Προγραμματισμός ΙΙ (ECE116)

#18 Διαδιεργασιακή επικοινωνία: αγωγοί (pipes)

Σπύρος Λάλης
lalis@uth.gr

Συνεργασία ανάμεσα σε διεργασίες

- Για ασφάλεια/ανεξαρτησία, το ΛΣ εξασφαλίζει πλήρη **απομόνωση** ανάμεσα στις διεργασίες

- Τι γίνεται αν κάποιες διεργασίες επιθυμούν να **συνεργαστούν / αλληλεπιδράσουν** μεταξύ τους;

- Το ΛΣ παρέχει ειδικούς **μηχανισμούς επικοινωνίας** μέσω των οποίων διαφορετικές διεργασίες μπορούν να αλληλεπιδράσουν / στείλουν δεδομένα
  - inter-process communication (IPC) mechanisms

Μηχανισμοί διαδιεργασιακής επικοινωνίας

- Αρχεία – files
- Αγωγοί – pipes
  - unnamed pipes, named pipes
- Σήματα – signals
  - ειδοποιήσεις
- Υποδοχείς – sockets
  - Unix domain sockets, Internet domain sockets

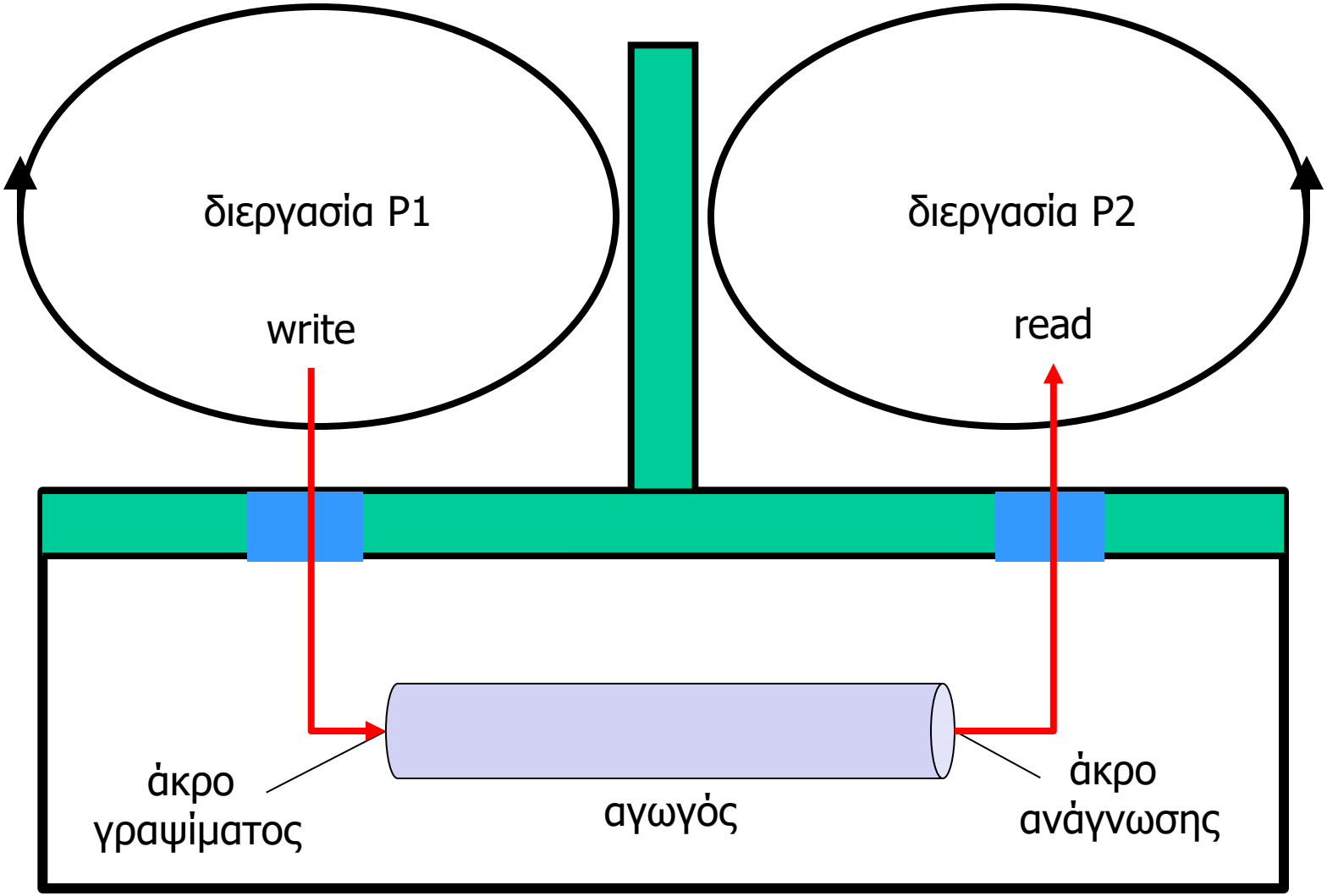
- Κοινόχρηστη μνήμη – shared memory
- Ουρές μηνυμάτων – message queues
- Σηματοφόροι – semaphores
  - συγχρονισμός / αμοιβαίος αποκλεισμός

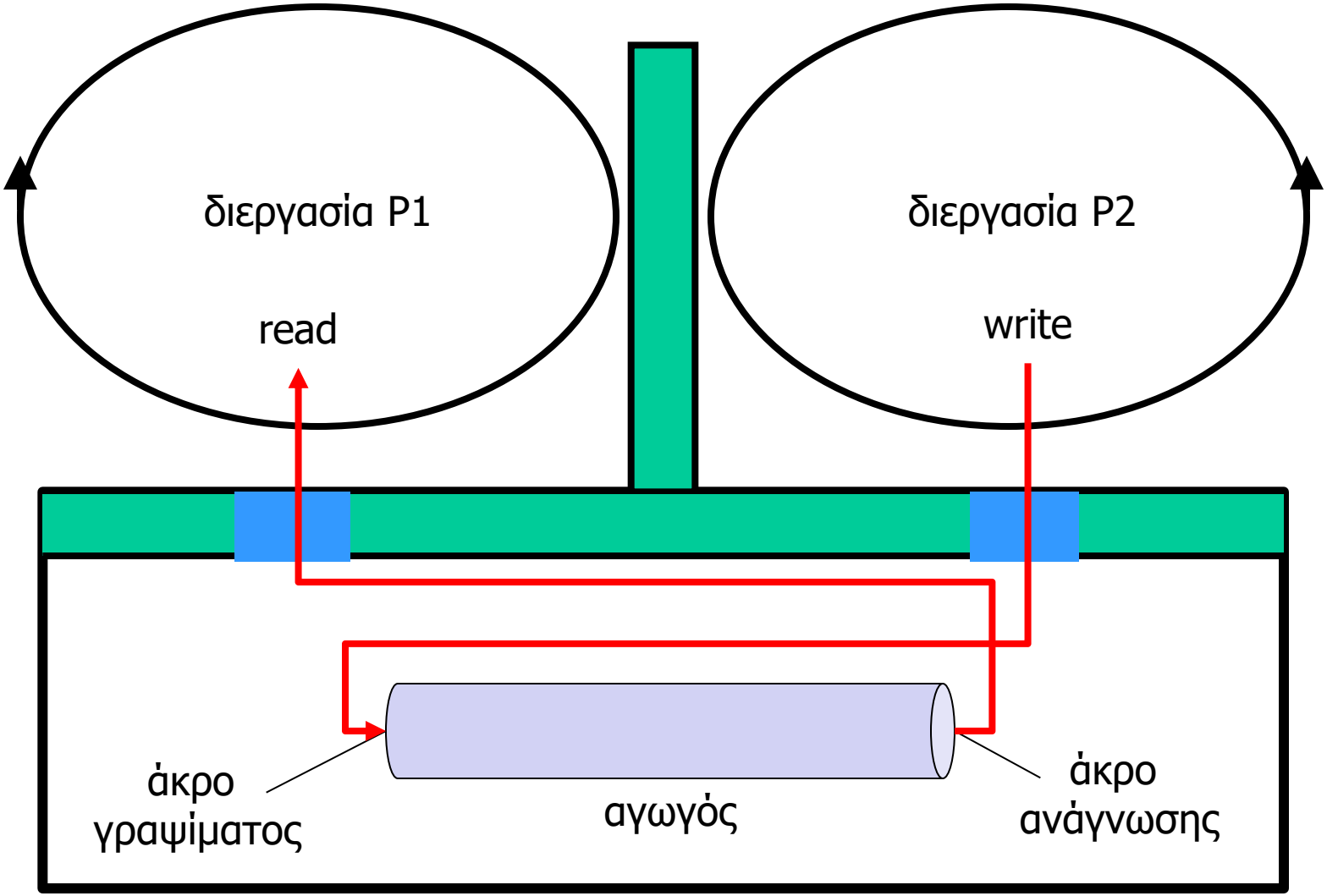
Αγωγός

- **Μηχανισμός/συσκευή επικοινωνίας** διεργασιών

- Μετάδοση μιας **ροής** από bytes (σε μια κατεύθυνση)
- **First-in-first-out (FIFO):** τα bytes βγαίνουν από τον αγωγό με την ίδια σειρά με την οποία μπήκαν

- Ένας αγωγός είναι μια ενδιάμεση **αποθήκη** bytes που την διαχειρίζεται το ΛΣ, με δύο άκρα
- **Άκρο γραψίματος**: «μπαίνουν» τα δεδομένα
- **Άκρο διαβάσματος**: «βγαίνουν» τα δεδομένα


διεργασία P1
write
διεργασία P2
read
άκρο
γραψίματος
αγωγός
άκρο
ανάγνωσης


διεργασία P1
read
διεργασία P2
write
άκρο
γραψίματος
αγωγός
άκρο
ανάγνωσης

Προσπέλαση αγωγού

- Για κάθε ένα από τα δύο άκρα του αγωγού επιστρέφεται ξεχωριστός **περιγραφέας αρχείου**

- **Γράψιμο** στον αγωγό: `write`
- **Διάβασμα** από τον αγωγό: `read`
- **Κλείσιμο** περιγραφέα: `close`

- Ισχύουν πολλά από όσα ήδη γνωρίζουμε για το γράψιμο/διάβασμα δεδομένων σε/από αρχεία δίσκου
- Υπάρχουν όμως και σημαντικές **διαφορές** ...

`read`, `write`, `close` σε αγωγό

- Η `read` **απομακρύνει** τα bytes από τον αγωγό
- Η `read` **μπλοκάρει** αν δεν υπάρχουν bytes μέσα στον αγωγό (μέχρι να γραφτούν bytes στον αγωγό)
  - όμως, βλέπε `close`
- Η `write` **μπλοκάρει** αν ο αγωγός έχει γεμίσει (μέχρι να διαβαστούν/απομακρυνθούν κάποια bytes)
- Η `close` έχει διάφορες **«παρενέργειες»**
- Αν κλείσουν όλοι οι περιγραφείς γραψίματος του αγωγού και δεν υπάρχουν bytes μέσα στον αγωγό, η `read` **επιστρέφει 0** (τότε και μόνο τότε)
- Αν κλείσουν όλοι οι περιγραφείς ανάγνωσης του αγωγού, η `write` **αποτυγχάνει**
  - στέλνεται **σήμα** στην διεργασία που οδηγεί στον **τερματισμό** της

Ανώνυμοι αγωγοί

- `int pipe(int fd[2])`
- Δημιουργεί έναν **ανώνυμο** αγωγό

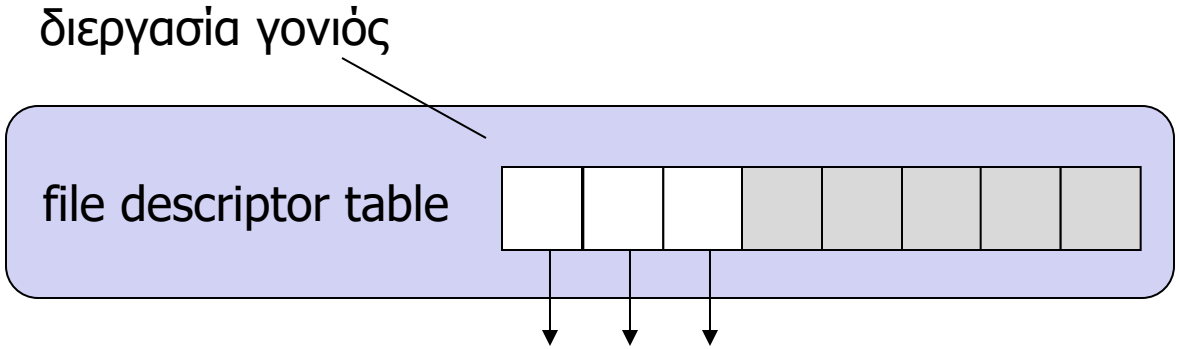
- Ο περιγραφέας για το **άκρο ανάγνωσης** αποθηκεύεται στο `fd[0]`
- Ο περιγραφέας για το **άκρο γραψίματος** αποθηκεύεται στο `fd[1]`

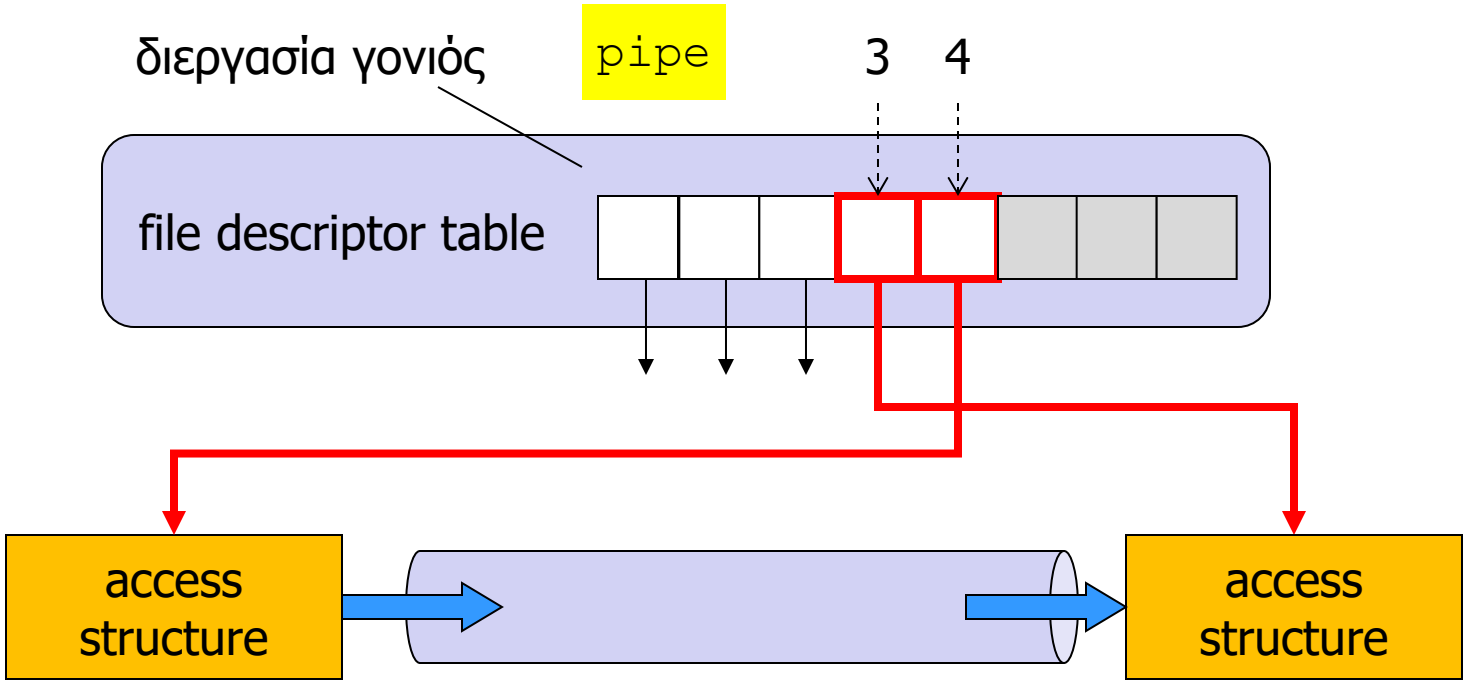
- Ο ανώνυμος αγωγός έχει **προσωρινή** υπόσταση
- Καταστρέφεται **αυτόματα** όταν κλείσει και ο τελευταίος περιγραφέας πάνω στον αγωγό

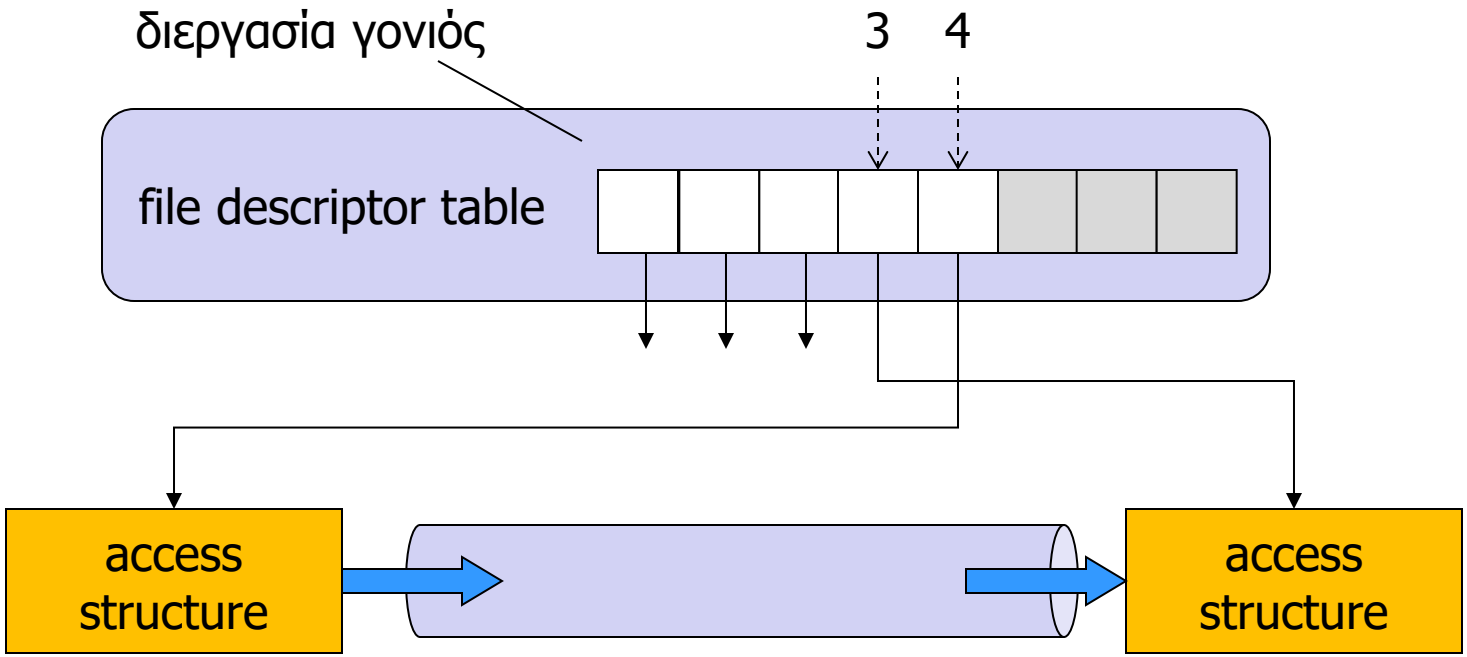
```
int main(int argc, char *argv[]) {
  int fd[2], i, val, count, n;

  count = atoi(argv[1]);
  pipe(fd);

  if (!fork()) {
    close(fd[0]); /* close pipe read end */
    for (i=1; i<=count; i++) {
      n = write(fd[1], &i, sizeof(int));
      printf("child: wrote %d (%d bytes)\n", i, n);
    }
    return(0);
  }

  close(fd[1]); /* close pipe write end */
  for (i=1; i<=count; i++) {
     n = read(fd[0], &val, sizeof(int));
     printf("child: read %d (%d bytes)\n", val, n);
  }
  return(0);
}
```

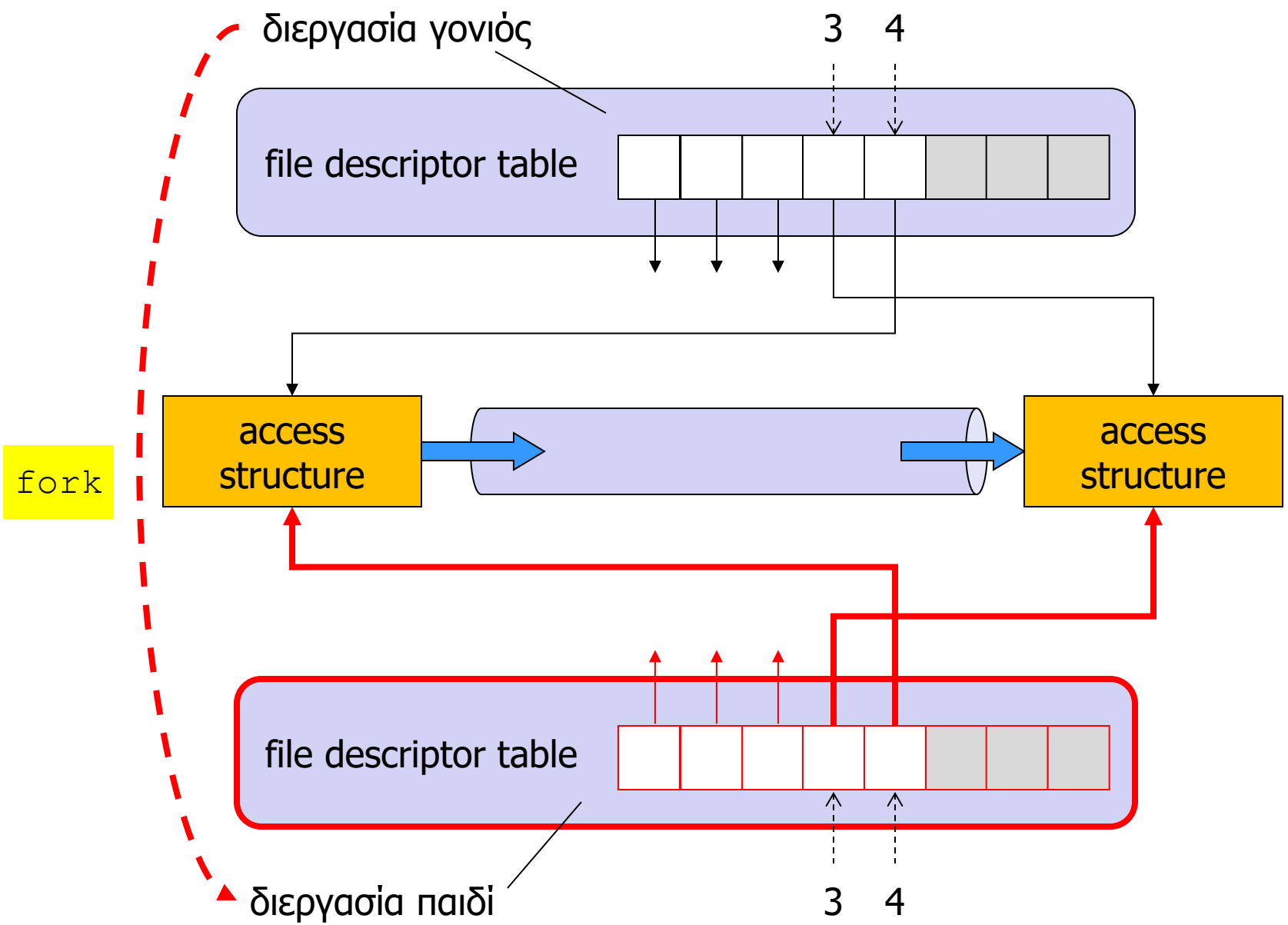
```
int main(int argc, char *argv[]) {
  int fd[2], i, count, n;

  count = atoi(argv[1]);
  pipe(fd);

  if (!fork()) {
    close(fd[0]); /* close pipe read end */
    for (i=1; i<=count; i++) {
      n = write(fd[1], &i, sizeof(int));
      printf("child: wrote %d (%d bytes)\n", i, n);
    }
    return(0);
  }

  close(fd[1]); /* close pipe write end */
  while ((n=read(fd[0], &i, sizeof(int))) > 0) {
    printf("parent: read %d (%d bytes)\n", i, n);
  }
  if (n == 0) { printf("parent: no data, bye\n"); }
  else { perror("read"); }
  return(0);
}
```

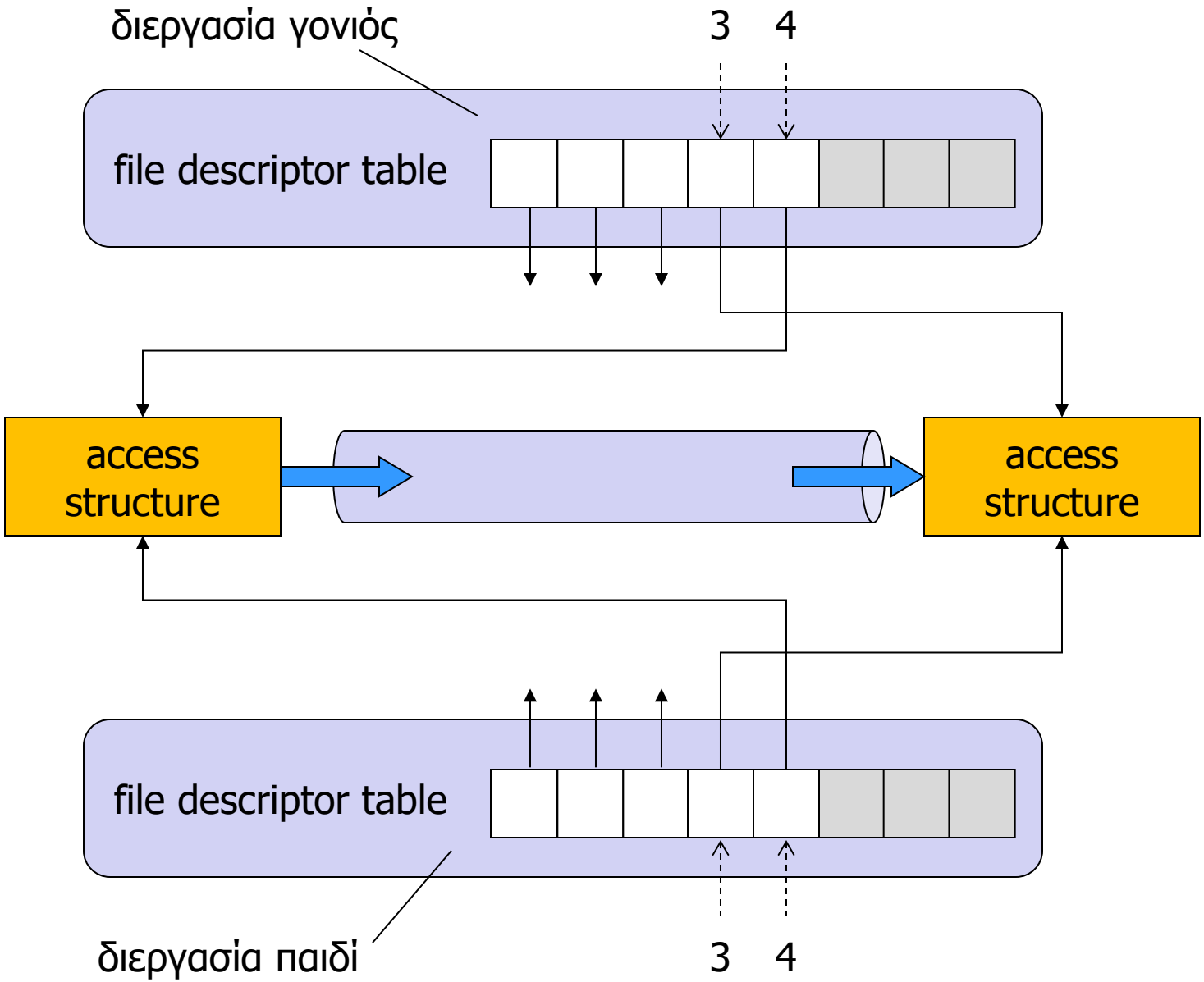
```
int main(int argc, char *argv[]) {
  int pid, fd[2], v1, v2, sum;

  pipe(fd);

  if (!(pid=fork())) {
    read(fd[0],&v1,sizeof(int));
    read(fd[0],&v2,sizeof(int));
    sum = v1+v2;
    write(fd[1],&sum,sizeof(int));
    return(0);
  }

  scanf("%d %d",&v1,&v2);
  write(fd[1],&v1,sizeof(int));
  write(fd[1],&v2,sizeof(int));

  waitpid(pid,NULL,0); // τι θα γίνει αν δεν υπάρχει αυτό;
  read(fd[0],&sum,sizeof(int));
  printf("sum is %d\n",sum);
  return(0);
}
```

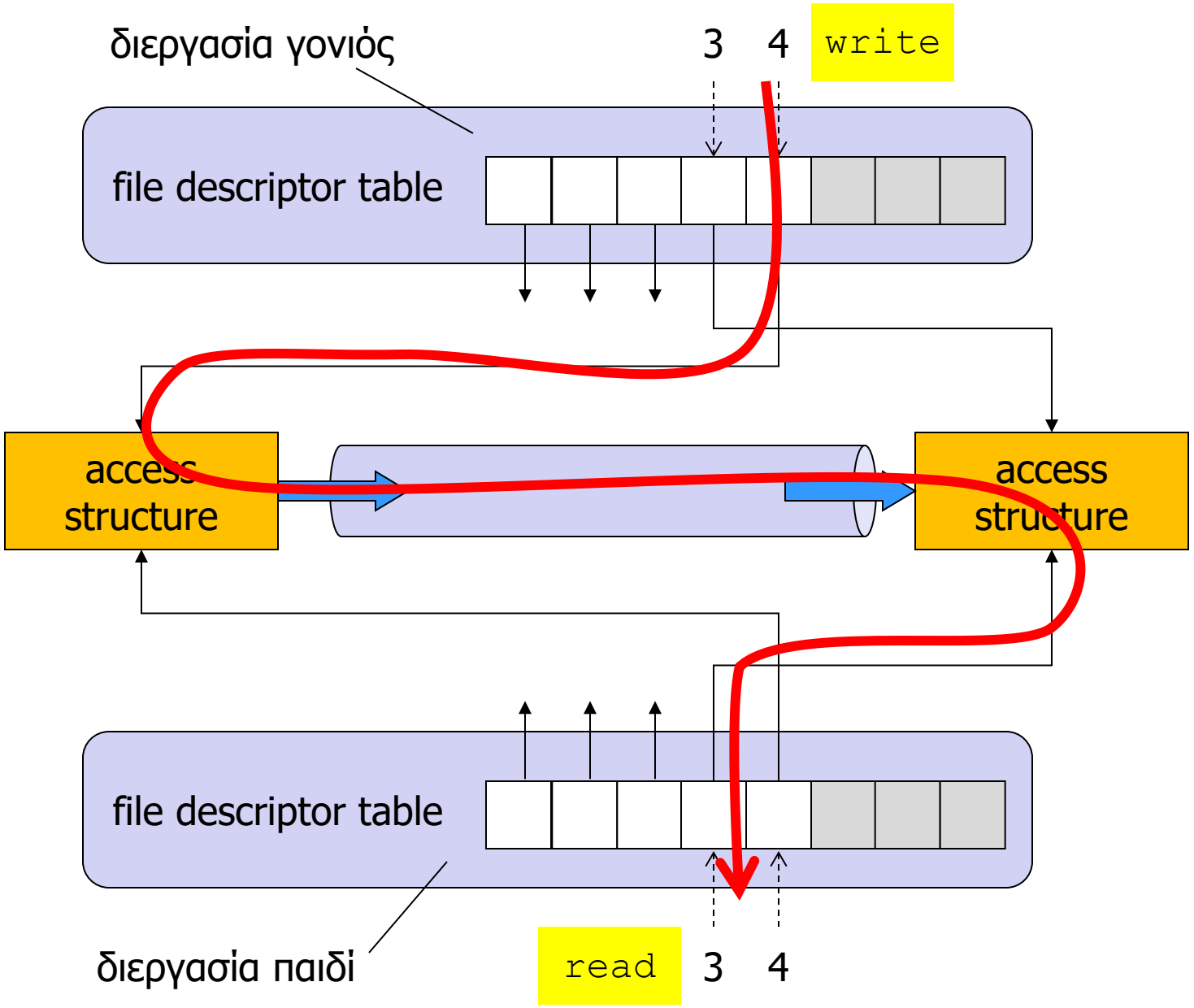

διεργασία γονιός
file descriptor table

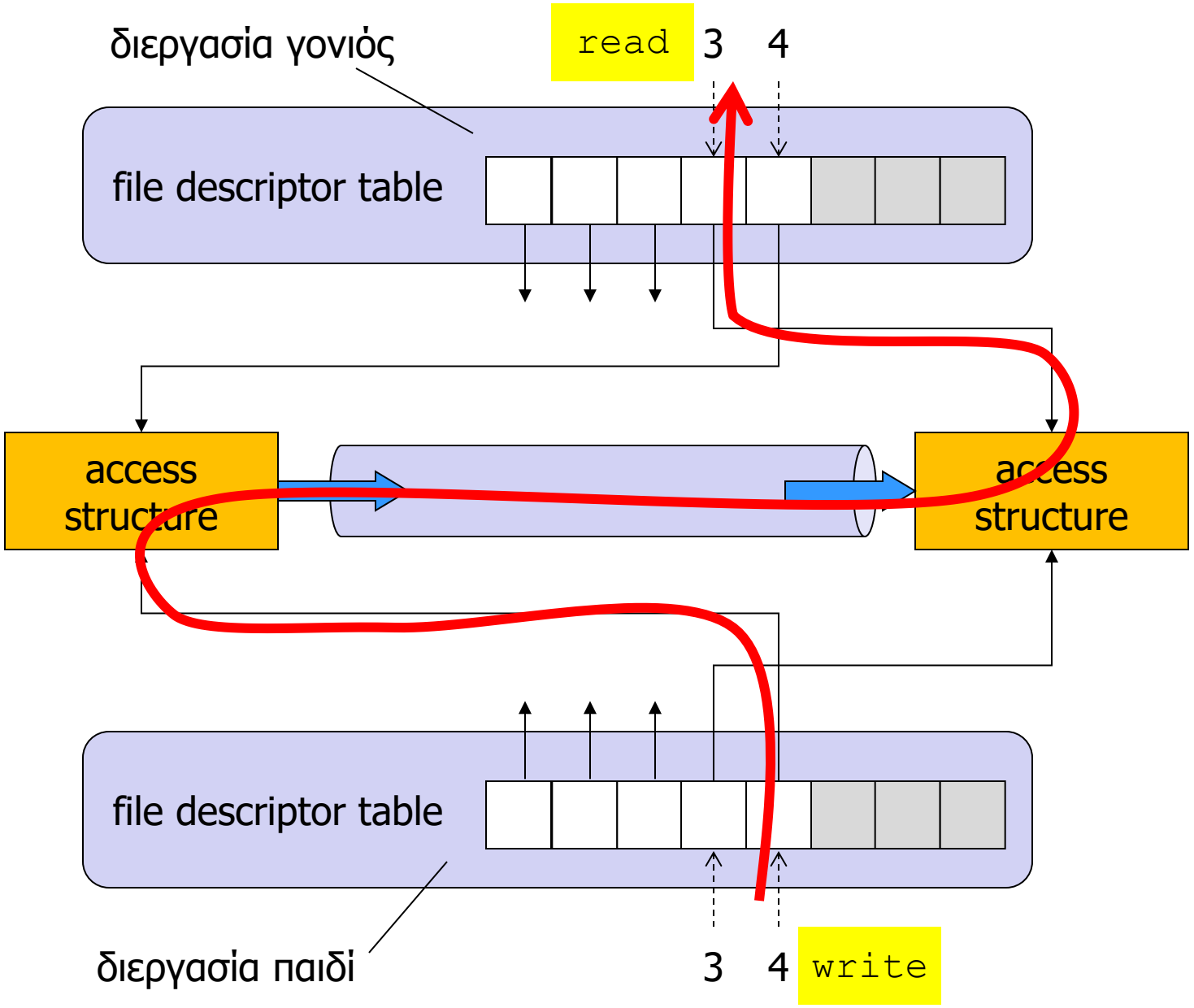

διεργασία γονιός
pipe
3
4
file descriptor table
access structure
access structure


διεργασία γονιός
3
4
file descriptor table
access structure
access structure


διεργασία γονιός
3
4
file descriptor table
fork
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
3
4
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
3
4
write
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
read
3
4


διεργασία γονιός
read
3
4
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
3
4
write

Ανακατεύθυνση εισόδου/εξόδου

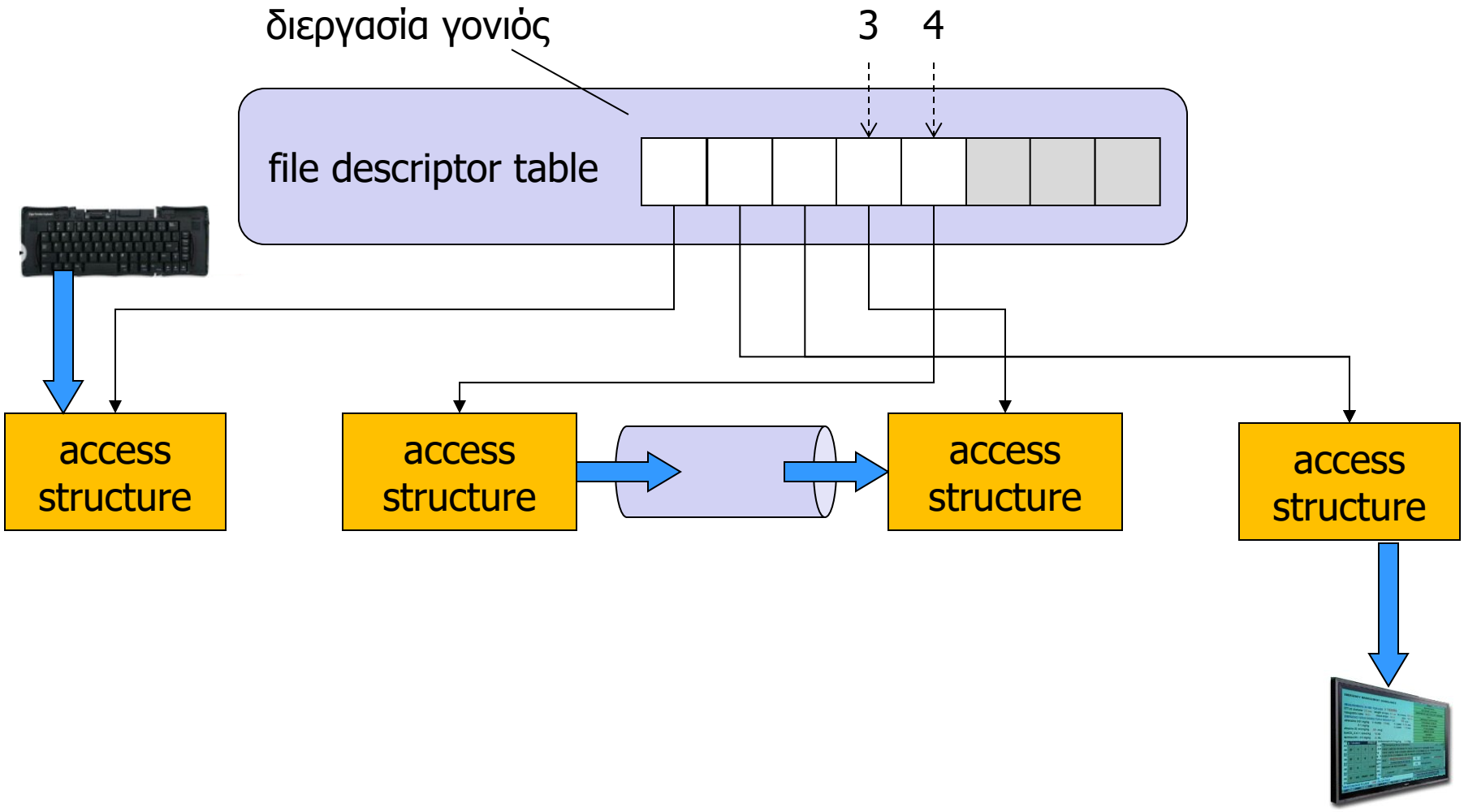
- Τα άκρα (ανάγνωσης/γραψίματος) ενός αγωγού προσπελάζονται μέσω περιγραφέων αρχείων
  - όπως και ένα οποιοδήποτε άλλο «αρχείο»

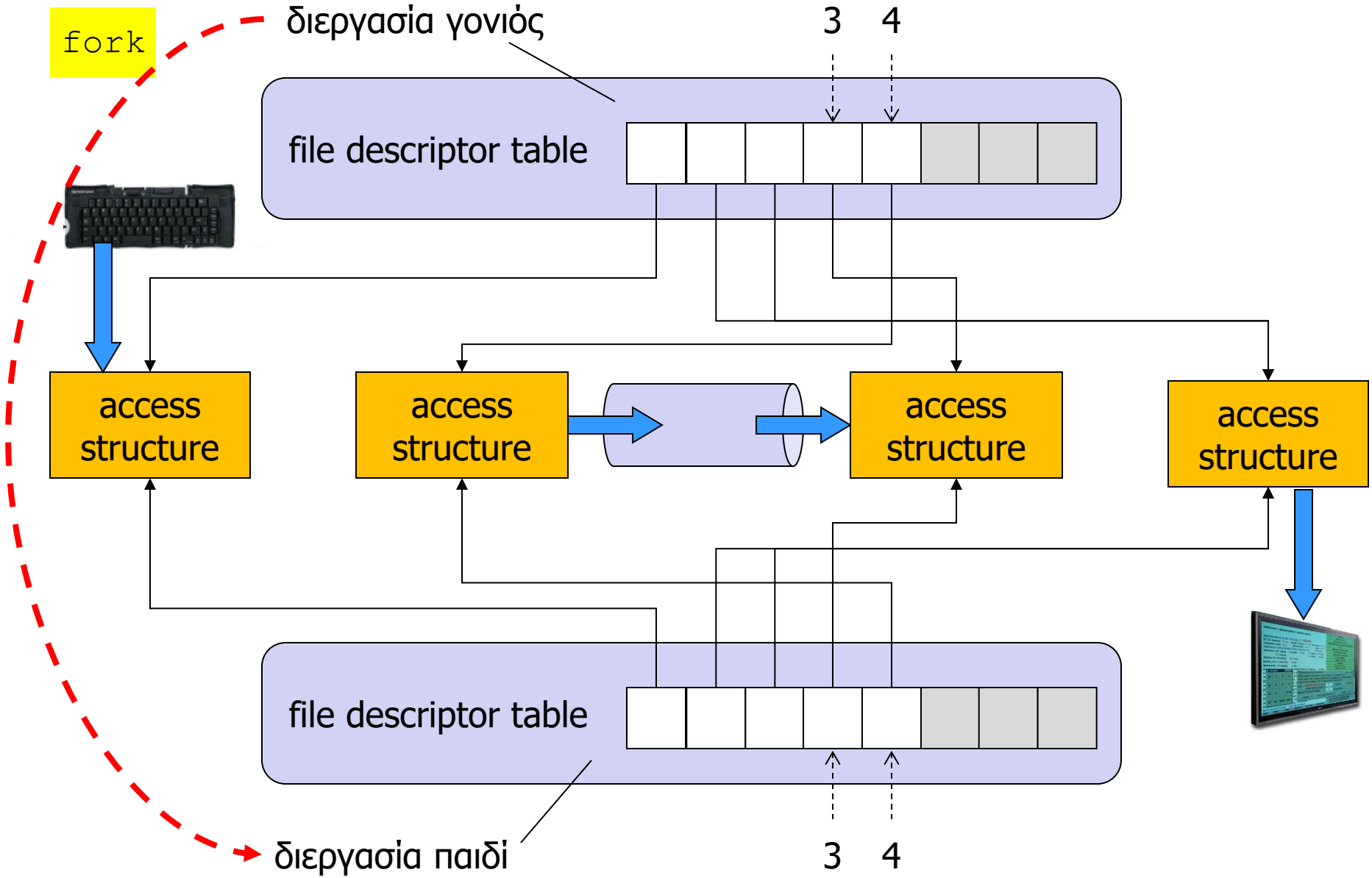
- Μπορεί να γίνει **ανακατεύθυνση** της καθιερωμένης Ε/Ε ενός προγράμματος σε έναν αγωγό, μέσω `dup2`
  - όπως γίνεται και σε ένα αρχείο

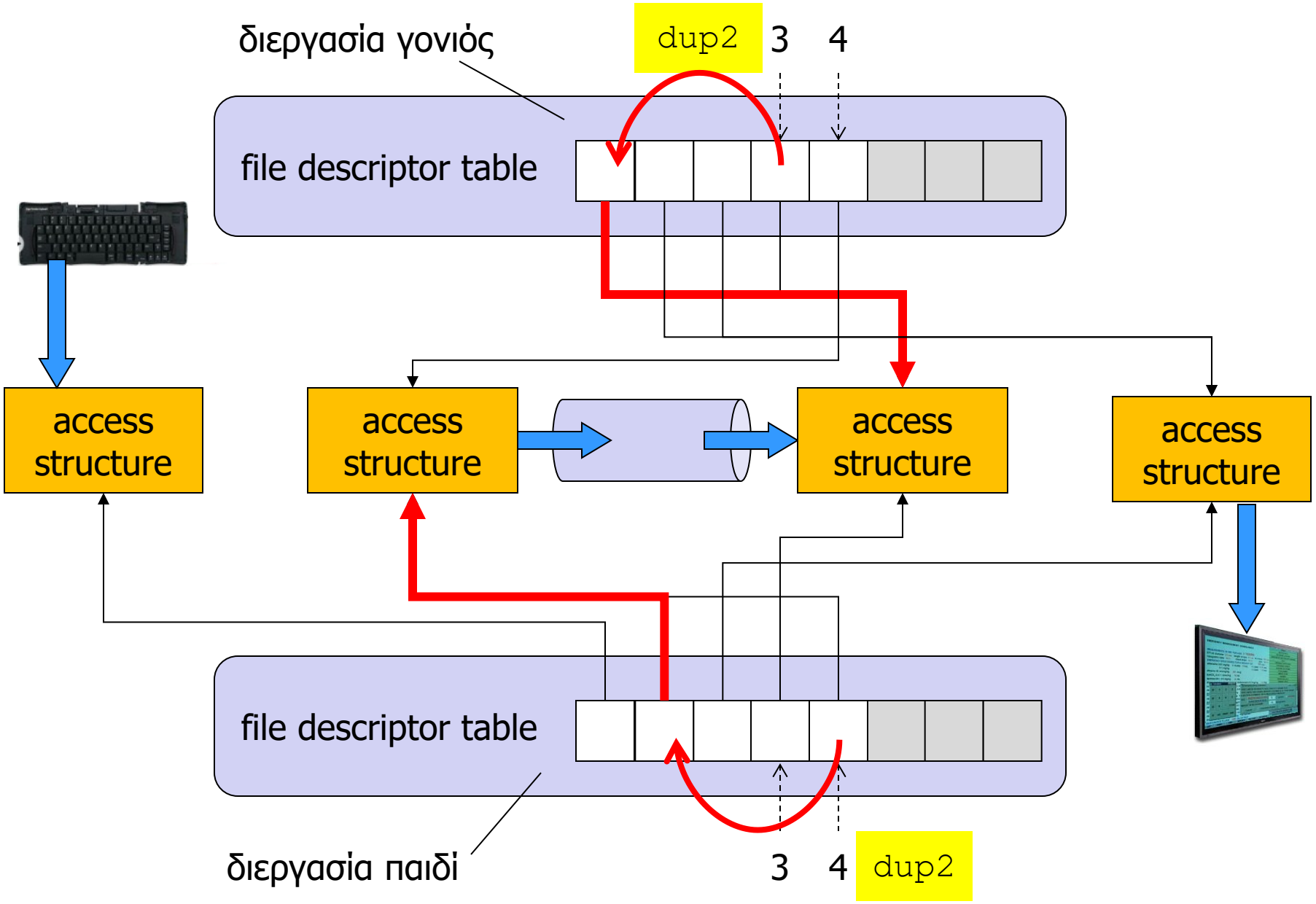
- Αντί το πρόγραμμα να διαβάζει δεδομένα από το τερματικό, μπορεί να τα διαβάζει από έναν αγωγό
- Αντί το πρόγραμμα να γράφει δεδομένα στο τερματικό, μπορεί να τα γράφει σε έναν αγωγό

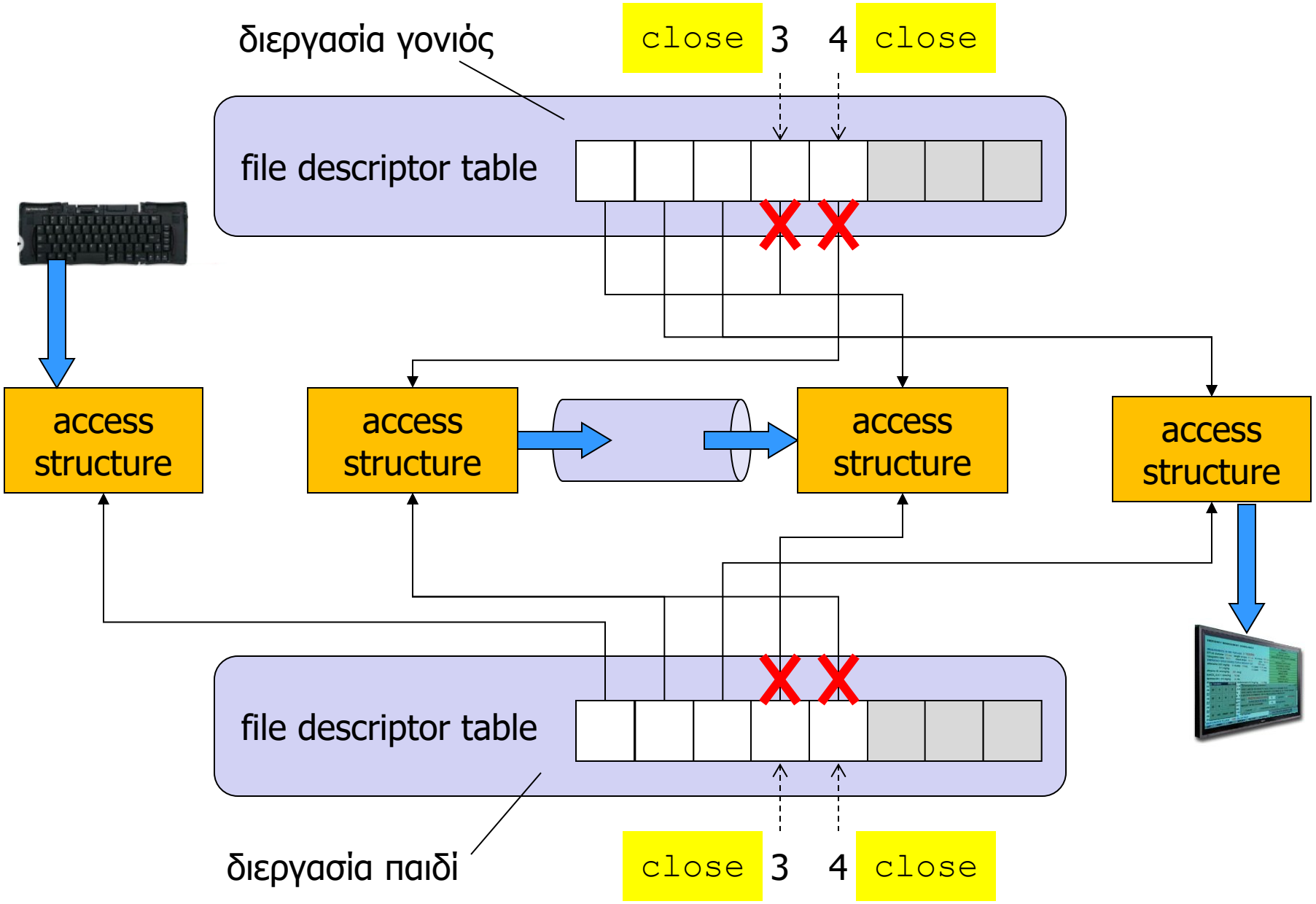
```
int main(int argc, char *argv[]) {
  int fd[2], i; char c;

  pipe(fd);

  if (!fork()) {
    dup2(fd[1], STDOUT_FILENO); /* redirect stdout */
    close(fd[0]); close(fd[1]);
    for (i=0; i<argc; i++) {
      printf("%s\n", argv[i]);
    }
    return(0);
  }

  dup2(fd[0], STDIN_FILENO); /* redirect stdin */
  close(fd[0]); close(fd[1]);
  while (scanf("%c", &c) != EOF) {
    printf("%c\n", c);
  }
  return(0);
}
```

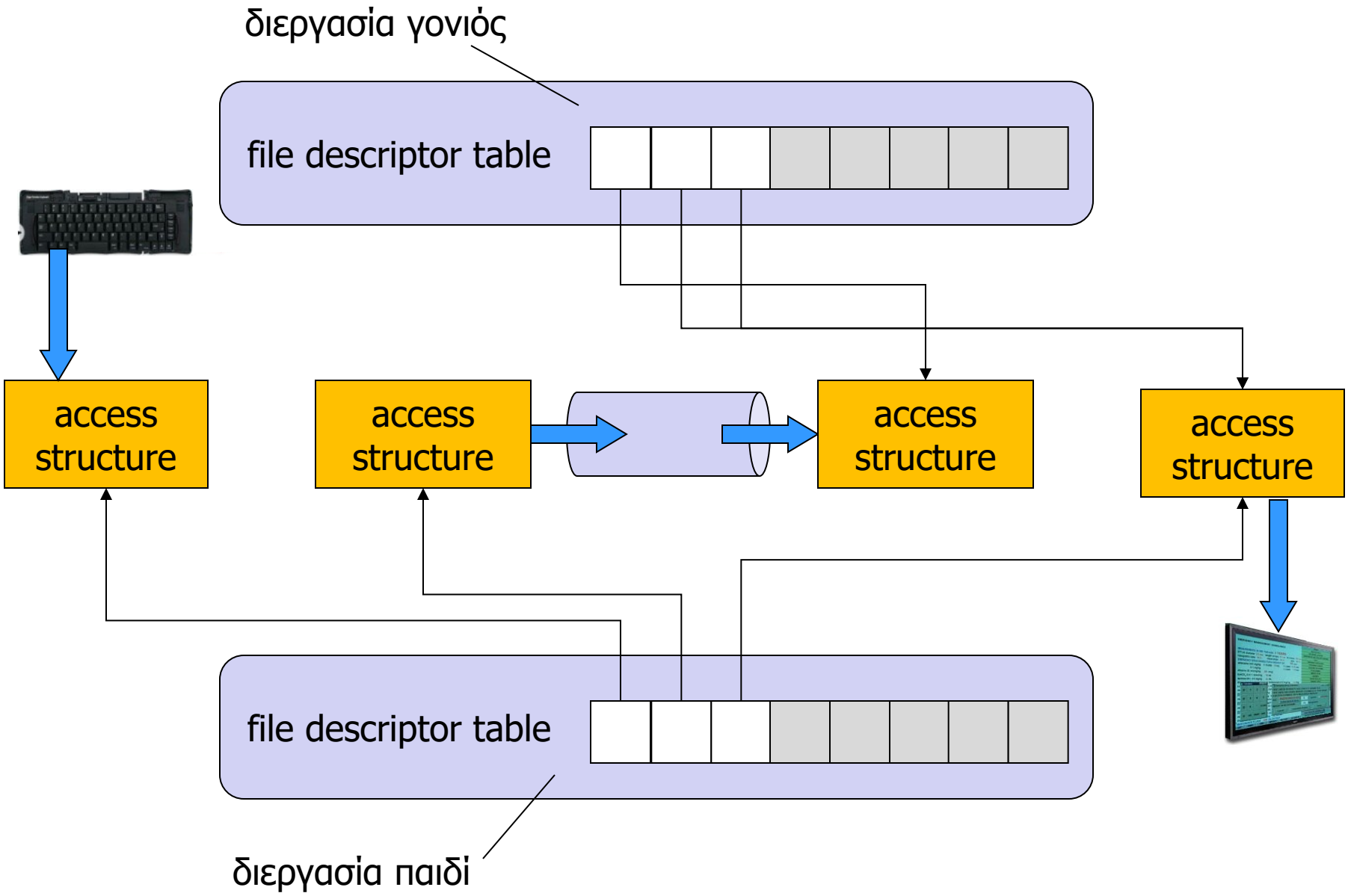
```
int main(int argc, char *argv[]) {
  int fd[2]; char c;

  pipe(fd);

  if (!fork()) {
    dup2(fd[1], STDOUT_FILENO); /* redirect stdout */
    close(fd[0]); close(fd[1]);
    execl("./add", "add", "123", "456", NULL);
    return(1);
  }

  dup2(fd[0], STDIN_FILENO); /* redirect stdin */
  close(fd[0]); close(fd[1]);
  while (scanf("%c", &c) != EOF) {
    printf("%c", c);
  }
  return(0);
}
```

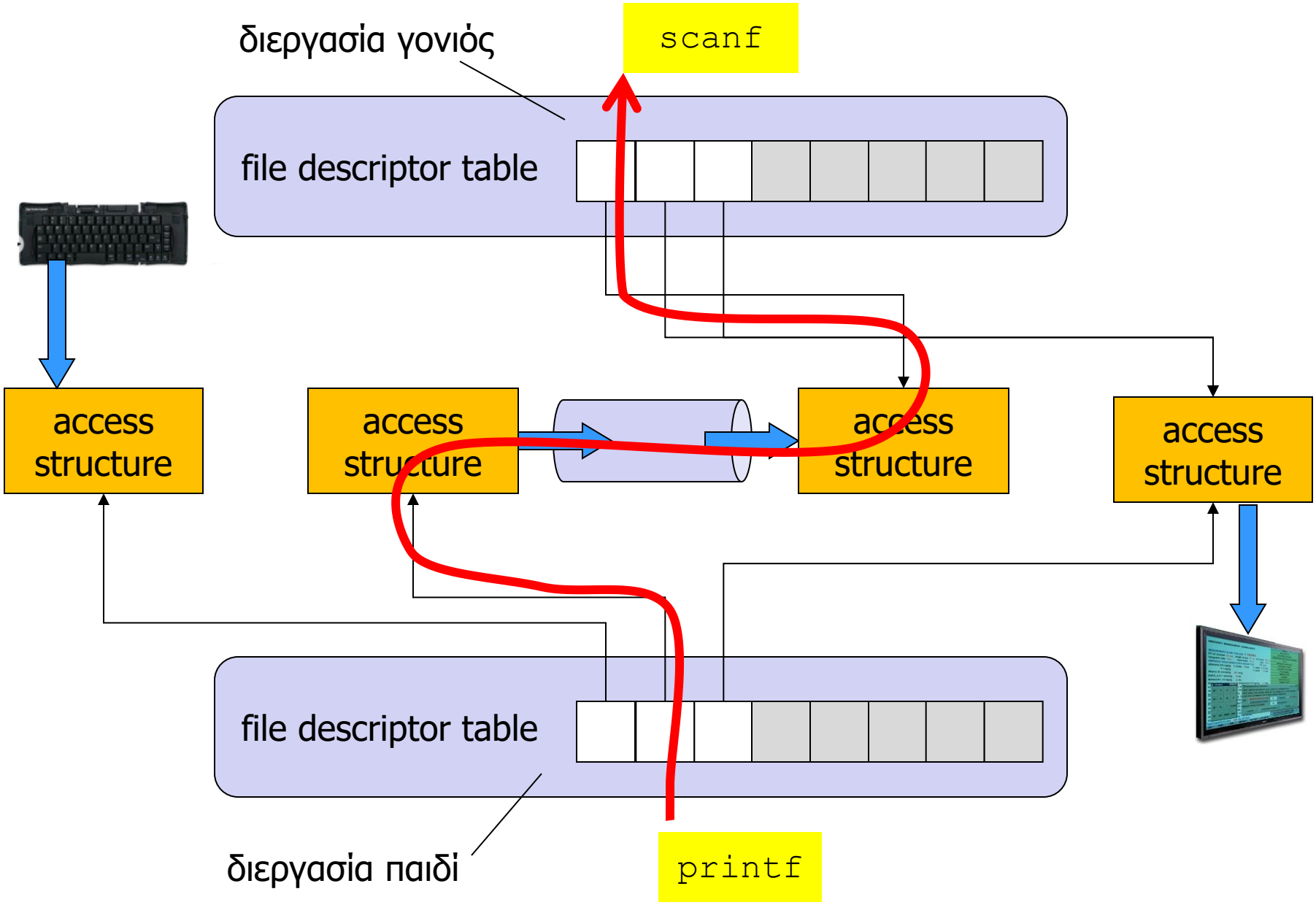

διεργασία γονιός
3
4
file descriptor table
access structure
access structure
access structure
access structure


fork
διεργασία γονιός
3
4
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
dup2
3
4
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
3
4
dup2


διεργασία γονιός
close
3
4
close
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
close
3
4
close


διεργασία γονιός
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί


διεργασία γονιός
scanf
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
printf

Επώνυμοι αγωγοί

- `int mkfifo(cons char *path,mode_t perms);`
- Δημιουργεί έναν **επώνυμο** αγωγό
  - το όνομα και οι άδειες δίνονται όπως για ένα αρχείο
  - η κλήση **αποτυγχάνει** αν ο αγωγός υπάρχει ήδη
- Η δημιουργία περιγραφέων για τα άκρα γραψίματος και ανάγνωσης του αγωγού γίνεται μέσω της `open`
- Άκρο **ανάγνωσης**: με `O_RDONLY` (**μπλοκάρει** αν δεν υπάρχει ανοιχτό άκρο γραψίματος για τον αγωγό)
- Άκρο **γραψίματος**: με `O_WRONLY` (**μπλοκάρει** αν δεν υπάρχει ανοιχτό άκρο ανάγνωσης για τον αγωγό)
- Ο επώνυμος αγωγός έχει **μόνιμη** υπόσταση
- Καταστρέφεται **ρητά** με `unlink` όπως τα αρχεία

client

```
int main(int argc, char *argv[]) {
  int fd, count;

  fd = open(argv[1], O_WRONLY); /* open pipe write end */
  if (fd < 0) {
    perror("client: open");
    return(1);
  }

  count = atoi(argv[2]);

  do {
    write(fd, &count, sizeof(int));
  } while (--count > 0);

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

server

```
int main(int argc, char *argv[]) {
  int fd, val, n;

  mkfifo(argv[1],S_IRWXU); /* create named pipe */

  while (1) {
    fd = open(argv[1], O_RDONLY); /* open pipe read end */
    if (fd < 0) {
      perror("server: open");
      return(1);
    }

    do {
      n = read(fd, &val, sizeof(int));
      printf("server read %d (%d bytes)\n", val, n);
    } while ((n > 0) && (val != -1));

    if (n > 0) {
      break;
    }
    else if (n < 0) {
      perror("server: read");
      return(1);
    }
    else {
      printf("server: end of input\n");
      close(fd);
    }
  }
  close(fd);
  unlink(argv[1]);
  return(0);
}
```

Θέματα συντονισμού/συγχρονισμού

- Ένας αγωγός μπορεί να χρησιμοποιείται **ταυτόχρονα** από **πολλές** διεργασίες
  - πολλές διεργασίες που επιθυμούν να γράψουν / διαβάσουν
- Παρόμοιο πρόβλημα υπάρχει και στην περίπτωση ταυτόχρονης πρόσβασης σε αρχεία
- Χρειάζεται **συντονισμός** έτσι ώστε οι διεργασίες να μην «μπλεχτούν» μεταξύ τους με ανεπιθύμητο τρόπο
  - αυτό ισχύει και για το γράψιμο σε αρχεία δίσκου ...
- Ειδική περίπτωση προβλήματος **συγχρονισμού**
  - βλέπε μάθημα Ταυτόχρονου Προγραμματισμού