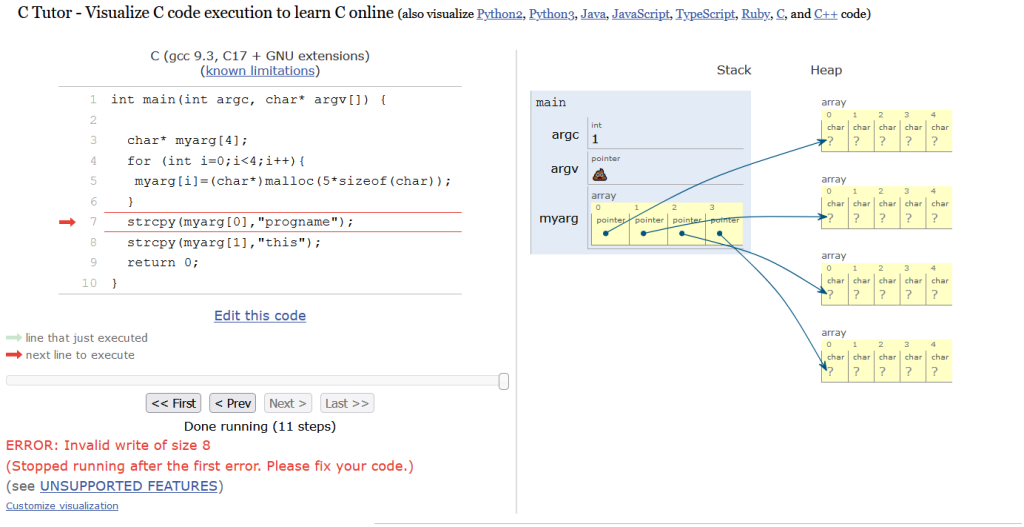
Πίνακες δύο διαστάσεων

- Είδαμε στο προηγούμενο εργαστήριο πίνακες δύο διαστάσεων ακεραίων. Παράδειγμα για char** ή char* arg[]

```
#define MAXARGS 5
#define MAXARGSIZE 20
int main(int argc, char* argv[]) {
        char* myarg[MAXARGS];
        for (int i=0;i<MAXARGS;i++){
                myarg[i]=(char*)malloc(MAXARGSIZE*sizeof(char));
        }
        //here we should also check the size to be less than MAXARGSIZE
        strcpy(myarg[0],"progname");
        strcpy(myarg[1],"this");
        return 0;
}
```

Προσοχή στις διαφορές του Tutor από τη πραγματική ζωή

- Σε πραγματικό πρόγραμμα τι θα γινόταν με τον παραπάνω κώδικα?
  - Θα γράφαμε στις θέσεις των επόμενων κελιών χωρίς να βγάζει σφάλμα αν δεν παραβιάζαμε το όριο της μνήμης



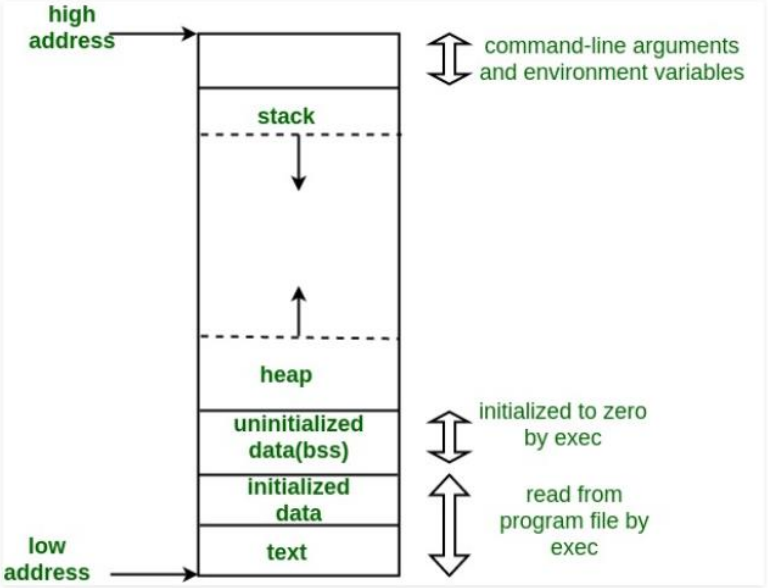
- Στον tutor βγάζει error κατευθείαν

Σημεία προσοχής

- Διαφορά character array από string
  - A char array is just a char array. It stores independent integer values (char is just a small integer type to represent the ASCII value). A char array does not have to end in \0. \0 has no special meaning in a char array. It is just a zero value.
    - Σε αυτή τη περίπτωση όμως πρέπει να προσέχετε στη χρήση του ώστε να φροντίσετε να σταματήσετε στα όρια του πίνακα
    - Επίσης να προσέχετε πως συμπεριφέρονται οι διάφορες συναρτήσεις της C και αν χρειάζονται να βρίσκουν το \0. Παράδειγμα με printf εκτύπωσης πέρα από τα όρια
  - A string is a sequence of characters terminated by \0. So, if you want to use your char array *as a string* you have to terminate your string with a \0.
    - Σε αυτή τη περίπτωση, αν προσπαθήσουμε να τυπώσουμε το char array μέσω της printf και δεν υπάρχει \0, θα τυπώσει από την αρχή του array μέχρι να βρει terminating character τυχαία στις επόμενες θέσεις μνήμης!
    - Επίσης αν στο ενδιάμεσο του char array υπάρχει \0, τότε θα σταματήσει σε αυτό το σημείο
- Η χρήση “ ” βάζει αυτόματα στο τέλος το \0 (char array[]=“mystring”)
- Η χρήση ‘ ‘ δεν το κάνει
- Δήλωση char* s=“hello” από char s[]=“hello”

Άσκηση 4 παραλλαγές/έξτρα (4.4 από literal και όχι command line)

- Διαφορές
- char* s="hello"
  - Δημιουργία στατικού string (σαν const)
    - Compile time, προσθέτει και το terminating character
  - Αποθηκεύεται σε read-only κομμάτι μνήμης και ο δείκτης σε αυτό αποθηκεύεται στη pointer μεταβλητή s στα initialized data
    - Μπορείτε να αλλάξετε τα περιεχόμενα του pointer να δείχνει αλλού, δεν μπορείτε να αλλάξετε τα περιεχόμενα του αρχικοποιημένου string
  - Προσπάθεια αλλαγής των περιεχομένων του "hello" καταλήγει σε undefined behavior (συνήθως segmentation fault)
    - Π.χ. s[0]='j' => undefined behavior
  - Αν θέλετε να προστατέψετε το πρόγραμμα από τέτοιο λάθος μπορείτε να το δηλώσετε σαν const char* s
    - Compile time error αν προσπαθήσετε να το κάνετε access στον κώδικα
- από char s[]="hello"
  - Δημιουργία κανονικού array στα υπόλοιπα κομμάτια μνήμης
  - Μπορούμε να το χειριστούμε όπως οποιοδήποτε array, να αλλάξουμε κάποιο κομμάτι του περιεχομένου
    - Πχ s[o]='j' => "jello"
  - Το πού θα δημιουργηθεί εξαρτάται από το που θα υπάρχει αυτή η δήλωση
    - Αν είναι σε function στο stack
    - Αν είναι στο κυρίως πρόγραμμα στο data
- Αν θέλετε να τα χρησιμοποιήσετε σαν function arguments είναι ισοδύναμα. Γιατί?
  - Όταν δηλώνουμε ένα array σαν function argument παίρνουμε στην ουσια τον pointer στην αρχή του. Οι πίνακες περνάνε πάντα σαν call by reference
  - Αρκεί βέβαια η λειτουργία του function να μην προσπαθεί να αλλάξει τα περιεχόμενα του string αν στην περίπτωση του char* s έχει αρχικοποιηθεί με τον παραπάνω τρόπο
    - Θα μπορούσαμε να αλλάξουμε τον pointer (char* s) να δείχνει στον array



Άσκηση 4.4

- Πολύ χρήσιμη η απεικόνιση από το tutor

- Θα δοκιμάσουμε μία παραλλαγή για να γράψουμε το δικό μας parser για arguments

Παραλλαγές δηλώσεων με const

- Το const μπορεί να σας προστατεύσει σε πολλές περιπτώσεις από το compile time αν φροντίσετε να το χρησιμοποιήσετε

- char* is a **mutable** pointer to a **mutable** character/string.
  - Με εξαίρεση αν το αρχικοποιήσετε με string literal
- const char* is a **mutable** pointer to an **immutable** character/string. You cannot change the contents of the location(s) this pointer points to. Also, compilers are required to give error messages when you try to do so.
- char* const is an **immutable** pointer (it cannot point to any other location) **but** the contents of location at which it points are **mutable**.
- const char* const is an **immutable** pointer to an **immutable** character/string.

Ερωτήσεις

- Τί θα γινόταν στην άσκηση 4.4 αν αντί για τα argv χρησιμοποιούσαμε ένα string literal μέσω pointer?

- Τί θα γινόταν στο παράδειγμα με την strtod (slide 16 Θεωρίας) αν χρησιμοποιούσαμε p αντί για &p?

- Από τη στιγμή που οι arrays είναι στην ουσία pointers στην αρχή του πίνακα, θα μπορούσαμε να πάρουμε διπλό pointer με το &array και να το χρησιμοποιήσουμε πχ στο strtod για να αρχικοποιήσουμε μέσω πίνακα το string και άρα να μπορούμε να το αλλάξουμε?

Ερωτήσεις-Απαντήσεις

- Τί θα γινόταν στην άσκηση 4.4 αν αντί για τα argv χρησιμοποιούσαμε ένα string literal μέσω pointer?
  - Segmentation fault αν έχει match, δηλαδή αν βρεθεί το token μέσα στη συμβολοσειρά
  - Η strtok αλλάζει τα περιεχόμενα του string που της δίνουμε στη θέση του match
- Τί θα γινόταν στο παράδειγμα με την strtod (slide 16 Θεωρίας) αν χρησιμοποιούσαμε p αντί για &p?
  - Η strtod αλλάζει το δεύτερο argument για να αποθηκεύσει το σημείο στο οποίο σταμάτησε
  - Segmentation fault γιατί ορίζουμε τον p να δείχνει στο string literal, που όπως είπαμε είναι στο Read-only κομμάτι
- Από τη στιγμή που οι arrays είναι στην ουσία pointers στην αρχή του πίνακα, θα μπορούσαμε να πάρουμε διπλό pointer με το &array και να το χρησιμοποιήσουμε πχ στο strtod για να αρχικοποιήσουμε μέσω πίνακα το string και άρα να μπορούμε να το αλλάξουμε?
  - Διαφορά array από &array
    - Δείχνουν στην ίδια διεύθυνση αλλά αλλάζει η λογική του pointer arithmetic
    - &array+1 παει μετά το μέγεθος του πίνακα
    - array+1 πάει τοσες θέσεις μετά όσο και το μέγεθος του τύπου της πληροφορίας
  - Άρα το &array δεν λειτουργεί σαν **, οπότε κατά πάσα πιθανότητα θα είχαμε segmentation fault

Εικονική μνήμη

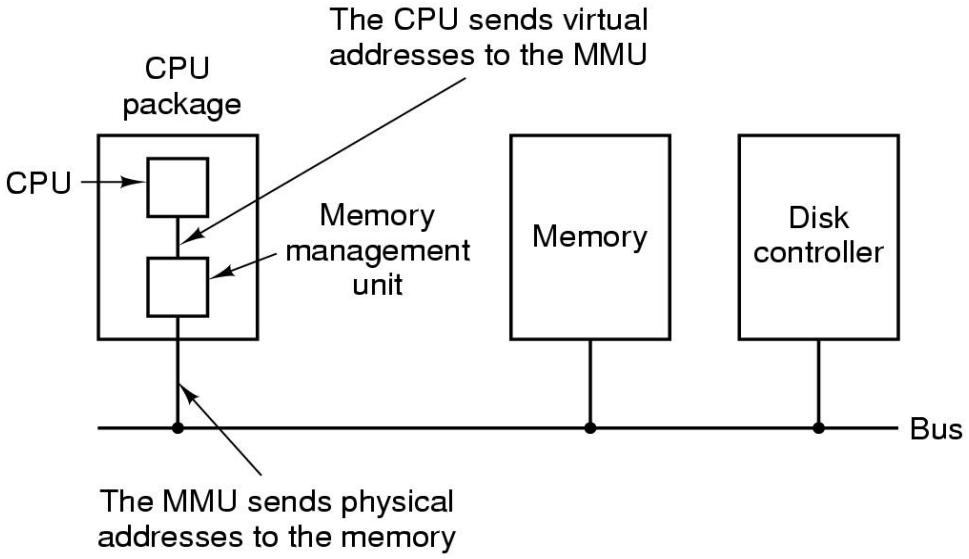
- Πώς μπορεί μια διεργασία/πρόγραμμα να αναφέρεται σε θέσεις μνήμης/διευθύνσεις από πριν, χωρίς να ενδιαφέρεται ποιες θα είναι οι πραγματικές διευθύνσεις που θα της δοθούν στην εκτέλεση
  - Οι οποίες πραγματικές μπορεί και να αλλάζουν κατά τη διάρκεια της εκτέλεσης λόγω ύπαρξης πολλών ταυτόχρονων προγραμμάτων που εκτελούνται στο σύστημα

- Τι γίνεται αν μια διεργασία θέλει τελικά περισσότερο χώρο από ότι είναι διαθέσιμος σαν φυσική μνήμη?
  - Κανονικά δεν θα μπορούσε να εκτελεστεί αν δεν μπορεί κάπως να σπάσει σε κομμάτια και να φορτώνεται εναλλάξ κάθε χρησιμοποιούμενο κομμάτι

- Θέλουμε 4 δυνατότητες
  - Διευθυνσιοδότηση μεγαλύτερου μεγέθους από τη διαθέσιμη μνήμη (virtual/logical memory)
    - Και χρήση δίσκου σαν αποθηκευτικό μέσο για τα κομμάτια που δεν χρησιμοποιούνται αυτή τη στιγμή αλλά έχουν χρησιμοποιηθεί στο παρελθόν
  - Διευθυνσιοδότηση ανεξάρτητα από την φυσική τοποθέτηση
  - Σπάσιμο χρησιμοποιούμενης μνήμης σε κομμάτια (paging) και
  - εναλλαγή κομματιών πλέον μεταξύ δίσκου μνήμης (swapping)

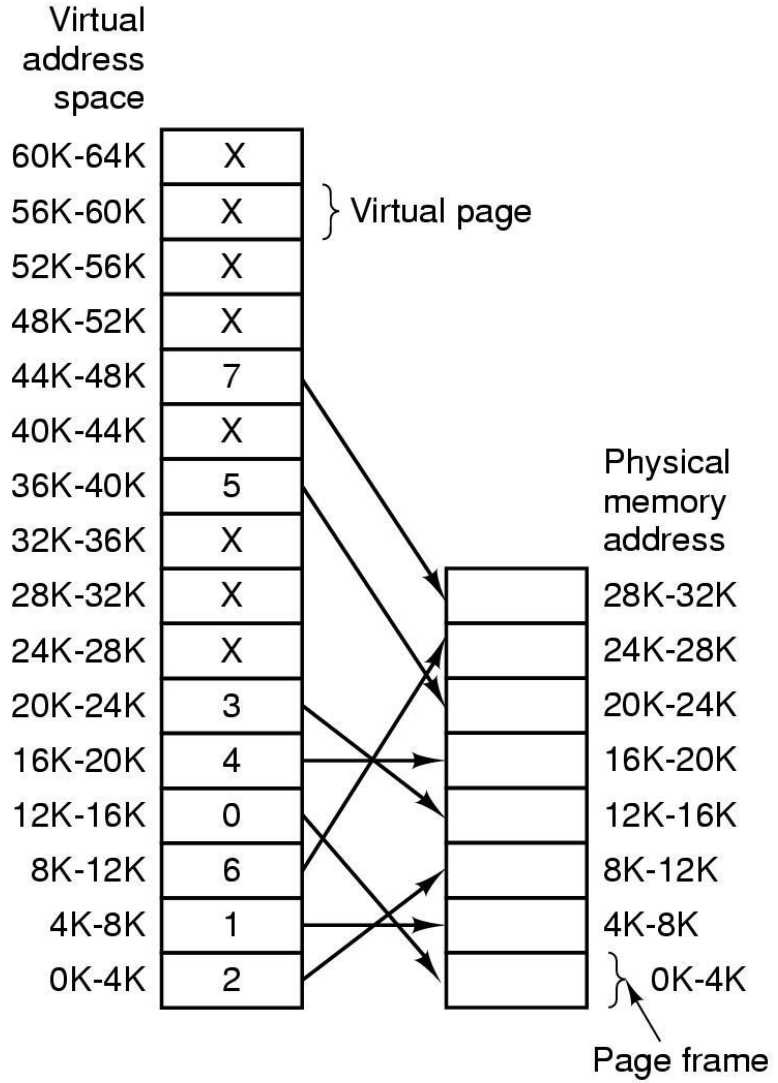
Εικονική μνήμη

- Διάκριση του χώρου διευθύνσεων σε
  - *Ιδεατό (Virtual) ή Λογικό χώρο διευθύνσεων* – αυτόν που βλέπουν οι προγραμματιστές. Γραμμικός ανά διεργασία, μεγάλου μεγέθους
  - *Φυσικό χώρο διευθύνσεων* – οι πραγματικές θέσεις στη μνήμη
- Εισαγωγή της *μονάδας διαχείρισης μνήμης (ΜΔΜ)*
  - Η ΜΔΜ μεταφράζει κάθε εικονική διεύθυνση σε φυσική, χρησιμοποιώντας πρόσθετες δομές
  - Συνεργασία ΛΣ και hardware



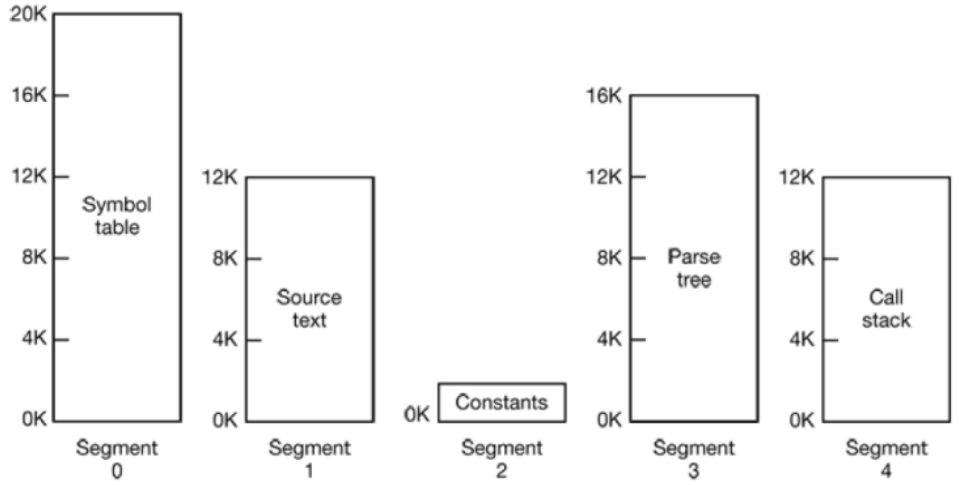
Εικονική μνήμη (συνολικά)

- Ο χώρος διευθύνσεων που είναι διαθέσιμος για τα προγράμματα χρήστη είναι μεγαλύτερος από τη φυσική μνήμη
  - Κάθε διεργασία έχει το δικό της χώρο
  - Οι απαιτήσεις της διεργασίας μπορεί να ξεπερνούν την διαθέσιμη μνήμη
  - Το λειτουργικό σύστημα φροντίζει να βρίσκονται στη μνήμη μόνο τα τμήματα της διεργασίας που χρειάζεται άμεσα – τα υπόλοιπα βρίσκονται στον δίσκο
    - Όταν χρειάζεται τμήμα που δεν βρίσκεται στη μνήμη, το Λ.Σ. αναλαμβάνει να το προσκομίσει, απομακρύνοντας ενδεχομένως κάποιο άλλο τμήμα
  - Είναι δυνατόν να φυλάσσονται στη μνήμη τμήματα από διαφορετικές διεργασίες
  - Η ύπαρξη μόνο του αναγκαίου κομματιού βοηθάει και την ύπαρξη περισσότερων διεργασιών σε μια χρονικη στιγμή στη μνήμη
  - Το ΛΣ επεμβαίνει και μεταφράζει την αναφορά του προγράμματος σε εικονική μνήμη στην πραγματική θέση μνήμης

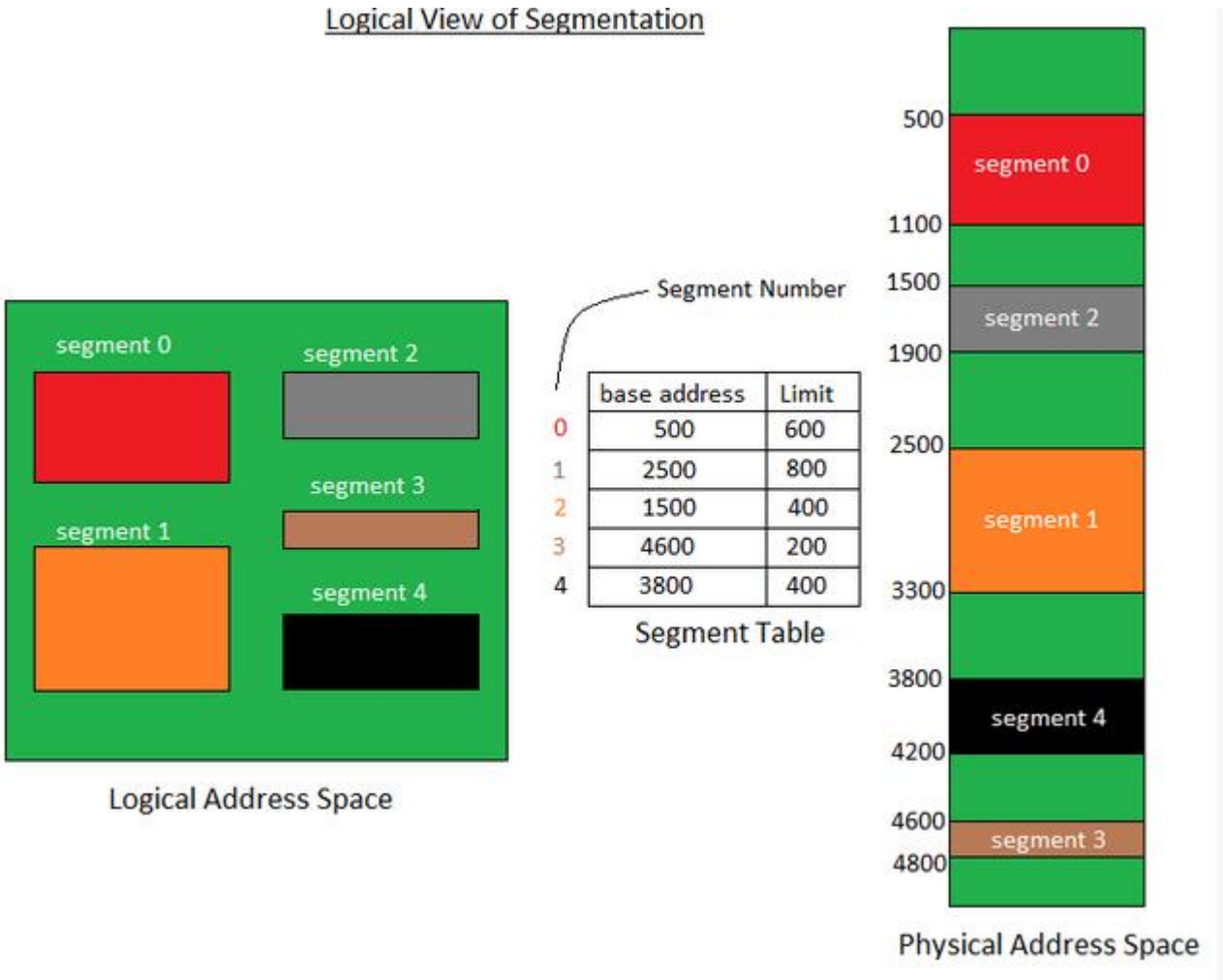


Κατάτμηση (Segmentation)

- Ξεχωριστοί χώροι για κάθε υποσύνολο πληροφορίας στον ίδιο εικονικό χώρο διευθύνσεων ενός προγράμματος
- Οι θέσεις καθορίζονται με βάση το segment number και την σχετική διεύθυνση από την αρχή του segment
- Ο χώρος των μεταβλητών αρχικοποιείται από τον compiler σε επίπεδο εικονικής διευθυνσιοδότητης
  - Για τις μεταβλητές που μπορεί να καθορίσει το μέγεθός τους
  - Όχι για τα δυναμικά κομμάτια μέσω malloc
  - Και όχι απαραίτητα ως προς το περιεχόμενο (initialized/uninitialized segments)



Λογική άποψη για την Κατάτμηση



Πλεονεκτήματα Κατάτμησης

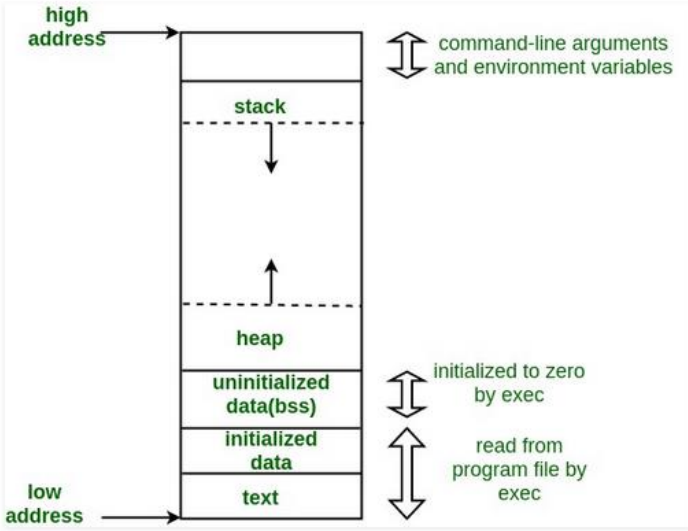
- Ανεξαρτησία προστασίας
  - Ο προγραμματιστής μπορεί να ορίσει δικαιώματα ανά segment
- Χωρισμός ανάλογα με τον τύπο της πληροφορίας
  - Ο προγραμματιστής γνωρίζει την εννοιολογική διαφορά τους
- Δυναμικά τμήματα μνήμης, βοηθά στην καλύτερη διαχείριση του fragmentation
  - Κάθε τμήμα μπορεί να αυξομειώνεται ανεξάρτητα από τα άλλα

Εφαρμογή κατάτμησης

- Η κατάτμηση συνήθως γίνεται στους compilers με βάση κάποιο τυπικό χωρισμό
- Αλλά μπορεί να επέμβει και ο προγραμματιστής με directives (πχ C) και να δώσει αντιστοιχες οδηγίες λογικού διαχωρισμού στο πρόγραμμα

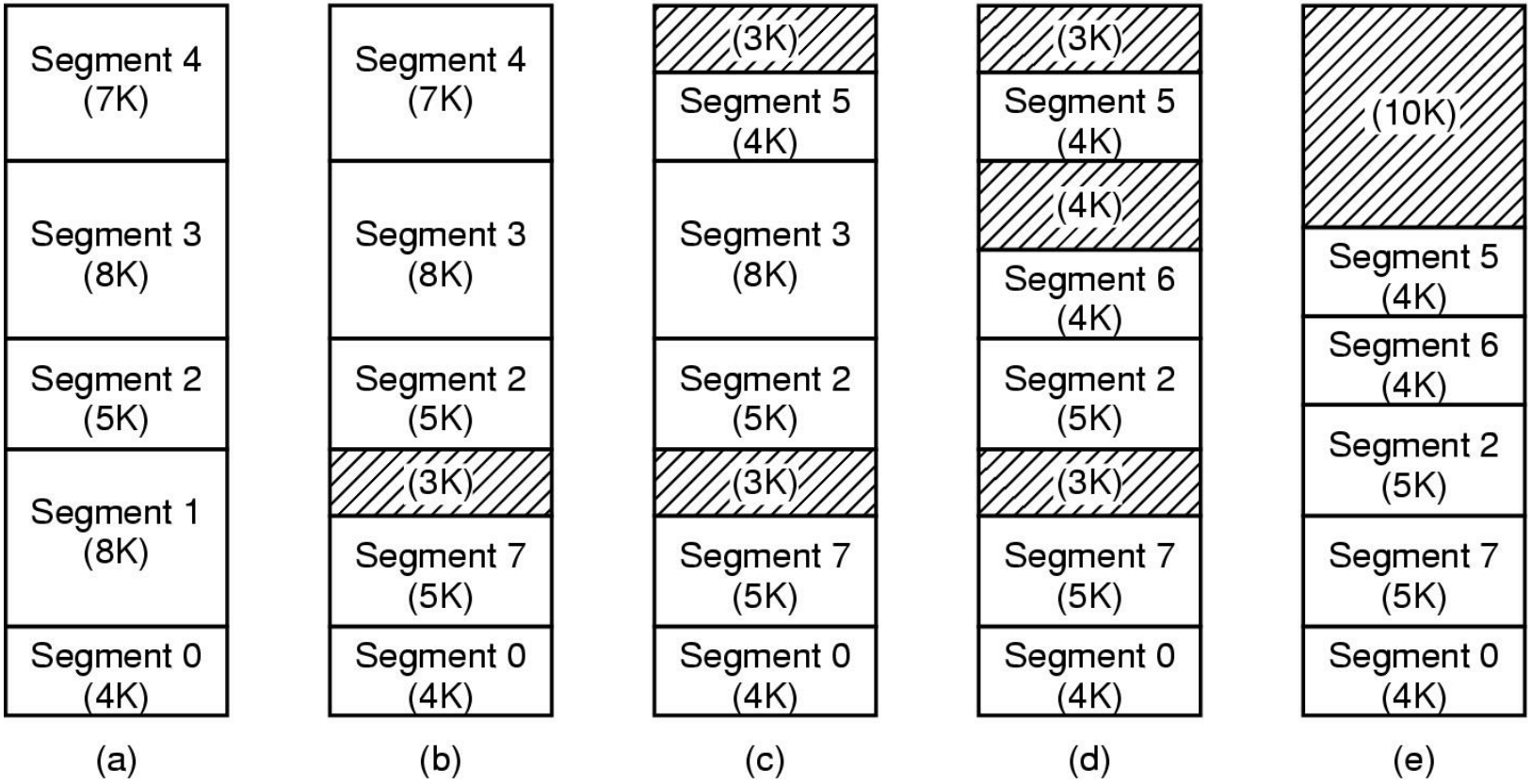
```
#pragma section("mydata$a", read, write)
__declspec(allocate("mydata$a")) int i = 0;

#pragma section("mydata$b", read, write)
__declspec(allocate("mydata$b")) int j = 0;
```



- Προσοχή ότι η διαδικασία είναι compiler & system specific
- Περισσότερες πληροφορίες
  - https://devblogs.microsoft.com/oldnewthing/20181107-00/?p=100155
  - https://www.geeksforgeeks.org/pragma-directive-in-c-c/

Κατάτμηση - Υλοποίηση



(a)-(d) Development of checkerboarding
(e) Removal of the checkerboarding by compaction

Segmentation fault

- Αυτό που παρατηρείτε πολύ συχνά σαν σφάλμα όταν δουλεύετε με τους pointers στη C. Πότε συμβαίνει?
- Παραβίαση δικαιωμάτων στο κομμάτι
  - Όπως θα δούμε σήμερα όταν προσπαθήσουμε να γράψουμε σε ένα κομμάτι string literal που αποθηκεύεται σε read-only περιοχή
  - Ή αν προσπαθήσουμε να κάνουμε access σε περιοχή που δεν έχουμε δικαίωμα
- Αν δεν έχει οριστεί το αντίστοιχο κομμάτι (δηλαδή δεν υπάρχει μέσα στον αντίστοιχο πίνακα του ΛΣ σαν κομμάτι μνήμης αυτής της διεργασίας)
  - Προφανώς αναφέρεται στα μη αυτόματα δημιουργούμενα κομμάτια αλλά σε αυτά που μπορούν να δημιουργηθούν από το προγραμματιστή άμεσα
- Αν παραβιάζονται τα όρια του κομματιού
  - Αυτό το παρατηρείτε συνήθως κυρίως λόγω σφάλματος χειρισμού του περιεχομένου των pointers