Λειτουργικά Συστήματα Υπολογιστών (τμήμα 1)

2η εργαστηριακή αναφορά

Ομάδα oslaba24	
Αναστάσιος Παπαζαφειρόπουλος	el18079
Νικόλαος Παγώνας	el18175

Πηγαίος Κώδικας

Ο πηγαίος κώδικας όλων των ασκήσεων βρίσκονται στο φάκελο “source-code” με ονόματα:

Άσκηση 1.1 ⟶ ask2-fork.c
Άσκηση 1.2 ⟶ ask2-tree.c
Άσκηση 1.3 ⟶ ask2-signals.c
Άσκηση 1.4 ⟶ ask2-pipes.c

Έξοδος εκτέλεσης των προγραμμάτων για διάφορα αρχεία εισόδου

Τα αρχεία εισόδου που χρησιμοποιήθηκαν βρίσκονται στον φάκελο “input-files” με ονόματα:

proc-1.tree, proc-2.tree, proc-3.tree
expr-1.tree, expr-2.tree, expr-3.tree

Τα πρώτα 3 αρχεία χρησιμοποιήθηκαν για τα προγράμματα **ask2-tree** και **ask2-signals**, ενώ τα υπόλοιπα 3 χρησιμοποιήθηκαν για το πρόγραμμα **ask2-pipes.**

Η έξοδος εκτέλεσης βρίσκεται στον φάκελο “execution-output”, σε αρχεία με ονόματα:

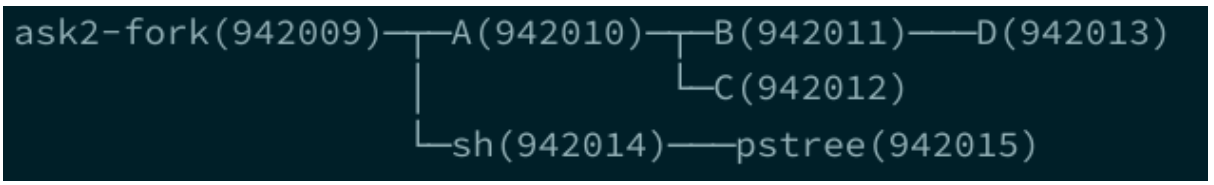
ask2-tree-1, ask2-tree-2, ask2-tree-3
ask2-signals-1, ask2-signals-2, ask2-signals-3
ask2-pipes-1, ask2-pipes-2, ask2-pipes-3

Όπου οι δείκτες 1,2,3 συμβολίζουν το αρχείο που χρησιμοποιήθηκε κάθε φορά, όπως αναγράφεται παραπάνω. Για παράδειγμα, το ask2-tree-1 αναφέρεται στην έξοδο εκτέλεσης του ask2-tree με είσοδο το αρχείο proc-1.tree.

Απαντήσεις στις ερωτήσεις

1.1

1. Αν τερματίσουμε πρόωρα τη διεργασία Α, μέσω της εντολής kill -KILL <pid>, τότε τα παιδιά της Α θα βρεθούν σε κατάσταση zombie, αφού πλέον δεν θα υπάρχει η διεργασία-πατέρας Α, που κανονικά θα έκανε wait. Η πρακτική που ακολουθείται τότε είναι τα παιδιά αυτά να υιοθετούνται από την init, δηλαδή την πρωταρχική διεργασία του λειτουργικού συστήματος (pid = 1), η οποία θα περιμένει να τερματιστούν.
2. Αν στη main έχουμε γράψει `show_pstree(getpid())` αντί για `show_pstree(pid)`,τότε η show_pstree θα μας εμφανίσει το δέντρο διεργασιών με ρίζα τη διεργασία ask2-fork, όπως φαίνεται και στην εικόνα.



Αυτό συμβαίνει διότι η show_pstree παίρνει ως παράμετρο το pid της διεργασίας-ρίζας του δέντρου που θέλουμε να εμφανίσουμε. Έτσι, ενώ το pid αναφέρεται στην διεργασία-παιδί της main (την A), το getpid() επιστρέφει το pid της ίδιας της main. Επομένως, μιας και η main:

1. κάνει fork την A
2. μέσω της εντολής system (που περιέχεται στην συνάρτηση show_pstree) ενεργοποιεί ένα terminal, το οποίο στη συνέχεια καλεί την εντολή pstree

προκύπτει η παραπάνω συμπεριφορά.

3. Αν δεν υπήρχαν όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης, όταν πρόκειται για συστήματα πολλαπλών χρηστών, τότε θα μπορούσε κάποιος χρήστης (εξ' αμελείας ή κακόβουλα) να κάνει πάρα πολλά fork, με αποτέλεσμα να εξαντλήσει όλα τα resources του συστήματος. Για παράδειγμα, ένα πρόγραμμα της μορφής:

```
while (1) fork();
```

Θα έκανε συνεχώς fork μέχρι να μην υπάρχουν άλλοι πόροι στο σύστημα.

1.2

1. Η σειρά εμφάνισης των μηνυμάτων έναρξης και τερματισμού των διεργασιών είναι τυχαία σε αυτή την άσκηση, και διαφέρει μεταξύ διαδοχικών εκτελέσεων. Αυτό συμβαίνει επειδή δεν έχουμε καθορίσει μέσω του προγράμματος κάποια σειρά προτεραιότητας (όπως γίνεται παρακάτω στην 1.3 μέσω σημάτων). Κάθε διεργασία δημιουργεί τα παιδιά της και τα παιδιά δημιουργούν τα δικά τους παιδιά, όμως το ποια διεργασία θα εκτελείται στον επεξεργαστή κάθε φορά είναι δουλειά του λειτουργικού συστήματος, εφόσον δεν έχουμε καθορίσει εμείς κάποιον τρόπο που θα ελέγχει τη σειρά εκτέλεσης. Το μόνο που ξέρουμε είναι ότι, σε περίπτωση που δεν έχουμε κάποιου είδους πρόωρη/αναπάντεχη διακοπή, πρώτα θα εμφανίσουν μήνυμα τερματισμού τα παιδιά και μετά οι γονείς (αν δηλαδή ο A έχει παιδιά τα B και C, τότε πρώτα θα εμφανίσουν μήνυμα τερματισμού τα B και C).

1.3

1. Στην περίπτωση της sleep() δεν είναι γνωστό εκ των προτέρων πόσο ακριβώς χρόνο χρειάζεται να περιμένουμε για να εκτελεστεί το τμήμα του κώδικα που επιθυμούμε, ώστε να τον θέσουμε ως παράμετρο και ενδέχεται είτε να περιμένουμε περισσότερο χρόνο άσκοπα, είτε να περιμένουμε λιγότερο χρόνο από τον απαιτούμενο και να υπάρξει σφάλμα στην εκτέλεση. Επιπλέον, με τη χρήση σημάτων μπορούμε να ασκούμε έλεγχο στη διαδικασία, κάτι το οποίο δε μπορούμε να κάνουμε με την εντολή sleep().
2. Ο ρόλος της wait_for_ready_children() είναι να περιμένει τα παιδιά να αναστείλουν τη λειτουργία τους και να ελέγξει αν αυτό έγινε με επιτυχία. Οπότε, η χρήση της ουσιαστικά μας διαβεβαιώνει ότι αυτό συνέβη επιτυχώς. Αν απουσίαζε θα μπορούσε κάποιο παιδί να μην έχει προλάβει να κάνει raise(SIGSTOP) όταν ο πατέρας του στείλει σήμα SIGCONT. Σε αυτή την περίπτωση, το σήμα SIGCONT θα αγνοηθεί και το συγκεκριμένο παιδί θα μπει σε μια ατέρμονη αναμονή.

1.4

1. Στη συγκεκριμένη άσκηση χρησιμοποιείται μία σωλήνωση ανά σχέση “πατέρας-παιδί”. Δηλαδή, κάθε διεργασία διαθέτει μία σωλήνωση για να επικοινωνεί με τον πατέρα της και δύο για να επικοινωνεί με καθένα από τα παιδιά της, αν υπάρχουν. Ωστόσο, εδώ θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση για όλες τις διεργασίες παιδιά, καθώς οι πράξεις που γίνονται είναι αντιμεταθετικές(πρόσθεση, πολλαπλασιασμός) οπότε δεν παίζει σημαντικό ρόλο με ποια σειρά θα στείλει το κάθε παιδί το αποτέλεσμά του στον πατέρα. Γενικά, αυτό μπορεί να συμβεί προφανώς μόνο για τους αριθμητικούς τελεστές που χρησιμοποιούνται στις αντιμεταθετικές πράξεις.
2. Σε ένα σύστημα πολλαπλών επεξεργαστών, η αποτίμηση της έκφρασης από δέντρο διεργασιών αντί της αποτίμησης από μία μόνο διεργασία δίνει τη δυνατότητα η διάσχιση του δέντρου να «κατανεμηθεί» στους επεξεργαστές. Δηλαδή, οι διεργασίες εκτελούνται σε διαφορετικούς επεξεργαστές, όχι απαραίτητα στον ίδιο. Έτσι έχουμε παράλληλη, και άρα ταχύτερη εκτέλεση.