ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Λειτουργικά Συστήματα

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 2:
Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

1 Ασκήσεις

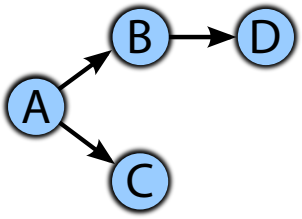
Στην παρούσα άσκηση θα ασχοληθείτε με τη διαχείριση διεργασιών και την επικοινωνία ανάμεσά τους. Η άσκηση έχει δύο μέρη. Στο πρώτο μέρος σας ζητείται πρόγραμμα κατασκευής δεδομένου δέντρου διεργασιών (§ 1.1) κι η επέκτασή του για οποιοδήποτε δέντρο, βάσει αρχείου εισόδου (§ 1.2). Στο δεύτερο μέρος ενεργοποιείτε με συγκεκριμένη σειρά τις διεργασίες, χρησιμοποιώντας σήματα (§ 1.3) και εκτελείτε παράλληλο υπολογισμό μιας αριθμητικής έκφρασης, χρησιμοποιώντας σωληνώσεις για τη διάδοση των ενδιάμεσων αποτελεσμάτων (§ 1.4).

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Ζητείται η κατασκευή προγράμματος το οποίο να δημιουργεί το δέντρο διεργασιών του Σχ. 1. Οι διεργασίες δημιουργούνται και διατηρούνται ενεργές για ορισμένο χρονικό διάστημα, ώστε ο χρήστης να έχει τη δυνατότητα παρατήρησης του δέντρου.

Οι διεργασίες-φύλλα εκτελούν τη κλήση `sleep()`, ενώ οι διεργασίες σε ενδιάμεσους κόμβους περιμένουν τον τερματισμό των διεργασιών-παιδιών τους. Κάθε διεργασία εκτυπώνει κατάλληλο μήνυμα κάθε φορά που περνά από φάση σε φάση (π.χ., εκκίνηση, αναμονή για τερματισμό παιδιών, τερματισμό), ώστε να είναι δυνατή η επαλήθευση της σωστής λειτουργίας του προγράμματος. Για να ξεχωρίσετε τις διεργασίες, κάθε μία θα τερματίζει με διαφορετικό κωδικό επιστροφής: `A` = 16, `B` = 19, `C` = 17, `D` = 13.

Δίδονται βοηθητικά αρχεία κώδικα στον κατάλογο `/home/oslab/code/forktree`:



Σχήμα 1: Δέντρο διεργασιών

- `fork-example.c`: παράδειγμα χρήσης των `fork()`, `wait()`.
- `proc-common.{h,c}`: βοηθητικές συναρτήσεις για το χειρισμό διεργασιών. Η συνάρτηση `explain_wait_status()` εκτυπώνει κατάλληλο μήνυμα ανάλογα με το επιστρεφόμενο `status` της `wait()`. Η συνάρτηση `show_pstree()` εμφανίζει το δέντρο διεργασιών με ρίζα δεδομένη διεργασία.
- `ask2-fork.c`: σκελετός του προγράμματος, απ'όπου μπορείτε να ξεκινήσετε. Κατασκευάζει μια διεργασία-παιδί (τη ρίζα του δέντρου σας), περιμένει να κατασκευαστεί το δέντρο διεργασιών (κοιμάται για λίγο) και καλεί την `show_pstree()`.

Ερωτήσεις:

1. Τι θα γίνει αν τερματίσετε πρόωρα τη διεργασία A, δίνοντας `kill -KILL <pid>`, όπου `<pid>` το Process ID της;
2. Τι θα γίνει αν κάνετε `show_pstree(getpid())` αντί για `show_pstree(pid)` στη `main()`; Ποιες επιπλέον διεργασίες φαίνονται στο δέντρο και γιατί;
3. Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί;

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Ζητείται η κατασκευή προγράμματος που δημιουργεί αυθαίρετα δέντρα διεργασιών βάσει αρχείου εισόδου. Το πρόγραμμα θα βασιστεί σε αναδρομική συνάρτηση, η οποία θα καλείται για κάθε κόμβο του δέντρου. Αν ο κόμβος του δέντρου έχει παιδιά, η συνάρτηση θα δημιουργεί τα παιδιά και θα περιμένει να τερματιστούν. Αν όχι, η συνάρτηση θα καλεί τη `sleep()` με προκαθορισμένο όρισμα.

Το αρχείο εισόδου περιέχει την περιγραφή ενός δέντρου κόμβο προς κόμβο, ξεκινώντας από τη ρίζα. Για κάθε κόμβο δίνεται το όνομά του, ο αριθμός των παιδιών του και τα ονόματά τους. Η περιγραφή ενός κόμβου από τον επόμενο χωρίζεται με κενή γραμμή. Η διαδικασία επαναλαμβάνεται αναδρομικά για κάθε ένα από τα παιδιά του, ενώ οι κόμβοι πρέπει να είναι διατεταγμένοι κατά βάθος. Για παράδειγμα, το το δέντρο διεργασιών του Σχ. 1 προέκυψε από το αρχείο εισόδου του Σχ. 2. Το σύμβολο ↩ δείχνει αλλαγή γραμμής.

```
A ↩ 2 ↩ B ↩ C ↩ ↩ B ↩ 1 ↩ D ↩ ↩ D ↩ 0 ↩ ↩ C ↩ 0 ↩
```

Σχήμα 2: Αρχείο περιγραφής δέντρου

Σας δίνεται βιβλιοθήκη `tree.{h,c}` η οποία αναλαμβάνει να διαβάσει το αρχείο εισόδου και να κατασκευάσει την αναπαράστασή του στη μνήμη. Κάθε κόμβος ορίζεται από τη δομή `struct tree_node`, που περιέχει τον αριθμό των παιδιών `nr_children`, το όνομα του κόμβου και δείκτη προς περιοχή όπου βρίσκονται συνεχόμενα αποθηκευμένες `nr_children` δομές, μία για κάθε κόμβο-παιδί. Η βιβλιοθήκη προσφέρει δύο συναρτήσεις:

- `get_tree_from_file(const char *filename)`: διαβάζει το δέντρο, κατασκευάζει την αναπαράστασή του στη μνήμη κι επιστρέφει δείκτη στη ρίζα.
- `print_tree(struct tree_node *root)`: διατρέχει το δέντρο με ρίζα `root` και εκτυπώνει τα στοιχεία του στο standard output.

Το `tree-example.c` περιέχει παράδειγμα χρήσης της βιβλιοθήκης. Διαβάζει το αρχείο εισόδου, κατασκευάζει το δέντρο και το εμφανίζει στην οθόνη. Για να το εμφανίσει, το διατρέχει ακολουθώντας τους δείκτες `children` των δομών `struct tree_node`.

Ερωτήσεις:

1. Με ποια σειρά εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών; γιατί;

1.3 Αποστολή και χειρισμός σημάτων

Ζητείται η επέκταση του προγράμματος της § 1.2 έτσι ώστε οι διεργασίες να ελέγχονται με χρήση σημάτων, για να εκτυπώνουν τα μηνύματά τους κατά βάθος (Depth-First).

Κάθε διεργασία δημιουργεί τις διεργασίες-παιδιά της και αναστέλλει τη λειτουργία της έως ότου λάβει κατάλληλο σήμα εκκίνησης (`SIGCONT`). Όταν το λάβει, εμφανίζει ενημερωτικό μήνυμα κι ενεργοποιεί διαδοχικά τις διεργασίες-παιδιά: ξυπνάει μια διεργασία, περιμένει τον τερματισμό της και εκτυπώνει ανάλογο διαγνωστικό. Η διαδικασία εξελίσσεται αναδρομικά ξεκινώντας από τη διεργασία-ρίζα του δέντρου. Η αρχική διεργασία του προγράμματος στέλνει `SIGCONT` στη διεργασία-ρίζα, αφού παρουσιάσει το δέντρο διεργασιών στο χρήστη.

Στην περίπτωση του δέντρου του Σχ. 1, τα μηνύματα ενεργοποίησης εκτυπώνονται με τη σειρά `A`, `B`, `D`, `C`.

Υποδείξεις:

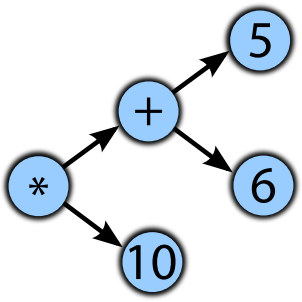
- Μια διεργασία αναστέλλει τη λειτουργία της με χρήση της εντολής `raise(SIGSTOP)`.
- Πριν το κάνει αυτό, έχει βεβαιωθεί ότι όλα τα παιδιά της έχουν αναστείλει τη λειτουργία τους με χρήση της συνάρτησης `wait_for_ready_ children()` που σας δίνεται. Χρησιμοποιήστε τη συνάρτηση `explain_wait_status()` μετά από `wait()` ή `waitpid()`.
- Κατά την ανάπτυξη του προγράμματος μπορείτε να στείλετε μηνύματα χειροκίνητα, με χρήση της εντολής `kill` από κάποιο άλλο παράθυρο. Χρησιμοποιήστε την εντολή `strace -f -p <pid>` για να παρακολουθήσετε τις κλήσεις συστήματος που εκτελεί η διεργασία <pid> κι όλα τα παιδιά της.
- Ξεκινήστε από τον σκελετό `ask2-signals.c` που σας δίνεται.

Ερωτήσεις:

1. Στις προηγούμενες ασκήσεις χρησιμοποιήσαμε τη `sleep()` για τον συγχρονισμό των διεργασιών. Τι πλεονεκτήματα έχει η χρήση σημάτων;
2. Ποιος ο ρόλος της `wait_for_ready_children()`; Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψή της;

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

Ζητείται η επέκταση του προγράμματος της § 1.2 ώστε να υπολογίζει δέντρα που αναπαριστούν αριθμητικές εκφράσεις. Για παράδειγμα, το δέντρο του Σχ. 3 αναπαριστά την έκφραση $10 \times (5 + 6)$.



Σχήμα 3: Δέντρο έκφρασης $10 \times (5 + 6)$

Θεωρήστε ότι τα αρχεία — και κατά συνέπεια τα παραγόμενα δέντρα — έχουν τους παρακάτω περιορισμούς:

- Κάθε μη τερματικός κόμβος[1] έχει ακριβώς δύο παιδιά, αναπαριστά τελεστή και το όνομά του είναι "+" ή "*".
- Το όνομα κάθε τερματικού κόμβου είναι ακέραιος αριθμός.

Η αποτίμηση της έκφρασης θα γίνεται παράλληλα, δημιουργώντας μία διεργασία για κάθε κόμβο του δέντρου. Κάθε τερματική διεργασία επιστρέφει στη γονική την αριθμητική τιμή που της αντιστοιχεί. Κάθε μη-τερματική διεργασία λαμβάνει από τις διεργασίες-παιδιά τις τιμές των υπο-εκφράσεών τους, υπολογίζει την τιμή της και την επιστρέφει στη γονική της διεργασία. Η διεργασία-ρίζα επιστρέφει το συνολικό αποτέλεσμα στην αρχική διεργασία του προγράμματος, η οποία το εμφανίζει στην οθόνη.

Υποδείξεις:

- Χρησιμοποιήστε σωληνώσεις του Unix (pipes) για την επικοινωνία από τις διεργασίες-παιδιά στη γονική διεργασία.
- Κάθε διεργασία οφείλει να περιμένει τον τερματισμό των παιδιών της και να εκτυπώνει κατάλληλο διαγνωστικό, για να εντοπίζετε εγκαίρως προγραμματιστικά σφάλματα.
- Εκτυπώνετε ενδιάμεσα αποτελέσματα από κάθε διεργασία, βοηθά στο debugging.

Ερωτήσεις:

1. Πόσες σωληνώσεις χρειάζονται στη συγκεκριμένη άσκηση ανά διεργασία; Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση για όλες τις διεργασίες παιδιά; Γενικά, μπορεί για κάθε αριθμητικό τελεστή να χρησιμοποιηθεί μόνο μια σωλήνωση;
2. Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορούν να εκτελούνται παραπάνω από μια διεργασίες παράλληλα. Σε ένα τέτοιο σύστημα, τι πλεονέκτημα μπορεί να έχει η αποτίμηση της έκφρασης από δέντρο διεργασιών, έναντι της αποτίμησης από μία μόνο διεργασία;

2 Αναφορά άσκησης

Η προθεσμία για την εξέταση της άσκησης έχει ανακοινωθεί στη σελίδα του μαθήματος.

[1] τερματικοί κόμβοι είναι οι κόμβοι που αντιστοιχούν σε φύλλα

Μετά την εξέταση της άσκησης η κάθε ομάδα θα πρέπει να συντάξει μια (σύντομη) αναφορά και να τη στείλει μέσω e-mail στους βοηθούς εργαστηρίου. Η προθεσμία για την αναφορά είναι μια εβδομάδα μετά την προθεσμία εξέτασης της άσκησης.

Η αναφορά αυτή θα περιέχει:

- Τον πηγαίο κώδικα (source code) των ασκήσεων.
- Την έξοδο εκτέλεσης των προγραμμάτων για διάφορα αρχεία εισόδου, ενδεικτικά.
- Σύντομες απαντήσεις στις ερωτήσεις.

3 Προαιρετικές ερωτήσεις (επιπλέον βαθμοί: 0)

1. Έστω σύστημα με N επεξεργαστές. Ποιοι παράγοντες καθορίζουν αν ο παράλληλος υπολογισμός μιας αριθμητικής έκφρασης όπως στην άσκηση θα είναι ταχύτερος από τον αντίστοιχο υπολογισμό σε μία μόνο διεργασία (σειριακή εκτέλεση).

2. Έστω υβριδική υλοποίηση του παράλληλου υπολογισμού αριθμητικής έκφρασης: από ένα συγκεκριμένο βάθος (μ) και μετά οι υπολογισμοί γίνονται στην ίδια διεργασία (σειριακά). Τι πλεονεκτήματα έχει η υβριδική υλοποίηση; Πως επηρεάζει η παράμετρος μ την επίδοση του υπολογισμού, ως προς τον χρόνο εκτέλεσης; Από ποιους παράγοντες εξαρτάται η βέλτιση τιμή της μ;

   Σκιαγραφήστε την παραπάνω υλοποίηση.