καταχώρηση που αντιστοιχεί στο αρχείο αυτό στον πίνακα ανοιχτών αρχείων. Τέλος, ενημερώνει στο δίσκο όλα τα μπλοκ του αρχείου που είναι που έχουν μεν ξεκαρφωθεί αλλά παραμένουν ``βρωμισμένα'' στη μνήμη. Επιστρέφει BFE_OK εάν το αρχείο κλείσει επιτυχώς, ειδ' αλλοιώς κάποιον κωδικό σφάλματος. Ένα από τα σφάλματα αντιστοιχεί σε απόπειρα κλεισίματος αρχείου ενώ κάποιο μπλοκ του είναι ``καρφωμένο'' στην ενδιάμεση μνήμη.

BF_GetFirstBlock()

```
int BF_GetFirstBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    *blockNum      /* αριθμός του πρώτου μπλοκ του αρχείου */
  char   **blockBuf     /* έμμεσος δείκτης προς την ενδιάμεση μνήμη */
)
```

Η ρουτίνα αυτή διαβάζει στην ενδιάμεση μνήμη το πρώτο έγγυρο μπλοκ του αρχείου με αναγνωριστικό αριθμό *fileDesc* και μετά βάζει τον δείκτη **blockBuf* να δείχνει σ' αυτό. Επίσης αναθέτει στο **blockNum* τον αριθμό του μπλοκ που διαβάστηκε. Προσέξτε ότι αυτό δεν είναι απαραίτητα το μπλοκ υπ' αριθμόν 1 (αν υποθέσουμε ότι η μέτρηση των μπλοκ ξεκινά από εκεί), καθότι το τελευταίο (και ίσως και άλλα) μπορεί να έχει επιστραφεί από το αρχείο στο σύστημα για ανακύκλωση. Το μπλοκ που διαβάστηκε πρέπει να ``καρφωθεί'' στη μνήμη και να παραμείνει σ' αυτήν την κατάσταση μέχρι να ``ξεκαρφωθεί'' μέσω της ρουτίνας *BF_UnpinBlock()*, η οποία περιγράφεται παρακάτω. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει, BFE_EOF εάν έχουμε φτάσει στο τέλος του αρχείου (δηλαδή, εάν το αρχείο δεν έχει κανένα μπλοκ), ή κάποιον κωδικό σφάλματος εάν κάτι δεν λειτουργήσει όπως πρέπει.

BF_GetNextBlock()

```
int BF_GetNextBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    *blockNum      /* αριθμός του μπλοκ του αρχείου το επόμενο του οποίου
αναζητείται */
  char   **blockBuf     /* έμμεσος δείκτης προς την ενδιάμεση μνήμη */
)
```

Η ρουτίνα αυτή διαβάζει στην ενδιάμεση μνήμη το πρώτο έγγυρο μπλοκ του αρχείου με αναγνωριστικό αριθμό *fileDesc* μετά από το μπλοκ με αριθμό **blockNum* και μετά βάζει τον δείκτη **blockBuf* να δείχνει σ' αυτό. Επίσης, στο τέλος της

διαδικασίας, ενημερώνει το **blockNum* ώστε να περιέχει τον αριθμό του νέου μπλοκ που διαβάστηκε. Προσέξτε και πάλι ότι αυτό δεν είναι απαραίτητα το αριθμητικά αμέσως επόμενο του **blockNum*, καθότι το τελευταίο (και ίσως και άλλα) μπορεί να έχει επιστραφεί από το αρχείο στο σύστημα για ανακύκλωση. Το μπλοκ που διαβάστηκε πρέπει να ``καρφωθεί'' στη μνήμη και να παραμείνει σ' αυτήν την κατάσταση μέχρι να ``ξεκαρφωθεί'' μέσω της ρουτίνας *BF_UnpinBlock()*, η οποία περιγράφεται παρακάτω. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει, BFE_EOF εάν έχουμε φτάσει στο τέλος του αρχείου (δηλαδή δεν υπάρχει επόμενο μπλοκ), BFE_INVALIDBLOCK εάν ο αρχικά δοσμένος αριθμός μπλοκ (**blockNum*) δεν αναφέρεται σε έγγυρο μπλοκ, ή κάποιον άλλον κωδικό σφάλματος εάν κάτι άλλο δεν λειτουργήσει όπως πρέπει.

BF_GetThisBlock()

```
int BF_GetThisBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    blockNum       /* αριθμός του αναζητούμενου μπλοκ του αρχείου */
  char   **blockBuf     /* έμμεσος δείκτης προς την ενδιάμεση μνήμη */
)
```

Η ρουτίνα αυτή διαβάζει στην ενδιάμεση μνήμη το μπλοκ με αριθμό *blockNum* του αρχείου με αναγνωριστικό αριθμό *fileDesc* και μετά βάζει τον δείκτη **blockBuf* να δείχνει σ' αυτό. Ο αριθμός *blockNum* πρέπει να αναφέρεται σε έγγυρο μπλοκ. Το μπλοκ που διαβάστηκε πρέπει να ``καρφωθεί'' στη μνήμη και να παραμείνει σ' αυτήν την κατάσταση μέχρι να ``ξεκαρφωθεί'' μέσω της ρουτίνας *BF_UnpinBlock()*, η οποία περιγράφεται παρακάτω. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει, BFE_INVALIDBLOCK εάν ο αρχικά δοσμένος αριθμός μπλοκ (*blockNum*) δεν αναφέρεται σε έγγυρο μπλοκ, ή κάποιον άλλον κωδικό σφάλματος εάν κάτι άλλο δεν λειτουργήσει όπως πρέπει.

Κάθε επίπεδο χρησιμοποιεί δική του, διαφορετική αρίθμηση για τα μπλοκ του αρχείου που διαχειρίζεται, καθώς χρειάζεται να κρύβει τις κεφαλίδες που χρησιμοποιεί. Έτσι, αν το ανώτερο επίπεδο ζητήσει από το επίπεδο αρχείου μπλοκ το μπλοκ με αριθμό 0, στην πραγματικότητα θα πάρει το πρώτο μπλοκ αμέσως μετά την κεφαλίδα (εφόσον αυτό είναι έγκυρο). Κάθε φορά λοιπόν που ζητείται ένα μπλοκ, πρέπει να γίνεται ο κατάλληλος μετασχηματισμός έτσι ώστε να αποκρύπτεται η ύπαρξη της κεφαλίδας.

BF_AllocBlock()

```
int BF_AllocBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    *blockNum      /* αριθμός του νέου μπλοκ του αρχείου */
  char   **blockBuf     /* έμμεσος δείκτης προς την ενδιάμεση μνήμη */
)
```

Η ρουτίνα αυτή προσθέτει ένα νέο άδειο μπλοκ στο αρχείο με αναγνωριστικό αριθμό *fileDesc*. Αναθέτει τον αριθμό του νέου μπλοκ στο **blockNum* και βάζει τον δείκτη **blockBuf* να δείχνει σε αντίστοιχο χώρο στην ενδιάμεση μνήμη. Το νέο

μπλοκ μπορεί να είναι πρωτοεμφανιζόμενο ή να προέρχεται από ανακύκλωση. Ανεξάρτητα από αυτό, το νέο μπλοκ πρέπει να αρχικοποιείται με μηδενικά. Επίσης το νέο μπλοκ πρέπει να ``καρφωθεί'' στη μνήμη και να παραμείνει σ' αυτήν την κατάσταση μέχρι να ``ξεκαρφωθεί'' μέσω της ρουτίνας *BF_UnpinBlock()*, η οποία περιγράφεται παρακάτω. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει ή κάποιον κωδικό σφάλματος εάν κάτι δεν λειτουργήσει όπως πρέπει.

BF_DisposeBlock()

```
int BF_DisposeBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    *blockNum      /* αριθμός του ανακυκλούμενου μπλοκ του αρχείου */
)
```

Η ρουτίνα αυτή επιστρέφει προς ανακύκλωση το μπλοκ με αριθμό **blockNum* του αρχείου με αναγνωριστικό αριθμό *fileDesc*. Το μπλοκ δεν μπορεί να είναι ``καρφωμένο'' στην ενδιάμεση μνήμη από κανέναν. Αν είναι, τότε η *BF_DisposeBlock()* πρέπει να αποτυγχάνει. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει ή κάποιον κωδικό σφάλματος εάν κάτι δεν λειτουργήσει όπως πρέπει.

BF_UnpinBlock()

```
int BF_UnpinBlock(
  int    fileDesc       /* αριθμός που αντιστοιχεί σε ανοιχτό αρχείο */
  int    blockNum       /* αριθμός του μπλοκ του αρχείου που ``ξεκαρφώνεται'' */
  int    dirty          /* ένδειξη αλλαγής του μπλοκ στην ενδιάμεση μνήμη */
)
```

Η ρουτίνα αυτή ειδοποιεί ότι το μπλοκ με αριθμό *blockNum* του αρχείου με αναγνωριστικό αριθμό *fileDesc* δεν χρειάζεται πια στην ενδιάμεση μνήμη και μπορεί να ``ξεκαρφωθεί''. Η παράμετρος dirty είναι ίση με *TRUE*=1 εάν το μπλοκ έχει αλλάξει στην ενδιάμεση μνήμη από τη στιγμή που ``καρφώθηκε'' εκεί (οπότε σε περίπτωση αντικατάστασής του πρέπει πρώτα να γραφτεί στη θέση του στο δίσκο), ειδ' αλλοιώς είναι ίση με *FALSE*=0. Η ρουτίνα επιστρέφει BFE_OK εάν επιτύχει ή κάποιον κωδικό σφάλματος εάν κάτι δεν λειτουργήσει όπως πρέπει. Αν το μπλοκ που ξεκαρφώνεται είναι ``βρωμισμένο'' τότε το σύστημα πρέπει κάποια στιγμή να ενημερώσει και το αντίστοιχο μπλοκ στο δίσκο. Μάλιστα, αν το μπλοκ έχει ένδειξη dirty=*TRUE* από ένα χρήστη και dirty=*FALSE* από έναν άλλο, τότε το πρώτο υπερισχύει. Η ενημέρωση στο δίσκο δε γίνεται από την ίδια την *BF_UnpinBlock()*, αλλά αμέσως πριν την αντικατάστασή του στα πλαίσια άλλων ρουτινών σύμφωνα με την πολιτική που εφαρμόζεται. Σε περίπτωση που υπάρχουν ``βρωμισμένα'' μπλοκ που έχουν ξεκαρφωθεί αλλά δεν έτυχε να αντικατασταθούν, τότε η ενημέρωσή τους στο δίσκο γίνεται από τη ρουτίνα *BF_CloseFile()*, που περιγράφεται παρακάτω.

BF_PrintError()

```
void BF_PrintError(
  char    *errString            /* κείμενο για εκτύπωση */
```

)

Η ρουτίνα τυπώνει το κείμενο που δείχνει η παράμετρος *errString*, και μετά τυπώνει το μήνυμα που αντιστοιχεί στο τελευταίο σφάλμα που προέκυψε από οποιαδήποτε από τις ρουτίνες. Για τον σκοπό αυτό, η ρουτίνα αυτή χρησιμοποιεί μία καθολική (global) μεταβλητή *BF_errno* η οποία αποθηκεύει πάντα τον κωδικό του πλέον πρόσφατου σφάλματος. Ο κωδικός αυτός σφάλματος πρέπει πάντα να ενημερώνεται σωστά σε όλες τις άλλες ρουτίνες. Η ρουτίνα αυτή δεν έχει δική της τιμή επιστροφής.

Διαχείριση Σφαλμάτων Επιπέδου Αρχείου Μπλοκ

Ακολουθεί μια λίστα με κωδικούς σφαλμάτων του επιπέδου Αρχείου Μπλοκ:

```
#define BFE_OK                  0    /* OK */
#define BFE_NOMEM               -1   /* έλλειψη μνήμης */
#define BFE_NOBUF               -2   /* έλλειψη χώρου ενδιάμεσης μνήμης */
#define BFE_BLOCKFIXED          -3   /* μπλοκ ήδη ``καρφωμένο'' στη μνήμη */
#define BFE_BLOCKNOTINBUF       -4   /* μπλοκ για ``ξεκάρφωμα'' δεν είναι στη μνήμη */
#define BFE_BLOCKINBUF          -5   /* μπλοκ ήδη στη μνήμη */
#define BFE_OS                  -6   /* γενικό σφάλμα Λειτουργικού Συστήματος */
#define BFE_INCOMPLETEREAD      -7   /* ατελής ανάγνωση μπλοκ */
#define BFE_INCOMPLETEWRITE     -8   /* ατελές γράψιμο σε μπλοκ */
#define BFE_INCOMPLETEHDRREAD   -9   /* ατελής ανάγνωση μπλοκ-κεφαλίδας */
#define BFE_INCOMPLETEHDRWRITE  -10  /* ατελές γράψιμο σε μπλοκ-κεφαλίδα */
#define BFE_INVALIDBLOCK        -11  /* μη έγκυρος αναγνωριστικός αριθμός μπλοκ */
#define BFE_FILEOPEN            -12  /* αρχείο ήδη ανοιχτό */
#define BFE_FTABFULL            -13  /* λίστα ανοιχτών αρχείων πλήρης */
#define BFE_FD                  -14  /* μη έγκυρος αναγνωριστικός αριθμός αρχείου */
#define BFE_EOF                 -15  /* τέλος αρχείου */
#define BFE_BLOCKFREE           -16  /* μπλοκ ήδη διαθέσιμο */
#define BFE_BLOCKUNFIXED        -17  /* μπλοκ ήδη ``ξεκαρφωμένο'' */
```

Δεν είναι απαραίτητο ότι η υλοποίησή σας θα χρησιμοποιήσει όλους τους

παραπάνω. Επίσης, αν κρίνετε ότι χρειάζονται και άλλοι κωδικοί για την δική σας υλοποίηση, μπορείτε να τους προσθέσετε στον δικό σας πρόγραμμα.

Σχόλια για την Υλοποίηση

Θα πρέπει να ορίσετε δύο δομές (π.χ., απλούς πίνακες ή πίνακες κατακερματισμού) στο πρόγραμμά σας. Στον πρώτο θα κρατάτε όποια στοιχεία χρειάζεστε για τα ανοιχτά αρχείο, με κάθε (μη κενή) θέση του πίνακα να περιέχει ή να δείχνει προς τα στοιχεία που αντιστοιχούν σε ένα αρχείο. Αν ο πίνακας είναι γεμάτος και ζητηθεί και άλλο άνοιγμα αρχείου, το αντίστοιχο μήνυμα σφάλματος πρέπει να παραχθεί. Στον δεύτερο θα κρατάτε όποια στοιχεία κρίνετε χρήσιμα για τα μπλοκ της ενδιάμεσης μνήμης και των εκάστοτε περιεχομένων τους. Κι εδώ, κάθε (μη κενή) θέση του πίνακα θα περιέχει ή θα δείχνει προς τα στοιχεία που αντιστοιχούν σε ένα μπλοκ. Ανάμεσα στα στοιχεία που θα κρατάτε πρέπει να είναι και οι πληροφορίες που χρειάζεστε για να εφαρμόσετε σωστά την πολιτική αντικατάστασης μπλοκ στην ενδιάμεση μνήμη (LRU).

Να επισημάνουμε ότι τα παραπάνω είναι οδηγίες αλλά όχι υποχρεωτικές. Εσείς μπορείτε να υλοποιήσετε τα πράγματα διαφορετικά αν θέλετε. Οι διεπαφές των ρουτινών και η λειτουργικότητά τους είναι τα μόνα πράγματα που δεν μπορείτε να αλλάξετε.

Δε χρειάζεται να υλοποιήσετε έλεγχο συνδρομικότητας. Αν το ανώτερο επίπεδο ζητήσει κάποιο μπλοκ που υπάρχει ήδη σε κάποια θέση της μνήμης, τότε θα πάρει ένα δείκτη στη θέση αυτή. Ακόμα και αν κάποιος άλλος έχει βρωμίσει το μπλοκ αυτό, το επίπεδο που υλοποιούμε δεν το γνωρίζει και δεν ενδιαφέρεται. Τελικά στο δίσκο θα φτάσουν οι αλλαγές του τελευταίου που έγγραψε ένα μπλοκ και αυτό είναι απόλυτα σωστό. Προσέξτε ότι για τον έλεγχο σχετικά με το αν ένα μπλοκ είναι ήδη στη μνήμη δε μας αρκεί ο αναγνωριστικός κωδικός του αρχείου. Ο έλεγχος πρέπει να βασίζεται στο όνομα του αρχείου, αφού μπορεί το ίδιο αρχείο να έχει ανοιχθεί περισσότερες από μία φορές και σε κάθε άνοιγμα αντιστοιχεί διαφορετικός κωδικός.

Σχολιασμός, Έλεγχος Σφαλμάτων, και Γενική Μορφοποίηση

Όπως πάντοτε, αναμένεται καλός σχολιασμός του προγράμματος, και εσωτερικός (ανάμεσα στις γραμμές κώδικα) και εξωτερικός (στην αρχή κάθε ρουτίνας). Ένας γενικός κανόνας είναι να σχολιάζετε τα προγράμματά σας σαν να πρόκειται να τα δώσετε σε κάποιον άλλον ο οποίος θα τα επεκτείνει και ο οποίος δεν έχει ιδέα για το τι κάνατε όταν τα γράφατε (και δεν μπορεί ούτε να σας βρει να σας ρωτήσει).

Επίσης, θα πρέπει να ελέγχετε για διάφορα σφάλματα που μπορούν να προκύψουν και να βεβαιωθείτε ότι ο κώδικάς σας τερματίζει ομαλά, με μηνύματα που έχουν

νόημα, σε όλες τις εισόδους που ικανοποιούν την παραπάνω περιγραφή.

Διαδικαστικές Λεπτομέρειες

Το τί ακριβώς θα παραδόσετε, πού θα βρείτε τα αρχεία με τα οποία θα ελέγξετε τελικά το πρόγραμμά σας, κτλ., θα ανακοινωθεί πολύ σύντομα.

Τα προγράμματά σας θα πρέπει να δουλεύουν στα μηχανήματα του τμήματος (είτε στα sun (Unix) είτε στα pc (Windows)) ώστε να μπορούν να ελεγχθούν. Ακόμη και αν δουλέψετε σε δικούς σας υπολογιστές θα πρέπει να βεβαιωθείτε ότι το τελικό αποτέλεσμα δουλεύει και εδώ τοπικά. Η έκδοση της C που θα χρησιμοποιήσετε πρέπει να έχει ακέραιους των 4 bytes και να επιτρέπει αλλαγή ερμηνείας τύπου δεδομένων (type casting).