ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Λειτουργικά Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:07

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

- Χειρισμός PID διεργασιών.
- Χειρισμός σημάτων.
- Υλοποίηση χρονομέτρων με σήματα.
- Ιεραρχία διεργασιών.
- Συγχρονισμός διεργασιών με wait.
- **Συναρτήσεις:** getpid(), getppid(), kill(), signal(), sigprocmask(), sigemptyset(), sigdelset(), sigfillset(), sigaddset(), sigismember(), alarm(), wait(), execl().

2. Παραδοτέα

(A) 11 ερωτήσεις
(C) 9 ασκήσεις

3. Χειρισμός PID

Κάθε διεργασία στο POSIX λειτουργικό σύστημα έχει ένα μοναδικό αριθμό. Ο αριθμός αυτός ονομάζεται **PID** *(process ID)*. Προκειμένου μια διεργασία να βρει τον αριθμό που τις έχει ανατεθεί θα πρέπει να καλέσει την κλήση συστήματος `getpid()`. Δείτε το man page αυτής της συνάρτησης. Αν η διεργασία θέλει να μάθει τον αριθμό που έχει ανατεθεί στη γονική διεργασία της τότε θα καλέσει την κλήση συστήματος `getppid()`. Σε περίπτωση που δεν υπάρχει το manpage στο ΛΣ που βρίσκεστε μπορείτε να χρησιμοποιείτε online τη βοήθεια:
http://www.freebsd.org/cgi/man.cgi

Για παράδειγμα για το `getpid()` μπορείτε να γράψετε getpid στο πεδίο *(set command name)* είτε τη σχετική σελίδα:
http://www.freebsd.org/cgi/man.cgi?query=getpid Η συνάρτηση `getpid()` τι τύπο μεταβλητής επιστρέφει όταν κληθεί; _____ **(A1)**

3.1 Παραδοτέο C1

(C1) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c1.c** το οποίο κάνει κλήση των συναρτήσεων `getpid()` και `getppid()` και εκτυπώνει το μήνυμα που ενημερώνει το χρήστη για το PID που έχει το ίδιο το εκτελέσιμο και το PID που έχει η γονική διεργασία. Προσέξτε ότι για να κάνετε κλήση των συγκεκριμένων συναρτήσεων πρέπει να κάνετε include συγκεκριμένα αρχεία. Αυτά τα αρχεία φαίνονται στο αντίστοιχο man page.

Στο C1 όταν εκτελείται εκτυπώνει ένα PID για τη γονική διεργασία. Ποια είναι αυτή η διεργασία *(όνομα διεργασίας)*;_____ **(A2)**

4. Χειρισμός Σημάτων

Τα σήματα είναι ποικίλες ειδοποιήσεις που στέλνει μια διεργασία σε μια άλλη για να την ειδοποιήσει για διάφορα σπουδαία γεγονότα. Διακόπτουν οτιδήποτε κάνει η διεργασία και την αναγκάζουν να τα χειριστεί άμεσα. Κάθε σήμα έχει έναν ακέραιο αριθμό και ένα συμβολικό όνομα *(man signal)*. Μέσα στη διεργασία μπορούμε προαιρετικά να τοποθετήσουμε κώδικα χειρισμού σήματος, διαφορετικά καλείται η προεπιλογή του ΛΣ. Μπορούμε να στείλουμε σήμα σε μια διεργασία, είτε με τον προγραμματιστικό τρόπο *(κλήση της* `kill()`*)*, είτε με το βοηθητικό πρόγραμμα `kill`, είτε με το πληκτρολόγιο για τα σήματα SIGINT *(πλήκτρα CTRL+C)*, SIGSTP *(πλήκτρα CTRL+Z)*, SIGQUIT *(πλήκτρα CTRL+\)*, SIGINFO *(πλήκτρα CTRL+T)*.

*** Όλα τα προγράμματα θα πρέπει να τοποθετούν στο exit code μια τιμή, σχετική με το λόγο τερματισμού. Παραδοσιακά, αν η τιμή επιστροφής είναι 0[1], τότε αυτό υποδηλώνει μια επιτυχή ολοκλήρωση, ενώ αν είναι διαφορετική του 0 τότε σημαίνει μη φυσιολογικό τερματισμό. Για παράδειγμα, δείτε την παράγραφο DIAGNOSTICS από το man grep.***

Αν θέλουμε να στείλουμε κάποιο σήμα *(signal)* σε μια διεργασία τότε θα χρησιμοποιήσουμε τη κλήση συστήματος `kill()` . Δείτε τη βοήθεια για τη συνάρτηση αυτή. Σημειώστε ότι μπορεί να υπάρχει μια βοήθεια για τη `kill` στο τμήμα 1 των man page *(man 1 kill ή από το σχετικό website με section 1 General Commands)* το οποίο περιγράφει το πρόγραμμα kill, ενώ μπορεί να υπάρχει και μια βοήθεια για τη kill στο τμήμα 2 των man page *(man 2 kill ή σχετικό website με section 2 System Calls)* . Υπενθυμίζεται ότι στα Linux/BSD ΛΣ υπάρχουν πολλαπλά τμήματα βοήθειας. Το κάθε τμήμα αναφέρεται σε ένα συγκεκριμένο κομμάτι του ΛΣ. Ο πίνακας δείχνει τα τμήματα που υπάρχουν.

Section	Description
1	General commands
2	System calls
3	C library functions
4	Special files (usually devices, those found in /dev) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

[1] *"...one of the main causes of the fall of the Roman Empire was that, lacking zero, they had no way to indicate successful termination of their C programs.",* Robert Firth

Όταν θέλουμε να χρησιμοποιήσουμε βοήθεια για **κλήσεις συστήματος** τότε θα πρέπει να χρησιμοποιούμε το **τμήμα 2**. Όταν θέλουμε να χρησιμοποιήσουμε βοήθεια για συναρτήσεις της C τότε θα χρησιμοποιήσουμε το τμήμα 3. Για να χρησιμοποιήσουμε ένα τμήμα τοποθετούμε αμέσως μετά την εντολή man τον αριθμό του τμήματος και στη συνέχεια το topic που θέλουμε βοήθεια. Για παράδειγμα:
`man 2 kill` .

Η κλήση συστήματος `kill()` πόσες παραμέτρους απαιτεί; Τι σημαίνει η κάθε παράμετρος; Τι τύπο μεταβλητής επιστρέφει η συνάρτηση; _____ **(A3)**

4.1 Παραδοτέο C2

(C2) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c2.c** *(μπορείτε να επαναχρησιμοποιήσετε κώδικα από το C1)* το οποίο κάνει κλήση των συναρτήσεων `getpid()` και `getppid()` και εκτυπώνει το μήνυμα που ενημερώνει το χρήστη για το PID που έχει το ίδιο το εκτελέσιμο και το PID που έχει η γονική διεργασία. Στη συνέχεια στέλνει στον εαυτό του το σήμα που προκαλεί παύση *(STOP Signal)*. Για να δείτε ποιο είναι το σήμα αυτό χρησιμοποιήστε τη βοήθεια για το signal. Μόλις το εκτελέσετε θα δείτε να εμφανίζεται η λέξη STOPPED *(ενδέχεται να μην εμφανιστεί το μήνυμα σας)*. Πατήστε `fg` για να συνεχίσει η εκτέλεση.

Εκτελέστε μερικές φορές το **C2** και δείτε αν εμφανίζεται πρώτα το μήνυμα που έχετε γράψει και μετά η λέξη Stopped. Μήπως πάντα γίνεται πρώτα Stopped και μετά μόλις πατήσετε fg εμφανίζεται το μήνυμα; Εσείς πρώτα όμως έχετε γράψει το `printf()` ή `puts()` ή `cout` και μετά ακολουθεί το `kill()`.
Γιατί στην εκτέλεση γίνεται αυτό; _____ **(A4)**

Αρκετές φορές είναι επιθυμητό να χειριζόμαστε τα σήματα σε μια διεργασία, προκειμένου να μην εκτελείται η προεπιλογή από το ΛΣ. Για παράδειγμα από το man signal βλέπουμε ότι το σήμα SIGINT *(σήμα με αριθμό 2)* έχει ως προεπιλογή τον τερματισμό της διεργασίας που θα το λάβει. Εκτός από τα σήματα με αριθμό 9, 17, 19, 23 τα οποία δε μπορούν να αγνοηθούν ή να αλλάξουν λειτουργία, όλα τα άλλα σήματα μπορούν να αγνοηθούν. Προκειμένου να χειριστούμε ένα σήμα στη διεργασία μας θα πρέπει:

- να ορίσουμε μια συνάρτηση χειρισμού σήματος
- μέσα στο κυρίως σώμα της διεργασίας να ορίσουμε με τη συνάρτηση signal() ότι το συγκεκριμένο σήμα θα χειρίζεται από τη συνάρτηση στο βήμα 1. Αυτή η ρύθμιση ισχύει μόνο για να χειριστεί μια φορά το συγκεκριμένο σήμα.
- στη συνάρτηση χειρισμού σήματος, εκτός από τις εντολές που θέλουμε να εκτελούνται, θα πρέπει να ξαναρυθμίζεται με τη signal ότι το συγκεκριμένο σήμα χειρίζεται από τη συνάρτηση του βήματος 1.

Από τη σελίδα βοήθειας της συνάρτησης `signal()` τμήματος 3 *(αφού είναι η συνάρτηση της C)* βρείτε και απαντήστε:

Πόσες παραμέτρους απαιτεί η signal(); Τι σημαίνει η κάθε παράμετρος; Τι τύπο μεταβλητής επιστρέφει η συνάρτηση; _____**(A5)**

4.2 Παραδοτέο C3

(C3) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c3.c** το οποίο:

- κάνει κλήση των συναρτήσεων `getpid()` και `getppid()`
- εκτυπώνει το μήνυμα που ενημερώνει το χρήστη για το PID που έχει το ίδιο το εκτελέσιμο και το PID που έχει η γονική διεργασία.
- Ρωτάει το χρήστη αν θέλει να χρησιμοποιήσει μια συνάρτηση χειρισμού σήματος και διαβάζει την απάντηση που πρέπει να είναι **y** ή **Y**, αλλιώς πρέπει να γράψει **n** ή **N**.
- Μέχρι να δώσει μια έγκυρη απάντηση *(y Y n N)* επαναλαμβάνεται η προηγούμενη ερώτηση *(χρησιμοποιήστε δομή* `case`*)*. Μόλις δώσει έγκυρη απάντηση συνεχίζεται η εκτέλεση.
    - Αν ο χρήστης έχει πατήσει n ή N τότε δε θα εγκατασταθεί κάποια συνάρτηση χειρισμού σήματος και η εκτέλεση θα συνεχίσει στο βήμα `for ( ;; ) pause();`
    - Αν ο χρήστης έχει πατήσει y ή Y τότε σε αυτό το σημείο θα εγκατασταθεί συνάρτηση χειρισμού σήματος και θα οριστεί ότι ως signal handler για το σήμα SIGINT, δηλαδή για το CONTROL+C που υπό άλλες συνθήκες τερματίζει τη διεργασία θα είναι η συνάρτηση `catch_sigint`. Η συνάρτηση χειρισμού σήματος `catch_sigint` θα τοποθετηθεί στην αρχή του αρχείου αμέσως μετά τα `#include`.
- Στη συνέχεια ακολουθεί ένας βρόχος που δεν ολοκληρώνεται ποτέ ως εξής:

  ```
  for ( ;; ) pause();
  ```

- Η συνάρτηση χειρισμού σήματος θα εκτυπώνει ένα μήνυμα στην οθόνη "`You cannot kill me with Control+C`" και θα ρυθμίζει πάλι με τη `signal()` ότι το σήμα SIGINT θα πρέπει να χειριστεί από τη `catch_sigint` *(μπορεί να μη χρειάζεται η επανα-ρύθμιση στο σύστημα σας, δείτε τη σχετική σελίδα βοήθειας της signal())*

Η συνάρτηση `pause()` τοποθετεί τη διεργασία στην κατάσταση BLOCKED περιμένοντας κάποιο σήμα. Είναι πολύ καλύτερη υλοποίηση από το να κάνουμε `for(;;){}` δηλαδή να εκτελεί συνεχώς το τίποτα *(no operation)*, που υλοποιεί ένα **busy waiting.**

Εκτελέστε το παραπάνω πρόγραμμα και δοκιμάστε το **Control+C** είτε έχετε ορίσει τη συνάρτηση χειρισμού σήματος είτε όχι και διαπιστώστε ότι η συνάρτηση χειρισμού σήματος έχει ενεργοποιηθεί με επιτυχία.

Αφού το Control+C έχει απενεργοποιηθεί με ποιον τρόπο μπορείτε να τερματίσετε την παραπάνω διεργασία; Δώστε την εντολή/εντολές._____ **(A6)**

Υπάρχουν δυο συναρτήσεις χειρισμού σήματος που μπορούν να χρησιμοποιηθούν, αντί για τις δικές μας.

- Η συνάρτηση **SIG_IGN** είναι η συνάρτηση Signal Ignore, δηλαδή η συνάρτηση που αγνοεί το συγκεκριμένο σήμα.
- Η συνάρτηση **SIG_DFL** είναι η συνάρτηση Signal Default, δηλαδή η συνάρτηση προεπιλογής του λειτουργικού συστήματος.

Μπορούμε να χρησιμοποιήσουμε αυτές τις συναρτήσεις με το συνηθισμένο τρόπο μέσα στη **`signal()`** ως εξής για παράδειγμα: **`signal(SIGINT, SIG_DFL)`**.

Ένα πρόβλημα που υπάρχει με τα σήματα είναι τι γίνεται όταν μια διεργασία χειρίζεται ένα σήμα με το χειριστή σήματος, και έρθει ένα νέο σήμα *(είτε ίδιου τύπου, είτε διαφορετικού)* που απαιτεί χειρισμό. Αυτή η ανεπιθύμητη κατάσταση μπορεί να αποφευχθεί με το να απενεργοποιήσουμε όσα σήματα μπορούμε μέσα στη συνάρτηση χειρισμού σήματος και αφού ολοκληρώσουμε την εργασία μας να άρουμε την εξαίρεση χειρισμού σημάτων. Η συνάρτηση που χρησιμοποιούμε για την αγνόηση των σημάτων είναι η **`sigprocmask().`**

Από τη σελίδα βοήθειας της συνάρτησης **`sigprocmask()`** βρείτε τι τιμή επιστρέφει και ιδιαίτερα τι τιμή επιστρέφει για αποτυχία ή επιτυχία: _____ **(A7)**

Η συνάρτηση **`sigprocmask()`** δέχεται τρεις παραμέτρους.

- **(πρώτη παράμετρος)** `int how`
  ορίζει αν θέλουμε να προσθέσουμε σήματα στη μάσκα *(SIG_BLOCK)*, αφαιρέσουμε σήματα από τη μάσκα *(SIG_UNBLOCK)* ή να αντικαταστήσουμε τελείως την μάσκα με μια άλλη *(SIG_SETMASK).*

- **(δεύτερη παράμετρος)** `const sigset_t *set`
  ορίζει το σετ των σημάτων πάνω στο οποίο θα κάνουμε την ενέργεια της πρώτης παραμέτρου.

- **(τρίτη παράμετρος)** `sigset_t *oldset`
  μπορεί να παραληφθεί. Αν δεν παραληφθεί τότε θα τοποθετηθεί η προηγούμενη μάσκα, δηλαδή η μάσκα πριν την κλήση της sigprocmask().

Προκειμένου να χειριστούμε τα σύνολα sigset χρησιμοποιούμε ειδικές συναρτήσεις. Οι συναρτήσεις είναι **`sigemptyset()`**, για αρχικοποίηση του συνόλου σε άδεια τιμή, **`sigaddset()`**, για προσθήκη σήματος στο σύνολο, **`sigdelset()`** για αφαίρεση σήματος από το σύνολο, **`sigismember()`** για έλεγχο αν ένα σήμα ανήκει μέσα στο σύνολο και **`sigfillset()`** για προσθήκη *(γέμισμα)* όλων των σημάτων στο σύνολο.

Με το παρακάτω παράδειγμα φαίνεται η χρήση των συναρτήσεων χειρισμού συνόλων σημάτων.

```
/* define a new mask set */
sigset_t mask_set;
/* first clear the set (i.e. make it contain no signal numbers) */
sigemptyset(&mask_set);
/* lets add the TSTP and INT signals to our mask set */
sigaddset(&mask_set, SIGTSTP);
sigaddset(&mask_set, SIGINT);
/* and just for fun, lets remove the TSTP signal from the set. */
sigdelset(&mask_set, SIGTSTP);
/* finally, lets check if the INT signal is defined in our set */
if (sigismember(&mask_set, SIGINT)
printf("signal INT is in our set\n");
else
printf("signal INT is not in our set - how strange...\n");
/* finally, lets make the set contain ALL signals available on our
system */
sigfillset(&mask_set)
```

4.3 Παραδοτέο C4

(C4) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c4.c** , το οποίο θα αγνοεί το σήμα CTRL+C για 4 φορές και μετά από 4 φορές θα εκτελείται η προεπιλογή από το ΛΣ:

- στο κυρίως πρόγραμμα θα τοποθετείτε ένα signal handler για το σήμα CTRL+C και ένα signal handler για το σήμα CTRL+\
- Το κυρίως πρόγραμμα θα εκτελεί έναν αέναο βρόχο με την **`pause()`**
- Να ορίσετε μια ακέραια μεταβλητή μετρητή που θα έχει global scope *(θα είναι προσβάσιμη από κάθε συνάρτηση),* αμέσως μετά τα **`#include`**
- Μέσα σε κάθε συνάρτηση χειρισμού σήματος θα πρέπει να τοποθετήσετε τον κώδικα για να αποφευχθεί το πρόβλημα ταυτόχρονων σημάτων που αναφέρθηκε προηγουμένως ως εξής:
    - αρχικά θα ορίζεται ένα σύνολο σημάτων.
    - θα τοποθετούνται όλα τα σήματα σε αυτό το σύνολο.
    - θα καλείται η συνάρτηση με πρώτη παράμετρο την αντικατάσταση της μάσκας, με δεύτερη παράμετρο το σύνολο του προηγούμενου βήματος και ασφαλώς ότι άλλα στοιχεία απαιτούνται.
- Η συνάρτηση χειρισμού του σήματος CTRL+C:
    - θα αυξάνει το μετρητή κατά 1
    - θα ελέγχει αν ο αριθμός είναι ίσος με 4
        - Αν είναι ίσος με 4 τότε θα εκτυπώνει ανάλογο μήνυμα, θα ζητάει το χρήστη να πληκτρολογήσει y ή Y αν θέλει πραγματικά να κάνει έξοδο και αν ναι θα καλεί την **`exit()`** και

θα σταματάει η εκτέλεση. Διαφορετικά θα μηδενίζεται ο μετρητής.

- Αν είναι μικρότερος από 4 τότε θα ενεργοποιεί πάλι με το signal ότι το CTRL+C θα χειριστεί από τη συγκεκριμένη συνάρτηση και θα εμφανίζει το μήνυμα ότι έχει επανατεθεί το signal handler με επιτυχία.

- Η συνάρτηση χειρισμού του σήματος CTRL+\ (`SIGQUIT`):
  - Θα εκτυπώνει την τιμή του μετρητή. Δε θα κάνει τίποτα άλλο.

Συνηθίζεται μετά από μια εντολή εκτύπωσης να τοποθετούμε την εντολή `fflush(stdout);` για να μεταφέρονται στην οθόνη κάθε στοιχείο που υπάρχει στην προσωρινή μνήμη.

5. Υλοποίηση χρονομέτρων με σήματα

Αρκετές φορές μας είναι χρήσιμο ενώ περιμένουμε μια είσοδο από το χρήστη να ενεργοποιήσουμε ένα χρονόμετρο, ώστε αν τύχει ο χρήστης να μη μας δώσει κάποια είσοδο τότε να συνεχίσει η εκτέλεση του προγράμματος μας κανονικά. Για να ενεργοποιήσουμε το χρονόμετρο θα χρησιμοποιήσουμε την κλήση συστήματος `alarm()`.

Από τη σελίδα βοήθειας του `alarm()` απαντήστε: Πόσοι είναι οι είσοδοι στη συνάρτηση `alarm()` και τι σημαίνει η κάθε παράμετρος.
Τι επιστρέφει η συνάρτηση; _____ **(A7)**

Από τη σελίδα βοήθειας του signal απαντήστε: Ποια είναι η προεπιλεγμένη ενέργεια που θα γίνει σε μια διεργασία όταν λάβει το σήμα SIGALRM; _____ **(A8)**

5.1 Παραδοτέο C5

(C5) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c5.c**. Το πρόγραμμα θα χρησιμοποιεί τη συνάρτηση `signal()` για να θέσει μια συνάρτηση χειρισμού του σήματος SIGALRM. Θα εκτυπώνει ένα μήνυμα αν ο χρήστης θέλει να εκτυπωθεί το pid της τρέχουσας διεργασίας ή όχι *(αποδεκτές τιμές y/Y/n/N)*. Στη συνέχεια θα κάνει κλήση της συνάρτησης `alarm()` με τιμή τα 10 sec. Μετά θα ακολουθεί η είσοδος.

- Αν ο χρήστης πατήσει y Y τότε θα απενεργοποιείται το χρονόμετρο *(παράμετρος 0),* θα εκτυπώνεται το pid της διεργασίας και θα τερματίζει
- Αν ο χρήστης πατήσει n N τότε θα σταματάει η εκτέλεση του προγράμματος αλλά θα του εμφανίζει πρώτα τον αριθμό των sec που είχαν απομείνει στο χρονόμετρο *(man alarm)* και θα απενεργοποιείται το χρονόμετρο.

- Αν ο χρήστης πατήσει κάποιο άλλο πλήκτρο τότε θα τίθεται πάλι το alarm σε 10 sec, θα εκτυπώνεται η ερώτηση και θα περιμένει να διαβάσει κάποια είσοδο το πρόγραμμα.
- Στη συνάρτηση χειρισμού του σήματος SIGALRM θα υπάρχει η εκτύπωση του μηνύματος ότι το χρονόμετρο έληξε και θα εκτελείται η exit().

Γιατί στο προηγούμενο πρόγραμμα απενεργοποιούμε το χρονόμετρο όταν δε το χρειαζόμαστε;_____ **(A9)**

6. Ιεραρχία Διεργασιών

Η πρώτη πολύ σημαντική έννοια που υλοποίησε τον πολυπρογραμματισμό και τη δυνατότητα εκτέλεσης πολλαπλών εργασιών στο ΛΣ είναι η έννοια της διεργασίας. Η δημιουργία μιας διεργασίας γίνεται με την κλήση συστήματος `fork()`. Μόλις εκτελεστεί η fork δημιουργείται μια διεργασία ως αντίγραφο της γονικής διεργασίας, που συνεχίζει την εκτέλεση από εκείνη τη γραμμή.

Αν έχουμε δώσει την εντολή: `child_pid = fork();`

τότε στη γονική διεργασία η μεταβλητή **child_pid** θα έχει την τιμή pid του παιδιού και θα είναι μεγαλύτερη από 0, ενώ στη θυγατρική διεργασία η μεταβλητή child_pid θα έχει τιμή 0. Έτσι αμέσως μετά την κλήση της fork συνήθως τοποθετείται ένας έλεγχος if για το αν είναι 0, που σημαίνει ότι ο κώδικας εκείνος θα εκτελεστεί μόνο από τη θυγατρική διεργασία, ή για το αν είναι διαφορετικό από 0, που σημαίνει ότι εκείνος ο κώδικας θα εκτελεστεί μόνο από τη γονική διεργασία.

(C6) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c6.c** το οποίο εκτελεί τη `fork()` και με κατάλληλη δομή ελέγχου εκτυπώνει η θυγατρική διεργασία το μήνυμα “hello from child” ενώ η γονική διεργασία “hello from parent”. Εκτελέστε μερικές φορές το πρόγραμμα και αποφανθείτε αν εκτελείται τις περισσότερες φορές πρώτη η θυγατρική ή όχι.

(C7) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c7.c** το οποίο δημιουργεί μια θυγατρική διεργασία και αναφέρει ότι “`είμαι η θυγατρική διεργασία με PID XXX ενώ η γονική διεργασία μου είναι η YYYY`”, η οποία στη συνέχεια δημιουργεί μια άλλη θυγατρική διεργασία που εκτυπώνει το ίδιο μήνυμα. Η αρχική διεργασία μόλις δημιουργήσει τη θυγατρική διεργασία, θα εκτυπώσει το μήνυμα “Είμαι ο πατέρας με PID XXXX ενώ η θυγατρική μου διεργασία είναι η YYYY”

7. Απλός συγχρονισμός με wait()

Μια απλή μέθοδος συγχρονισμού της γονικής και της θυγατρικής διεργασίας επιτυγχάνεται με την κλήση συστήματος `wait()`. Διαβάστε τη σελίδα βοήθειας για την κλήση συστήματος `wait()` *(προσοχή: τμήμα 2 manpage)*. Αυτή η συνάρτηση κάνει τον πατέρα να περιμένει τον τερματισμό της θυγατρικής διεργασίας και μόνο

όταν τερματίσει η θυγατρική συνεχίζει την εκτέλεση. Αν έχει ήδη τερματίσει η θυγατρική τότε δεν περιμένει καθόλου η γονική.

Με την κλήση της συνάρτησης wait σε ποια κατάσταση από το διάγραμμα καταστάσεων μεταβαίνει η γονική διεργασία και γιατί; _____ **(A10)**

(C8) Δημιουργήστε το πρόγραμμα σε γλώσσα C **c8.c**, το οποίο δημιουργεί 2 διεργασίες που περιμένουν κάποιο τυχαίο χρόνο από 3 έως 12 sec και στη συνέχεια μόλις ολοκληρώνεται η κάθε διεργασία μας αναφέρει ποια διεργασία ολοκληρώθηκε. Συγκεκριμένα:

- Θα εκτελεί μια φορά τη `fork()` και θα ελέγχει τον κωδικό επιστροφής της `fork()` αν έχει δημιουργηθεί με επιτυχία η πρώτη διεργασία.
- Θα εκτελεί ακόμη μια φορά τη `fork()` και θα ελέγχει τον κωδικό επιστροφής της `fork()` αν έχει δημιουργηθεί με επιτυχία η διεργασία.
- Το κυρίως πρόγραμμα αφού επιβεβαιώσει ότι και τα 2 `fork()` έχουν γίνει με επιτυχία θα κάνει κλήση της `wait(&status)` όπου `&status` έχει δηλωθεί ως `int status;` και θα τοποθετηθεί ο κωδικός επιστροφής της κάθε θυγατρικής συνάρτησης.
- Το κυρίως πρόγραμμα θα κάνει κλήση της `wait(&status)` και αμέσως μετά θα υπάρχει μια δομή if που θα ελέγχει κατά πόσο η διεργασία που ενεργοποίησε το wait ήταν η πρώτη διεργασία ή η δεύτερη και θα εκτυπώνει το κατάλληλο μήνυμα μαζί με την τιμή που επέστρεψε η κάθε συνάρτηση *(hint: ελέγξτε το pid της κάθε θυγατρικής με την τιμή που επιστρέφει η* `wait(&status)`*).* Συμβουλευτείτε το man page της `wait()` για τη σωστή σύνταξη.
- Μετά θα υπάρχει πάλι μια κλήση *(για δεύτερη φορά)* της `wait(&status)` και αμέσως μετά θα υπάρχει μια δομή if που θα ελέγχει κατά πόσο η διεργασία που ενεργοποίησε το wait ήταν η πρώτη ή η δεύτερη και θα εκτυπώνει το κατάλληλο μήνυμα μαζί με την τιμή που επέστρεψε η κάθε συνάρτηση.
- Η κάθε διεργασία θα καλεί τη `random()` **ή** `rand()` *(απαιτείται και* `srand()`*)*, ενώ θα ακολουθεί την πράξη modulo10 + 3 *(αν χρησιμοποιήσετε τη* `random()`*)*, για να μας δώσει μια τιμή από 3 έως 12 και στη συνέχεια η συνάρτηση `sleep()` για να γίνει block η συνάρτηση τόσο χρονικό διάστημα όσο έχει υπολογισθεί από 3 έως 12 sec. Στο τέλος θα κάνει `exit()` με τιμή επιστροφής 4.
- Συνήθως μετά τη `fork()` εκτελείται η εντολή `execl()` η οποία αντικαθιστά πλήρως τη διεργασία με κάποια άλλη διεργασία. Έτσι, χρησιμοποιούμε μια δομή if στην οποία ελέγχουμε αν είναι η θυγατρική διεργασία να εκτελεστεί η νέα διεργασία *(εκτελέσιμο αρχείο).*

Η συνάρτηση `execl()` πόσες παραμέτρους παίρνει και τι επιστρέφει;

_____ **(A11)**

Ένα παράδειγμα χρήσης της **`execl()`** είναι το:
`execl("/bin/ls","/bin/ls","/tmp",(char *)0);` .
Με αυτή την εντολή λοιπόν θα αντικατασταθεί πλήρως η τρέχουσα διεργασία με το /bin/ls το οποίο θα εκτελεστεί με την παράμετρο /tmp και στη συνέχεια δε θα υπάρχει άλλη παράμετρος, αφού το (char *) 0, δηλώνει το τέλος των παραμέτρων. Παρατηρήστε ότι πρέπει το εκτελέσιμο πρόγραμμα να τοποθετείται στην πρώτη και στη δεύτερη παράμετρο.

(C9) Δημιουργήστε το πρόγραμμα σε γλώσσα C **C9.c** το οποίο θα δημιουργεί δύο θυγατρικές διεργασίες.

- Η πρώτη θυγατρική θα ανοίγει ένα αρχείο (open()) και θα γράφει write() τους αριθμούς 0 – 9 κάθε 1 sec. Στο τέλος θα κλείνει το αρχείο και θα τερματίζει.
- Η δεύτερη θυγατρική θα εκτελεί την execl με παράμετρο το tail -f για να ακολουθεί τα δεδομένα του αρχείου. Αυτή η διεργασία δεν τερματίζει, γιατί το tail -f δεν τερματίζει.
- Μόλις τερματίσει η πρώτη θυγατρική διεργασία θα ενημερώνεται η γονική (κλήση της **`wait()`**) και θα στέλνει το σήμα SIGKILL στο PID της δεύτερης θυγατρικής διεργασίας, που σημαίνει ότι θα σκοτώνει τη δεύτερη διεργασία.