**Άσκηση 3:**

**Μηχανισμοί Εικονικής Μνήμης**

Συντάκτες: Νικόλαος Λάππας 03121098

Γεώργιος Πνευματικός 03121058

1.1 Κλήσεις Συστήματος και Βασικοί Μηχανισμοί του ΛΣ για την

διαχείριση της Εικονικής Μνήμης (Virtual Memory – VM)

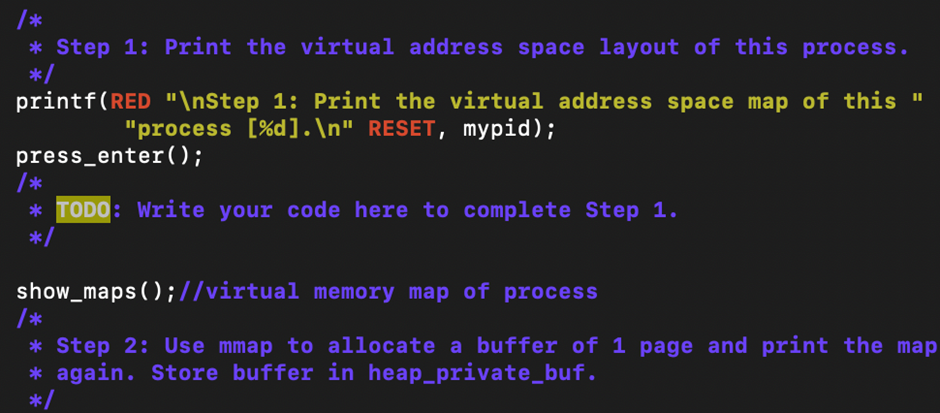
*αρχείο mmap.c στον κατάλογο third\_ex*

Αρχικά φέρνουμε το περιεχόμενο του προγράμματος mmap.c από τον φάκελο /home/oslab/code/mmap στον δικό μας χώρο. Παράλληλα έχουμε φέρει και το Makefile καθώς και το help.h που αποτελεί ένα header file εντός του οποίου δηλώνονται συναρτήσεις αλλά και το help.c στο οποίο οι συναρτήσεις αυτές ορίζονται. Φυσικά στο αρχείο mmap.c γίνεται include το αρχείο help.h.

Το πρόγραμμα mmap.c είναι ημιτελές και περιέχει βήματα τα οποία εμείς καλούμαστε να συμπληρώσουμε :

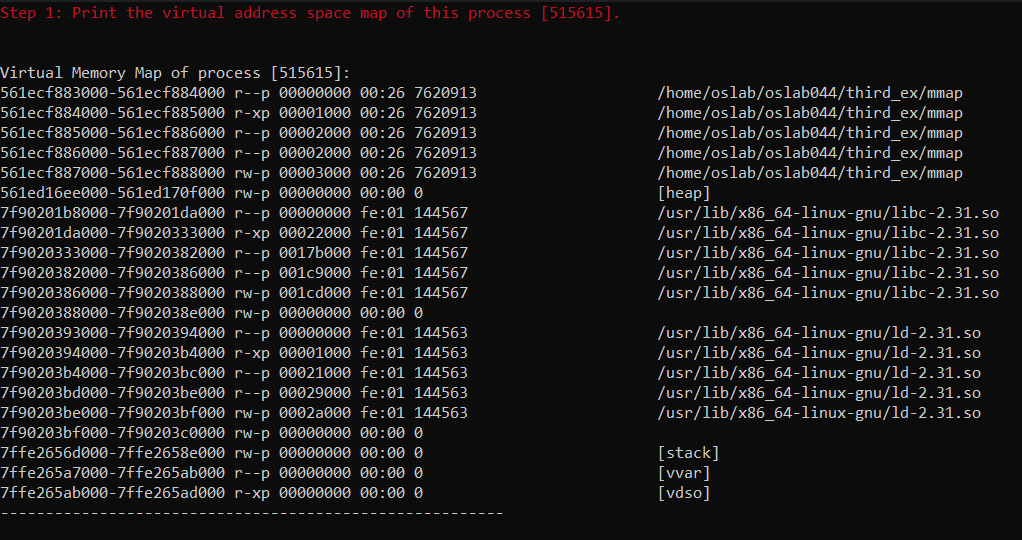
1. Στο πρώτο βήμα καλούμαστε να τυπώσουμε τον χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας. Αυτό το επιτυγχάνουμε κάνοντας χρήση της συνάρτησης show\_maps() ορίζεται στο αρχείο help.c.

Παραθέτουμε την αλλαγή στον κώδικα

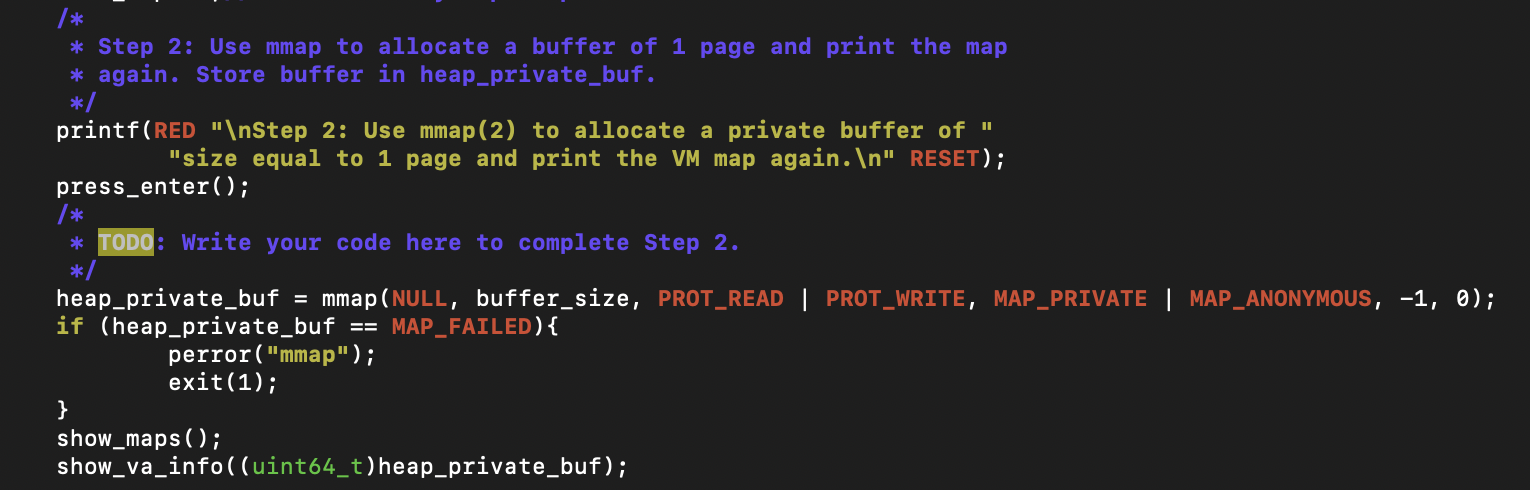


Μέσω του Makefile μεταγλωττίζουμε και έπειτα εκτελούμε το πρώτο βήμα

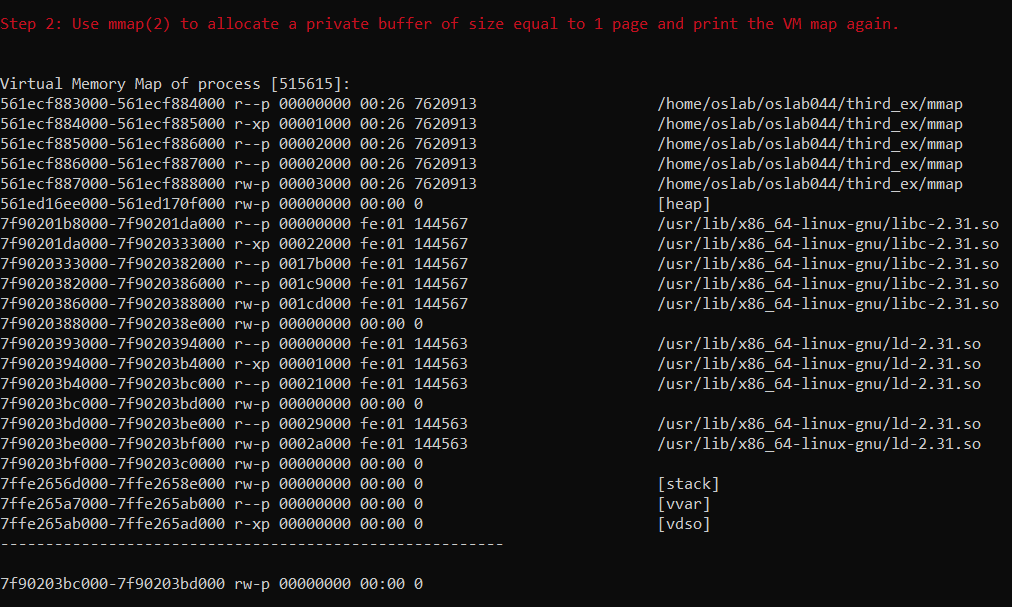
Παραθέτουμε το αποτέλεσμα:



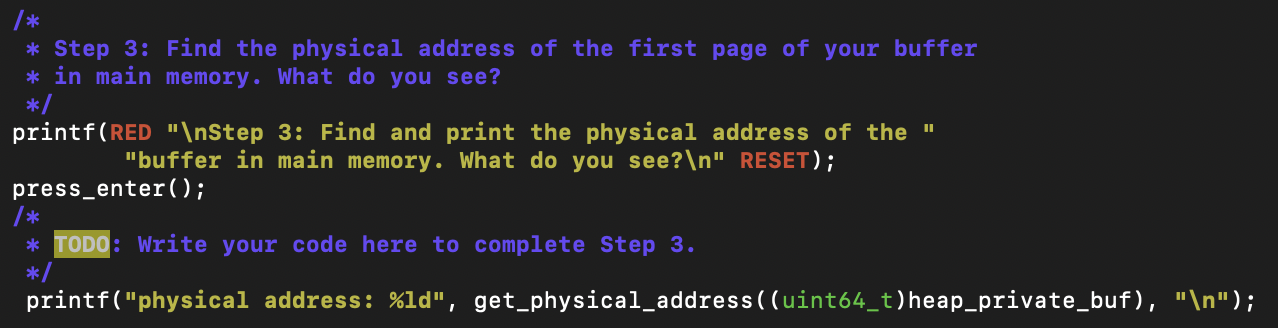
2. Δεσμεύουμε buffer (προσωρινή μνήμη) μεγέθους μιας σελίδας με την χρήση της εντολής mmap () και τυπώνουμε ξανά τον χάρτη. Έπειτα με χρήση της εντολής show\_va\_info() εντοπίζουμε στον χάρτη μνήμης τον χώρο των εικονικών διευθύνσεων που δεσμεύσαμε.

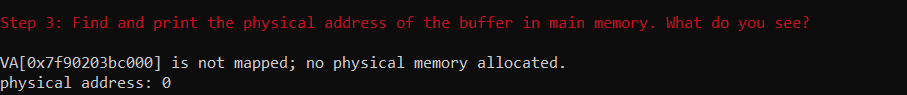


Παραθέτουμε τα αποτελέσματα του βήματος



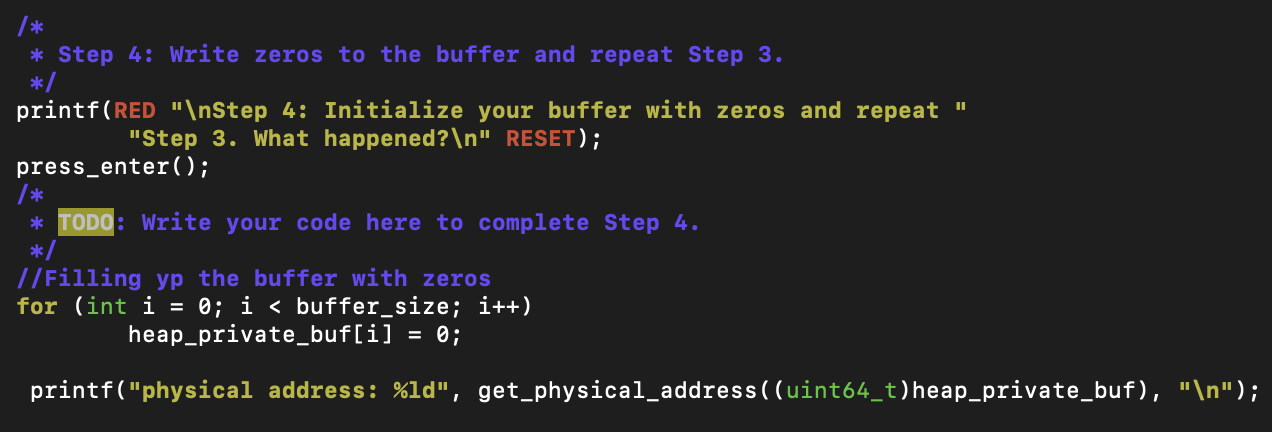
3. Τυπώνουμε τη φυσική διεύθυνση της μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer. Αυτό το επιτυγχάνουμε με χρήση της get\_physical\_address()



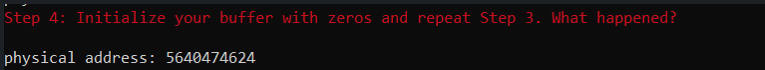


Αυτό που παρατηρούμε είναι ότι ενώ φαίνεται να έχει δεσμευτεί η μνήμη, όπως φάνηκε και παραπάνω στο χάρτη μνήμης, δεν έχει απεικονιστεί η εικονική μνήμη σε φυσική μνήμη. Αυτό συμβαίνει γιατί κατά την δυναμ εντολή mmap έχουμε paging on demand. Δηλαδή την στιγμή που πάμε να προσπελάσουμε εικονική μνήμη που δεν έχει ακόμη απεικονιστεί τότε παίρνουμε page fault και εφόσον η εικονική μνήμη βρίσκεται στον χάρτη μνήμης το ΛΣ επιλέγει ένα χωρίο στο οποίο γίνεται η απεικόνιση της φυσικής μνήμης.

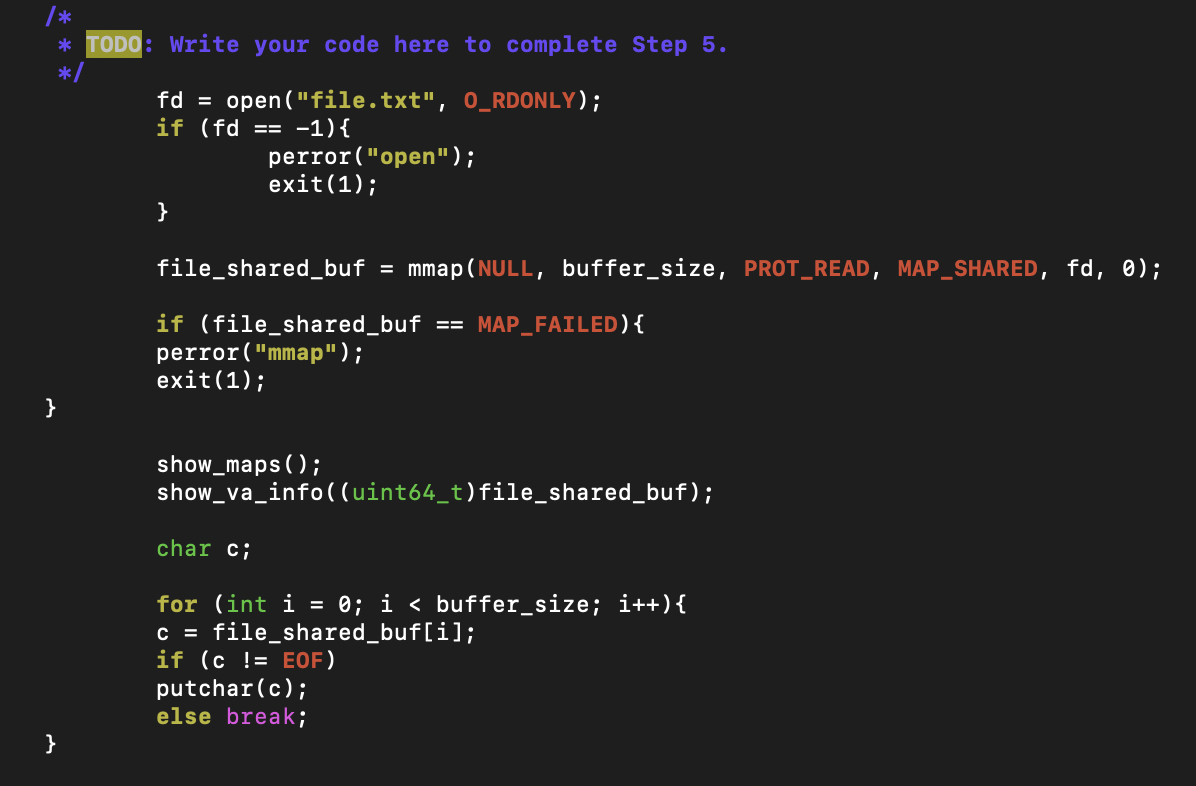
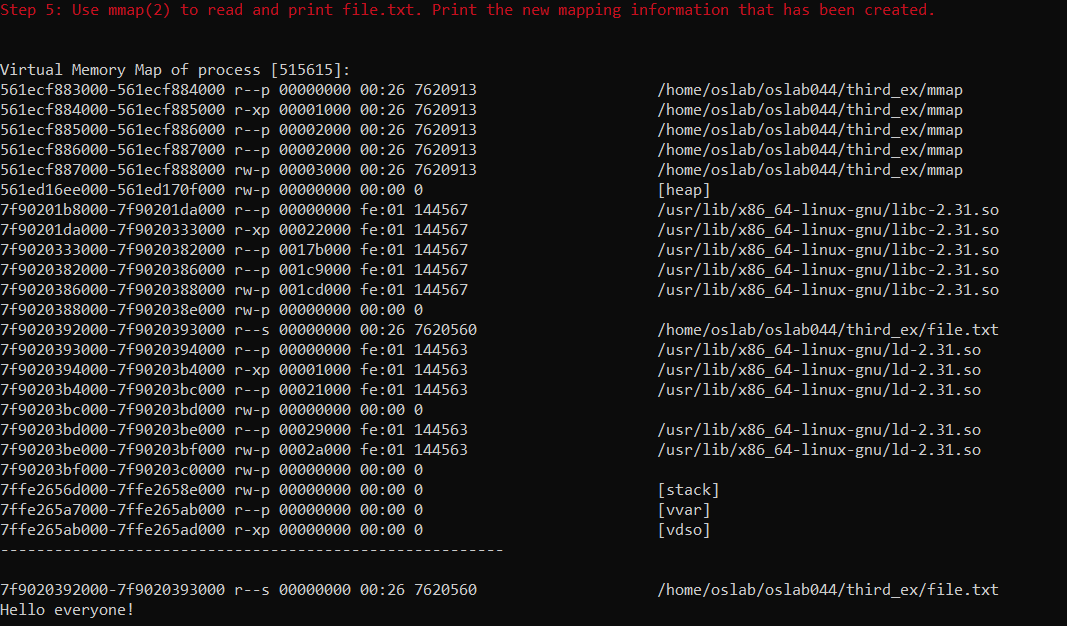
4. Γεμίζουμε με μηδενικά τον buffer και επαναλαμβάνουμε το βήμα 3



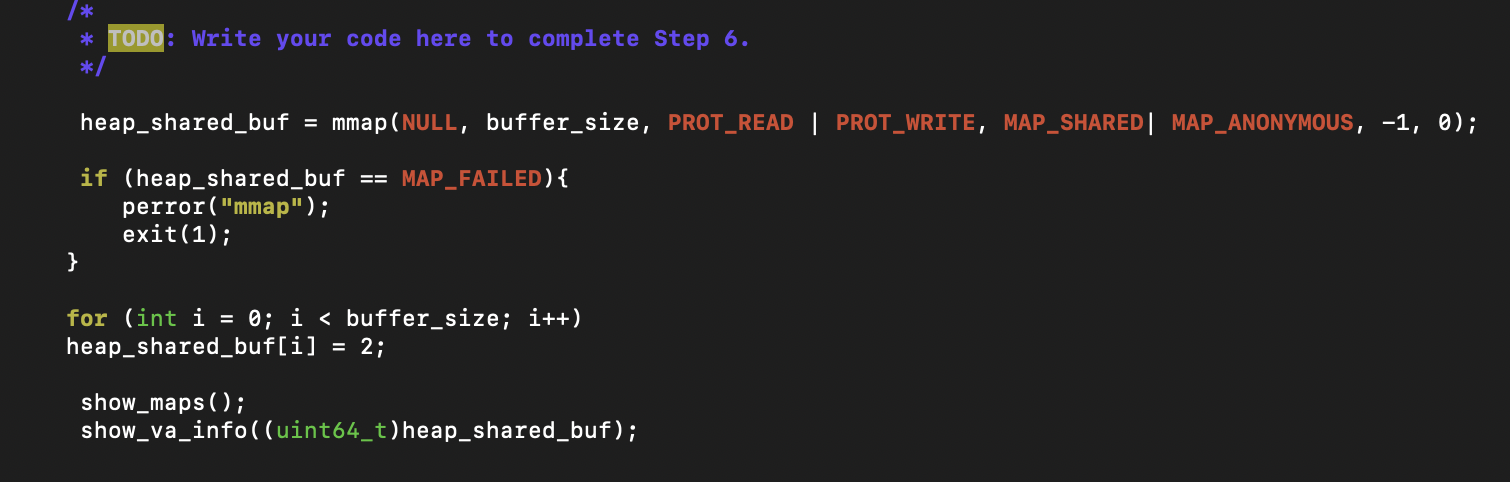
Στην περίπτωση αυτή παρατηρούμε μια ορθή αντιστοίχιση στην φυσική μνήμη καθώς πλέον έχουμε προσπελάσει τον buffer.

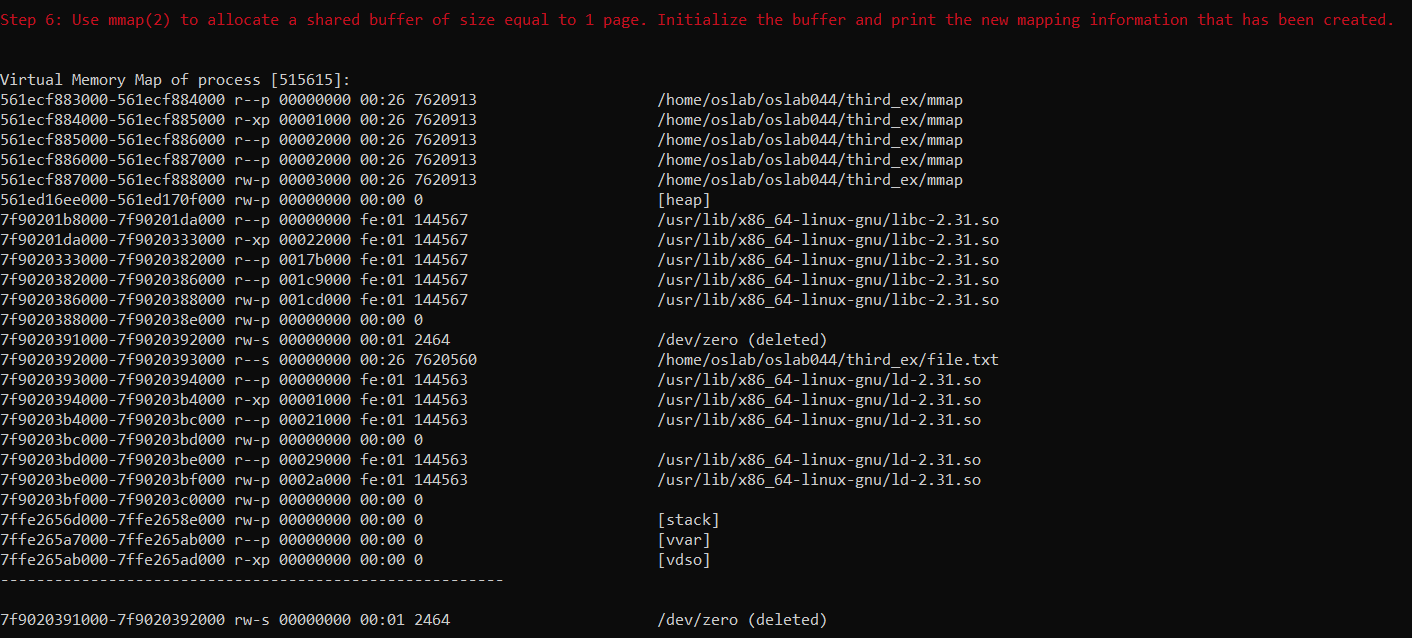


5.Χρησιμοποιούμε την mmap() για να απεικονίσουμε το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας και τυπώνουμε το περιεχόμενό του. Παράλληλα μέσω της εντολής show\_va\_info() εντοπίζουμε και την νέα απεικόνιση στον χάρτη μνήμης.



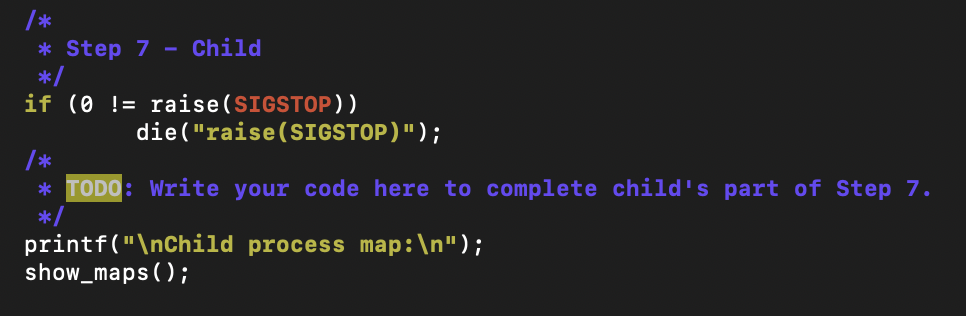
6. Χρησιμοποιούμε την mmap προκειμένου να δεσμεύσουμε έναν νέο buffer διαμοιραζόμενο, (shared) αυτήν την φορά, μεταξύ διεργασιών. Παράλληλα εντοπίζουμε πάλι την νέα απεικόνιση στον χάρτη της μνήμης και προσπελάσουμε, κιόλας, τον διαμοιραζόμενο buffer για να τον χρησιμοποιήσουμε έπειτα σε επόμενα βήματα καθώς όπως είδαμε στο ερώτημα 3 έχουμε paging on demand.

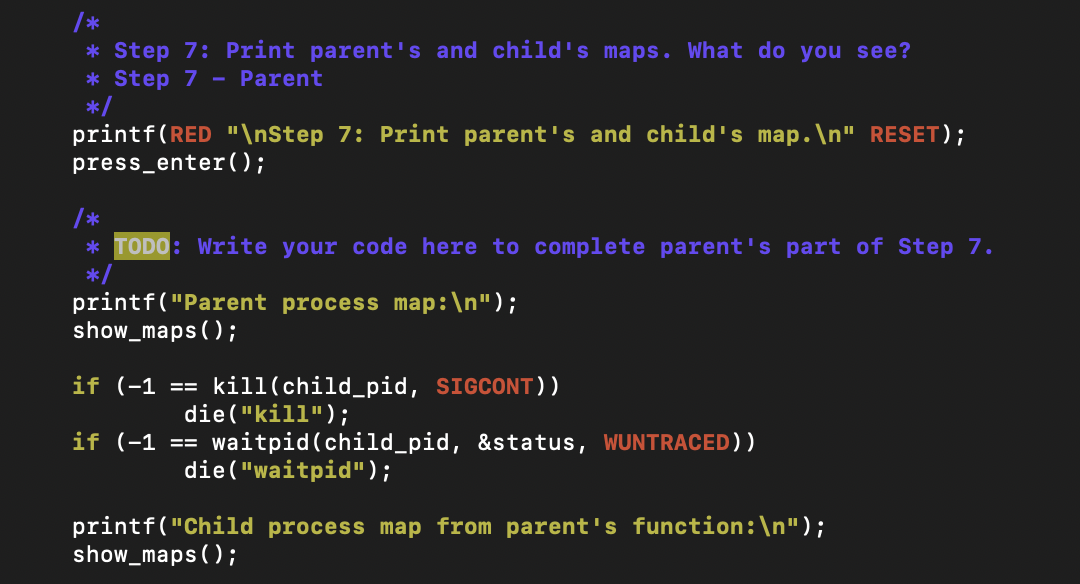
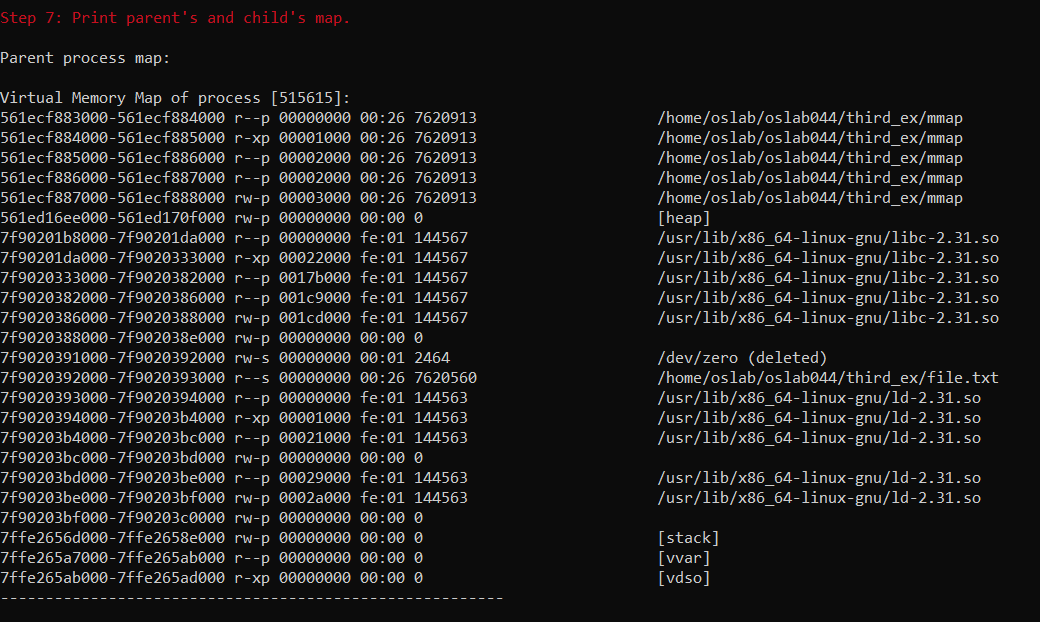


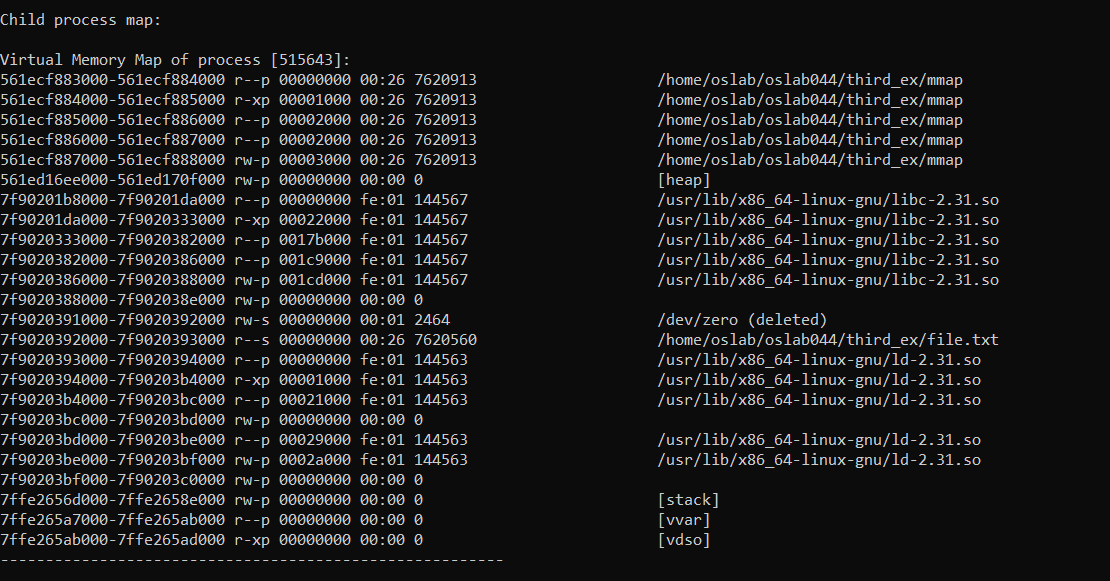


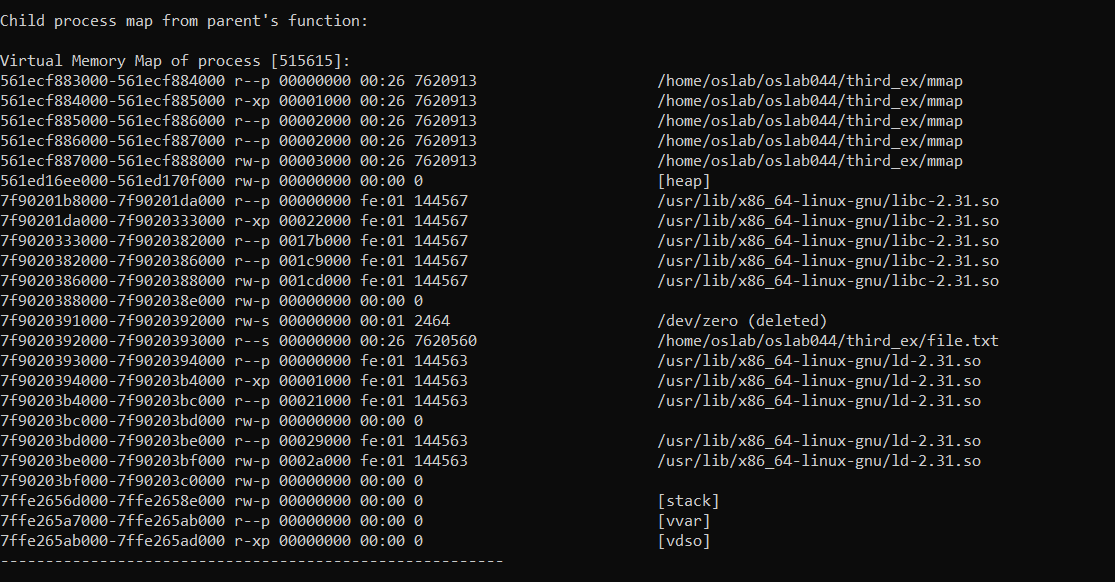
Πλεόν καλείται η fork και έχουμε δύο διεργασίες. Τη διεργασία γονέα και την διεργασία παιδί.

7.Τυπώνουμε αρχικά τον χάρτη εικονικής μνήμης των δύο διεργασίων. Αξίζει να σημειωθεί πώς με τον κώδικα που παραλάβαμε σε κάθε βήμα πάντα τυπώνει πρώτα η διεργασία πατέρα και μετά τυπώνει η διεργασία παιδί. Αυτό φυσικά επιτυγχάνεται με το σχήμα συγχρονισμού που έχει υλοποιηθεί κατά το οποίο χρησιμοποιούνται οι εντολές raise(SIGSTOP) από το παιδί και οι εντολές kill(child\_pid ,SIGCONT) και waitpid =(child\_pid, WUNTRACED) από τον πατέρα





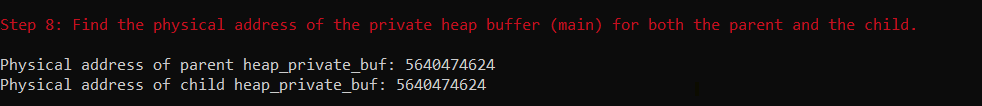




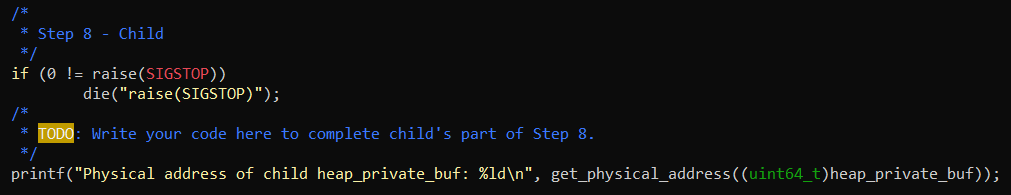
Αυτό που παρατηρούμε είναι πώς η νέα διεργασία, αρχικά που δημιουργείται μέσω της fork() αποτελεί αντίγραφο της γονεϊκής διεργασίας, ο χάρτης δηλαδή της διεργασίας παιδιού είναι ολοιδιος με αυτόν της διεργασίας γονέα, είτε τυπωθεί από την γονεική διεργασία είτε από την ίδια του παιδιού. Αυτό είναι και το αναμενόμενο αφού με την fork() δημιουργείται μια διεργασία αντίγραφο της γονεϊκής, που έχει τα ίδια δικαιώματα ανάγνωσης και τις ίδιες απεικονίσεις στην εικονική μνήμη (ακόμα). Βλέπουμε ότι ακόμα και το αρχείο ‘file.txt’ που είχε επεξεργαστεί η διεργασία πριν την κλήση της συνάρτησης fork(), είναι προσβάσιμο για ανάγνωση και από την διεργασία παιδί. Αξίζει να σημειωθεί πώς μετά από την κλήση της fork στον πίνακα σελίδων και των δύο διεργασιών αφαιρούνται τα δικαιώματα εγγραφής στους ιδιωτικούς buffers Copy on Write(COW).

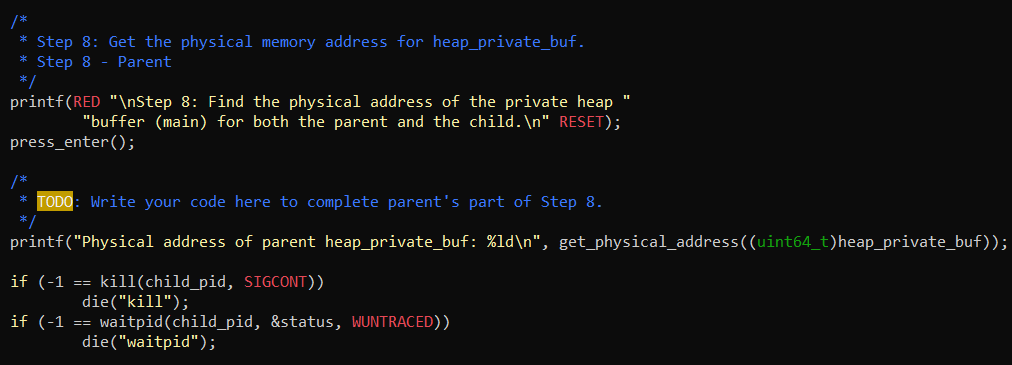
8. Εδώ καλούμαστε να βρούμε και να τυπώσουμε την φυσική διεύθυνση στην κύρια μνήμη του private buffer αντίστοιχα με το βήμα 3. Αυτό που παρατηρούμε είναι ότι πλέον έχει δεσμευτεί ο private buffer και πλέον παίρνουμε την θέση στην μνήμη όπου έχει απεικονιστεί, σε αντίθεση με το βήμα 3 όπου η διεύθυνση δεν είχε δεσμευτεί.

Κάτι ακόμα που παρατηρούμε είναι ότι οι δυο διεργασίες έχουν την ίδια φυσική διεύθυνση στην μνήμη. Αυτό συμβαίνει καθώς η διεργασία παιδί αποτελεί ακριβές αντίγραφο της διεργασίας γονιού, μέχρι να αρχίσει να εκτελεί δικές του συναρτήσεις και εντολές. Τότε ορισμένες μεταβλητές, ακόμα και ίδιες να είναι με αυτές του γονιού, θα αποθηκευτούν σε διαφορετική θέση στην μνήμη.

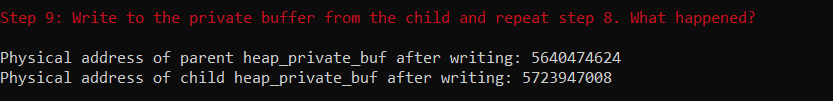


Ακολουθούν οι γραμμές κώδικα:

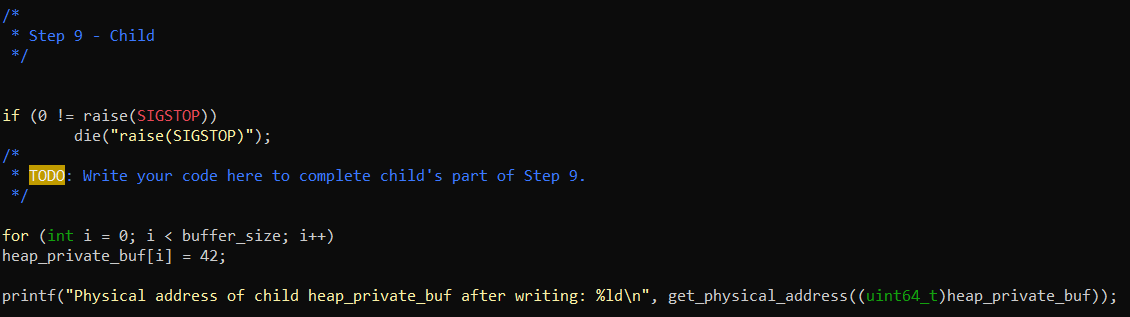


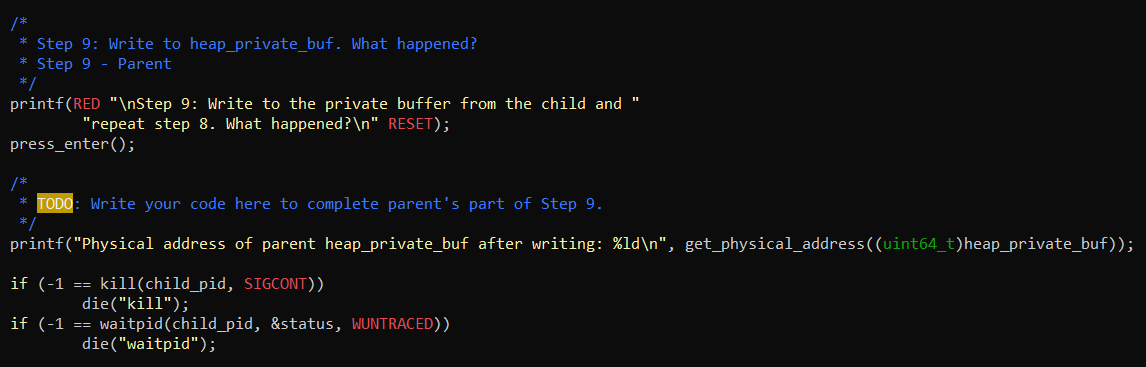


9. Σε αυτό το βήμα εγγράφουμε στον private buffer δεδομένα από την διεργασία παιδί, συγκεκριμένα εισάγουμε σε εύρος μιας σελίδας τον αριθμό 42, και εκτελούμε ξανά το βήμα 8.

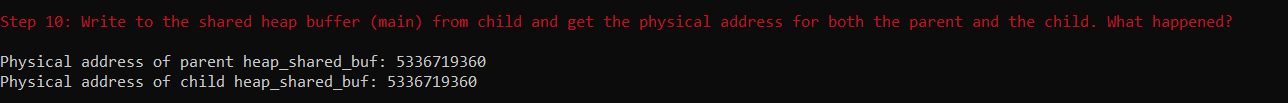


Αυτό που παρατηρούμε είναι ότι πλέον οι θέσεις μνήμης που αποθηκεύονται τα δεδομένα του κάθε buffer διαφέρει, όπως είναι λογικό αφού ο buffer είναι private για κάθε διεργασία. Από την στιγμή που εισάγουμε δεδομένα στον buffer από την διεργασία παιδί, παύουν πλέον να δείχνουν στην ίδια διεύθυνση μνήμης οι εκάστοτε δείκτες των buffer και αποκτούν την δική τους ιδιωτική θέση στην μνήμη.

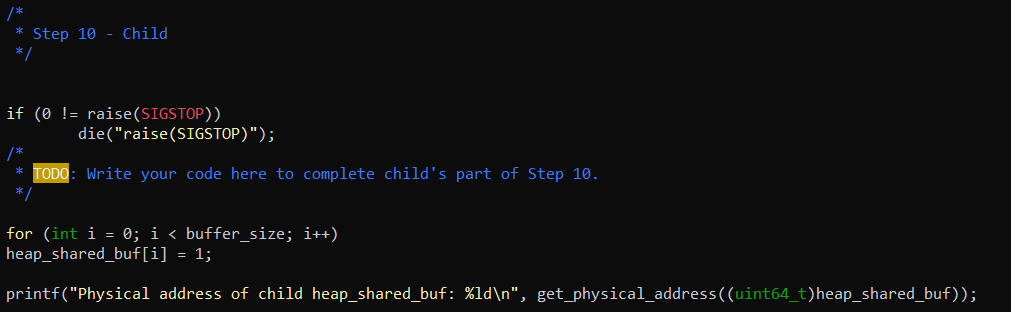


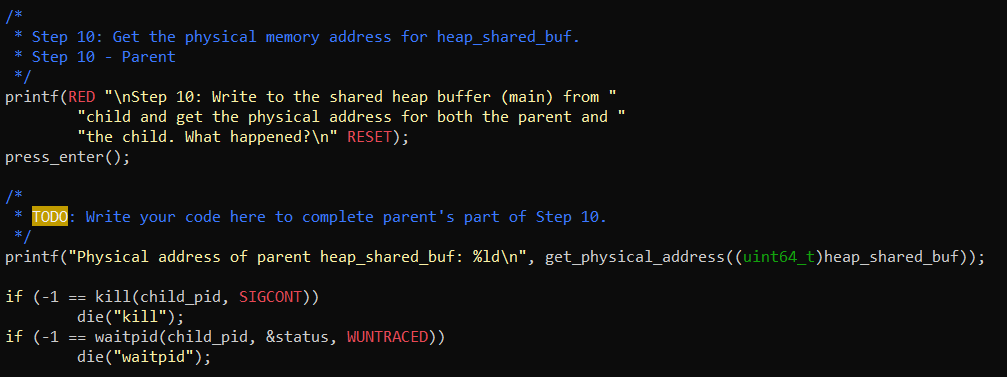


10. Σε αυτό το βήμα εγγράφουμε στον shared buffer δεδομένα από την διεργασία παιδί, συγκεκριμένα εισάγουμε σε εύρος μιας σελίδας τον αριθμό 1, και εκτελούμε ξανά το βήμα 6.



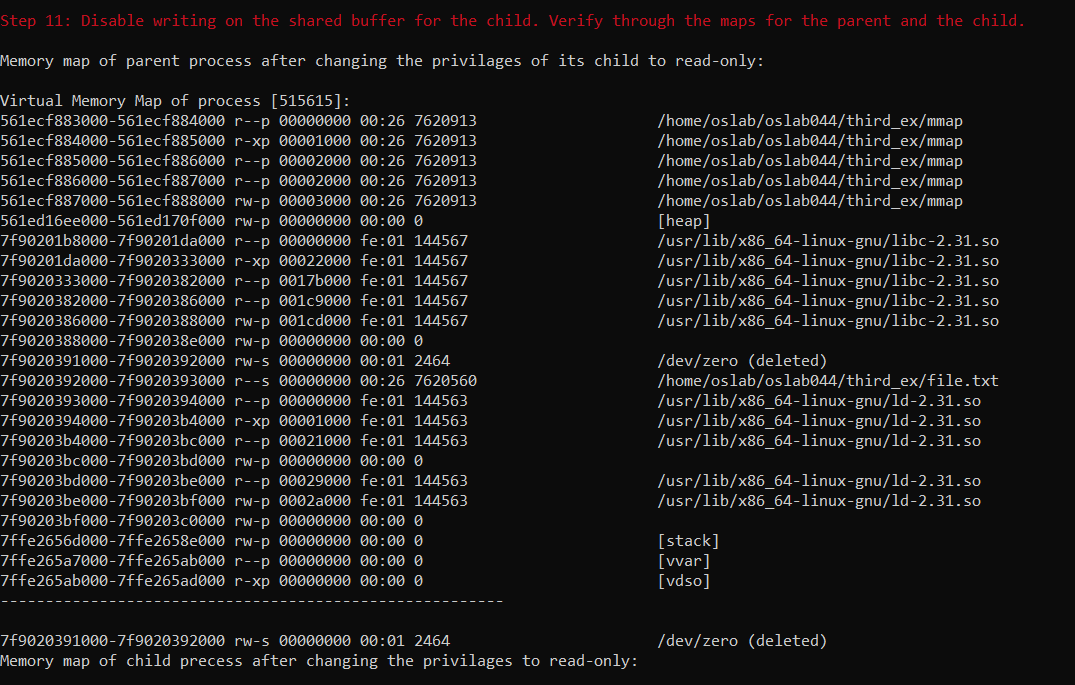
Αυτό που παρατηρούμε είναι ότι τώρα εγγράψαμε δεδομένα σε έναν κοινόχρηστο buffer και όπως είναι λογικό αυτός θα έχει και για τις δυο διεργασίες την ίδια φυσική διεύθυνση. Θα ήταν σπατάλη χώρου αν δημιουργείται αντίγραφο για τον κοινόχρηστο buffer, αφού όπως λέει και το όνομά του θέλουμε να είναι προσβασιμος και εγγράψιμος με κοινά δεδομένα και από τις δυο διεργασίες. Βλέπουμε, δηλαδή, ότι εδώ διαφοροποιείται το αποτέλεσμα σε σχέση με το προηγούμενο βήμα. Εκεί είχαμε private buffer ο οποίος είναι προσωπικός για κάθε διεργασία και τα δεδομένα του εκάστοτε private buffer αποθηκεύονται σε διαφορετική θέση στην μνήμη, εδώ έχουμε shared buffer ο οποίος είναι κοινόχρηστος και για τις δυο διεργασίες και τα δεδομένα του αποθηκεύονται στο ίδιο εύρος μνήμης, είτε γίνει εγγραφή/τροποποίηση από την διεργασία παιδί είτε από την διεργασία γονέα.

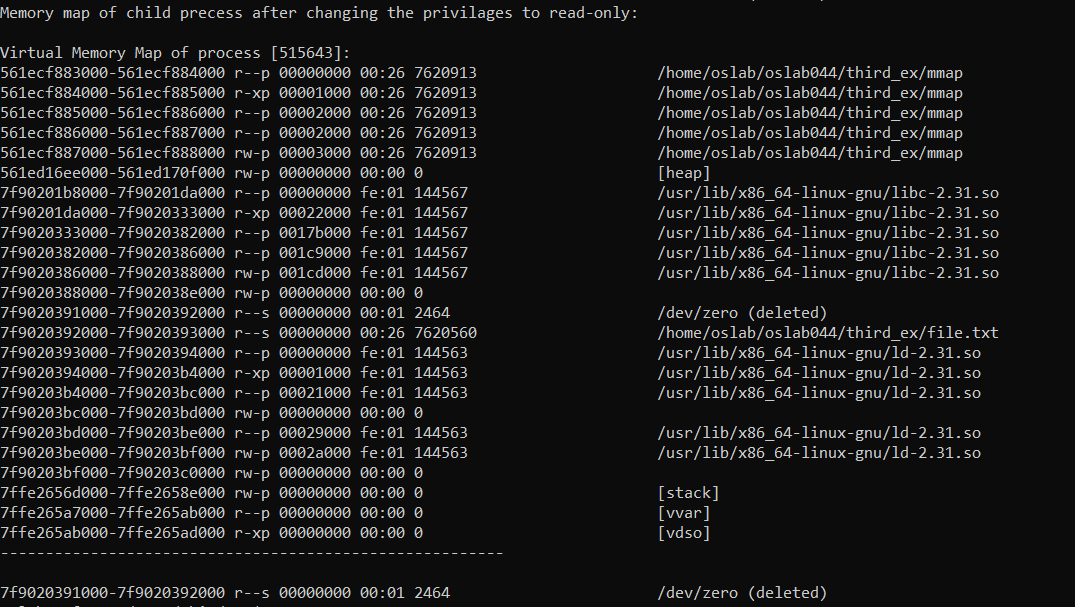


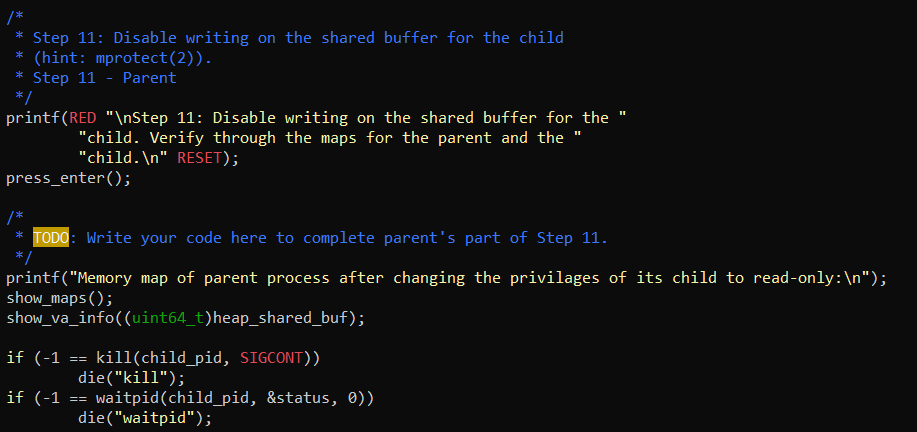


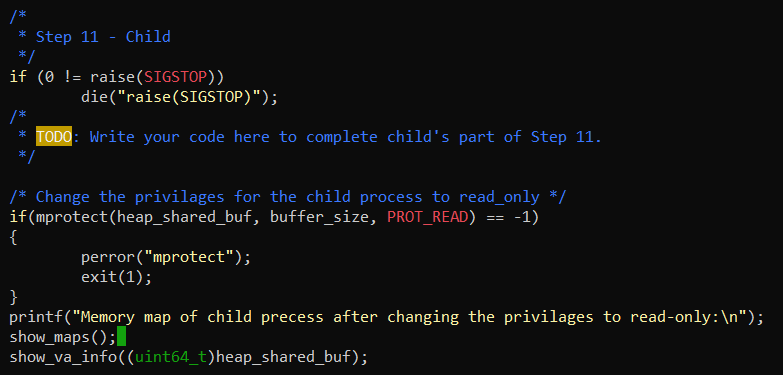
11. Στο ερώτημα αυτό θέλουμε να τροποποιήσουμε τα δικαιώματα χρήσης της διεργασίας παιδιού. Συγκεκριμένα, θέλουμε να της απαγορεύσουμε το δικαίωμα εγγραφής στον shared buffer. Αυτό το επιτυγχάνουμε με την συνάρτηση mprotect() της οποίας τα ορίσματα αναφέρονται σε: int mprotect(void \*addr, size\_t len, int prot);

Όπου στο πεδίο addr βάζουμε τον δείκτη που δείχνει στην διεύθυνση του buffer του οποίου θέλουμε να τροποποιήσουμε τα δικαιώματα, μετά ακολουθεί το μέγεθος του buffer και τέλος, ορίζουμε τι δικαιώματα επιτρέπουμε στην διεργασία που κάνει κλήση αυτής της συνάρτησης να έχει, μόνο ανάγνωση στην προκειμένη περίπτωση. Ταυτόχρονα ελέγχουμε για πιθανό σφάλμα κατά την κλήση και αν προκύψει πετάμε perror και κάνουμε exit(1).





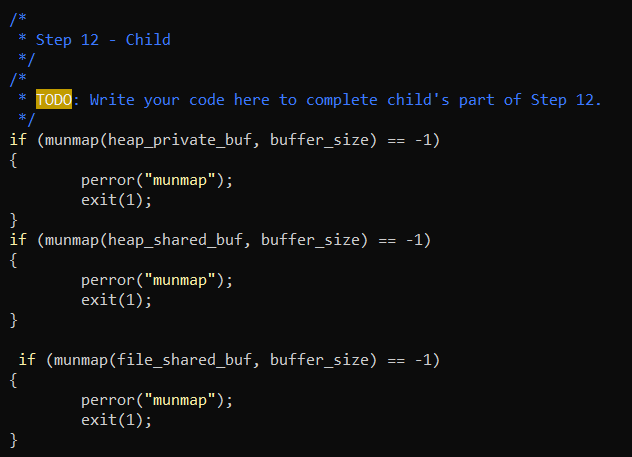


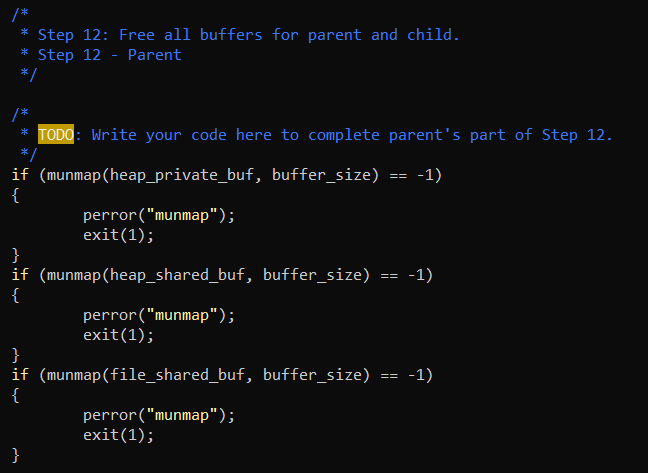


12. Εδώ θέλουμε να αποδεσμεύσουμε όλους τους buffers στις δυο διεργασίες. Αυτό το υλοποιούμε με την χρήση της συνάρτησης munmap() της οποίας τα ορίσματα είναι τα εξής: int munmap(void \*addr, size\_t length);

Όπου σαν \*addr βάζουμε τον δείκτη που δείχνει στην διεύθυνση μνήμης που θέλουμε να αποδεσμεύσουμε (εμείς τον δείκτη του εκάστοτε buffer) και στο length βάζουμε το μέγεθος του buffer.

Αποδεσμεύουμε και στις δυο διεργασίες τον private buffer αφού είναι ιδιωτικής χρήσης για κάθε διεργασία και άρα θα πρέπει η κάθε διεργασία ξεχωριστά να αποδεσμεύσει την μνήμη όταν πλέον δεν τον χρειάζεται. Αποδεσμεύουμε, επίσης, τον shared buffer και το file shared buffer και στις δυο διεργασίες και αυτό γιατί, παρόλο που αυτά είναι κοινόχρηστα και των δυο διεργασιών, δεν αρκεί η αποδέσμευση σε μια από τις δυο, γιατί πρέπει να αποδεσμευτεί και η εικονική μνήμη αλλά και η αντιστοίχιση της εικονικής μνήμης στην φυσική στην οποία βρίσκεται ο shared\_buffer για όλες τις διαφορετικές διεργασίες. Ταυτόχρονα η μνήμη που δεσμεύεται για έναν κοινόχρηστό buffer μεταξύ των διεργασιών, μέσω της mmap(), αποδεσμεύεται πλήρως μόνο όταν όλες οι διεργασίες που έχουν αντίστοιχη αντιστοίχιση την αποδεσμεύσουν. Πολύ σωστά γίνεται και η αποδέσμευση του αρχείου που ανοίξαμε για ανάγνωση προτού τελειώσει το πρόγραμμά μας και κάνουμε return 0.





1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

Αρχικά φέρνουμε το περιεχόμενο του προγράμματος που θα εμπλουτίσουμε από τον φάκελο /home/oslab/code/sync-mmap στον δικό μας χώρο. Ύστερα θα προχωρήσουμε στην υλοποίηση του προγράμματος υπολογισμού του Mandelbrot set με δυο τρόπους κάνοντας χρήση διεργασιών αντί για νήματα αυτή τη φορά. Ο πρώτος τρόπος γίνεται με semaphores πάνω από διαμοιραζόμενη μνήμη και ο δεύτερος χωρίς σημαφόρους.

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

*αρχείο mandel-fork.c στον κατάλογο semaphore*

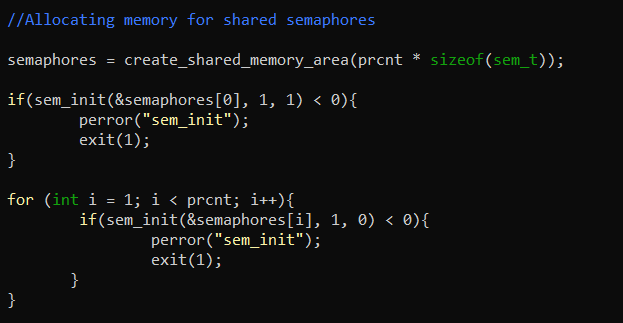
Να επισημάνουμε ότι κατά την υλοποίηση ζητείται οι σημαφόροι να βρίσκονται σε κοινή μνήμη στην οποία θα έχουν πρόσβαση όλες οι διεργασίες. Αυτό θα αναφερθεί και πιο κάτω. Δίνονται, επίσης, δυο βοηθητικές συναρτήσεις τις οποίες θα πρέπει να συμπληρώσουμε κατάλληλα.

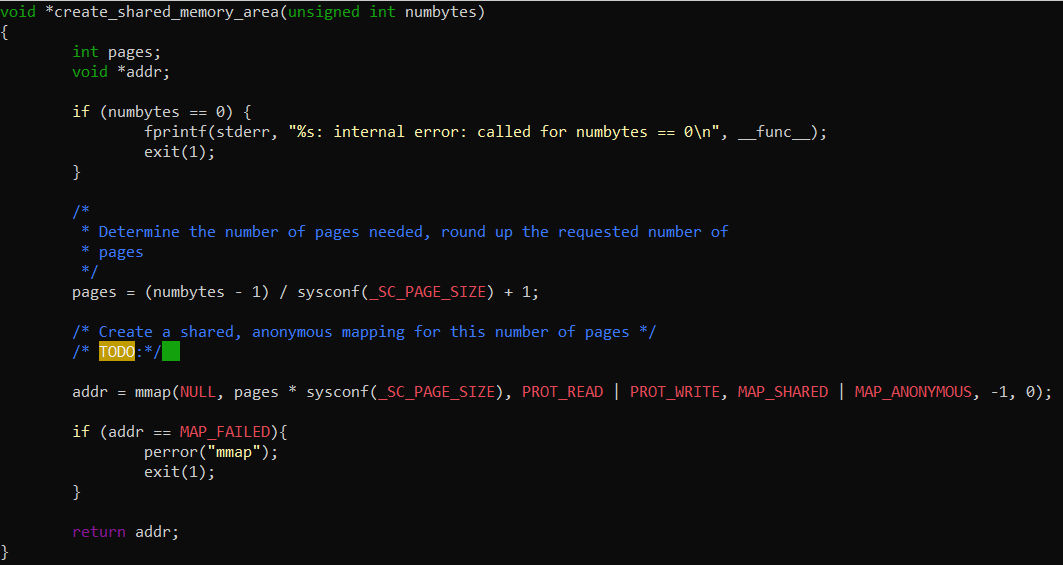
Ξεκινάμε, λοιπόν, με με τον ορισμό τριων μεταβλητών, της prcnt η οποία αποτελεί ουσιαστικά τον προκαθορισμένο από την χρήστη αριθμό των διεργασιών που θα δημιουργήσουμε. Την συμβολοσειρά αυτή που παίρνουμε μέσω από το argv[1], την μετατρέπουμε στον δεκαδικό αριθμό που εισάγει ο χρήστης μέσω της συνάρτησης safe\_atoi(). Ορίζουμε, επίσης, την μεταβλητή process\_number, η οποία αρχικοποιείται μέσα στο fork() και δηλώνει το αναγνωριστικό κάθε διεργασίας, προκειμένου ο σημαφόρος να καλεί την κατάλληλη κάθε φορά για την εκτύπωση (κρίσιμο τμήμα). Η τελευταία μεταβλητή που ορίζουμε είναι η sem\_t \*semaphores. Αυτή χρησιμοποιείται για το κλείδωμα του κρίσιμου τμήματος, αυτό της εκτύπωσης και για το ξεκλείδωμα κάθε φορά της κατάλληλης διεργασίας.

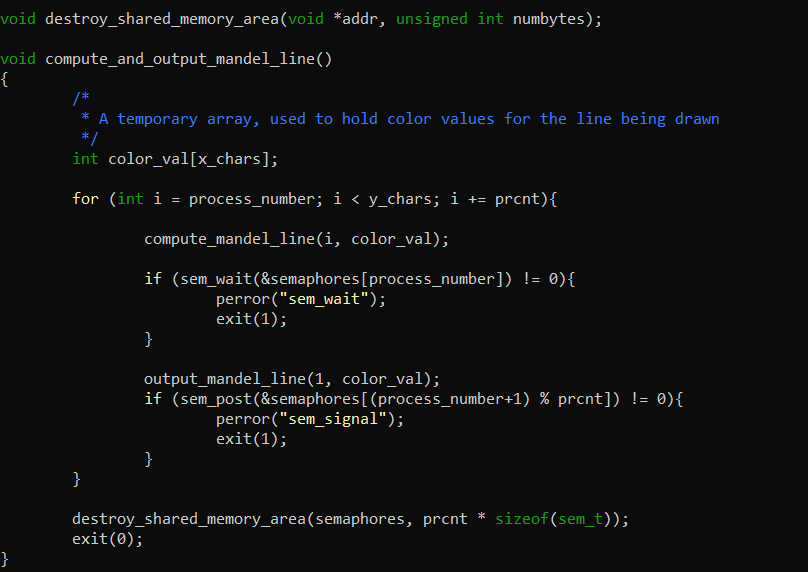
Οι σημαφόροι που θα χρησιμοποιήσουμε καταλαμβάνουν μια κοινόχρηστη περιοχή στην μνήμη, η οποία δεσμεύεται μέσω της συνάρτησης create\_shared\_memory\_area() χάρη στην χρήση της mmap(). Με την mmap() καθορίζουμε ότι η μνήμη αυτή θα είναι κοινόχρηστη, MAP\_SHARED, μεταξύ των διεργασιών και καθορίζουμε, επίσης, ότι αυτή θα λαμβάνει δικαιώματα εγγραφής και ανάγνωσης, PROT\_READ, PROT\_WRITE. Αν η δημιουργία της περιοχής μνήμης είναι επιτυχής, η συνάρτηση επιστρέφει τον δείκτη στην περιοχή της μνήμης. Δεσμεύουμε τόσο χώρο όσες οι διεργασίες επί το μέγεθος του τύπου του σημαφόρου (sem\_t). Αφότου δεσμεύσουμε την κοινόχρηστη μνήμη, περνάμε στην αρχικοποίηση των σημαφόρων. Συγκεκριμένα, με την συνάρτηση sem\_init() αρχικοποιούμε τον σημαφόρο της πρώτης διεργασίας (&semaphore[0]) στην τιμή ‘1’, ξεκλείδωτος δηλαδή προκειμένου να εισέλθει απευθείας στο κρίσιμο τμήμα όταν υπολογίσει το χωρίο της, και όλους τους υπόλοιπους σημαφόρους στην τιμή ‘0’, κλειδωμένοι. Προσέχουμε ότι για όλους τους σημαφόρους ο πρώτος άσσος κατά την αρχικοποίηση σημαίνει ότι είναι κοινόχρηστος μεταξύ των διεργασιών, είναι δηλαδή ορατός σε όλες τις διεργασίες.

Τέλος, είναι σημαντικό να μην ξεχάσουμε να αποδεσμεύσουμε τις θέσεις μνήμης που δεσμεύσαμε. Κάνουμε, αρχικά, sem\_destroy() καταστρέφοντας έναν έναν του σημαφόρους και στην συνέχεια καλούμε την δοθείσα συνάρτηση destroy\_shared\_memory\_area(), αποδεσμεύοντας έτσι την κοινόχρηστη μνήμη που καταλάμβαναν οι σημαφόροι. Η συνάρτηση αυτή κάνει χρήση της συνάρτησης munmap().

Παρακάτω ακολουθούν οι φωτογραφίες με την υλοποίηση των χωρίων που αναφέραμε:







Στην τελευταία φωτογραφία φαίνεται ότι έχουμε ορίσει πάνω πάνω την συνάρτηση destroy\_shared\_memory\_area() προκειμένου να είναι ορατή από την συνάρτηση compute\_and\_output\_mandel\_line() που ακολουθεί, η οποία την καλεί αποδεσμεύοντας την θέση της μνήμης που καταλαμβάνει ο σημαφόρορος κάθε διεργασίας που ολοκληρώνει. Είναι εμφανές, επίσης, ότι κάθε διεργασία αναλαμβάνει να υπολογίσει διαφορετικό χωρίο από κάθε άλλη διεργασία, και αυτό ξεκινώντας με την σειρά που καθορίζει το χαρακτηριστικό της (αριθμός διεργασίας) και προχωράει prcnt γραμμές παρακάτω μετά, όπου prcnt ο αριθμός των διεργασιών που έχουν δημιουργηθεί ή θα δημιουργηθούν.

1.2.2 Υλοποίηση χωρίς semaphores

*αρχείο mandel-fork.c στον κατάλογο no\_sema*

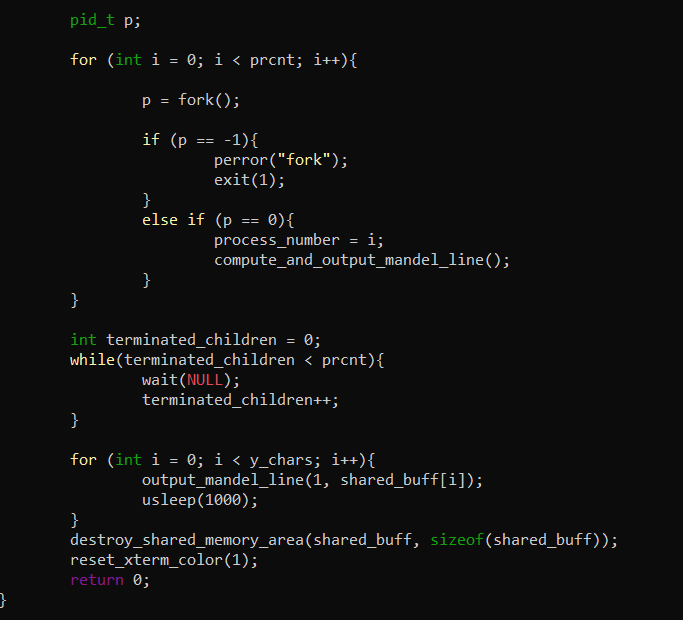
Εδώ, σε αυτή την υλοποίηση θέλουμε να αποφύγουμε την χρήση σημαφόρων. Για τον σκοπό αυτό έχουμε ορίσει έναν δισδιάστατο πίνακα, τον \*\*shared\_buff, στον οποίο κάθε διεργασία θα αποθηκεύει το αποτέλεσμά της με τον κατάλληλο τρόπο. Συγκεκριμένα, με την ακόλουθη γραμμή κώδικα shared\_buff = create\_shared\_memory\_area(y\_chars \* sizeof(int)), δεσμεύουμε μια κοινόχρηστη περιοχή στην μνήμη, κατασκευάζοντας έναν πίνακα δεικτών με μέγεθος όσο και οι γραμμές του Mandelbrot (y\_chars). Αυτός ο πίνακας θα περιέχει τους δείκτες σε καθένα από τους υποπίνακες μεγέθους x\_chars, δεμεύοντας πάλι μια κοινόχρηστη περιοχή στην μνήμη (for (int i = 0; i < y\_chars; i++){ shared\_buff[i] = create\_shared\_memory\_area(x\_chars \* sizeof(char)); }).

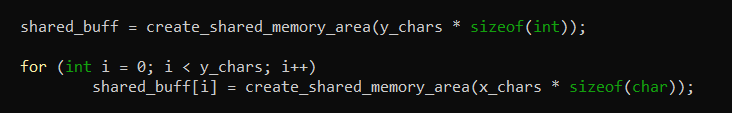
Με αυτόν τον τρόπο κάθε διεργασία θα αποθηκεύει στην κατάλληλη θέση την γραμμή που υπολογίζει, ώστε να είναι σε θέση η διεργασία γονέα να τυπώσει σωστά το Mandelbrot set. Η εκτύπωση, δηλαδή, του Mandelbrot set δεν θα ξεκινήσει προτού ολοκληρώσουν όλες οι διεργασίες την εγγραφή των αποτελεσμάτων τους στην διαμοιραζόμενη θέση μνήμης.

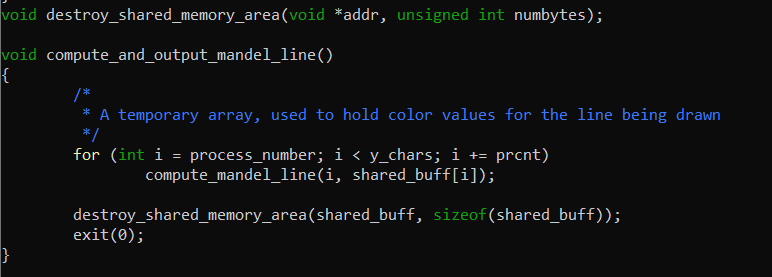
Περνάμε, τώρα, στην κλήση της fork() δημιουργώντας τόσες διεργασίες παιδιά όσα έχει καθορίσει ο χρήστης μέσω του argv[1]. Όταν p==0, βρισκόμαστε στην διεργασία παιδί, αποθηκεύουμε το αναγνωριστικό της στην μεταβλητή process\_number και καλούμε την συνάρτηση compute\_and\_output\_mandel\_line(). Εδώ δεν χρησιμοποιούμε κάποιο κλείδωμα, αλλά όταν μια διεργασία έχει υπολογίσει την εκάστοτε γραμμή της την αποθηκεύει στην κοινόχρηστη δισδιάστατη μεταβλητή-πίνακα στην μνήμη που έχουμε δεσμεύσει. Να αναφέρουμε ότι σε κάθε διεργασία αναθέτουμε διαφορετικές γραμμές, και συγκεκριμένα κάθε διεργασία θα ξεκινήσει με την γραμμή που ορίζει το αναγνωριστικό της (το ποια διεργασία δηλαδή είναι, πχ για την πρώτη, process\_number = 0) και μετά παίρνει την γραμμή i+=prcnt, δηλαδή την γραμμή prcnt θέσεις πιο κάτω από αυτή που μόλις υπολόγισε.

Όταν ολοκληρώνει μια μια διεργασία παιδί τον υπολογισμό των γραμμών που της έχουν ανατεθεί, αποδεσμεύει τους δείκτες στις θέσεις μνήμης που έχει κληρονομήσει και κάνει exit(0).

Ακολουθούν ορισμένες φωτογραφίες από τα σημεία κώδικα που αναφέραμε:





****

1.3 (Προαιρετική) Επέκταση Άσκησης 1

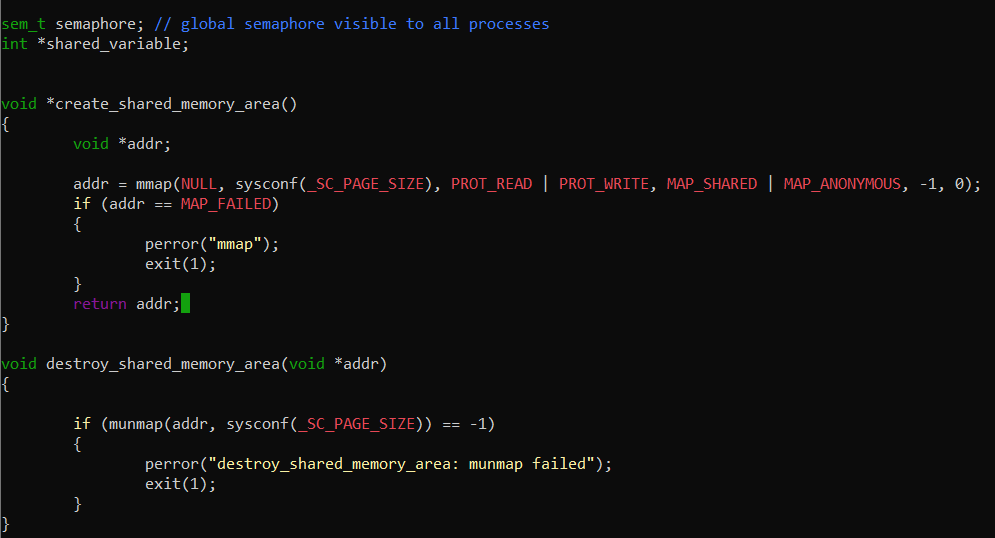
*αρχείο ask3final.c στον κατάλογο /third\_ex/ex\_3*

Σε αυτή την άσκηση μας ζητείται να τροποποιήσουμε κατάλληλα το πρόγραμμα της άσκησης 3 της πρώτης σειράς ασκήσεων ώστε πλέον οι διεργασίες παιδιά να χρησιμοποιούν μια κοινή μεταβλητή (πάνω από διαμοιραζόμενη μνήμη) για την μέτρηση των εμφανίσεων του χαρακτήρα στόχου. Οι προσβάσεις σε αυτή την κοινή μεταβλητή θα γίνεται με την χρήση ενός σημαφόρου πάνω από διαμοιραζόμενη μνήμη, κοινό για όλες τις διεργασίες. Με αυτόν τον τρόπο, πλέον, όταν το πρόγραμμά μας θα δέχεται το σήμα Ctrl+C (SIGINT) θα μπορεί να τυπώσει και τον αριθμό εμφάνισης του χαρακτήρα-στόχου μέχρι εκείνη την στιγμή.

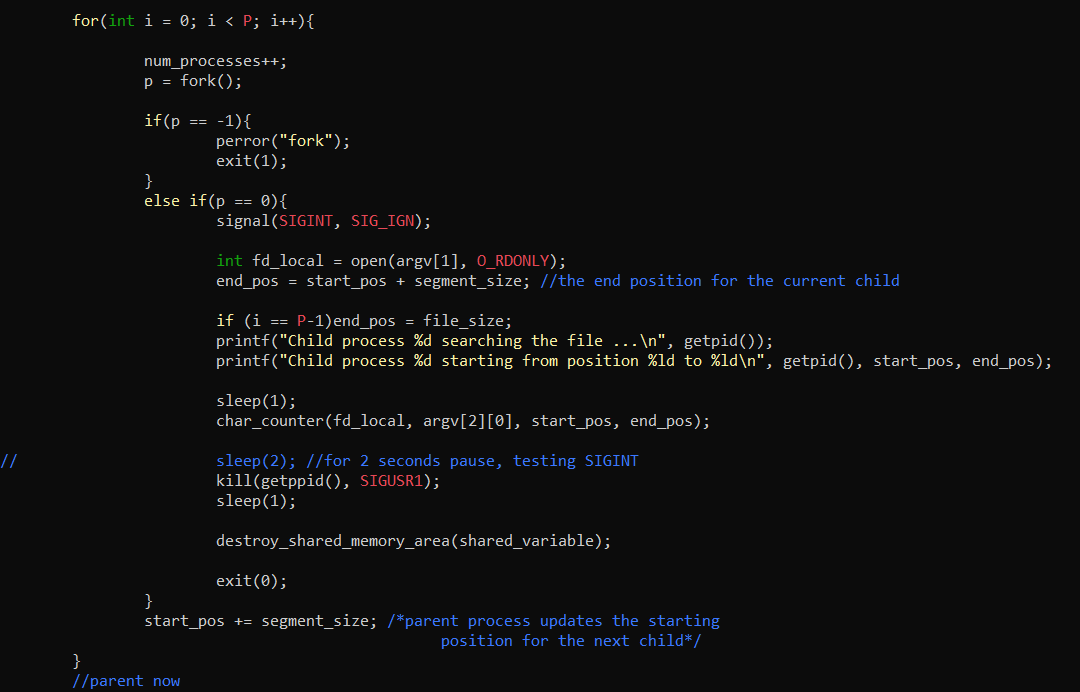
Θα προχωρήσουμε με μια σύντομη παρουσίαση του σκεπτικού πίσω από τις τροποποιήσεις που κάναμε και θα παραθέσουμε τις αντίστοιχες φωτογραφίες με τον κώδικα.

Αρχικά, θέλουμε να αποφύγουμε την επικοινωνία μέσω pipes και στην θέση τους να χρησιμοποιήσουμε μια κοινόχρηστη μεταβλητή, την int \*shared\_variable, η οποία θα δείχνει σε ένα κοινόχρηστο χώρο στην μνήμη όπου θα αποθηκεύονται οι εμφανίσεις του χαρακτήρα στόχου. Για την δέσμευση του χώρου αυτού στην μνήμη χρησιμοποιούμε την void \*create\_shared\_memory\_area(), η οποία επιστρέφει έναν δείκτη στην θέση μνήμης που δεσμεύσαμε. Για την αποδέσμευση της θέσης αυτής έχουμε την void destroy\_shared\_memory\_area().

Ακολουθούν οι δυο αυτές συναρτήσεις:

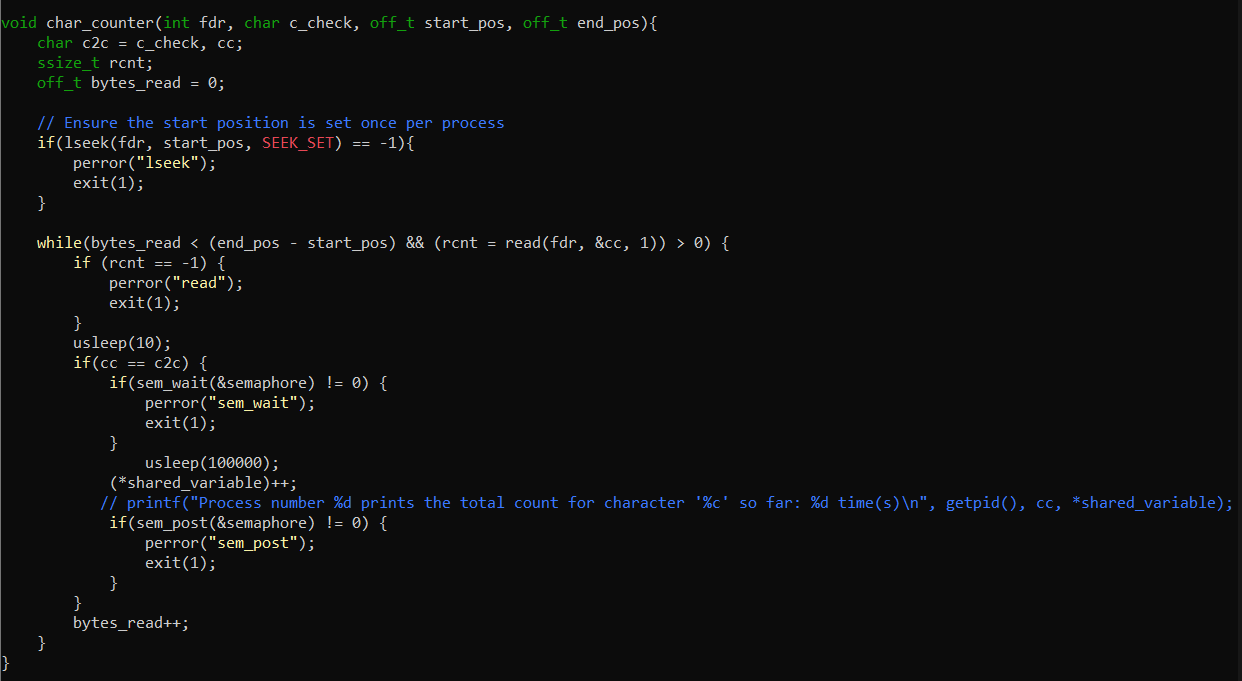


Συνεχίζοντας, στην main() συνάρτηση αφαιρέσαμε τις σωληνώσεις και προκειμένου να αποφύγουμε race conditions κατά την αύξηση του μετρητή εμφάνισης του χαρακτήρα-στόχου (κρίσιμο τμήμα), χρησιμοποιήσαμε έναν κοινόχρηστο σημαφόρο, τον οποίο αρχικοποιήσαμε με την sem\_init() στην τιμή ‘1’ (unlocked) ως shared (πρώτος ‘1’). Αυτόν τον χρησιμοποιούμε στην συνάρτηση void char\_counter() ώστε να επιτρέπει σε μια μόνο διεργασία την εγγραφή στην θέση μνήμης το πλήθος χαρακτήρων-στόχου που έχει βρει μέχρι τότε. Προτού αναφερθούμε λίγο περισσότερο στην συνάρτηση μέτρησης, είναι σημαντικό να σχολιάσουμε μια ακόμα αλλαγή που κάναμε σε σχέση με το αντίστοιχο αρχείο μας στην πρώτη σειρά ασκήσεων. Συγκεκριμένα, συναντήσαμε πρόβλημα όταν το αρχείο-στόχο για την εύρεση του χαρακτήρα-στόχου το άνοιγε αποκλειστικά και μόνο η γονεϊκή διεργασία και κληρωνομούσαν τα δικαιώματα ανάγνωσης οι διεργασίες παιδιά χωρίς να το ανοίγουν εκ νέου. Αυτό συνέβαινε γιατί όταν ανοίγουμε ένα αρχείο με την συνάρτηση open() και μετά καλούμε fork(), αυτό το αρχείο με τα δικαιώματα που του είχαμε ορίσει (ανάγνωσης στην περίπτωσή μας) και τον file descriptor κληρονομείται από τις διεργασίες παιδιά. Δηλαδή, κάθε διεργασία παιδί θα έχει πρόσβαση στο ανοιγμένο αρχείο με δικαίωμα ανάγνωσης μέσω ενός δικού της file descriptor ο οποίος θα δείχνει εκεί που δείχνει και της γονεϊκής διεργασίας. Οι διεργασίες παιδιά κληρονομούν κατ’ επέκταση το file description του αρχείου, με αποτέλεσμα αλλαγή στο offset σε μια διεργασία (μέσω της lseek() για παράδειγμα) να επηρεάζει και το file offset στις υπόλοιπες διεργασίες. Αυτό ήταν το πρόβλημα που είχαμε, διότι κάθε κληρονομημένος file descriptor που είχε κάθε διεργασία παιδί έδειχνε σε ένα συγκεκριμένο file description, με αποτέλεσμα κάθε φορά που μια διεργασία πείραζε το offset ώστε να ψάξει το χωρίο της, επηρέαζε και τις ήδη υπάρχουσες διεργασίες που αναζητούσαν το δικό τους χωρίο. Αυτό είχε σαν συνέπεια να παίρνουμε ακαθόριστο αριθμό εμφάνισης του χαρακτήρα-στόχου. Έτσι, τελικά, λύσαμε το πρόβλημα ορίζοντας ένα int local\_fd για κάθε διεργασία παιδί, ώστε να έχει κάθε διεργασία, όταν αυτή δημιουργείται, το δικό της file description.



Τελειώνοντας, θα αναφερθούμε στην αποθήκευση του μετρητή στην μνήμη μέσα στην συνάρτηση char\_counter() και την εκτύπωση του αποτελέσματος από την γονεϊκή διεργασία. Αφού δεσμεύσουμε την θέση στην μνήμη όπου θα αποθηκεύεται αυτός ο μετρητής, τον αρχικοποιούμε στην τιμή 0, μέσω του (\*shared\_variable) = 0, και ύστερα κάθε διεργασία που εισέρχεται στο κρίσιμο τμήμα (είμαστε στην char\_counter() πλέον), δηλαδή όταν μια διεργασία βρίσκει τον χαρακτήρα-στόχο (cc == c2c) και προλαβαίνει το κρίσιμο τμήμα ξεκλείδωτο, αυξάνει τον μετρητή κατά ένα μέσω της εντολής (\*shared\_variable)++. Αν τυχόν βρει τον χαρακτήρα-στόχο αλλά βρει κλειδωμένο το κρίσιμο τμήμα (εγγράφει άλλη διεργασία στην κοινόχρηστη θέση μνήμης) τότε περιμένει μέχρι να ελευθερωθεί (unlock) το κρίσιμο τμήμα με την συνάρτηση sem\_post(&semaphore), η οποία αυξάνει την τιμή του κοινόχρηστου σημαφόρου από 0 σε 1, δείχνοντας ότι το κρίσιμο τμήμα είναι ξεκλείδωτο.

Αφού ολοκληρώσουν όλες οι διεργασίες παιδιά την αναζήτηση του χαρακτήρα-στόχου στα χωρία που τους έχουμε ορίσει, τότε η γονεϊκή διεργασία τυπώνει την μεταβλητή (\*shared\_variable).



**Απαντήσεις στις Ερωτήσεις:**

1.2.1

1)Περιμένουμε να έχει καλύτερη απόδοση η παράλληλη υλοποίηση με νήματα και όχι με τις διεργασίες. Αυτό γιατί η δημιουργία διεργασιών είναι πολύ πιο χρονοβόρα σε σχέση με την δημιουργία νημάτων καθώς κατά την δημιουργία μιας νέας διεργασίας πρέπει να δημιουργηθούν τα PCB, να υποστηριχθεί το copy on write και φυσικά μέσω της χρονοδρομολόγησης να αφαιρείται μια διεργασία από τον επεξεργαστή προκειμένου να επέλθει σε αυτόν μια άλλη διεργασία. Το τελευταίο φυσικά συμβαίνει και στα νήματα τα οποία αφαιρούνται από τους πυρήνες του επεξεργαστή προκειμένου να αντικατασταθούν από άλλο νήμα, ωστόσο η διαδικασία αυτή στα νήματα υλοποιείται σημαντικά ταχύτερα από ότι με τις διεργασίες. Επιπλέον, το γεγονός ότι τα semaphores βρίσκονται σε διαμοιραζόμενη μνήμη μεταξύ των διεργασιών δηλώνει πώς πρέπει να χρησιμοποιήσουμε την create\_shared\_memory\_area() (η οποία καλεί την mmap και επιστρέφει δείκτη στον δεσμευμένο χώρο) και εντέλει να αποδεσμεύσουμε την μνήμη αυτή μέσω της destroy\_shared\_memory\_area() (η οποία καλεί την munmap) και μάλιστα από όλες τις διεργασίες ώστε να αφαιρούμε και τους δείκτες της εικονικής μνήμης της διεργασίας αυτής από την φυσική μνήμη. Όλη αυτή η διαδικασία επιβραδύνει αρκετά την υλοποίηση με διεργασίες σε σχέση με την υλοποίηση με νήματα καθώς τα νήματα έχουν εν γένει μεταξύ τους κοινή μνήμη με αποτέλεσμα να επιταχύνουν την διαδικασία.

1.2.1

2)Το mmap() interface δεν μπορεί όπως το εξετάσαμε μέχρι τώρα (χρησιμοποιώντας τις σημαίες MAP\_ANONYMOUS και MAP\_SHARED) να χρησιμοποιηθεί για τον διαμοιρασμό μνήμης μεταξύ διεργασιών που δεν έχουν κοινό ancestor. Αυτό συμβαίνει καθώς για τον διαμοιρασμό μνήμης μεταξύ των διεργασιών κατ’ αυτόν τον τρόπο πρέπει μια αρχική γονική διεργασία να δημιουργήσει ένα segment στην μνήμη που θα είναι κοινό για όλες τις διεργασίες και έπειτα μετά την κλήση της fork θα το κληρονομούν και οι διεργασίες παιδιά και θα μπορούν να αλληλεπιδράσουν σε αυτό το κοινό segment. Αυτό το segment δηλαδή δεν θα είναι διαθέσιμο σε άλλες διεργασίες που δεν έχουν δημιουργηθεί μέσω fork() από την αρχική διεργασία. Προκειμένου, λοιπόν, να δημιουργήσουμε διαμοιραζόμενη μνήμη μεταξύ διεργασιών που δεν προέρχονται από τον ίδιο ancestor θα πρέπει να αναζητήσουμε κάποια άλλη μέθοδο αντί του mmap() interface που εξετάσαμε μέχρι τώρα.

1.2.2

1)Στην συγκεκριμένη υλοποίηση το σχήμα συγχρονισμού το οποίο έχουμε υλοποιήσει συνίσταται στο εξής: Κάθε διεργασία υπολογίζει τις γραμμές που τις αντιστοιχούν με βάση την αρίθμησή της (η nοστη διεργασία υπολογίζει της γραμμές n + i\*prcnt όπου i = 0,1,2… και prcnt ο αριθμός των διεργασιών) και τις τοποθετεί σε έναν κοινόχρηστο shared\_buff στις αντίστοιχες θέσεις του. Η γονική διεργασία περιμένει ώσπου να τερματίσουν όλες οι διεργασίες παιδιά και όταν αυτό συμβεί τότε εκτυπώνει όλες τις γραμμές. Αξίζει να σημειωθεί πώς με την συγκεκριμένη υλοποίηση δεν έχουμε κρίσιμο τμήμα και έτσι ο υπολογισμός είναι τελείως παράλληλος μεταξύ των διεργασιών καθώς δεν τίθεται ποτέ η περίπτωση μια διεργασία (παιδί) να περιμένει να εισέλθει στο κρίσιμο τμήμα (ύπνος) για να εκτυπώσει το χωρίο της. Άρα η υλοποίηση αυτή φανερώνει επιτάχυνση σε σχέση με την προηγούμενη υλοποίηση με τους σημαφόρους ως πρός τον υπολογισμό. Ωστόσο, η υλοποίηση αυτή έχει το πρόβλημα πώς η γονική διεργασία έπειτα θα πρέπει να διατρέξει ολόκληρο τον shared\_buff για να εκτυπώσει τους υπολογισμούς όλων των διεργασιών. Στο σημείο αυτό είναι που υστερεί.

Σε περίπτωση που ο shared\_buff είχε διαστάσεις NPROCS x x\_chars τότε κάθε διεργασία θα είχε την δική της γραμμή σε αυτόν το shared\_buff. Θα έπρεπε η κάθε διεργασία να υπολογίσει την iοστη γραμμή που της αντιστοιχεί και να την τοποθετήσει στην γραμμή της στον shared\_buff. Όταν και οι NPROCS διεργασίες τελειώνουν με τον υπολογισμό μιας γραμμής (αφου την έχουν τοποθετήσει στον shared\_buff) τότε σταματούν την εκτέλεσή τους ώσπου μια άλλη διεργασία να εκτυπώσει αυτές τις NPROCS γραμμές του shared\_buff. Αυτό το σχήμα συγχρονισμού θα επαναληφθεί για τον υπολογισμό όλων των γραμμών. Ένας τρόπος με τον οποίο θα μπορούσαμε να πετύχουμε το παραπάνω σχήμα συγχρονισμού θα ήταν να ακολουθήσουμε το σχήμα συγχρονισμού που είδαμε στο αρχείο mmap.c. Δηλαδή, κάθε φορά που η διεργασία υπολογίζει ένα χωρίο που της αντιστοιχεί να το βάζει στον shared\_buff και έπειτα να καλεί raise(SIGSTOP); Η γονική διεργασία θα καλεί waitpid παίρνοντας ως παράμετρο WUNTRACED (το σημείο αυτό απαιτεί τροποποίηση σε σχέση με την mmap καθώς εδώ πρέπει να υπολογιστούν και οι NPROCS γραμμές από τις αντίστοιχες διεργασίες) και έπειτα θα τυπώνει τα χωρία. Μετά θα ξυπνά πάλι όλες τις διεργασίες μέσω kill με παράμετρο SIGCONT και θα επαναλαμβάνεται η διαδικασία που εξετάσαμε.