**ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**
**Τμήμα Πληροφορικής και Τηλεπικοινωνιών**
**Κ24: Προγραμματισμός Συστήματος – Εαρινό Εξάμηνο 2013**
**2η Προγραμματιστική Εργασία**
**Ημερομηνία Ανακοίνωσης: 28/3/2013**
**Ημερομηνία Υποβολής: 25/4/2013**

## Εισαγωγή στην Εργασία

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με τον προγραμματισμό συστήματος σε Unix, και συγκεκριμένα με την δημιουργία διεργασιών, την επικοινωνία μεταξύ τους με σωλήνωση, καθώς και την δημιουργία bash scripts.

Θα υλοποιήσετε έναν πολύ βασικό debugger χρησιμοποιώντας την κλήση συστήματος *ptrace*, ο οποίος θα ενημερώνει τον χρήστη τα system calls που εκτελέστηκαν από το παρακολουθούμενο πρόγραμμα. Επίσης θα υλοποιήσετε ένα bash script το οποίο θα αυτοματοποιεί την όλη διαδικασία χρήσης του picodb με συγκεκριμένο πρόγραμμα στόχο (σας δίνεται), εξάγοντας μετρήσεις από την έξοδο του.

**Διαδικαστικά:**

- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux της σχολής. Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε τη λίστα του μαθήματος και το URL: www.di.uoa.gr/~mema/courses/k24/k24.html.
- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση, κτλ) είναι οι: Serafeim Chatzopoulos, Sotirios-Efstathios (Stathis) Maneas, Georgia Soulioti, Nikolas Stratis. Email θα βρείτε στην ιστοσελίδα του μαθήματος.
- Εγγραφείτε στην ηλεκτρονική λίστα (mailman) του μαθήματος και παρακολουθείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση (σημείωση: η η-λίστα αυτή δεν έχει καμία σχέση με την hard-copy λίστα που κυκλοφορεί/ησε στα πρώτα μαθήματα και στην οποία θα έπρεπε να γράψετε το όνομά σας και το Unix user-id σας).

**Τι πρέπει να παραδοθεί:**

1. Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα source files, header files, makefile. ΠΡΟΣΟΧΗ: φροντίστε να φτιάξετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με chmod 755 OnomaEponymoProject1.tar (περισσότερα στη σελίδα του μαθήματος).
2. Μια σύντομη περιγραφή (2-3 σελίδες maximum) για τις επιλογές που κάνατε στο σχεδιασμό της άσκησης, σε μορφή PDF.
3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στην παραπάνω αναφορά.

## Υπόβαθρο

Η ptrace είναι ένα system call που επιτρέπει σε μία διεργασία-πατέρας να παρακολουθήσει αλλά και να παρέμβει στην εκτέλεση μιας διεργασίας-παιδιού του. Χρησιμοποιείται κυρίως

από debuggers και προγράμματα παρακολούθησης system calls (βλ. strace). Όταν μία διεργασία παρακολουθείται, σε κάθε signal που δέχεται, σταματάει την εκτέλεσή της και η διεργασία-πατέρας θα ειδοποιηθεί στην επόμενη εκτέλεση της wait. Με την σειρά της, μπορεί να κάνει όποιες ενέργειες της προσφέρει η ptrace και στην συνέχεια να ειδοποιήσει την διεργασία-παιδί να συνεχίσει την εκτέλεσή της. Η υπογραφή της συνάρτησης βρίσκεται στο sys/ptrace.h header file και είναι η εξής:

```
long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

Η τιμή του πρώτου ορίσματος, ορίζει το ποια ενέργεια θέλουμε να πραγματοποιήσουμε. Για παράδειγμα, προκειμένου η διεργασία-παιδί να δηλώσει πως θέλει να παρακολουθηθεί, το request θα έχει τιμή PTRACE_TRACEME, ή για να δηλώσει η διεργασία-πατέρας πως θέλει να ειδοποιείται για τα system calls που εκτελεί το παιδί, το request θα πρέπει να έχει την τιμή PTRACE_SYSCALL. Μπορείτε να βρείτε όλες τις τιμές που μπορεί να πάρει το request, καθώς και την περιγραφή του τί κάνουν, στην man page της ptrace. Το δεύτερο όρισμα είναι το pid του παθητικού process. Το όρισμα address χρησιμοποιείται για να δεικτοδοτήσει μία διεύθυνση που κάποιο ptrace request μπορεί να χρησιμοποιήσει. Τέλος, το όρισμα data, χρησιμοποιείται όταν κάποιο request χρειάζεται δεδομένα από την διεργασία που καλεί την ptrace. Όταν τα ορίσματα addr και data είναι αδιάφορα για ένα request, τους περνάμε NULL τιμές. Η ptrace, επιστρέφει μία τιμή τύπου long, που διαφέρει ανάλογα με το request που εκτέλεσε.

### Χειρισμός system calls σε επίπεδο assembly

Προκειμένου να εκτελεστεί ένα system call, πρέπει να εκτελεστεί η εντολή assembly int $0x80. Αυτή η εντολή θα εκτελέσει το interrupt 80, το οποίο δείχνει σε κώδικα που χειρίζεται τα system calls. Αυτός ο κώδικας, περιμένει να βρει στους καταχωρητές της CPU, τον αριθμό που αντιστοιχεί στο system call που πρόκειται να εκτελεστεί και τα ορίσματα με τα οποία θα πάρει το system call. Για παράδειγμα, στην i386 αρχιτεκτονική, ο αριθμός του system call πρέπει να βρίσκεται στον καταχωρητή eax (στην x64 αρχιτεκτονική, στον RAX) και τα ορίσματα στους καταχωρητές ebx, ecx, edx, esi και edi. Όταν το system call εκτελεστεί, θα τοποθετήσει την τιμή επιστροφής του στον καταχωρητή eax. Επίσης, το λειτουργικό σύστημα, φυλάει τις τιμές των καταχωρητών σε μία περιοχή μνήμης που λέγεται User Area, προκειμένου να μπορεί η ptrace να δει τις τιμές των καταχωρητών μίας διεργασίας-παιδιού. Επομένως, αν για παράδειγμα θέλουμε να δούμε το περιεχόμενο του καταχωρητή eax μίας διεργασίας-παιδιού που θέλει να εκτελέσει ένα system call, θα κάναμε το εξής:

```
long eax_value = ptrace(PTRACE_PEEKUSER, child_pid, EAX * 4, NULL);
```

και θα επιστρέψει την τιμή του καταχωρητή eax. Η EAX δήλωση στο όρισμα address, είναι μία τιμή ορισμένη στο header sys/reg.h, που έχει την τιμή της θέσης μέσα στην User Area που αντιστοιχεί στον EAX καταχωρητή. Αυτή την τιμή, την πολλαπλασιάζουμε με 4 για να γίνει διεύθυνση μέσα στη δομή, γιατί σε ένα 32-bit σύστημα κάθε λέξη έχει μήκος τεσσάρων bytes. Σε ένα 64-bit σύστημα, η αντίστοιχη τιμή (RAX) θα πρέπει να πολλαπλασιαστεί με 8.

Σημειώστε ότι σε περίπτωση syscall, θα γίνουν δύο διακοπές (αλλιώς wait() events): μία

κατά την είσοδο στο syscall (πριν το λειτουργικό εκτελέσει την ζητούμενη λειτουργία), και μια μετά (αφού το λειτουργικό την εκτελέσει και επιστρέψει, αλλά πριν η εφαρμογή δει το αποτέλεσμα). Αυτό θα πρέπει να το διαχειριστεί ανάλογα το πρόγραμμα-πατέρας, για παράδειγμα μετρώντας τα wait() events και χωρίζοντάς τα σε ζεύγη. Σημειώστε ότι το πρώτο wait() event συνήθως είναι η επιστροφή από το πρώτο syscall και όχι η είσοδος σε αυτό.

Επίσης, παρατηρήστε (στο /usr/include/sys/reg.h) και την ύπαρξη του ORIG_EAX (ORIG_RAX αντίστοιχα για 64 bit αρχιτεκτονική) η οποία μας δίνει την τιμή του αιτούμενου system call τόσο κατά την είσοδο (πρώτο event) όσο και κατά την έξοδο (δεύτερο event).

Μιά λύση για την υποστήριξη τόσο 32 όσο και 64 bit αρχιτεκτονικών είναι η χρήση του predefined macro "__x86_64__". Αν αυτό είναι ενεργό, τότε είμαστε σε 64 bit αρχιτεκτονική, αλλιώς σε 32.

Για περισσότερες πληροφορίες ανατρέξτε στην man page της ptrace (man ptrace).
Μία παρουσίαση καθώς και παραδείγματα μπορείτε να βρείτε στον άρθρο Playing with ptrace, Part 1, του Linux journal στο:

`http://www.linuxjournal.com/article/6100`

## Αναλυτική περιγραφή

### Θέμα 1, Μονάδες 80

Αναπτύξτε την εφαρμογή picodb, η οποία θα μπορεί να εκκινεί και να παρακολουθεί ένα άλλο εκτελέσιμο, παρακολουθώντας όλες τις κλήσεις συστήματος που κάνει. Ο picodb θα δέχεται "εντολές" από την είσοδο του τερματικού.

Ο picodb θα δέχεται ως όρισμα το μονοπάτι ενός μεταγλωττισμένου εκτελέσιμου, το οποίο και θα είναι το εκτελέσιμο το οποίο θα παρακολουθήσει. Επίσης θα προσφέρει ένα Command Line Interface (CLI), με τις εξής εντολές:

| **Εντολή** | **Περιγραφή** |
|---|---|
| **trace <category>** | Αιτείται την παρακολούθηση των system calls της κατηγορίας <category>.<br>Η παράμετρος <category> μπορεί να είναι:<br>• "process-control": Συγκεκριμένη κατηγορία syscalls που ορίζεται παρακάτω<br>• "file-management": Συγκεκριμένη κατηγορία syscalls που ορίζεται παρακάτω<br>• "all": Παρακολουθεί όλες τις παραπάνω κατηγορίες system calls. Πρακτικά είναι σαν να έδωσε τις εντολές "trace process-control" και "trace file-management" ο χρήστης. |

| | |
|---|---|
| **redirect <stream> <filename>** | Αιτείται την ανακατεύθυνση ενός από τα τρία standard input/output streams του προγράμματος που θα εκτελεσθεί από/σε αρχείο.<br>Η παράμετρος <filename> είναι όνομα αρχείου.<br>Το <stream> μπορεί να είναι:<br>• stdin: Αιτείται την ανακατεύθυνση της εισόδου του προγράμματος που θα εκτελεσθεί από το αρχείο <filename>. Το αρχείο πρέπει να υπάρχει, αλλιώς εμφανίζεται μήνυμα λάθους.<br>• stdout: Αιτείται την ανακατεύθυνση της εξόδου stdout του προγράμματος που θα εκτελεσθεί στο αρχείο <filename>. Το αρχείο αυτό δημιουργείται αν δεν υπάρχει. Αν υπάρχει, πρέπει το παλιό περιεχόμενο να διαγραφεί με την έναρξη της εκτέλεσης.<br>• stderr: Αιτείται την ανακατεύθυνση της εξόδου stderr του προγράμματος που θα εκτελεσθεί στο αρχείο <filename>. Το αρχείο αυτό δημιουργείται αν δεν υπάρχει. Αν υπάρχει, πρέπει το παλιό περιεχόμενο να διαγραφεί με την έναρξη της εκτέλεσης. |
| **blocking-mode <mode>** | Θέτει την παράμετρο blocking-mode στην κατάσταση <mode><br>Η παράμετρος <mode> μπορεί να είναι:<br>• on: για να παγώνει η εκτέλεση του παρακολουθούμενου προγράμματος κάθε φορά που ένα από τα system calls που παρακολουθούνται καλείται και να ερωτηθεί ο χρήστης αν θέλει να συνεχιστεί η παρακολουθούμενη εκτέλεσή. Για να συνεχιστεί η εκτέλεση, ο χρήστης θα απαντάει με **y** και με **n** για να σταματήσει.<br>• off: το πρόγραμμα εκτελείται χωρίς διακοπή |
| **limit-trace <number>** | Ορίζει ένα όριο για το πόσα system calls κάθε κατηγορίας θα παρακολουθηθούν. Αν οριστεί το limit-trace, μετά από <number> κλήσεις system calls της κάθε κατηγορίας, θα αγνοούνται οι υπόλοιπες κλήσεις της κατηγορίας αυτής. Αν δεν οριστεί το limit-trace, ο picodb θα παρακολουθεί όλες τις κλήσεις των system calls των κατηγοριών που ορίστηκαν μέσω του CLI. |
| **go** | Εκτελεί το πρόγραμμα που δόθηκε στον picodb σαν όρισμα και το παρακολουθεί σύμφωνα με τις εντολές που έδωσε ο χρήστης στο CLI. Όταν τερματίσει το παρακολουθούμενο πρόγραμμα, θα τυπώσει τις πληροφορίες που ζήτησε ο χρήστης μέσω του CLI, πριν δώσει την εντολή *go.* |
| **quit** | Τερματίζει τον picodb |
| **help** | Τυπώνει όλες τις CLI εντολές που υποστηρίζει ο picodb στον χρήστη, μαζί με μία σύντομη περιγραφή. |

Σε όλες τις παραπάνω εντολές θα δίνεται μήνυμα λάθους σε περίπτωση λάθους τιμής σε οποιοδήποτε όρισμα.
Όλες οι παραπάνω εντολές θα πρέπει να μπορούν να συντμηθούν μόνο στο πρώτο τους γράμμα (το οποίο είναι μοναδικό). Για παράδειγμα η εντολή

```
trace all
```

να μπορεί να δοθεί και σαν

t all

Οι κατηγορίες των system calls είναι οι εξής:

**Process Control**
- execve (SYS_execve)
- fork (SYS_fork)
- wait4 (SYS_wait4)
- kill (SYS_kill)

**File Management**
- open (SYS_open)
- close (SYS_close)
- read (SYS_read)
- write (SYS_write)

Οι αιτήσεις ανακατεύθυνσης (redirect) θα πρέπει να υλοποιηθούν μέσω pipes και forked processes, έτσι ώστε να μην μπλοκάρει η διεργασία του picodb. Δηλαδή, θα πρέπει να γίνει fork μία διεργασία για κάθε redirected file handle, έτσι ώστε είτε να γράφει είτε να διαβάζει το ζητούμενο αρχείο <filename> και να το ανακατευθύνει από/στο αντίστοιχο pipe.

Επιπλέον, θα πρέπει να διαχειριστείτε τα σήματα *SIGHUP*, *SIGINT* και *SIGTERM*, μεταφέροντάς τα στο παρακολουθούμενο πρόγραμμα.
Η υλοποίηση του *picodb* θα πρέπει να γίνει με την βοήθεια του *ptrace* system call, το οποίο είναι το κύριο συστατικό των debuggers αλλά και προγραμμάτων παρακολούθησης συστημάτων.

Συνολικά, ο picodb ξεκινά με παράμετρο το όνομα ενός εκτελέσιμου και περιμένει εντολές από την είσοδο. Μόλις δοθεί η εντολή *go*, ρυθμίζει όλες τις απαραίτητες λειτουργίες του και εκκινεί το ζητούμενο πρόγραμμα-στόχο. Ενόσω το τελευταίο εκτελείται, τυπώνει στην έξοδο όλα τα syscalls που έχει ζητηθεί να παρακολουθούνται.

### Θέμα 2, Μονάδες 20

Σας δίνεται το πρόγραμμα readfile (σε πηγαίο κώδικα, στο τέλος της εκφώνησης) το οποίο δέχεται είσοδο από το τερματικό και υλοποιεί δύο μεθόδους ανάγνωσης αρχείου, μία με την συνάρτηση *read* και μια με την συνάρτηση *fread*.
Φτιάξτε ένα bash shell script, το οποίο να εκκινεί τον picodb με το πρόγραμμα readfile, έτσι ώστε να διαβαστεί με τους δύο τρόπους το αρχείο */etc/services*.
Δηλαδή, στο readfile θα δοθεί:

read /etc/services
fread /etc/services
quit me

Το script σας θα καταγράφει την έξοδο του readfile και θα επιβεβαιώνει ότι διαβάστηκε ο σωστός αριθμός από χαρακτήρες.
Το script σας επίσης θα καταγράφει όλη την έξοδο του picodb και στο τέλος θα μετρήσει τα syscalls που χρειάστηκαν για να διαβαστεί το αρχείο με τον κάθε έναν από τους δύο τρόπους ανάγνωσης. Τελικά θα δώσει τα εξής στατιστικά (με bold είναι το δείγμα το οποίο εσείς θα αντικαταστήσετε με τα πραγματικά μεγέθη):

| | read method | fread method | gain % |
|---|---|---|---|
| open | **1** | **1** | **0** |
| read | **1000** | **100** | **90** |
| close | **1** | **1** | **0** |

Όπου “gain %” = (read – fread) * 100 / read

### Θέμα 3 (Extra credit), Μονάδες 10

Τροποποιήστε το πρόγραμμα picodb έτσι ώστε να κρατάει ένα αρχείο log στο οποίο θα καταγράφει πληροφορίες σχετικά με τα system calls διαχείρισης αρχείων που πραγματοποιούν τα προγράμματα που δέχεται σαν όρισμα. Πιο συγκεκριμένα, για κάθε πρόγραμμα που αναλαμβάνει να παρακολουθήσει το picodb, μετά το πέρας της εκτέλεσής του, θα προσαρτά στο αρχείο log μία πλειάδα (γραμμή) της παρακάτω μορφής:

<program_name> <args>[] #open #read #close #write total_execution_time

Το πρώτο κομμάτι της γραμμής είναι το όνομα του προγράμματος που εκτελέστηκε ακολουθούμενο από έναν πίνακα που περιγράφει τα ορίσματα του. Στην συνέχεια, ακολουθεί μια καταγραφή του αριθμού των file management system calls που πραγματοποίησε το πρόγραμμα <program_name>, όπως ο αριθμός των αρχείων που άνοιξε (#open), ο αριθμός των κλήσεων συστήματος ανάγνωσης (#read) κλπ. Το τελευταίο κομμάτι μιας πλειάδας του αρχείου log είναι ο συνολικός χρόνος εκτέλεσης του υπό παρακολούθηση προγράμματος. Η έξοδος του picodb ουσιαστικά μας δίνει πληροφορίες σχετικά με το “κόστος” ανάγνωσης και επεξεργασίας ενός αρχείου ακολουθώντας διάφορες στρατηγικές, π.χ ανάλογα με το αν χρησιμοποιούμε την read ή την fread και ανάλογα με το μέγεθος του buffer. Συνεπώς μπορούμε να αναπτύξουμε ένα περιβάλλον σύγκρισης ανάμεσα στους δύο αυτούς τρόπους διαχείρισης αρχείων και να εξάγουμε συμπεράσματα για την συμπεριφορά τους.

Τροποποιήστε το πρόγραμμα readfile έτσι ώστε να δέχεται σαν όρισμα το μέγεθος του buffer που θα χρησιμοποιήσει.

Στα πλαίσια αυτής της εργασίας, θα δίνουμε σαν είσοδο στο picodb το πρόγραμμα readfile με διάφορες τιμές για την παράμετρο buffer size και χρησιμοποιώντας από την μία την τριάδα open/read/close και από την άλλη την fopen/fread/fclose για την ανάγνωση αρχείων. Τα στατιστικά που θα παράγει το picodb στο log αρχείο του θα αναλαμβάνει να τα επεξεργαστεί και να τα παρουσιάσει στον χρήστη ένα bash script. Το script αυτό θα συγκεντρώνει από το αρχείο log όλες τις πλειάδες που αφορούν ένα συγκεκριμένο όνομα αρχείου (<filename>), το οποίο θα το λαμβάνει σαν παράμετρο,

| Buffer Size | Read | Fread | Gain % |
|---|---|---|---|
| .... | .... | .... | .... |
| .... | .... | .... | .... |
| .... | .... | .... | .... |

Πίνακας 1: Χρόνος εκτέλεσης και το gain του προγράμματος readfile για το ίδιο αρχείο εισόδου ανάλογα με το buffer size και την στρατηγική ανάγνωσης (read/fread).

| Mean Number of | Read | Fread |
|---|---|---|
| I/O Syscalls | | |

Πίνακας 2: Μέσος αριθμός των system calls διαχείρισης αρχείου που πραγματοποιήθηκαν από το readfile πρόγραμμα για όλα τις διαθέσιμες μετρήσεις.

| Cumulative Statistics |
|---|

| Mean Execution Time | Best Execution time | Standard Deviation | Optimal Buffer Size |
|---|---|---|---|
| | | | |

Πίνακας 3: Συγκεντρωτικά στατιστικά όπως ο μέσος χρόνος εκτέλεσης, η τυπική απόκλιση κλπ.

## Διαδικαστικά

**Επισημάνσεις/Παραδοχές:**

- Η υλοποίηση του *picodb* θα πρέπει να γίνει με την χρήση του *ptrace* system call.
- Το πρόγραμμα σας θα πρέπει να τρέχει στα linux της σχολής (άρα σε 32 bit αρχιτεκτονική). Αν το αναπτύξετε σε 64-bit υπολογιστή, χρησιμοποιείστε ανάλογα τις επισημάνσεις στο υπόβαθρο της ptrace για να υποστηρίξετε και τις δύο αρχιτεκτονικές.

**Τι θα βαθμολογηθεί:**

1. Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης
2. Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα
3. Η χρήση Makefile και η κομματιαστή σύμβολο-μετάφραση (separate compilation).
4. Η αναφορά που θα γράψετε και θα υποβάλετε μαζί με τον πηγαίο κώδικα σε μορφή PDF.

**Άλλες σημαντικές παρατηρήσεις:**

1. Οι εργασίες είναι **ατομικές.**
2. Όποιος υποβάλλει / δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο **μηδενίζεται** στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που **δεν επιτρέπεται** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα **απλά παίρνει μηδέν** στο μάθημα. Αυτό ισχύει για **όλους όσους εμπλέκονται**, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
4. Το πρόγραμμά σας θα πρέπει να γραφτεί σε C (ή C++ **χωρίς όμως** STL extensions) και θα πρέπει να τρέχει σε Ubuntu-Linux, αλλιώς **δεν θα βαθμολογηθεί**.
5. Σε καμία περίπτωση τα MS-Windows **δεν είναι επιλογή** πλατφόρμας για την παρουσίαση αυτής της άσκησης.

Πρόγραμμα readfile.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

#define BUFFER_SIZE 1
#define MAX_TOKEN_LEN 128

size_t do_read(char *filename) {
  char buffer[BUFFER_SIZE];
  size_t total_read = 0, chars_read;
  int fh = open(filename, O_RDONLY);
  if( fh >= 0 ) {
    while( (chars_read = read(fh, buffer, BUFFER_SIZE)) > 0 )
      total_read += chars_read;
    close(fh);
  } else {
```

```
    total_read = -1;
  }
  return total_read;
}

size_t do_fread(char *filename) {
  char buffer[BUFFER_SIZE];
  size_t total_read = 0, chars_read;
  FILE *fp = fopen(filename, "rb");
  if( fp != NULL ) {
    while( (chars_read = fread(buffer, 1, BUFFER_SIZE, fp)) > 0 )
      total_read += chars_read;
    fclose(fp);
  } else {
    total_read = -1;
  }
  return total_read;
}

int main() {
  char command[MAX_TOKEN_LEN];
  char parameter[MAX_TOKEN_LEN];
  size_t ret;
  while(1) {
    printf(">");
    scanf("%s %s", command, parameter);
    if( !strncmp(command, "read", MAX_TOKEN_LEN) ) {
      ret = do_read(parameter);
      printf("==>READ %zu\n", ret);
    } else if( !strncmp(command, "fread", MAX_TOKEN_LEN) ) {
      ret = do_fread(parameter);
      printf("==>FREAD %zu\n", ret);
    } else if( !strncmp(command, "quit", MAX_TOKEN_LEN) ) {
      printf("Bye!\n");
      return 0;
    } else {
      printf("Unknown command. Possible commands are:\n"
        "read <file>\n"
        "fread <file>\n"
        "quit me\n");
    }
  }
}
```