ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Λειτουργικά Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:09

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

1. Σκοπός της άσκησης

- Διεργασίες Zombie.
- Διαδιεργασιακή επικοινωνία με pipes, sockets, ψευδοτερματικά, ports.
- **Συναρτήσεις:** pipe(), write(), read(), perror(), open(), openpty(), socketpair(), socket(), listen(), accept(), send(), write(), recv(), read().Grep.

2. Παραδοτέα

(A) 18 ερωτήσεις
(C) 11 ασκήσεις

3. Δημιουργία διεργασίας σε κατάσταση Zombie

Από τη σελίδα βοήθειας του προγράμματος ps βρείτε και σημειώστε πως σημειώνεται μια διεργασία που βρίσκεται στην κατάσταση Zombie:_____ **(A01)**

Αφού διαβάσετε τα σχετικά έγγραφα στο COMPUS ή στο άρθρο "Zombie Process" της wikipedia, απαντήστε:

Τι είναι η κατάσταση Zombie; Γιατί χρειάζεται;_____ **(A02)**
Γιατί η διεργασία διατηρεί καταχώρηση στον πίνακα διεργασιών του ΛΣ; _____ **(A03)**
Ποιο είναι το πρόβλημα στο ΛΣ με το να υπάρχουν διεργασίες σε κατάσταση Zombie; _____ **(A04)**
Πως εμφανίζεται το όνομα μιας διεργασίας Zombie; _____ **(A05)**

3.1 Παραδοτέο C1

(C1) Δημιουργήστε το πρόγραμμα **c1.c**, το οποίο δημιουργεί μια διεργασία σε κατάσταση zombie ως εξής:

- Θα καλείται η `fork()`
  - το παιδί θα εκτυπώνει το PID του και το μήνυμα "child will now exit and create zombie" και θα καλεί την `exit()`
  - ο πατέρας θα εκτυπώνει το μήνυμα "parent ps xuw", θα εκτελεί την εντολή "`ps xuw`" με τη `system()` και θα κάνει `sleep(10)`. Θα πρέπει να εμφανιστεί η διεργασία με το ανωτέρω PID σε κατάσταση Zombie.

Να τοποθετήσετε με αντιγραφή - επικόλληση τη συγκεκριμένη γραμμή του **`ps axuw`** που δείχνει ότι η διεργασία τοποθετήθηκε σε κατάσταση zombie _____ **(A06)**

4. Διαδιεργασιακή επικοινωνία με χρήση διασωλήνωσης

Με την τεχνική αυτή μπορούν διεργασίες που βρίσκονται στο ίδιο μηχάνημα να μεταδίδουν δεδομένα μεταξύ τους. Ένας σωλήνας είναι ένα κανάλι επικοινωνίας μιας κατεύθυνσης και όχι αμφίδρομης. Αν θέλουμε να δημιουργήσουμε αμφίδρομη επικοινωνία απαιτείται η χρήση δύο σωλήνων επικοινωνίας. Στο φλοιό η διασωλήνωση επιτυγχάνεται με τη χρήση της κάθετης μπάρας | .

Η χρήση της διασωλήνωσης επιτυγχάνεται με την κλήση συστήματος **`pipe()`**.

Από τη σελίδα βοήθειας της **`pipe()`** απαντήστε. Τι τιμή επιστρέφει και τι παραμέτρους δέχεται *(περιγράψτε την κάθε παράμετρο)*: _____ **(A07)**

Η εγγραφή στη διασωλήνωση επιτυγχάνεται με τη **`write()`** *(τμήμα 2 manpage)* ενώ η ανάγνωση από τη διασωλήνωση επιτυγχάνεται με τη **`read()`** *(τμήμα 2 manpage).*

Ποιοι είναι οι παράμετροι της **`write()`** και τι συμβολίζει η κάθε παράμετρος; Τι επιστρέφει;_____ **(A08)**

Ποιοι είναι οι παράμετροι της **`read()`** και τι συμβολίζει η κάθε παράμετρος; Τι επιστρέφει;_____ **(A09)**

Αρχικά θα δημιουργήσουμε ένα πρόγραμμα το οποίο χρησιμοποιεί την **`pipe()`** για να γράψει και στη συνέχεια να διαβάσει από το pipe αυτά που έγραψε. Κατά την κλήση του pipe δημιουργούνται δυο file descriptors *(pointers)*. Στο ένα file descriptor *(στο No1)* γράφουμε και στο δεύτερο *(στο No0)* διαβάζουμε. Δε μπορεί να γίνει διαφορετικά.

4.1 Παραδοτέα C2 και C3

(C2) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c2.c** το οποίο ως διεργασία:

- θα ορίζει ένα buffer[200] ως πίνακα χαρακτήρων.
- Θα κάνει κλήση της pipe() και θα ελέγχει με if αν έχει δημιουργηθεί, διαφορετικά θα αναφέρει κατάλληλο μήνυμα.
- Θα γράφει στο pipe *(στο σωστό file descriptor)* το ονοματεπώνυμο σας.
- Θα διαβάζει από το pipe *(από το σωστό file descriptor)* τι έχει γραφεί και θα τα τοποθετεί στο buffer [200].
- Με τη χρήση της write θα έχει ήδη διαβάσει πόσα στοιχεία βρίσκονταν στο pipe και τον αριθμό αυτό θα το τοποθετήσει στη συνέχεια στο buffer [ ].

Το μέγεθος των στοιχείων που μπορεί να τοποθετηθεί στο buffer έχει οριστεί από το ΛΣ μια προεπιλεγμένη τιμή. Βρείτε από το Internet για το ΛΣ που χρησιμοποιείται ποιο είναι το όριο σε ένα pipe. Όταν έχει γεμίσει το pipe τότε η επόμενη εγγραφή *(write())* τοποθετεί σε κατάσταση blocked τη συγκεκριμένη διεργασία, μέχρι κάποια άλλη διεργασία διαβάσει από αυτό και απομακρύνει στοιχεία.

Πόσο είναι το μέγεθος σε Bytes της διασωλήνωσης στο ΛΣ που χρησιμοποιείτε και που το βρήκατε; _____ **(A10)**

Να δημιουργήσετε το παρακάτω πρόγραμμα το οποίο θα επιβεβαιώνει το όριο.

(C3) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c3.c** το οποίο ως διεργασία:

- θα δημιουργεί ένα pipe.
- θα γράφει ένα χαρακτήρα *(δηλαδή 1 Byte)* στο pipe.
- Μόλις το γράψει θα αυξάνει την τιμή ενός μετρητή κατά 1.
- Μόλις το γράψει θα εκτυπώνει το μετρητή.
- Θα επαναλαμβάνεται η εγγραφή ενός χαρακτήρα.
- Μόλις σταματήσει να εκτυπώνεται η τιμή του μετρητή τότε η διεργασία θα έχει μπει σε κατάσταση blocked και μπορείτε να τη σταματήσετε μόνο με CTRL+C.

Αναγράψτε την τελευταία τιμή του μετρητή και σχολιάστε αν είναι η αναμενόμενη: _____**(A11)**[1]

Σε κάθε ΛΣ υπάρχει και ένα όριο ως προς το μέγιστο αριθμό των file descriptors που μπορεί να έχει μια διεργασία. Αυτό γίνεται για να μη καταναλωθούν όλοι οι πόροι από μια διεργασία.

Πόσο είναι το όριο των file descriptors που υπάρχει στο δικό σας ΛΣ και πως το βρήκατε *(εντολή ή άρθρο στο Internet):*_____**(A12)**

Να δημιουργήσετε το παρακάτω πρόγραμμα το οποίο θα επιβεβαιώνει το όριο. Προσέξτε ότι ένα `pipe()` μπορεί να αποτύχει να δημιουργηθεί, είτε αν δεν υπάρχουν διαθέσιμα file descriptors, είτε αν δεν υπάρχει διαθέσιμη μνήμη στο ΛΣ.

4.2 Παραδοτέα C4, C5 και C6

(C4) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c4.c** το οποίο ως διεργασία:

- θα δημιουργεί ένα pipe και θα ελέγχει αν έχει δημιουργηθεί με επιτυχία.
- θα αυξάνει το μετρητή κατά 1
- θα εκτυπώνει την τιμή του μετρητή

[1] Μπορείτε να δείτε και το αρχείο pipe.h αν υπάρχει στο σύστημά σας.

- θα επαναλαμβάνει τη εντολή δημιουργίας νέου pipe.
- Μόλις αποτύχει η δημιουργία pipe τότε θα καλείται κατάλληλα η συνάρτηση **`perror()`** *(τμήμα 3 manpage)* και θα εκτυπώνει το μήνυμα λάθους. Επίσης να εκτυπώσετε την τιμή errno, δηλαδή την τιμή επιστροφής σφάλματος της **`pipe()`**.

Ποια είναι η τελευταία τιμή του μετρητή σας και για ποιο λόγο δε μπόρεσε να δημιουργηθεί ένα νέο pipe;_____ **(A13)**

Η συνάρτηση **`pipe()`** δε μπορεί να χρησιμοποιηθεί για να διασωληνώσει διεργασίες που ήδη εκτελούνται. Μπορεί όμως να χρησιμοποιηθεί από νέες διεργασίες που κάνουν **`fork()`**, γιατί μόλις γίνει fork() και η γονική και η θυγατρική διεργασία έχουν τους ίδιους file descriptors. Έτσι, αρχικά η διεργασία δημιουργεί ένα pipe και στη συνέχεια καλεί τη **`fork()`**. Αυτό έχει ως συνέπεια όταν κάποια διεργασία να γράφει στο file descriptor 1, τότε αυτό να καταλήγει στο file descriptors 0 και των δυο διεργασιών. Για να μη δημιουργηθεί ένα πρόβλημα, αμέσως μετά η γονική διεργασία κλείνει το file descriptor 0 *(αν η γονική θέλει να γράφει μόνο)* και η θυγατρική κλείνει το file descriptor 1 *(αν η θυγατρική θέλει να διαβάζει μόνο)*. Ασφαλώς, μπορεί αυτό να γίνει και αντίστροφα. Σε κάθε περίπτωση η μια διεργασία θα πρέπει να κλείσει το δικό της περιγραφέα αρχείου 0 και η άλλη να κλείσει το δικό της περιγραφέα αρχείου 1.

Ποιοι είναι οι παράμετροι της **`close()`** *(τμήμα 2 manpage)* και τι συμβολίζει η κάθε παράμετρος; Τι επιστρέφει; _____ **(A14)**

(C5) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c5.c** το οποίο ως διεργασία:

- Θα δημιουργεί ένα pipe.
- Θα καλείται η **`fork()`**.
- Η γονική θα κλείνει το file descriptor 0, γιατί θα γράφει μόνο
- Η θυγατρική θα κλείνει το file descriptor 1, γιατί θα διαβάζει μόνο
- Η θυγατρική θα κάνει **`sleep(1)`** και θα διαβάζει από το pipe. Ότι διαβάζει θα το εκτυπώνει στη οθόνη με το μήνυμα "`Child read: XXXX`" , όπου **XXXX** αυτό που έχει διαβάσει από το pipe. Θα διαβάζει μέχρι να συναντήσει τους χαρακτήρες "`00`" το οποίο θα σηματοδοτεί το τέλος των δεδομένων. Μόλις ανιχνευτεί το τέλος θα στέλνει το σήμα τερματισμού στον πατέρα και η ίδια θα καλεί την **`exit()`**.
- Η γονική διεργασία ύστερα από προτροπή θα ζητάει από το χρήστη να πληκτρολογήσει κάτι. Ότι πληκτρολογήσει ο χρήστης θα στέλνεται μέσω pipe στη θυγατρική. Η διαδικασία αυτή θα επαναλαμβάνεται επ αόριστον *(ασφαλώς μόλις ο χρήστης πατήσει 00 τότε η θυγατρική θα λαμβάνει το 00 και θα τερματίζει και τις δυο διεργασίες).*

Σε περίπτωση που θέλουμε να επιτύχουμε αμφίδρομη επικοινωνία, τότε θα χρησιμοποιήσουμε 2 διασωληνώσεις. Αυτό θα το δούμε με το παρακάτω παράδειγμα.

(C6) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c6.c** το οποίο ως διεργασία:

- Θα δημιουργεί δυο pipe.
- Θα καλείται η `fork()`.
- Η γονική θα κλείνει το file descriptor 0 του πρώτου pipe και το file descriptor 1 του δεύτερου pipe, γιατί θα γράφει μόνο στο πρώτο και θα διαβάζει μόνο από το δεύτερο.
- Η θυγατρική θα κλείνει το file descriptor 1 του πρώτου pipe και το file descriptor 0 του δεύτερου pipe, γιατί θα διαβάζει μόνο από το πρώτο pipe και θα γράφει μόνο στο δεύτερο.
- Η γονική θα ρωτάει το χρήστη να πληκτρολογήσει κάτι. Αυτό που πληκτρολογεί θα στέλνεται στη θυγατρική και θα κάνει `sleep(1)`. Στη συνέχεια θα διαβάζει από το δεύτερο pipe *(αν δεν έχουν έρθει δεδομένα θα περιμένει σε κατάσταση blocked).* Μόλις διαβάσει τα δεδομένα θα τα εκτυπώνει με το μήνυμα "`Parent Read: XXX`", όπου **XXX** το μήνυμα που έστειλε η θυγατρική. Αυτό θα εκτελείται σε ένα ατέρμονα βρόχο.
- Η θυγατρική θα διαβάζει από το πρώτο pipe θα εκτυπώνει το μήνυμα "`Child Read: XXX`", όπου **XXX** το μήνυμα που έχει διαβάζει, θα κάνει sleep(1) και θα στέλνει στο δεύτερο pipe το μήνυμα αφού έχει προσθέσει και τη λέξη -child- . Αυτό θα εκτελείται σε ένα ατέρμονα βρόχο.

5. Διαδιεργασιακή επικοινωνία με χρήση ψευδο-τερματικών

Υπάρχουν περιπτώσεις που απαιτείται η χρήση ενός τερματικού, όπως για παράδειγμα επεξεργαστές κειμένου κ.ο.κ. Υπάρχει περίπτωση να χρησιμοποιηθεί ένα ψευδοτερματικό για την επικοινωνία. Η επικοινωνία αυτή είναι αμφίδρομη σε αντίθεση με της διασωλήνωσης. Τα ψευδοτερματικά είναι ειδικά αρχεία που βρίσκονται στον κατάλογο /dev και βρίσκονται σε ζευγάρια. Υπάρχουν τα slave ψευδοτερματικά με ονόματα /dev/ttyp[a-z][0-9] και τα αντίστοιχα master ψευδοτερματικά με ονόματα /dev/pts[a-z][0-9] ή σε νεότερες διανομές UNIX /dev/pts/[0-9]. Με αυτόν τον τρόπο μια διεργασία γράφει στο /dev/tty??? και τα αποτελέσματα εμφανίζονται στο /dev/pty???, ενώ επίσης μπορεί να γράψει στο /dev/pty?? και τα αποτελέσματα να εμφανιστούν στο /dev/tty???

Για να τα χρησιμοποιήσουμε θα πρέπει να καλέσουμε τη συνάρτηση:

```
int fd = open("/dev/ptypb7",2); ή
int tty_fd = open("/dev/ttypb7",2);
```

Προκειμένου να βρούμε ένα τερματικό για να το χρησιμοποιήσουμε χρησιμοποιούμε συνήθως την τεχνική του ψαρέματος. Δηλαδή, προσπαθούμε να ανοίξουμε ένα pty???? και αν η συνάρτηση `open()` επιστρέψει σφάλμα προσπαθούμε με το επόμενο κ.ο.κ. Μόλις, ανοίξουμε ένα pty?? τότε η άλλη διεργασία μπορεί να ανοίξει το αντίστοιχο tty και έτσι επιτυγχάνεται η διαδιεργασιακή επικοινωνία.

Αυτή η μορφή επικοινωνίας εξαρτάται από τα ΛΣ. Για παράδειγμα στο FreeBSD δε χρησιμοποιείται η συνάρτηση open για να ανοιχθεί ένα pty, αλλά η συνάρτηση `openpty()` η οποία επιστρέφει το αμέσως επόμενο διαθέσιμο pty που βρίσκεται στο σύστημα.

Θα πρέπει να δείτε το manpage του pty στο ΛΣ που χρησιμοποιείτε για να δείτε τι υποστηρίζεται.

5.1 Παραοτέο C7

(C7) Να δημιουργήσετε ένα πρόγραμμα **c7.c** το οποίο ως διεργασία:

- θα ανοίγει ένα pty.
- θα κάνει fork.
- η γονική διεργασία θα γράφει το όνομα σας στο pty.
- η θυγατρική διεργασία θα διαβάζει από το αντίστοιχο tty και θα εκτυπώνει αντίστοιχο μήνυμα.
- η θυγατρική διεργασία θα γράψει το επώνυμο στο tty.
- η γονική διεργασία θα διαβάζει από το pty και θα εκτυπώνει το αντίστοιχο μήνυμα.

6. Διαδιεργασιακή επικοινωνία με χρήση sockets (υποδοχών)

Εκτός από τους παραπάνω τρόπους επικοινωνίας, δύο ή περισσότερες διεργασίες μπορούν να επικοινωνήσουν με τη χρήση sockets. Μάλιστα τα sockets επιτρέπουν την επικοινωνία *(με κατάλληλη ρύθμιση)* ανάμεσα σε διεργασίες διαφορετικών μηχανημάτων. Οι υποδοχές μπορούν να είναι είτε τοπικές *(local)* είτε δικτυακές *(network)*. Χρησιμοποιούνται οι κλήσεις συστήματος read/write και είναι αμφίδρομης κατεύθυνσης. Προκειμένου να κλείσει ένα socket θα πρέπει η διεργασία που το έχει δημιουργήσει να καλέσει την κλήση συστήματος shutdown() *(τμήμα 2 manpage)*.

Για τη δημιουργία ενός socket χρησιμοποιείται η κλήση συστήματος `socketpair()`.

Ποιοι είναι οι παράμετροι της `socketpair()` *(τμήμα 2 manpage)* και τι συμβολίζει η κάθε παράμετρος; Τι επιστρέφει; _____ **(A15)**

Με την κλήση της `socketpair()` δημιουργούνται δύο file descriptors *(0 και 1)*. Μπορούμε να γράψουμε στο 0 και να τα διαβάσουμε στο 1, ή να γράψουμε στο 0 και να τα διαβάσουμε στο 1.

6.1 Παραδοτέα C8 και C9

(C8) Να δημιουργήσετε ένα πρόγραμμα **c8.c**, το οποίο ως διεργασία:

- δημιουργεί με το socketpair ένα socket με τις παραμέτρους AF_UNIX ή PF_UNIX προκειμένου να δημιουργηθεί μια υποδοχή στον τοπικό υπολογιστή και παράμετρο πρωτοκόλλου SOCK_STREAM.
- Στη συνέχεια θα γράφει στο file descriptor 1 το όνομά σας.
- Μετά θα διαβάζει από το file descriptor 0 τι έχει γραφεί και θα το εκτυπώνει *(θα πρέπει να είναι το ίδιο)*.
- Μετά θα γράφει στο file descriptor 0 το επίθετό σας.
- Ακολούθως, θα διαβάζει από το file descriptor 1 τι έχει γραφεί και θα το εκτυπώνει *(θα πρέπει να είναι το ίδιο με αυτό που γράψατε πριν)*.

(C9) Να δημιουργήσετε ένα πρόγραμμα **c9.c**, το οποίο ως διεργασία:

- δημιουργεί με το socketpair ένα socket με τις παραμέτρους AF_UNIX ή PF_UNIX για να δημιουργηθεί μια υποδοχή στον τοπικό υπολογιστή και παράμετρο πρωτοκόλλου SOCK_STREAM.
- Δημιουργεί με το fork δυο διεργασίες, οπότε η κάθε διεργασία υιοθετεί τους file descriptors.
- Η κάθε διεργασία θα κλείσει τον ένα από τους δύο file descriptors. Η άλλη θα κλείσει τον άλλο.
- Η πρώτη διεργασία θα γράψει τον αριθμό μητρώου στο file desciptor, η άλλη διεργασία θα το διαβάζει και θα το εκτυπώνει.
- Αμέσως μετά, η δεύτερη διεργασία θα κάνει το ίδιο.

Μερικές φορές θέλουμε να δημιουργήσουμε μια υποδοχή δικτυακού τύπου, δηλαδή μια υποδοχή που περιμένει *(βρίσκεται σε κατάσταση ακρόασης)* να εξυπηρετήσει κάποιους πελάτες από το δίκτυο. Σε αυτήν την περίπτωση θα πρέπει να ακολουθήσουμε τα εξής βήματα.

Αρχικά θα πρέπει να κάνετε include κάποια header files. Συγκεκριμένα, θα πρέπει να κάνετε

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

(μπορεί να μη χρειάζεται, αρχικά μην το κάνετε include)

Στη συνέχεια θα πρέπει να κάνετε κλήση τη συνάρτηση

```
int socket(int domain, int type, int protocol)
```

η οποία δέχεται τρεις παραμέτρους. Η πρώτη παράμετρος θα είναι το domain το οποίο αν θέλουμε να είναι δικτυακού τύπου θα είναι PF_INET *(αν θέλαμε μια τοπική socket σε κατάσταση ακρόασης θα βάζαμε PF_UNIX).* Η δεύτερη παράμετρος *(type)* θα πρέπει να είναι SOCK_STREAM αν θέλουμε να χρησιμοποιήσουμε TCP ή SOCK_DGRAM αν θέλουμε να χρησιμοποιήσουμε UDP. Η τρίτη παράμετρος θα είναι (0) για την προεπιλογή ρυθμίσεων *(μπορεί ένα πρωτόκολλο να υποστηρίζει πολλαπλές ρυθμίσεις).*

Από τη σελίδα βοήθειας της socket αναφέρετε τι τύποι socket επιτρέπονται στο σύστημά σας:_____ **(A16)**

Από την ίδια σελίδα γράψτε ποιοι τύποι επιτρέπουν εγγυημένη *(reliable)* μετάδοση δεδομένων και ποιοι όχι:_____ **(A17)**

Πότε αποτυγχάνει η συγκεκριμένη κλήση; Αναφέρετε ενδεικτικά δυο καταστάσεις που μπορεί να οδηγήσουν σε αποτυχία:_____ **(A18)**

Αφού δημιουργηθεί με το socket η θύρα υποδοχής, θα πρέπει στη συνέχεια να την τοποθετήσουμε σε μια συγκεκριμένη διεύθυνση. Αυτό επιτυγχάνεται με την κλήση bind() *(σελίδα βοήθειας 2 του manpage).* Η συνάρτηση αυτή δέχεται 3 παραμέτρους. Η πρώτη παράμετρος αντιστοιχεί στο file descriptor της υποδοχής που δημιουργήσαμε με το socket. Η δεύτερη παράμετρος είναι struct, δηλαδή μια σύνθετη μεταβλητή. Θα πρέπει πριν, να έχετε ορίσει μια μεταβλητή struct sockaddr_in myaddr στην οποία θα τοποθετήσετε τις παρακάτω τιμές *(για δικτυακή ακρόαση).*

```
name.sin_family=PF_INET
```

```
name.sin_addr.s_addr=htonl(INADDR_ANY)
```
για ακρόαση σε οποιαδήποτε διεύθυνση έχει το μηχάνημά μας

```
name.sin_port=htons(port)
```
όπου port είναι ο αριθμός θύρας που χρησιμοποιούμε

Σε περίπτωση που είναι TCP και θέλουμε να την τοποθετήσουμε σε κατάσταση ακρόασης θα χρησιμοποιήσουμε τη συνάρτηση:

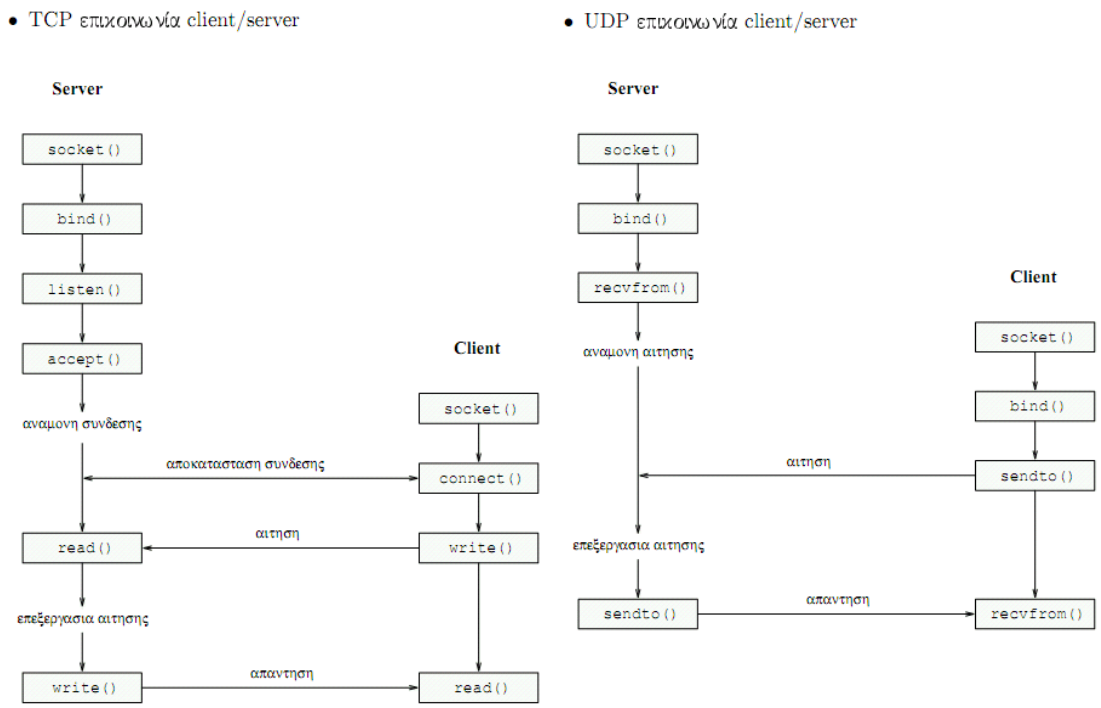
```
int listen(int fd, int queuelength)
```

η οποία ορίζει μια ουρά μήκους queuelength σε ένα server στην οποία μπορούν να συσσωρεύονται αιτήσεις από clients για σύνδεση στην υποδοχή που αντιστοιχεί στον περιγραφέα αρχείου fd.

Αμέσως μετά τη `listen()` στο πρωτόκολλο TCP πρέπει να εκτελεστεί η accept() η οποία αποδέχεται μια σύνδεση που έχει γίνει στη θύρα. Η `accept()` επιστρέφει έναν νέο περιγραφέα αρχείου ο οποίος πρέπει να χρησιμοποιηθεί από το διακομιστή

για επικοινωνία με το client. Δέχεται τρεις παραμέτρους. Η πρώτη είναι ο αρχικός περιγραφέας αρχείου. Η δεύτερη παράμετρος που είναι pointer είναι η θέση που καταγράφεται η διεύθυνση του client που επιστρέφει η accept(). Η τρίτη παράμετρος είναι pointer και γράφει η accept() το μέγεθος της δεύτερης παραμέτρου.

Αφού ολοκληρωθεί η σύνδεση τότε μπορεί να χρησιμοποιηθεί είτε η **`send()`** ή **`write()`** είτε η **`recv()`** ή **`read()`** για να σταλούν ή να ληφθούν δεδομένα από τη συγκεκριμένη υποδοχή. Αν θέλουμε να χρησιμοποιήσουμε έναν πελάτη *(δηλαδή κάποια διεργασία που θα συνδεθεί σε μια θύρα που είναι ήδη σε ακρόαση)* UDP ή TCP τότε η σειρά είναι διαφορετική και είναι η παρακάτω:



6.2 Παραδοτέα C10 και C11

(C10) Να δημιουργήσετε έναν απλό TCP server (**c10.c**), ο οποίος θα βρίσκεται σε κατάσταση ακρόασης στη θύρα TCP 9900. Συγκεκριμένα:

- Θα κάνει κλήση της **`socket()`**.
- Θα δημιουργεί την κατάλληλη δομή της διεύθυνσης για ακρόαση στη θύρα 9999
- Θα καλεί τη **`bind()`**.
- Θα καλεί τη **`listen()`**.

- Θα μπαίνει σε ένα ατέρμονο βρόχο στον οποίο:
    - Θα εκτελεί την `accept()` η οποία θα περιμένει μια καινούργια σύνδεση.
    - Θα διαβάζει τα δεδομένα αμέσως μετά την `accept()` με τη συνάρτηση `read()`.
    - Θα εκτυπώνει τα δεδομένα στο τερματικό.
    - Θα κλείνει τη σύνδεση με `close()`.

(C11) Να δημιουργήσετε έναν απλό TCP echo server (**c11.c**) *(δηλαδή ένα διακομιστή που θα εκτυπώνει ότι του στέλνουμε)*, ο οποίος θα βρίσκεται σε κατάσταση ακρόασης στη θύρα TCP 9999. Συγκεκριμένα:

- θα κάνει κλήση της `socket()`.
- Θα δημιουργεί την κατάλληλη δομή της διεύθυνσης για ακρόαση στη θύρα 9999
- Θα καλεί τη `bind()`.
- Θα καλεί τη `listen()`.
- Θα μπαίνει σε ένα ατέρμονο βρόχο στον οποίο:
    - θα εκτελεί την `accept()` η οποία θα περιμένει μια καινούργια σύνδεση.
    - Θα διαβάζει τα δεδομένα αμέσως μετά την `accept()` με τη συνάρτηση `read()`.
    - Θα γράφει πίσω στο κανάλι τα δεδομένα με τη συνάρτηση `write()`.
    - Θα κλείνει τη σύνδεση με `close()`.

Δοκιμάστε τα παραπάνω προγράμματα ως εξής:

- Εκτελέστε το σε ένα τερματικό.
- Από ένα άλλο τερματικό δώστε "**`telnet localhost 9999`**" για να συνδεθείτε στη διεργασία.
- Πληκτρολογήστε ένα κείμενο και πατήστε enter
- Το κείμενο θα πρέπει να σας εμφανιστεί ξανά ακριβώς από κάτω.