

Άσκηση 1

Κλήσεις Συστήματος, Διαχείριση Διεργασιών και
Διαδιεργασιακή Επικοινωνία

Λειτουργικά Συστήματα - 6ο εξάμηνο

ακαδημαϊκό έτος 2023 - 2024

Εργαστήριο Υπολογιστικών Συστημάτων



1η Άσκηση - Σύνοψη

Πρόγραμμα μέτρησης εμφανίσεων ενός χαρακτήρα σε αρχείο εισόδου.

1. Ανάγνωση και εγγραφή αρχείων
 - από τη χρήση της `libc` σε κλήσεις συστήματος
2. Δημιουργία διεργασιών
 - με `fork()` / `wait()`
3. Διαδιεργασιακή επικοινωνία
 - με σήματα
 - με σωληνώσεις
4. Εφαρμογή παράλληλης καταμέτρησης χαρακτήρων

Θεωρία - Μηχανισμοί

- Χρήση του `make`, `Makefile`
- Χρήση κλήσεων συστήματος για ανάγνωση/εγγραφή αρχείων
 - `open()`, `read()`, `write()`, `close()`
- Διαχείριση διεργασιών στο UNIX
 - `fork()`, `wait()`
- Σήματα στο UNIX
 - `kill()`, `signal handlers`
- Διαδιεργασιακή επικοινωνία με UNIX pipes

1η Άσκηση - Σύνοψη

Πρόγραμμα μέτρησης εμφανίσεων ενός χαρακτήρα σε αρχείο εισόδου.

1. **Ανάγνωση και εγγραφή αρχείων**
 - από τη χρήση της `libC` σε κλήσεις συστήματος
2. Δημιουργία διεργασιών
 - με `fork()` / `wait()`
3. Διαδιεργασιακή επικοινωνία
 - με σήματα
 - με σωληνώσεις
4. Εφαρμογή παράλληλης καταμέτρησης χαρακτήρων

Αρχεία

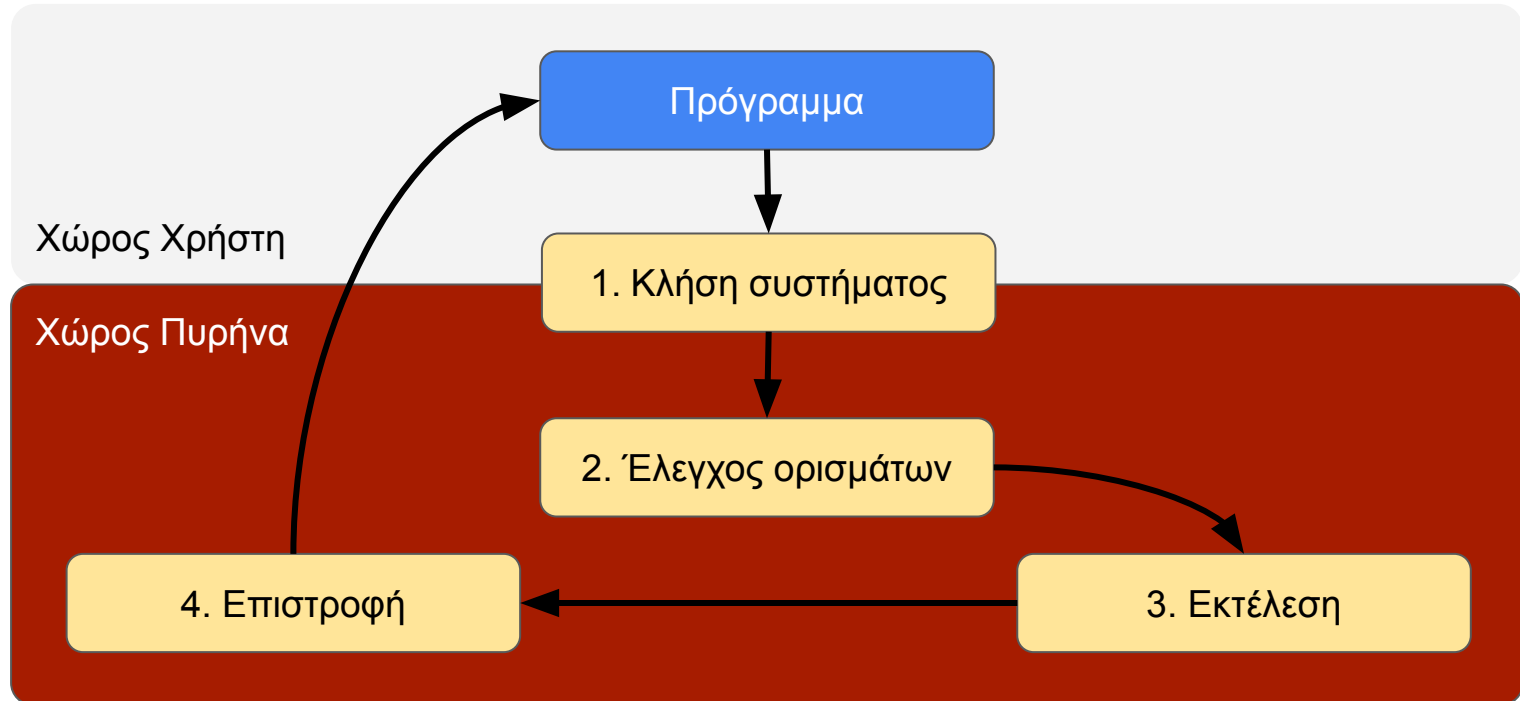
Τα αρχεία έχουν τα εξής χαρακτηριστικά:

- όνομα / μονοπάτι
(π.χ. `/home/oslab/oslabXX/file.txt`)
- δεδομένα / περιεχόμενα
(π.χ. `int main() ...`)
- μετα-δεδομένα
(π.χ. ημερομηνία τελευταίας πρόσβασης, μέγεθος)
- είναι persistent
παραμένουν μετά το κλείσιμο του υπολογιστή

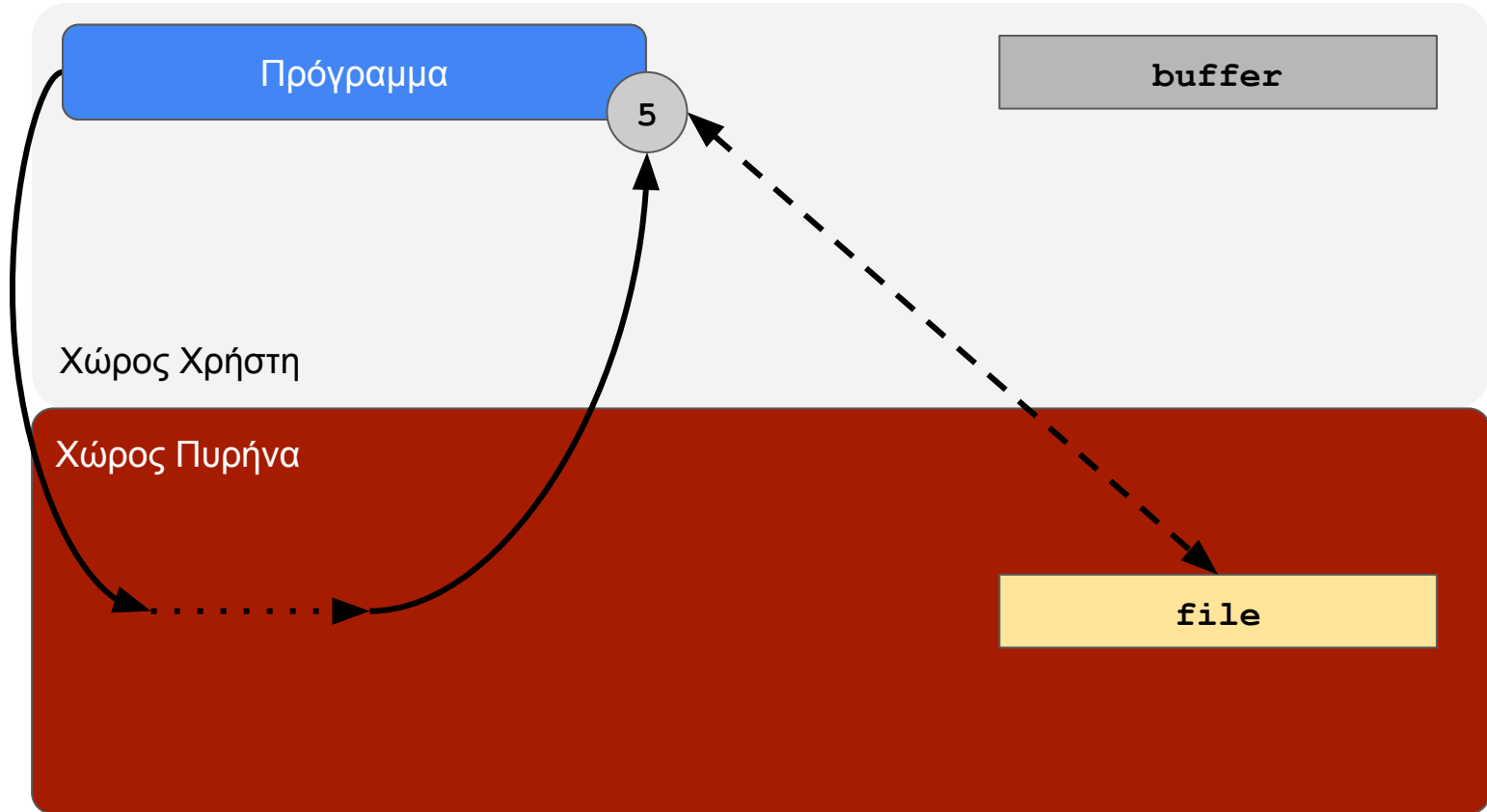
Για την πρόσβαση στα αρχεία χρησιμοποιούμε *κλήσεις συστήματος (system calls)*

Κλήσεις συστήματος (system calls)

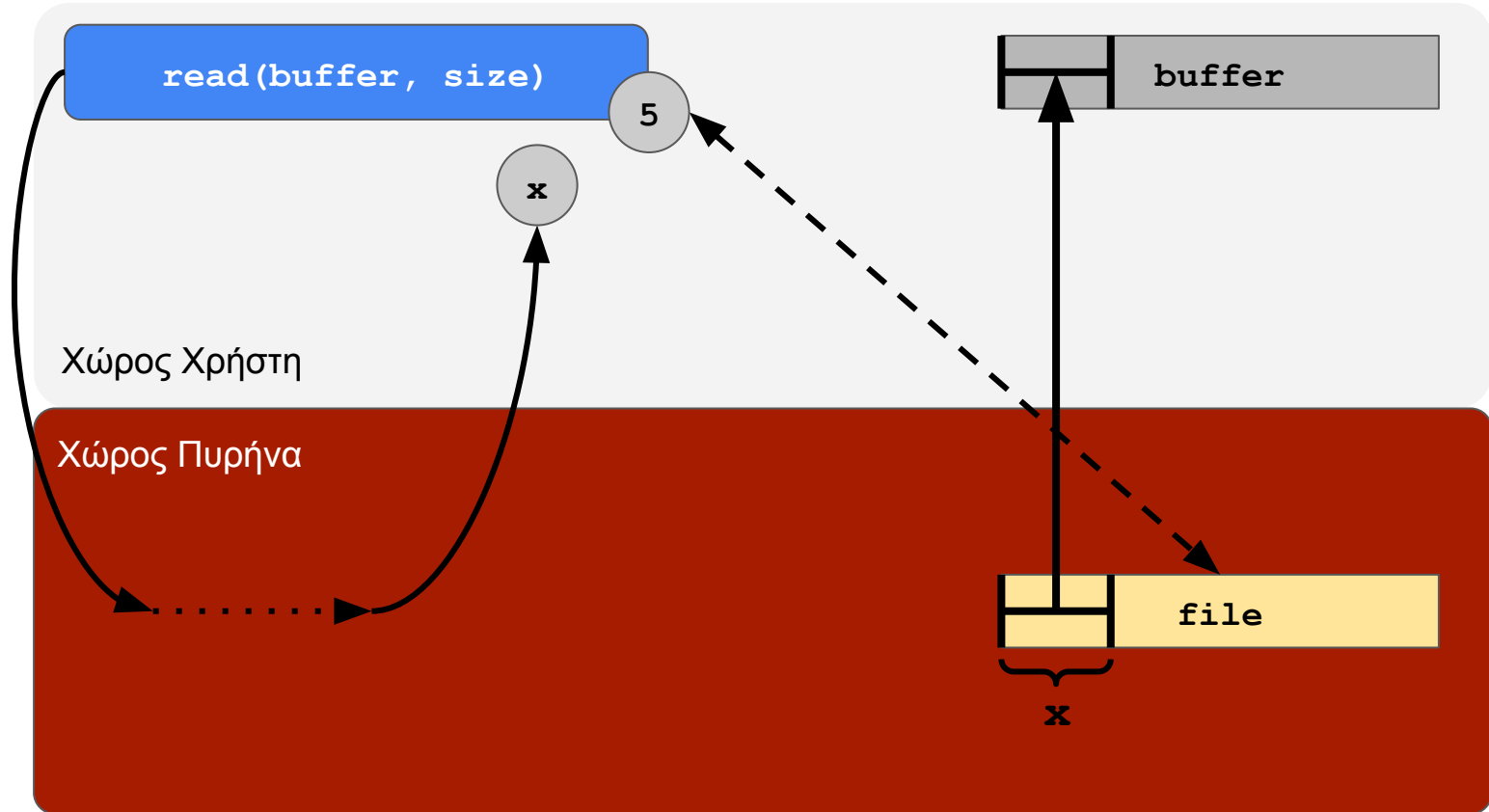
Προγραμματιστική διεπαφή για τις υπηρεσίες που προσφέρει το ΛΣ στις εφαρμογές.



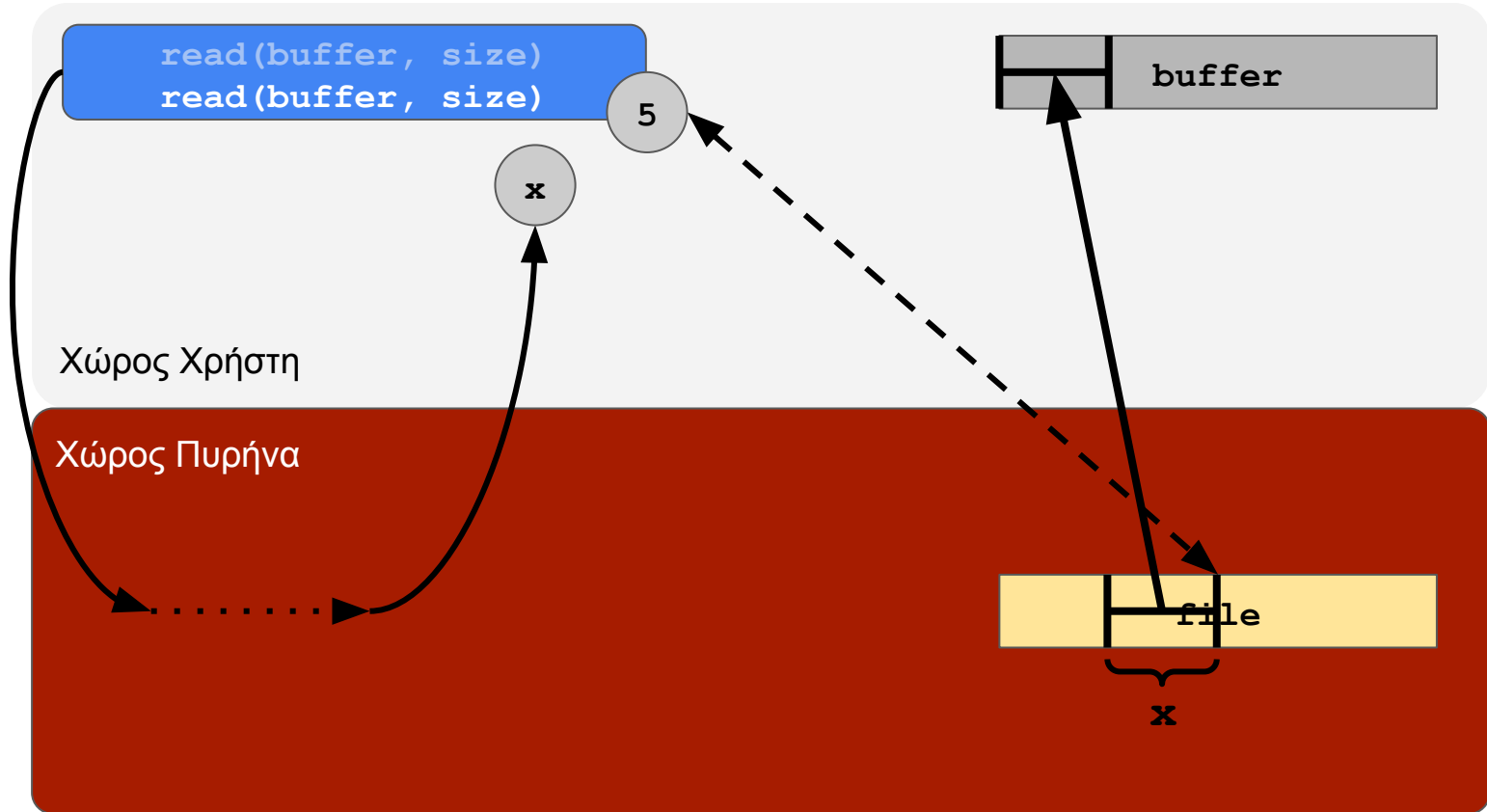
Βασικές κλήσεις Ε/Ε (`open`, `read`, `close`)



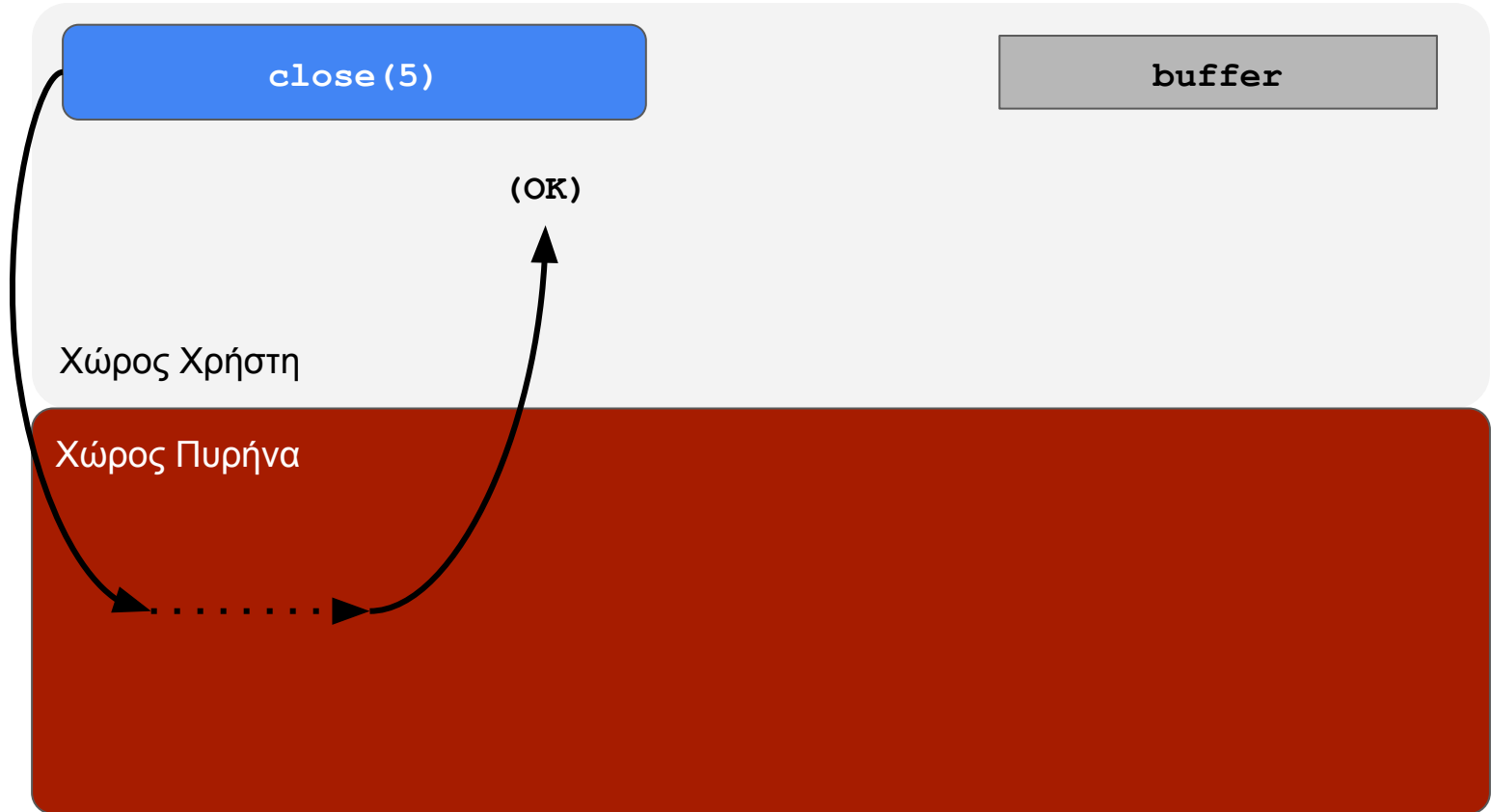
Βασικές κλήσεις Ε/Ε (open, **read**, close)



Βασικές κλήσεις Ε/Ε (open, **read**, close)



Βασικές κλήσεις Ε/Ε (open, read, **close**)



Βασικές κλήσεις Ε/Ε

Ανάγνωση:

- Άνοιγμα `open`
- Ανάγνωση σε μνήμη `read`
- Κλείσιμο `close`

Εγγραφή:

- Άνοιγμα `open`
- Εγγραφή από μνήμη `write`
- Κλείσιμο `close`

Από (εγγραφή) και προς (ανάγνωση) τη μνήμη με εξαιρέσεις (π.χ. `sendfile()`)

File descriptor: τιμή επιστροφής `open`, αναγνωριστικό για τις `read`, `write`, `close`

Διαθέσιμα αναγνωριστικά στην έναρξη του προγράμματος:

0: είσοδος (`stdin`)

1: έξοδος (`stdout`)

2: έξοδος σφαλμάτων (`stderr`)

Παράδειγμα:

read

read.c

```
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    char buff[1024];
    ssize_t rcnt;
    for (;;) {
        rcnt = read(0, buff, sizeof(buff) - 1);
        if (rcnt == 0)    /* end-of-file */
            return 0;
        if (rcnt == -1) { /* error */
            perror("read");
            return 1;
        }
        buff[rcnt] = '\0';
        fprintf(stdout, "%s", buff);
    }
}
```

Παράδειγμα: write

write.c

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv){
    char buff[1024];
    size_t len, idx;
    ssize_t wcnt;
    for (;;) {
        if (fgets(buff, sizeof(buff), stdin) == NULL)
            return 0;
        idx = 0;
        len = strlen(buff);
        do {
            wcnt = write(1, buff + idx, len - idx);
            if (wcnt == -1) { /* error */
                perror("write");
                return 1;
            }
            idx += wcnt;
        } while (idx < len);
    }
}
```

Παράδειγμα:

open

(ανάγνωση)

open-read.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd;
    fd = open("filename", O_RDONLY);
    if (fd == -1) {
        perror("open");
        exit(1);
    }

    // perform read(...)

    close(fd);
    return 0;
}
```

Παράδειγμα:

open (εγγραφή)

open-write.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
int
{
    - oflags:
        - O_CREAT : Δημιουργία αρχείου αν δεν υπάρχει.
        - O_WRONLY: Εγγραφή (μόνο)
        - O_TRUNC  : Μηδενισμός αρχείου αν υπάρχει.
    - mode:
        - S_IRUSR : δικαίωμα ανάγνωσης στον κάτοχο
        - S_IWUSR : δικαίωμα εγγραφής στον κάτοχο

    exit(1);
}

// perform write(...)

close(fd);
return 0;
}
```

strace

- Εφαρμογή
- Εκτέλεση προγράμματος που δίνεται ως όρισμα
- Καταγραφή των κλήσεων συστήματος που πραγματοποιούνται
- Χρήσιμη για εντοπισμό λαθών

Παράδειγμα: strace

Γραμμή εντολών

```
$ echo 'Hello World!' > hello
$ cat hello
Hello World!
$ strace cat hello
execve("/bin/cat", ["cat", "hello"], [/* 52 vars */]) = 0
...
open("hello", O_RDONLY)                = 3
read(3, "Hello World!\n", 32768)       = 13
write(1, "Hello World!\n", 13Hello World!
)                                       = 13
read(3, "", 32768)                     =0
...
```

Στοίβα (stack)

- Αυτόματη αύξηση μεγέθους
- Όχι για πολλά δεδομένα (8 MB)

```
void foo(double *);
```

stack.c

```
int main(int argc, char **argv)
{
    double matrix[1048576];
    foo(matrix);
    return 0;
}
```

```
sub    $0x800008,%rsp
mov    %rsp,%rdi
callq  f <main+0xf>
xor    %eax,%eax
add    $0x800008,%rsp
retq
```

stack.s

Σωρός (Heap) - malloc / free

```
#include <stdio.h>
#include <stdlib.h>
void foo(double *);

int main(int argc, char **argv)
{
    double *array;
    array = malloc(1048576*sizeof(double));
    if (!array){
        printf("error in allocation\n");
        return 1;
    }
    foo(array);
    free(array);
    return 0;
}
```

malloc.c

1η Άσκηση - Σύνοψη

Πρόγραμμα μέτρησης εμφανίσεων ενός χαρακτήρα σε αρχείο εισόδου.

1. Ανάγνωση και εγγραφή αρχείων
 - από τη χρήση της `libc` σε κλήσεις συστήματος
2. Δημιουργία διεργασιών
 - με `fork()` / `wait()`
3. Διαδιεργασιακή επικοινωνία
 - με σήματα
 - με σωληνώσεις
4. Εφαρμογή παράλληλης καταμέτρησης χαρακτήρων

Δημιουργία διεργασίας στο UNIX: `fork()`

```
p = ?          mypid = -1
```

```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```

PID = 981

Δημιουργία διεργασίας στο UNIX: `fork()`

```
p = ?          mypid = -1
```

```
pid_t p, mypid;  
mypid = -1;
```

```
...
```

```
p = fork();
```

-1 error,

errno π.χ. OOM

```
→ if (p < 0) {  
    perror("fork");  
    exit(1);
```

```
} else if (p == 0) {  
    mypid = getpid();  
    child();
```

```
} else {  
    mypid = getpid();  
    father();
```

```
}
```

PID = 981

Δημιουργία διεργασίας στο UNIX: fork ()

p = ?

mypid = -1

```
pid_t p, mypid;
mypid = -1;
...
→ p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```

PID = 981

p = ?

mypid = -1

```
pid_t p, mypid;
mypid = -1;
...
→ p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```

PID = 987

Δημιουργία διεργασίας στο UNIX: `fork()`

p = 987

`mypid = -1`

```
pid_t p, mypid;  
mypid = -1;
```

...

→ `p = fork();` **child's PID = 987**

```
if (p < 0) {  
    perror("fork");  
    exit(1);  
} else if (p == 0) {  
    mypid = getpid();  
    child();  
} else {  
    mypid = getpid();  
    father();  
}
```

PID = 981

p = 0

`mypid = -1`

```
pid_t p, mypid;  
mypid = -1;
```

...

→ `p = fork();` **p = 0**

```
if (p < 0) {  
    perror("fork");  
    exit(1);  
} else if (p == 0) {  
    mypid = getpid();  
    child();  
} else {  
    mypid = getpid();  
    father();  
}
```


PID = 987

Δημιουργία διεργασίας στο UNIX: `fork()`

p = 987

mypid = -1

```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```




PID = 981

p = 0

mypid = -1

```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```



PID = 987

Δημιουργία διεργασίας στο UNIX: `fork()`

p = 987

mypid = 981

```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```

PID = 981

p = 0

mypid = 987


```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```

PID = 987

Δημιουργία διεργασίας στο UNIX: `fork()`

p = 987 **mypid = 981**


```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```



PID = 981

p = 0 **mypid = 987**

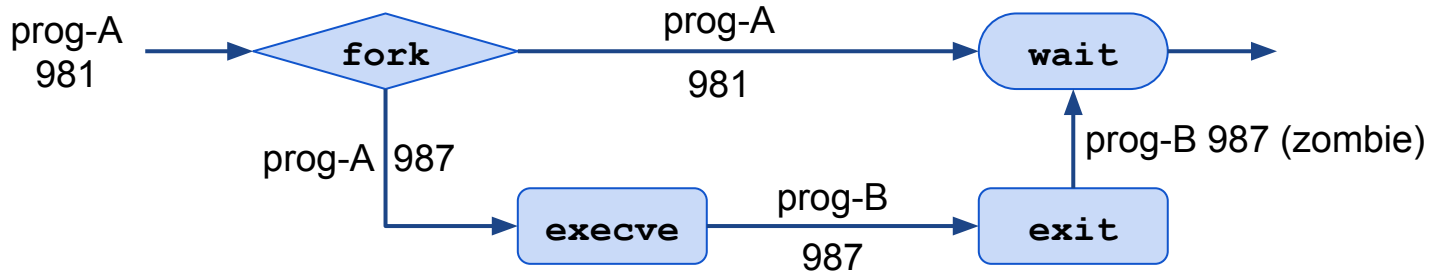
```
pid_t p, mypid;
mypid = -1;
...
p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    mypid = getpid();
    child();
} else {
    mypid = getpid();
    father();
}
```



PID = 987

Δημιουργία διεργασίας στο UNIX: `fork()`

- Όλες οι διεργασίες προκύπτουν με `fork()` (σχεδόν όλες)
 - Ίδιο πρόγραμμα με γονική διεργασία
 - Αντίγραφο χώρου μνήμης
 - Κληρονομεί ανοιχτά αρχεία, συνδέσεις και δικαιώματα πρόσβασης
- Αντικατάσταση προγράμματος διεργασίας: `execve()`
- Η γονική διεργασία ενημερώνεται για το θάνατο του παιδιού με `wait()` > συλλογή τιμής τερματισμού (exit status)
 - Μέχρι τότε, ένα παιδί που έχει καλέσει `exit()` λέγεται zombie
 - Αν ο γονέας πεθάνει πρώτα, η διεργασία γίνεται παιδί της `init` (PID = 1, κάνει συνεχώς `wait()`)



`wait()` / `waitpid()`

- Για κάθε `fork()` πρέπει να γίνει ένα `wait()`
- `wait(&status)`
 - Μπλοκάρει έως οποιοδήποτε παιδί πεθάνει
- Το `status` κωδικοποιεί πώς πέθανε η διεργασία
 - Κανονικά (`exit()`)
 - Λόγω κάποιου σήματος (`SIGTERM`, `SIGKILL`)
- Χρήσιμες μακροεντολές για την ερμηνεία του `status`
 - `WIFEXITED()`, `WEXITSTATUS()`, `WIFSIGNALED()`, `WTERMSIG()`
- Μια πιο ευέλικτη `wait()`: `waitpid()`
 - Περιμένει για αλλαγή κατάστασης συγκεκριμένου ή οποιουδήποτε PID διεργασίας - παιδιού
 - Συμπεριφορά ελεγχόμενη από flags (`WNOHANG`, `WUNTRACED`)

Βοηθητική συνάρτηση: `explain_wait_status()`

```
void explain_wait_status(pid_t pid, int status) {
    if (WIFEXITED(status))
        fprintf(stderr, "Child with PID = %ld terminated normally, exit
                        status = %d\n", (long)pid, WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        fprintf(stderr, "Child with PID = %ld was terminated by a signal,
                        signo = %d\n", (long)pid, WTERMSIG(status));
    else if (WIFSTOPPED(status))
        fprintf(stderr, "Child with PID = %ld has been stopped by a signal,
                        signo = %d\n", (long)pid, WSTOPSIG(status));
    else {
        fprintf(stderr, "%s: Internal error: Unhandled case, PID = %ld,
                        status = %d\n", __func__, (long)pid, status);

        exit(1);
    }
    fflush(stderr);
}
```

Βοηθητική συνάρτηση: `explain_wait_status()`

```
void explain_wait_status(pid_t pid, int status) {
    if (WIFEXITED(status))
        fprintf(stderr, "Child with PID = %ld terminated normally, exit
                                (status));

    else if (WIFSIGNALED(status))
        fprintf(stderr, "Child with PID = %ld terminated by a signal,
                                (us));
        explain_wait_status(pid, status);

    else if (WIFEXITED(status) || WIFSIGNALED(status))
        fprintf(stderr, "Child with PID = %ld terminated by a signal,
                                (us));

    else {
        fprintf(stderr, "%s: Internal error: Unhandled case, PID = %ld,
                                status = %d\n", __func__, (long)pid, status);

        exit(1);
    }

    fflush(stderr);
}
```

Παράδειγμα:

explain_wait_status(pid, status);

Παράδειγμα:

`fork()`

`wait()`

```
void child(void) {
    compute(...);
}

int main(void) {
    pid_t p;
    int status;

    p = fork();
    if (p < 0) {
        perror("fork");
        exit(1);
    }
    if (p == 0) {
        child();
        exit(1);
    }

    p = wait(&status);
    explain_wait_status(p, status);
    return 0;
}
```


1η Άσκηση - Σύνοψη

Πρόγραμμα μέτρησης εμφανίσεων ενός χαρακτήρα σε αρχείο εισόδου.

1. Ανάγνωση και εγγραφή αρχείων
 - από τη χρήση της `libC` σε κλήσεις συστήματος
2. Δημιουργία διεργασιών
 - με `fork()` / `wait()`
3. Διαδιεργασιακή επικοινωνία
 - με σήματα
 - με σωληνώσεις
4. Εφαρμογή παράλληλης καταμέτρησης χαρακτήρων

Σήματα στο UNIX

Αποστολή: `kill()`, `raise()`

Παράδειγμα:

```
if (kill(pid, SIGUSR1) < 0) {  
    perror("kill");  
    exit(1);  
}
```

Χειρισμός: `signal()` με `SIG_IGN`, `SIG_DFL` ή handler

Παράδειγμα:

```
void sighandler(int signum) {  
    got_sigusr1 = 1;  
}  
  
if (signal(SIGUSR1, signal_handler) < 0 {  
    perror("could not establish SIGUSR1 handler");  
    exit(1);  
}
```

Σήματα στο UNIX

- Αναξιόπιστα
 - Τι θα γίνει αν έρθουν πολλά σήματα;
 - Η συνάρτηση χειρισμού θα τρέξει από 1 έως n φορές
 - Τι θα γίνει αν το σήμα έρθει ενώ η συνάρτηση χειρισμού εκτελείται;
- Race conditions: αυτό θα δουλέψει;

Παράδειγμα:

```
void sighandler(int signum) {  
    got_sigusr1 = 1;  
}  
  
...  
got_sigusr = 0;  
while (!got_sigusr1)  
    pause(); // Αναμονή έως ότου ληφθεί κάποιο σήμα
```

Σήματα στο UNIX

- Η `signal()` δεν είναι φορητή
- Ο handler ακυρώνεται όταν εκτελείται (System V)
 - και πρέπει να επανεγκατασταθεί
 - ή όχι... BSD. Στο Linux; εξαρτάται... libC vs. kernel
- Καλύτερη, φορητή λύση: **`sigaction()`**

Παράδειγμα:

```
struct sigaction sa;
sigset_t sigset;

sa.sa_handler = sigchld_handler;
sa.sa_flags = SA_RESTART;
sa.sa_mask = sigset;
if (sigaction(SIGCHLD, &sa, NULL) < 0) {
    perror("sigaction");
    exit(1);
}
```

Σήματα στο UNIX

Χρήσιμες εντολές: kill, ps, pstree, killall, grep

```
$ kill -l
```

```
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
```

```
...
```

```
$ ps -ef | grep oslab001 | grep bash
```

```
oslab001      4277      4276      0   12:17   pts/0    00:00:00   bash
```

```
$ kill -TERM 4277
```

```
$ kill -9 4277
```

```
$ killall -9 bash
```

Γραμμή εντολών

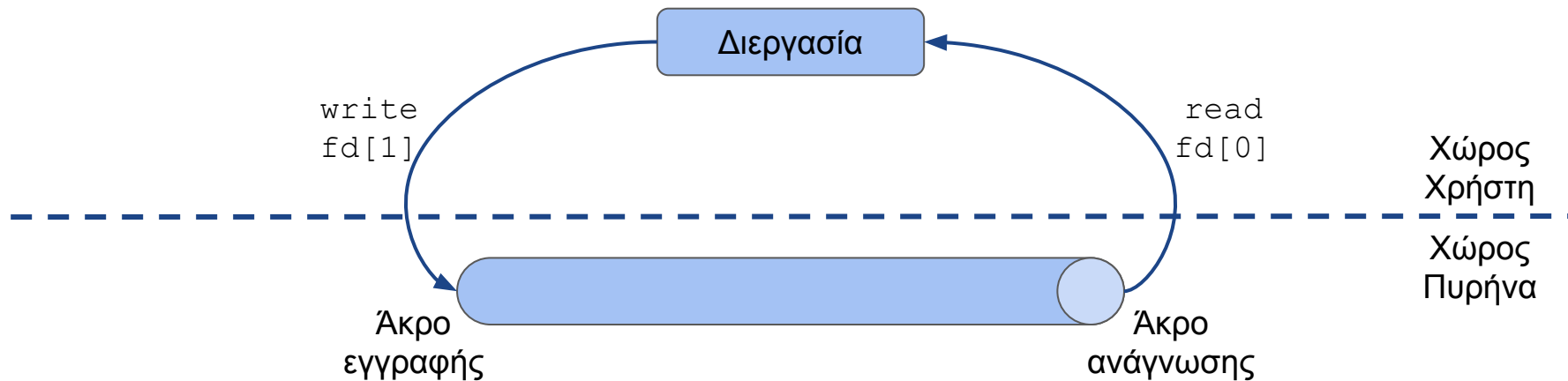
Όλες οι λεπτομέρειες στα man pages

1η Άσκηση - Σύνοψη

Πρόγραμμα μέτρησης εμφανίσεων ενός χαρακτήρα σε αρχείο εισόδου.

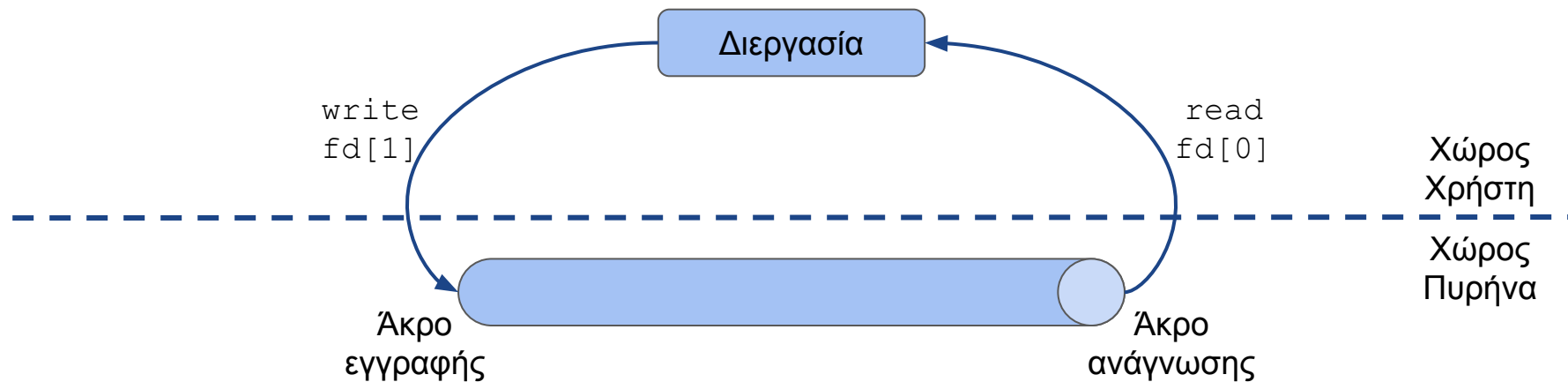
1. Ανάγνωση και εγγραφή αρχείων
 - από τη χρήση της `libC` σε κλήσεις συστήματος
2. Δημιουργία διεργασιών
 - με `fork()` / `wait()`
3. Διαδιεργασιακή επικοινωνία
 - με σήματα
 - με σωληνώσεις
4. Εφαρμογή παράλληλης καταμέτρησης χαρακτήρων

Σωληνώσεις στο UNIX



- Ένας από τους βασικότερους μηχανισμούς στο UNIX
- Μονόδρομη μεταφορά δεδομένων
- Από το άκρο εγγραφής στο άκρο ανάγνωσης
 - Δημιουργία με `pipe()`, επικοινωνία με `write()` και `read()`
 - Αν η σωλήνωση είναι άδεια: η `read()` μπλοκάρει

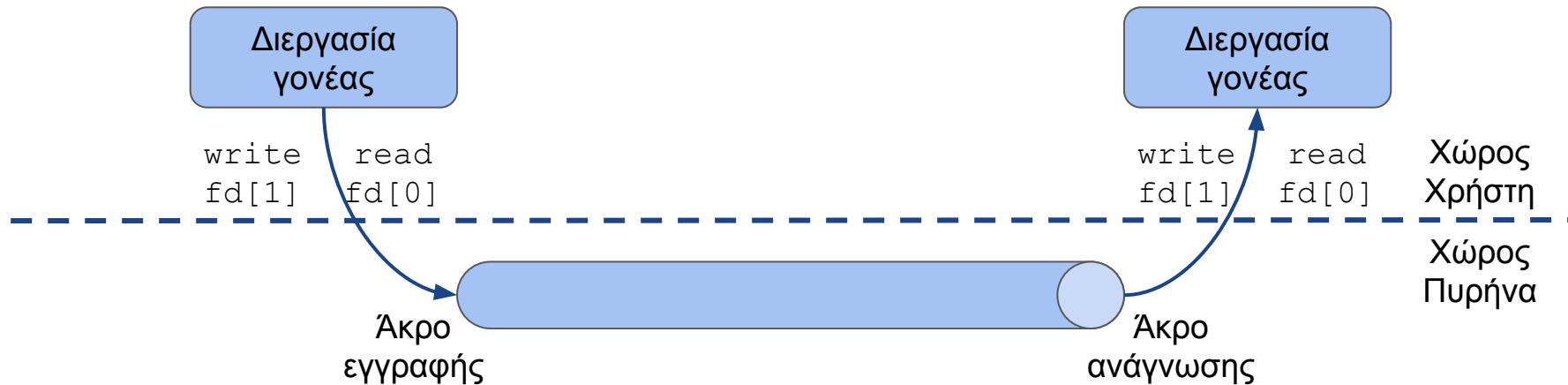
Σωληνώσεις στο UNIX



```
int fd[2];  
int num1, num2;
```

```
➡ pipe(fd);  
➡ write(fd[1], &num1, sizeof(num1));  
➡ read(fd[0], &num2, sizeof(num2));
```


Σωληνώσεις στο UNIX



- ➔ `pipe (fd) ;`
- ➔ `fork () ;`
- ➔ Ο γονέας κλείνει το άκρο ανάγνωσης
- ➔ Το παιδί κλείνει το άκρο εγγραφής

Παράδειγμα IPC με UNIX pipes

```
double value;
int pfd[2];
pid_t p;

if (pipe(pfd) < 0) {
    perror("pipe");
    exit(1);
}

p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    if (read(pfd[0], &value, sizeof(value)) != sizeof(value)) {
        perror("read from pipe");
    }
}
```

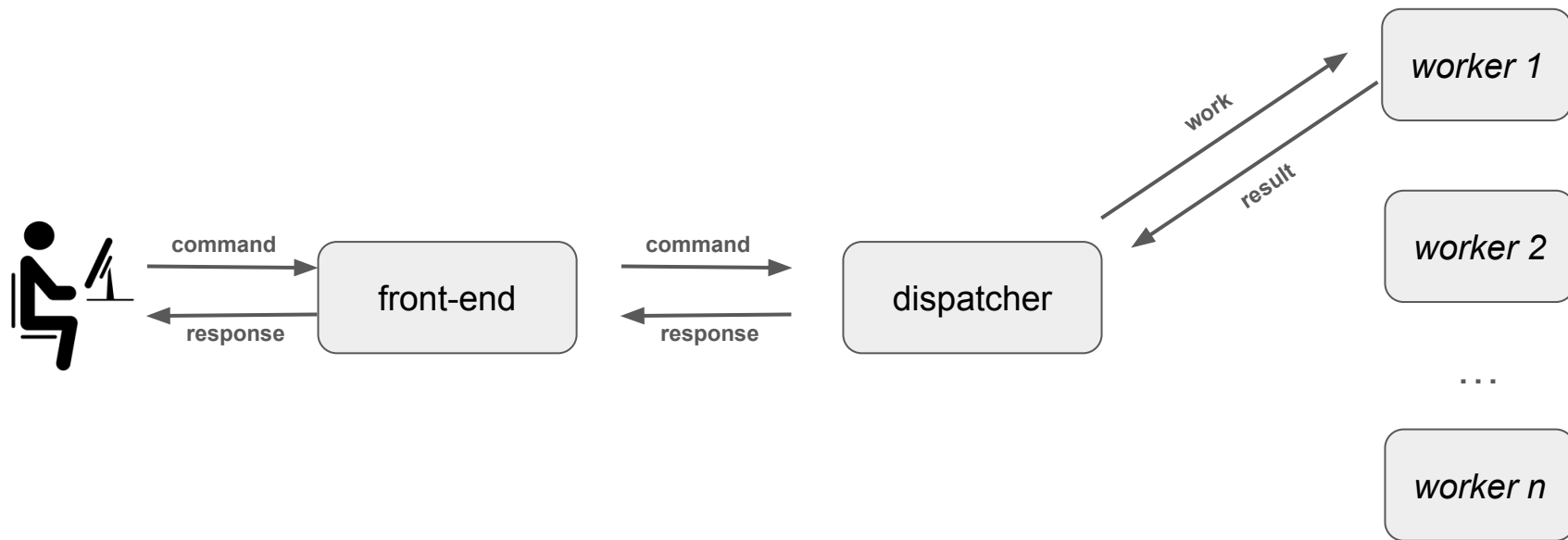
Παράδειγμα IPC με UNIX pipes

```
        exit(1);
    }
    p = fork();
    if (p < 0) {
        perror("fork");
        exit(1);
    } else if (p == 0) {
        if (read(pfd[0], &value, sizeof(value)) != sizeof(value)) {
            perror("read from pipe");
            exit(1);
        }
        printf("child received value: value = %f\n", value);
        exit(0);
    } else {
        compute_value(&value);
        if (write(pfd[1], &value, sizeof(value)) != sizeof(value)) {
```

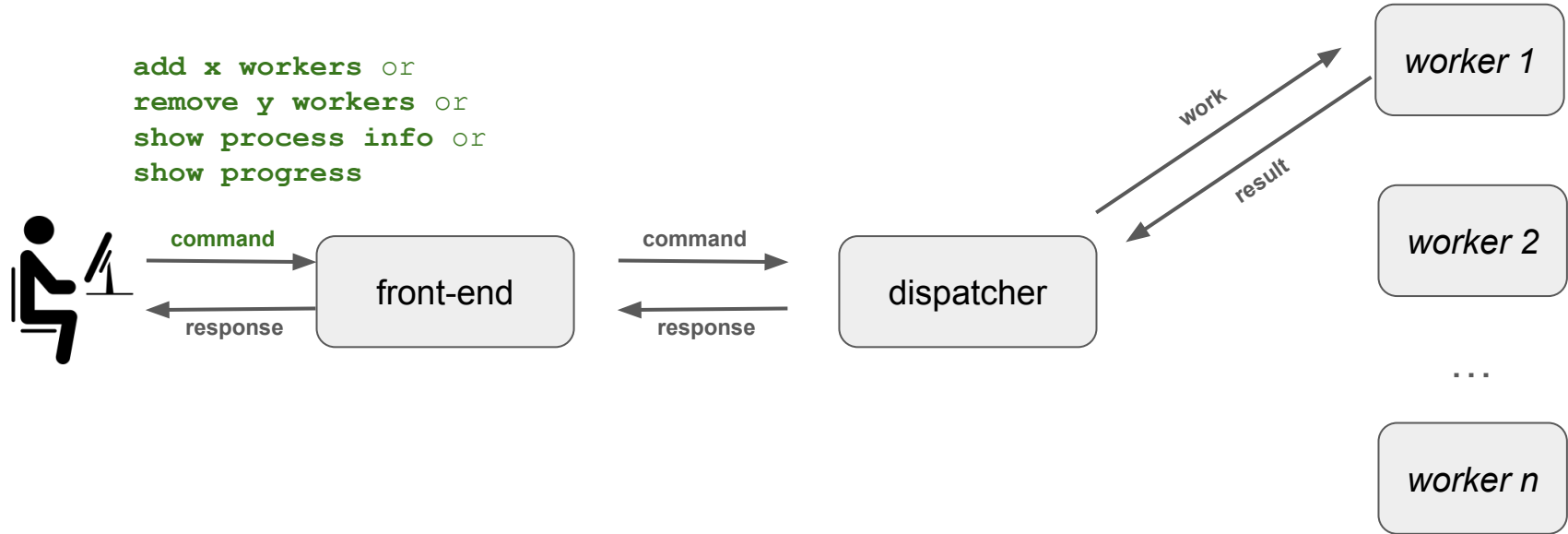
Παράδειγμα IPC με UNIX pipes

```
} else if (p == 0) {  
    if (read(pfd[0], &value, sizeof(value)) != sizeof(value)) {  
        perror("read from pipe");  
        exit(1);  
    }  
    printf("child received value: value = %f\n", value);  
    exit(0);  
} else {  
    compute_value(&value);  
    if (write(pfd[1], &value, sizeof(value)) != sizeof(value)) {  
        perror("write to pipe");  
        exit(1);  
    }  
    exit(0);  
}
```

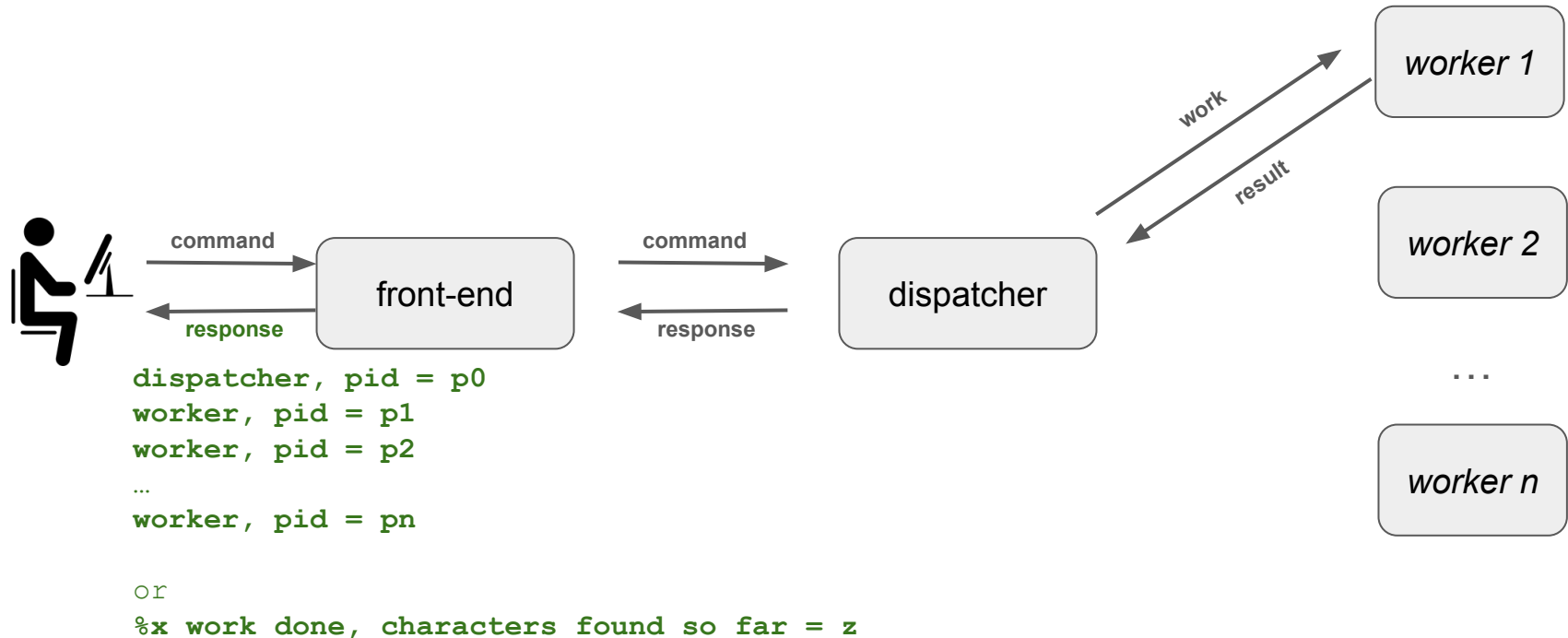
Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



Χρήσιμη συνάρτηση: show_pstree

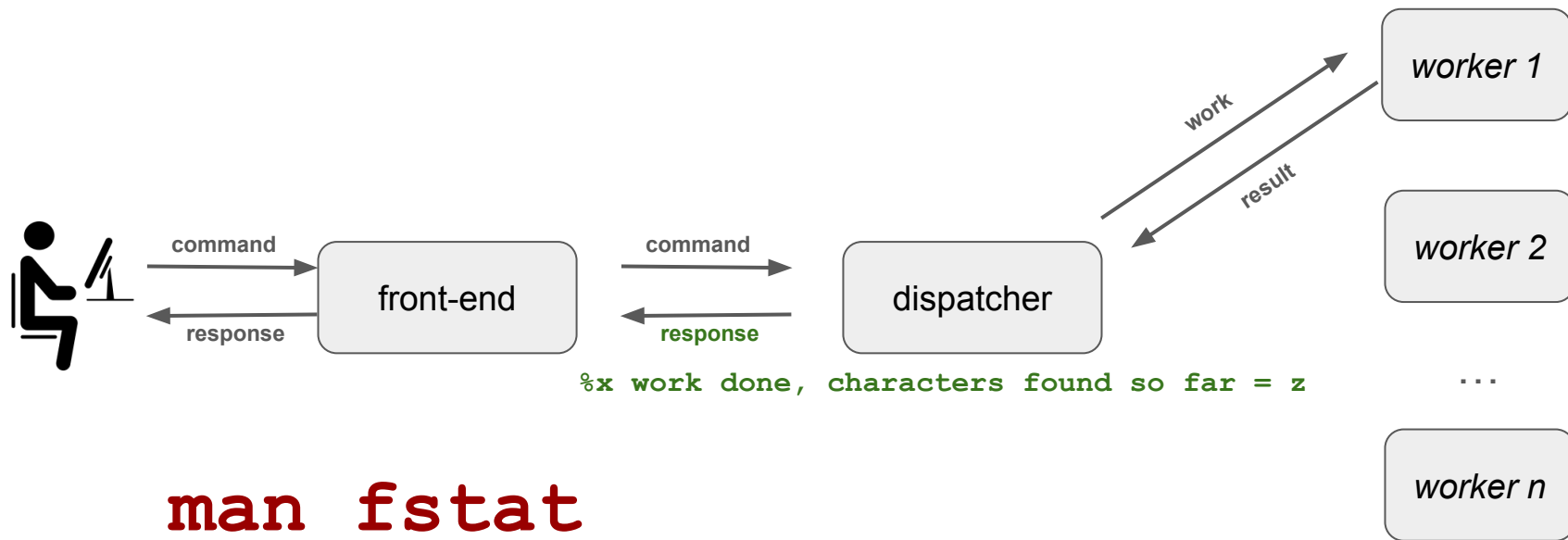
```
void show_pstree(pid_t p)
{
    int ret;
    char cmd[1024];

    snprintf(cmd, sizeof(cmd), "echo; echo; pstree -a -G -c -p %ld; echo; echo",
        (long)p);
    cmd[sizeof(cmd)-1] = '\0';
    ret = system(cmd);
    if (ret < 0) {
        perror("system");
        exit(104);
    }
}
```

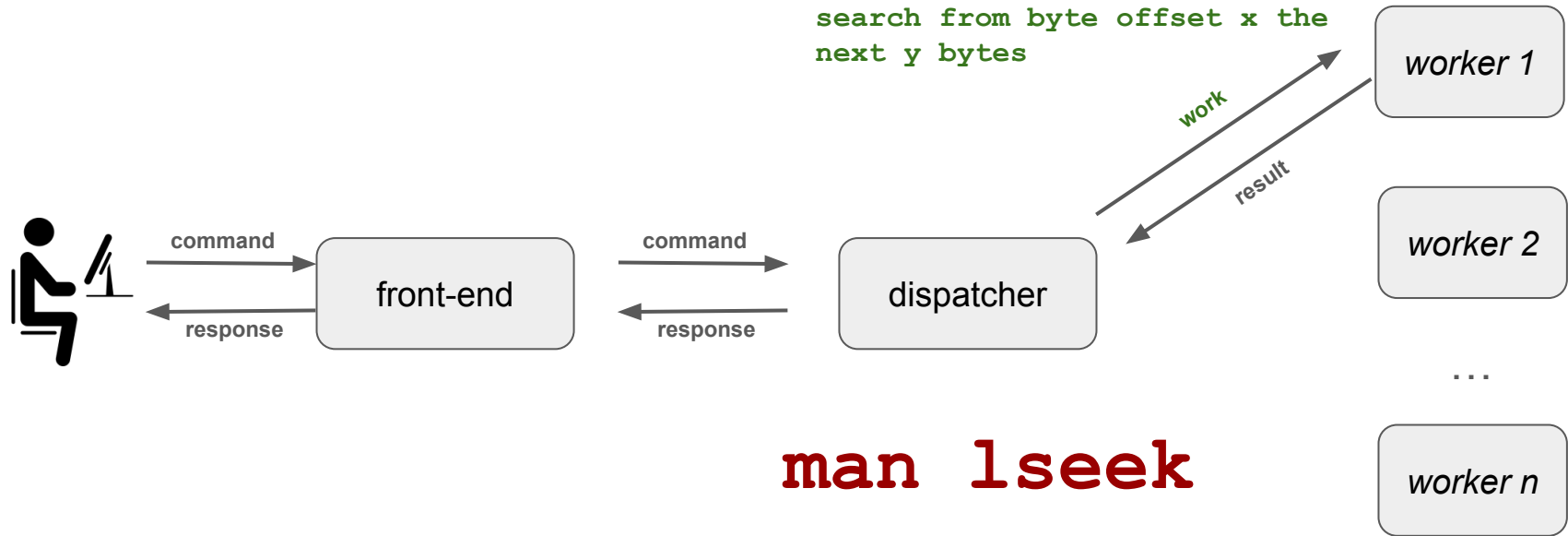

Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



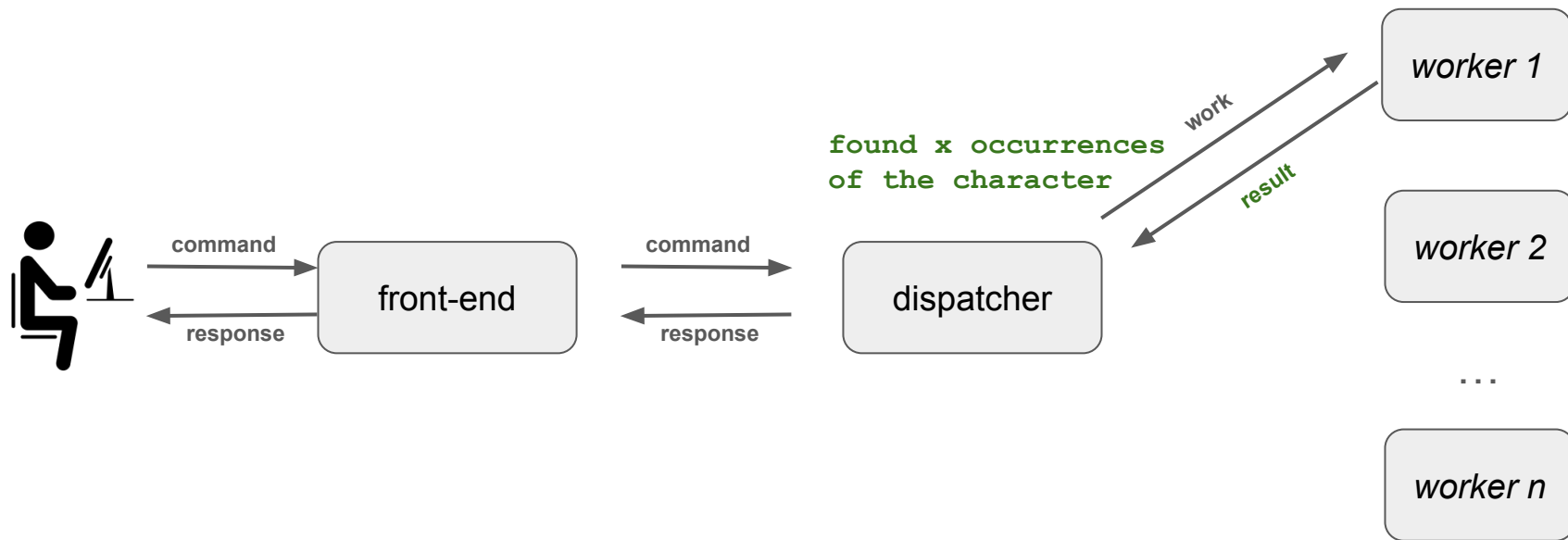
Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



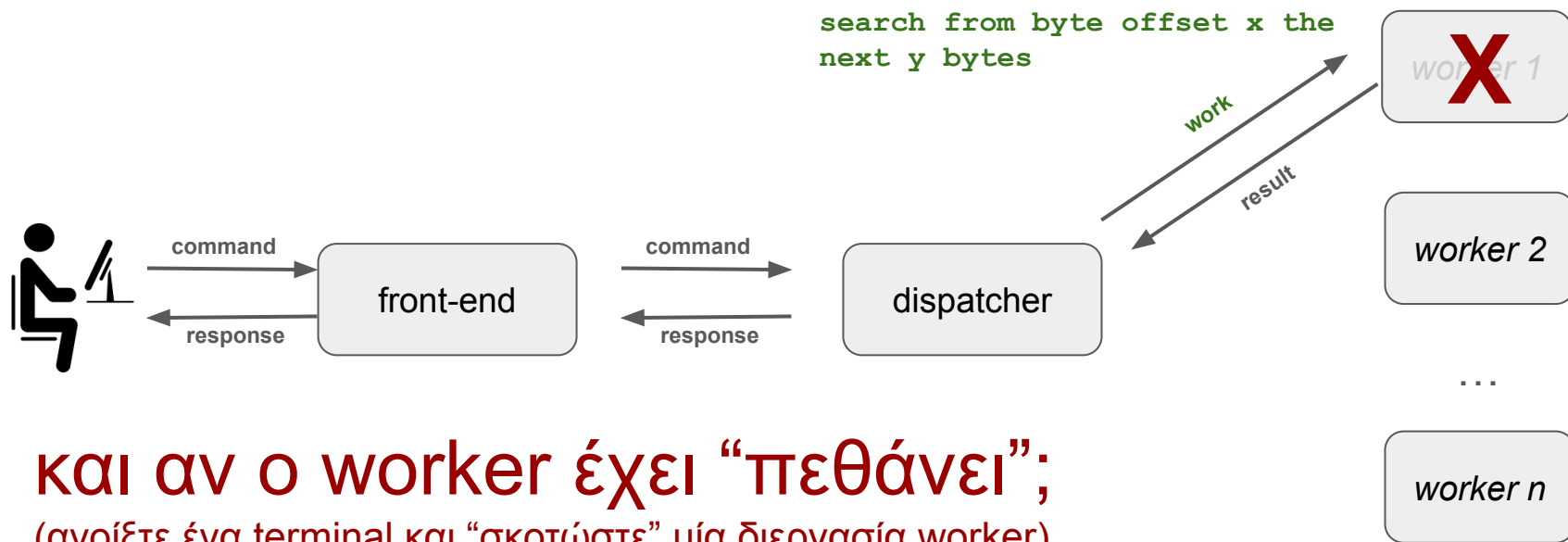
Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο



Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο

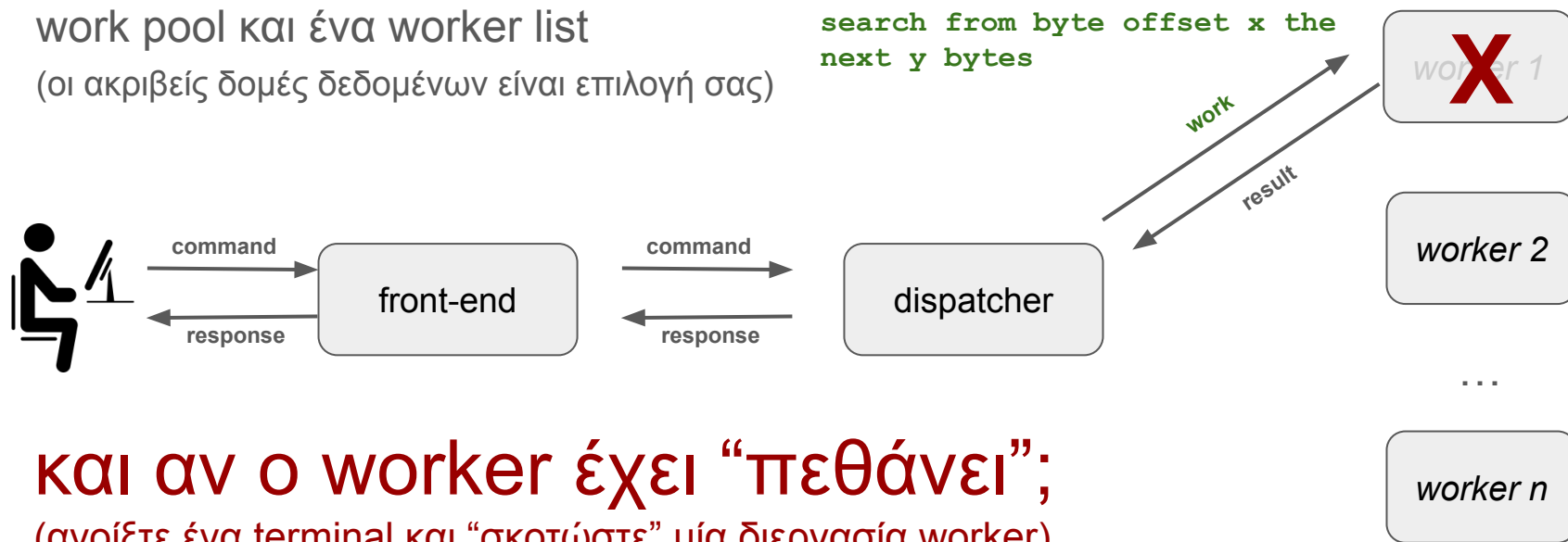


και αν ο worker έχει “πεθαίνει”;
(ανοίξτε ένα terminal και “σκοτώστε” μία διεργασία worker)

Εφαρμογή παράλληλης αναζήτησης χαρακτήρων σε αρχείο

ο dispatcher θα πρέπει να κρατά ένα work pool και ένα worker list

(οι ακριβείς δομές δεδομένων είναι επιλογή σας)



και αν ο worker έχει “πεθαίνει”;

(ανοίξτε ένα terminal και “σκοτώστε” μία διεργασία worker)