Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Λειτουργικά Συστήματα (Κ22) / Περίοδος 2024-2025
2η Εργασία

Επέκταση λειτουργιών συστήματος αρχείων στο xv6

Σε αυτή την εργασία θα προσθέσετε υποστήριξη για μεγάλα αρχεία και συμβολικούς συνδέσμους (symbolic links) στο σύστημα αρχείων του xv6. Για την οργάνωση του συστήματος αρχείων στο xv6, μελετήστε το Κεφάλαιο 8 από το βιβλίο του xv6, καθώς και τον αντίστοιχο κώδικα.

Μεγάλα αρχεία

Σε αυτή την άσκηση θα αυξήσετε το μέγιστο μέγεθος ενός αρχείου στο xv6. Στην παρούσα κατάσταση, τα αρχεία xv6 περιορίζονται σε 268 blocks ή 268*BSIZE bytes (BSIZE=1024 στο xv6). Το όριο αυτό προκύπτει από το γεγονός ότι ένας xv6 inode περιέχει 12 "άμεσους" αριθμούς block και έναν "απλά-έμμεσο" αριθμό block, που αναφέρεται σε ένα block που περιέχει άλλους 256 αριθμούς block, για ένα σύνολο 12+256=268 blocks.

Η εντολή bigfile δημιουργεί το μεγαλύτερο αρχείο που μπορεί και αναφέρει το μέγεθος του:

```
$ bigfile

..

wrote 268 blocks

bigfile: file is too small

$
```

Ο έλεγχος αποτυγχάνει γιατί η bigfile αναμένει να μπορεί να δημιουργήσει ένα αρχείο με 65803 blocks, αλλά το μη τροποποιημένο xv6 περιορίζει τα αρχεία σε 268 blocks.

Η αλλαγή που θα πραγματοποιήσετε στον κώδικα του συστήματος αρχείων του xv6, θα υποστηρίζει ένα "διπλά-έμμεσο" block σε κάθε inode, που θα περιέχει 256 διευθύνσεις "απλά-έμμεσων" blocks, κάθε μία από τις οποίες μπορεί να περιέχει μέχρι 256 διευθύνσεις από blocks δεδομένων. Το αποτέλεσμα θα είναι ότι το μέγιστο μέγεθος ενός αρχείου να είναι μέχρι 65803 blocks, δηλαδή 256*256+256+11 blocks (11 αντί για 12, γιατί χρησιμοποιούμε ένα από τα blocks που προορίζονταν για δεδομένα προκειμένου να το κάνουμε "διπλά-έμμεσο").

Αναλυτικά

Το πρόγραμμα mkfs δημιουργεί την εικόνα του δίσκου στο σύστημα αρχείων του xv6 και καθορίζει πόσα συνολικά blocks έχει το σύστημα αρχείων. Το μέγεθος αυτό καθορίζεται από τη σταθερά FSSIZE στο αρχείο kernel/param.h. Θα παρατηρήσετε ότι το FSSIZE στο αποθετήριο για αυτή την εργασία είναι 200000 blocks. Θα πρέπει να δείτε την ακόλουθη έξοδο από το mkfs/mkfs στην έξοδο της make:

```
nmeta 70 (boot, super, log blocks 30 inode blocks 13, bitmap blocks
25) blocks 199930 total 200000
```

Η παραπάνω γραμμή περιγράφει το σύστημα αρχείων που δημιούργησε η mkfs/mkfs. Περιέχει 70 blocks μετα-δεδομένων (blocks που περιγράφουν το σύστημα αρχείων) και 199930 blocks δεδομένων, συνολικά 200000 blocks.

Σημειώστε ότι η εντολή make qemu χτίζει ένα νέο fs.img και αποθηκεύει το υπάρχον στο fs.img.bk. Αν θέλετε να εκτελέσετε το xv6 με το υπάρχον fs.img αντί να χτίσετε νέο, εκτελέστε την εντολή make qemu-fs.

Λεπτομέρειες

Η μορφή του inode στον δίσκο καθορίζεται από το struct dinode στο αρχείο fs.h. Δείτε τα στοιχεία NDIRECT, NINDIRECT, MAXFILE και το addrs[] του struct dinode. Δείτε επίσης το Σχήμα 8.3 στο βιβλίο του xv6 για ένα διάγραμμα του xv6 inode.

Ο κώδικας που βρίσκει τα δεδομένα ενός αρχείου στον δίσκο βρίσκεται στη συνάρτηση bmap() στο fs.c. Μελετήστε και κατανοήστε τη λειτουργία της. Η bmap() καλείται τόσο κατά την ανάγνωση όσο και κατά την εγγραφή ενός αρχείου. Κατά την εγγραφή, η bmap() αναθέτει όσα νέα blocks χρειάζεται για να αποθηκευτούν τα περιεχόμενα του αρχείου, καθώς και την ανάθεση ενός έμμεσου block αν χρειάζεται για να αποθηκευτούν block διευθύνσεις.

Η bmap() χειρίζεται δύο ειδών αριθμούς block. Το όρισμα bn είναι ένας "λογικός αριθμός block" - ένας αριθμός block μέσα στο αρχείο, σχετικός με την αρχή του αρχείου. Οι αριθμοί block στο ip->addrs[], και το όρισμα στην bread() είναι αριθμοί block στον δίσκο. Μπορείτε να θεωρήσετε την bmap() ως τη συνάρτηση που απεικονίζει τους λογικούς αριθμούς block ενός αρχείου σε αριθμούς block στον δίσκο.

Τροποποιήστε την bmap(), ώστε να υλοποιεί ένα διπλά-έμμεσο block εκτός του απλά-έμμεσου και των άμεσων blocks. Θα υπάρχουν μόνο 11 άμεσα blocks αντί για 12, ώστε να μείνει χώρος για το διπλά-έμμεσο block. Δεν επιτρέπεται να αλλάξετε το μέγεθος ενος inode στον δίσκο. Τα πρώτα 11 στοιχεία του ip->addrs[] θα πρέπει να είναι άμεσα blocks, το 12ο να είναι απλά-έμμεσο (όπως το υπάρχον) και το 13ο το νέο διπλά-έμμεσο block. Η υλοποίηση θα θεωρείται ολοκληρωμένη όταν η bigfile γράφει 65803 blocks επιτυχώς και η usertests δεν αποτυγχάνει.

```
$ bigfile
```

```
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..................................

wrote 65803 blocks

done; ok

$ usertests -q

...

ALL TESTS PASSED

$
```

Η εκτέλεση της bigfile θα διαρκέσει αρκετά (τουλάχιστον 1 λεπτό). Επίσης η εκτέλεση της usertests κατά τη διάρκεια του ελέγχου είναι αρκετά χρονοβόρα.

Υποδείξεις

- Βεβαιωθείτε ότι κατανοήσατε τη λειτουργία της bmap(). Σχεδιάστε τις σχέσεις μεταξύ των ip->addrs[], του έμμεσου block, του διπλά-έμμεσου block και των απλά-έμμεσων blocks στα οποία δείχνει και των blocks δεδομένων.
- Σχεδιάστε πώς θα δεικτοδοτήσετε το διπλά-έμμεσο block και τα έμμεσα blocks στα οποία δείχνει με τον λογικό αριθμό block.
- Αν αλλάξετε τον ορισμό του NDIRECT, θα πρέπει να τροποποιήσετε και τη δήλωση των addrs[] στο αρχείο file.h. Βεβαιωθείτε ότι το struct inode και το struct dinode έχουν τον ίδιο αριθμό στοιχείων στα addrs[] arrays.
- Αν αλλάξετε τον ορισμό του NDIRECT, θα πρέπει να δημιουργήσετε ένα νέο fs.img, αφού η mkfs χρησιμοποιεί το NDIRECT για να κατασκευάσει το σύστημα αρχείων.
- Αν το σύστημα αρχείων σας περιέλθει σε κακή κατάσταση (ίσως από κατάρρευση του συστήματος), διαγράψτε το fs.img (από το Linux, όχι από το xv6). Η make θα δημιουργήσει μία νέα εικόνα συστήματος αρχείων.
- Χρησιμοποιείτε την brelse() για κάθε block που διαβάζετε με την bread().
- Θα πρέπει να αναθέτετε έμμεσα blocks και διπλά-έμμεσα blocks μόνο αν είναι απαραίτητο, όπως η αρχική bmap().
- Βεβαιωθείτε ότι η itrunc απελευθερώνει όλα τα blocks σε ένα αρχείο, συμπεριλαμβανομένων των διπλά-έμμεσων blocks.
- Η usertests καθυστερεί αρκετά σε αυτή την εργασία, αφού το FSSIZE είναι σημαντικά μεγαλύτερο και τα μεγάλα αρχεία είναι μεγαλύτερα.

Προαιρετική (bonus) επέκταση

Υποστήριξη τριπλά-έμμεσων blocks.

Συμβολικοί σύνδεσμοι (symbolic links)

Σε αυτή την άσκηση θα προσθέσετε συμβολικούς συνδέσμους στο xv6. Οι συμβολικοί σύνδεσμοι αναφέρονται σε ένα συνδεδεμένο αρχείο ή κατάλογο (directory) μέσω ονόματος μονοπατιού. Όταν ανοίγει ένας συμβολικός σύνδεσμος, ο πυρήνας ψάχνει το όνομα συνδεδεμένο όνομα. Οι συμβολικοί σύνδεσμοι μοιάζουν με hard links, αλλά οι hard links περιορίζονται μόνο σε αρχεία στον ίδιο δίσκο, δεν μπορούν να συνδεθούν σε καταλόγους και συνδέονται μόνο σε ένα συγκεκριμένο inode, αντί για οτιδήποτε βρίσκεται στο όνομα που συνδέεται μέσω συμβολικού συνδέσμου. Για την υλοποίηση των συμβολικών συνδέσμων, θα πρέπει να κατανοήσετε πώς λειτουργεί η αναζήτηση των ονομάτων μονοπατιών.

Δεν απαιτείται υλοποίηση συμβολικών συνδέσμων για καταλόγους για αυτή την άσκηση. Η μόνη κλήση συστήματος που θα πρέπει να γνωρίζει πώς ακολουθούνται οι συμβολικοί σύνδεσμοι είναι η open().

Αναλυτικά

Θα υλοποιήσετε την κλήση συστήματος symlink(char *target, char *path), η οποία δημιουργεί ένα νέο συμβολικό σύνδεσμο στο path που αναφέρεται στο αρχείο target. Για περισσότερες πληροφορίες man symlink. Για τον έλεγχο της υλοποίησης, προσθέστε το symlinktest στο Makefile και εκτελέστε το. Η υλοποίηση θα είναι πλήρης όταν οι έλεγχοι παράγουν το ακόλουθο αποτέλεσμα (και η usertests επιτυγχάνει):

```
$ symlinktest

Start: test symlinks

test symlinks: ok

Start: test concurrent symlinks

test concurrent symlinks: ok

$ usertests -q

...

ALL TESTS PASSED

$
```

Υποδείξεις

- Αρχικά, δημιουργήστε μία νέα κλήση συστήματος symlink, προσθέστε μία εγγραφή στα user/usys.pl, user/user.h και υλοποιήστε μία κενή sys_symlink στο kernel/sysfile.c
- Προσθέστε ένα νέο τύπο αρχίου (T_SYMLINK) στο kernel/stat.h που αναπαριστά ένα συμβολικό σύνδεσμό.
- Προσθέστε μία νέα σημαία στο kernel/fcntl.h (O_NOFOLLOW), η οποία μπορεί να χρησιμοποιηθεί με την κλήση συστήματος open(). Σημειώστε ότι οι σημαίες που περνάν στην open() συνδυάζονται μέσω ενός bitwise τελεστή OR. Η νέα σας σημαία δεν θα πρέπει να συγκρούεται με κάποια άλλη. Με τον τρόπο αυτό θα μπορείτε να μεταγλωττίσετε το user/symlinktest.c όταν θα το προσθέσετε στο Makefile.
- Υλοποιήστε την κλήση συστήματος symlink(target, path) για τη δημιουργία ενός νέου συμβολικού συνδέσμου στο path που αναφέρεται στο target. Σημειώστε ότι το target δεν χρειάζεται να υπάρχει για να επιτύχει η κλήση συστήματος. Θα πρέπει να επιλέξετε ένα μέρος να αποθηκεύετε το target ενός συμβολικού συνδέσμου, για παράδειγμα στα blocks δεδομένων ενός inode. Η κλήση συστήματος symlink θα πρέπει να επιστρέφει έναν ακέραιο που να συμβολίζει επιτυχία (0) ή αποτυχία (-1), όμοια με την link() ή την unlink().
- Τροποποιήστε την κλήση συστήματος open() για να χειρίζεται την περίπτωση, όπου το path είναι συμβολικός σύνδεσμος. Αν το αρχείο δεν υπάρχει, η open() πρέπει να αποτυγχάνει. Αν μία διεργασία καθορίζει τη O_NOFOLLOW στις σημαίες της open(), η open() θα πρέπει να ανοίγει τον συμβολικό σύνδεσμο, και όχι να ακολουθεί τον συμβολικό σύνδεσμο.
- Αν το συνδεδεμένο αρχείο είναι επίσης συμβολικός σύνδεσμος, θα πρέπει να ακολουθήσετε αναδρομικά την αλυσίδα μέχρι να βρεθεί αρχείο που δεν είναι σύνδεσμος. Αν οι σύνδεσμοι σχηματίζουν κύκλο, θα πρέπει να επιστρέψετε ένα κωδικό σφάλματος. Μπορείτε να προσεγγίσετε αυτή τη συμπεριφορά, αν επιστρέφετε κωδικό σφάλματος αν το βάθος των συνδέσμων ξεπεράσει κάποιο κατώφλι (π.χ. 10).
- Οι άλλες κλήσεις συστήματος (π.χ. link() και unlink()) δεν πρέπει να ακολουθούν τους συμβολικούς συνδέσμους. Αυτές οι κλήσεις συστήματος λειτουργούν πάνω στον ίδιο τον συμβολικό σύνδεσμο.

Κώδικας

Χρησιμοποιήσετε τον κώδικα για την εργασία ως εξής:

```
$ git clone git://gallagher.di.uoa.gr/xv6-project-2024
Cloning into 'xv6-project-2024'...
...
$ cd xv6-project-2024
```

Τα αρχεία που θα χρειαστείτε για αυτή την εργασία διανέμονται μέσω του συστήματος ελέγχου πηγαίου κώδικα git. Μπορείτε να βρείτε πληροφορίες για το git στο βιβλίο git ή σε άλλες δημόσιες πηγές. Το git επιτρέπει να διατηρείτε πληροφορία για όλες τις αλλαγές που έχετε κάνει στον κώδικα. Για παράδειγμα, αν τελειώσετε ένα μέρος της εργασίας και θέλετε να καταχωρήσετε τοπικά τις αλλαγές σας, μπορείτε να καταγράψετε (commit) τις αλλαγές σας μέσω της εντολής

```
$ git commit -am 'my solution for k22 project'
Created commit 60d2135: my solution for k22 project
 1 files changed, 1 insertions(+), 0 deletions(-)
$
```

Ημερομηνία Παράδοσης: 19 Ιαν 2025
Τρόπος παράδοσης: υποβολή στο eclass, θα πρέπει να παραδοθεί ένα αρχείο tar με περιεχόμενο όλα τα σχετικά αρχεία.
Συνοδευτικό υλικό: τεκμηρίωση 2-3 σελίδων που να εξηγεί τον τρόπο με τον οποίο εργαστήκατε.
Υλοποίηση: η εργασία είναι ατομική.
Η εργασία θα εξεταστεί σε συμβατό xv6 emulator (Linux, Windows WSL) με πρόγραμμα που θα ανακοινωθεί μετά την ημερομηνία παράδοσης. Σημειώστε ότι η έκδοση του qemu θα πρέπει να είναι τουλάχιστον 7.2. Σε περιβάλλον Ubuntu, η έκδοση αυτή παρέχεται από την έκδοση 24.04 και μετά.