ΙΩΑΝΝΗΣ ΚΑΠΕΤΑΝΓΕΩΡΓΗΣ **Α.Μ. : 1115201800061**

Κ22: Λειτουργικά Συστήματα (Περιττοί) – Χειμερινό '20

Project 2

Γενικές παρατηρήσεις:

- Όλα τα αρχεία κώδικα είναι πλήρως σχολιασμένα και εξηγείται η κάθε λειτουργία που γίνεται στο κάθε σημείο. Στο documentation αυτό, οι λειτουργιές που γίνονται σε κάθε σημείο εξηγούνται περιγραφικά. Οι προγραμματιστικές λεπτομέρειες εξηγούνται στα σχόλια του κώδικα έτσι ώστε αν γίνει πλήρως κατανοητός.
- Όση μνήμη έχει δεσμευθεί δυναμικά κατά την εκτέλεση του προγράμματος, αποδεσμεύεται πριν το τέλος αυτού με ελεγχόμενο τρόπο και χωρίς να υπάρχουν απώλειες μνήμης η errors.
- Έχουν υλοποιηθεί όλα όσα ζητούνται στην εκφώνηση αλλά και αναφέρθηκαν στο piazza

Αρχεία που περιέχονται στο directory

- root.c : στο αρχείο αυτό περιέχεται η λειτουργία του κόμβου ρίζα. Μέσω αυτού δημιουργείται και το εκτελέσιμο το οποίο "τρέχουμε" την άσκηση.
- middleNodes.c : στο αρχείο αυτό περιέχεται η λειτουργία των ενδιάμεσων κόμβων. Μέσω αυτού δημιουργείται το εκτελέσιμο το οποίο καλείται από τα παιδιά της root μέσω της execl().
- leafNodes.c : στο αρχείο αυτό περιέχεται η λειτουργία των κόμβων φύλλων (workers). Μέσω αυτού δημιουργείται το εκτελέσιμο το οποίο καλείται από τα παιδιά των ενδιάμεσων κόμβων μέσω της execl().
- prime1,c , prime2.c prime 3.c : Τα 3 αρχεία που περιέχουν τις αντίστοιχες 3 συναρτήσεις που χρησιμοποιούνται για την εύρεση πρώτων αριθμών (εξηγούνται παρακάτω αναλυτικά). Στο αρχείο primeFunctions.h υπάρχουν οι δηλώσεις των τριών αυτών συναρτήσεων και γίνονται include στο αρχείο leafNodes.c το οποίο είναι και το μόνο που τις χρησιμοποιεί.
- sortlist.c : στο αρχείο αυτό υλοποιείται μια ταξινομημένη λίστα με τις αντίστοιχες συναρτήσεις που χρειαζόμαστε πάνω στην λίστα. Στο αρχείο sortlist.h υπάρχουν οι απαραίτητες δηλώσεις έτσι ώστε να μπορούν άλλα αρχεία να χρησιμοποιούν την λίστα αυτή και τις συναρτήσεις της. Τα αρχεία που την χρησιμοποιούν είναι τα root.c και middleNodes.c στα οποία γίνεται include το σχετικό .h αρχείο.
- Makefile: εξηγείται αμέσως παρακάτω.

Makefile

- Εφαρμόζεται separate compilation. Αρχικά δημιουργούνται όλα τα .o αρχεία από τα αντίστοιχα .c αρχεία και έπειτα συνδέονται κατάλληλα έτσι ώστε να παραχθούν τα αντίστοιχα εκτελέσιμα.
- Παράγονται 3 εκτελέσιμα προγράμματα. Το myprime το οποίο ουσιαστικά είναι το εκτελέσιμο για τον root κόμβο. Επίσης δημιουργείται το εκτελέσιμο middleNodes το οποίο καλείται μέσω της execl για τους μεσαίους κόμβους δηλαδή τα παιδιά που δημιουργεί ο root. Τέλος παράγεται το εκτελέσιμο leafNodes το οποίο καλείται από τα παιδιά που δημιουργούν οι ενδιάμεσοι κόμβοι έτσι ώστε να εκτελέσουν τις λειτουργίες των workers.
- Έχει προστεθεί και η λειτουργία make clean έτσι ώστε να διαγράφονται όλα τα object files και τα εκτελέσιμα αρχεία.

Εκτέλεση προγράμματος

- Για την εκτέλεση του προγράμματος "τρέχουμε" το εκτελέσιμο του root κόμβου δηλαδή το ./myprime
- Κατά την εκτέλεση πρέπει να δοθούν οι παράμετροι που ζητούνται από την εκφώνηση, δηλαδή:

./myprime -l lb -u ub -w NumofChildren

βέβαια γίνεται ο κατάλληλος έλεγχος σε περίπτωση που δοθούν λάθος οι παράμετροι. Στην περίπτωση αυτή τυπώνεται το αντίστοιχο μήνυμα και τερματίζεται η εκτέλεση.
Σε περίπτωση που το εύρος του διαστήματος που θα δοθεί είναι μικρότερο από NumofChildren^2 (δηλαδή τον αριθμό των κόμβων-φύλλων που θα δημιουργηθούν), εκτυπώνεται αντίστοιχο μήνυμα και τερματίζεται η εκτέλεση.

Δομή για την αποθήκευση δεδομένων

Η δομή που χρησιμοποιώ για να αποθηκεύω τους πρώτους αριθμούς λέγεται node και η δήλωση της υπάρχει στο αρχείο types.h. Χρησιμοποιώ αυτήν την δομή καθώς με κάθε πρώτο αριθμό αποθηκεύω και τον χρόνο που χρειάστηκε ο αλγόριθμος εύρεσης πρώτων αριθμών για να βρει τον συγκεκριμένο αριθμό. Επίσης αποθηκεύω και σε ποιόν κόμβο-φύλλο βρέθηκε κυρίως για λόγους ελέγχού των αποτελεσμάτων.
Επίσης σε αυτήν την δομή αποθηκεύω και άλλες 2 πληροφορίες σε κάποια σημεία του προγράμματος. Η μια πληροφορία είναι ο συνολικός χρόνος που χρειάστηκε ο κάθε κόμβος-φύλλο για να βρει όλους τους πρώτους αριθμούς. Αυτό επιτυγχάνεται μέσω ενός flag (isTotalTime) που είναι χαρακτηριστικό του node. Έτσι αν το isTotalTime είναι 1 σημαίνει ότι το node αυτό περιέχει έναν totalTime ενός κόμβου φύλλου και όχι έναν prime. Ο χρόνος αυτός αποθηκεύεται στην μεταβλητή που αποθηκεύεται ο χρόνος εύρεσης του κάθε prime (όταν το node περιέχει prime). Η άλλη πληροφορία που μπορεί να αποθηκευτεί σε έναν κόμβο node είναι ένας counter. Ομοίως υπάρχει ένα flag (isCount) που δηλώνει ότι η μεταβλητή τύπου node περιέχει έναν counter και όχι έναν prime. Ο counter αυτός χρησιμοποιείται κατά την μεταφορά batches μέσω pipe (εξηγείται στην παράγραφο των ενδιάμεσων κόμβων).

Η επιλογή να χρησιμοποιηθεί μια δομή για την αποθήκευση τριών διαφορετικών πληροφοριών έγινε καθώς και τα τρία δεδομένα μεταφέρονται μέσω write σε ένα pipe από το παιδί στον πατέρα. Έτσι, κατά το διάβασμα των δεδομένων δεν υπάρχει σύγχυση για το αν το δεδομένο που διαβάζουμε είναι ένας prime (μαζί με το time του), ένας counter ή ένα totalTime καθώς και τα 3 είναι τύπου node. Έτσι, κάθε φορά διαβάζουμε από το pipe ένα αντικείμενο τύπου node (ασχέτως το τι περιέχει) και αφού το διαβάσουμε μέσω τον flag βρίσκουμε τι πραγματικά περιέχει και διαχειριζόμαστε ανάλογα το αντικείμενο.

Pipes

Για την μεταφορά δεδομένων μεταξύ διεργασιών χρησιμοποιώ απλά pipes. Επίσης δημιουργώ διαφορετικά pipes για κάθε σετ πατέρα-γονιού. Πιο συγκεκριμένα ο κόμβος root δημιουργεί numOfChildren διαφορετικά pipes, δηλαδή ένα για κάθε παιδί. Αντίστοιχα, ένας ενδιάμεσος κόμβος δημιουργεί numOfChildren διαφορετικά pipes δηλαδή ένα για κάθε παιδί-φύλλο που δημιουργεί.
Επίσης χρησιμοποιείται παντού η συνάρτηση poll έτσι ώστε να κάνουμε monitor τα pipes και να γνωρίζουμε εάν έχει γίνει write σε ένα pipe. Δηλαδή, διαβάζουμε πάντα μόνο από pipes τα οποία είναι έτοιμα για ανάγνωση (δηλαδή έχει γίνει ήδη read κάποιο δεδομένο το οποίο δεν έχουμε διαβάσει). Έτσι, πετυχαίνουμε να μην περιμένουμε ποτέ σε ένα pipe στο οποίο δεν έχει γίνει εγγραφή, ενώ σε άλλα pipes μπορεί να έχουν ήδη γίνει και να είναι έτοιμα για ανάγνωση.
Οι λεπτομέρειες για την χρήση της poll εξηγούνται πλήρως στα σχόλια του κώδικα στα αντίστοιχα σημεία που χρησιμοποιείται.
Η poll χρησιμοποιείται με μηδενικό timeout έτσι ώστε οι διεργασίες να είναι non-blocking.
Το πώς γίνεται το read και write στα pipe αυτά εξηγείται παρακάτω για κάθε διεργασία ξεχωριστά.

Κόμβοι-φύλλα/Workers (leafNodes.c)

Το εκτελέσιμο των κόμβων-φύλλων δέχεται σαν παραμέτρους το id του κόμβοι (π.χ. ο W1 έχει id 1, ο W2 έχει 2 κ.ο.κ.), τον παράγοντα διακλάδωσης (numOfChildren), τον file descriptor του pipe για την επικοινωνία με τον πατέρα (middleNode) καθώς και τα low bound και upper bound τα οποία προσδιορίζουν σε ποιο διάστημα πρέπει να ψάξει για πρώτους αριθμούς. Μέσω ενός if statement επιλέγεται με βάση του id και του branchingFactor για τον ποια από τις 3 συναρτήσεις εύρεσης πρώτων αριθμών πρέπει να χρησιμοποιήσει. Οι αλγόριθμοι αναζήτησης δέχονται σαν όρισμα το εύρος στο οποίο πρέπει να ψάξουν καθώς και μία μεταβλητή time μέσω της οποίας επιστρέφεται ο χρόνος που χρειάστηκε για να βρεθεί ένας αριθμός. Από την εκφώνηση ζητείται με το που βρεθεί ένας πρώτος αριθμός κατευθείαν να στέλνεται στον ενδιάμεσο κόμβο. Έτσι η συνάρτηση εύρεσης πρώτων με το που βρει έναν αριθμό τερματίζεται επιστρέφοντας τον αριθμό αυτόν και τον χρόνο που χρειάστηκε για να βρεθεί. Αποθηκεύουμε τον αριθμό και τον χρόνο του σε μία μεταβλητή τύπου node και την γράφουμε στο pipe μέσω του write. έπειτα συνεχίζουμε ξανακαλώντας την συνάρτηση prime αλλάζοντας το low bound στην επόμενο αριθμό από τον πρώτου αριθμό που βρέθηκε (δηλαδή η συνάρτηση συνεχίζει από εκεί που είχε μείνει). Επαναλαμβάνεται αυτή η διαδικασία έως ότου δοθεί ένα εύρος στο οποίο δεν περιέχεται κανένας πρώτος αριθμός. Στην περίπτωση αυτή ο αλγόριθμος επιστρέφει -1 καθώς και τον χρόνο που χρειάστηκε για να ελέγξει το διάστημα αυτό.
Τέλος, αφού έχουμε βρει (και γράψει στο pipe) όλους τους πρώτους αριθμούς στο διάστημα που ανατέθηκε στο συγκεκριμένο leafNode πρέπει να υπολογίσουμε τον συνολικό χρόνο που χρειάστηκε για την εύρεση όλων των πρώτων αριθμών. Αυτό επιταχύνεται ως εξής: έχουμε έναν αθροιστή και κάθε φορά που βρίσκουμε έναν πρώτο αριθμό στην παραπάνω διαδικασία προσθέτουμε τον χρόνο του αριθμού αυτού στο άθροισμα. Τέλος, προσθέτουμε τον χρόνο που μας επέστρεψε η συνάρτηση κατά την τελευταία κλήση (οπού δεν βρέθηκε κανένας πρώτος αριθμός) και έτσι έχουμε τον συνολικό χρόνο που ξοδεύτηκε για την αναζήτηση πρώτων αριθμών. Χρησιμοποιούμε μια μεταβλητή τύπου node (με το flag isTotalTime=1) για να αποθηκεύσουμε τον χρόνο αυτόν καθώς και το id του συγκεκριμένου κόμβου-φύλου (έτσι ώστε να ξέρουμε σε ποιόν κόμβο αναφέρεται ο χρόνος αυτός) και γράφουμε την μεταβλητή αυτήν στο pipe.
Έτσι έχουμε ολοκληρώσει όλες τις λειτουργίες που χρειάζεται να κάνει ένας κόμβος-φύλλο, οπότε κλείνουμε το pipe (αφού τελειώσαμε το γράψιμο) και τέλος στέλνουμε ένα signal USR1 στην ενδιάμεσο κόμβο μέσω του ppid ο οποίος θα προωθήσει το σήμα αυτό στον κόμβο-ρίζα.

Ενδιάμεσοι κόμβοι (middleNodes.c)

Το εκτελέσιμο των ενδιάμεσων κόμβων δέχεται σαν παραμέτρους το id του κόμβοι (π.χ. ο I1 έχει id 1, ο I2 έχει 2 κ.ο.κ.), τον παράγοντα διακλάδωσης (numOfChildren), τον file descriptor του pipe για την επικοινωνία με τον πατέρα (ρίζα) καθώς και τα low bound και upper bound τα οποία προσδιορίζουν το συνολικό διάστημα στο οποίο πρέπει να ψάξουν τα παιδιά του (κόμβοι-φύλλα). Μέσα σε μια επανάληψη numOfChildren φορών ο ενδιάμεσος κόμβος δημιουργεί τα παιδιά του τα οποία είναι κόμβοι-φύλλα και πρέπει να εκτελέσουν τις αντίστοιχες λειτουργίες. Πιο συγκεκριμένα, σε κάθε επανάληψη αρχικά υπολογίζουμε σε ποιό διάστημα θα εργαστεί το παιδί που πρόκειται να φτιάξουμε (το οποίο είναι υποδιάστημα του διαστήματος που δόθηκε σαν όρισμα στον middleNode). Έπειτα, δημιουργείται το pipe μέσω του οποίου θα επικοινωνεί ο middleNode με το συγκεκριμένο παιδί και στην συνέχεια γίνεται ένα fork έτσι ώστε να δημιουργηθεί η διεργασία παιδί.

Μετά το fork ο πατέρας συνεχίζει με την επόμενη επανάληψη (δηλαδή με την δημιουργία του επόμενου παιδιού) ενώ η διεργασία παιδί ακολουθεί την ακόλουθη διαδικασία για να καλέσει το εκτελέσιμου του leafNode έτσι ώστε να εκτελεστούν οι λειτουργίες που πρέπει να κάνει το παιδί-φύλλο που μόλις δημιουργήθηκε. Μετατρέπει όλες τις παραμέτρους οι οποίες πρέπει να δοθούν στο εκτελέσιμο leafNode (εξηγήθηκαν παραπάνω) σε strings καθώς οι παράμετροι που δίνονται σε ένα εκτελέσιμο είναι πάντα τύπου string, κάνει duplicate το pipe μέσω της dup2(), κλείνει το read του pipe καθώς το παιδί πρόκειται μόνο να γράψει στο pipe και τέλος καλεί το εκτελέσιμο leafNode μέσω του system call execl(). Από εκεί και πέρα ξεκινάει η εκτέλεση του εκτελέσιμου leafNode και εκτελούνται οι λειτουργίες του κόμβου-φύλλου που περιγράφηκαν παραπάνω.
Όσον αφορά τον ενδιάμεσο κόμβο, αφού δημιουργήσει όλα τα παιδιά (κόμβους-φύλλα) (δηλαδή αφού τελειώσει η παραπάνω επαναληπτική διαδικασία) ξεκινάει να διαβάζει τα δεδομένα που γράφουν οι κόμβοι φύλλα στα pipes. Για την αποθήκευση χρησιμοποιούμε μια ταξινομημένη λίστα (sortlist.c , εξηγείται παρακάτω) καθώς θέλουμε αφού διαβάσουμε όλα τα δεδομένα από τους κόμβους φύλλα, οι πρώτοι αριθμοί να είναι ταξινομημένοι έτσι ώστε να σταλθούν ταξινομημένοι στην ρίζα. Έτσι, κάθε φορά που βάζουμε έναν πρώτο αριθμό που μόλις διαβάσαμε στην λίστα, μπαίνει στην σωστή θέση και η λίστα είναι μονίμως ταξινομημένη.
Για να διαβάσουμε από τα pipes, χρησιμοποιούμε την poll έτσι ώστε να κάνουμε monitor τα pipes για το αν έχει γραφτεί κάποιο δεδομένο και να διαβάζουμε μόνο από pipes που έχουν ήδη δεδομένα για ανάγνωση και να μην χρειαστεί να περιμένουμε κάποιο παιδί να γράψει σε περίπτωση που αυτό δεν έχει γράψει ακόμα στο pipe.
Πιο συγκεκριμένα, διαβάζουμε τα δεδομένα με την εξής επαναληπτική διαδικασία:

1. Καλείται η poll και μας επιστρέφει ποια από τα pipes είναι έτοιμα για ανάγνωση.
2. Αν η poll επιστρέψει τουλάχιστον ένα pipe τότε:
   - Διαβάζουμε ένα δεδομένο από το κάθε pipe που μας επέστρεψε η poll και βάζουμε το δεδομένο που μόλις διαβάσαμε στην ταξινομημένη λίστα. Συνεχίζουμε με την επόμενη επανάληψη (δηλαδή ξανά στο βήμα 1).
3. Αν η poll δεν επιστρέψει κανένα pipe τότε
   - Είτε έχουμε διαβάσει όλα τα δεδομένα από όλα τα pipes είτε οι κόμβοι παιδιά αργούν να γράψουν κάποιο δεδομένο λόγω καθυστέρησης του αλγορίθμου εύρεσης πρώτων αριθμών. Για να βρούμε τι από τα δύο ισχύει ελέγχουμε με έναν επιπλέον τρόπο το αν έχουμε τελειώσει το διάβασμα από όλα τα pipes (εξηγείται παρακάτω). Αν έχουμε διαβάσει όλα τα δεδομένα από όλα τα pipes τότε τερματίζουμε την επανάληψη και συνεχίζουμε με τις απόμενες λειτουργίες. Αν όχι, τότε επιστρέφουμε στο βήμα 1 συνεχίζοντας τις επαναλήψεις.

Ο έλεγχος για τον αν έχουμε διαβάσει όλα τα δεδομένα από ένα παιδί γίνεται ελέγχοντας αν έχει σταλθεί το totalTime του παιδιού. Όπως εξηγήθηκε στην παράγραφο των κόμβων-φύλλων αφού ένα παιδί γράψει όλους τους πρώτους αριθμούς στο pipe γράφει και το totalTime σαν τελευταίο δεδομένο. Έτσι, αν δεν έχουμε διαβάσει ακόμα το totalTime ενός παιδιού σημαίνει ότι δεν έχουμε διαβάσει όλα τα δεδομένα που θα γράψει το παιδί αυτό, ενώ αν το έχουμε διαβάσει σημαίνει ότι ήδη έχουμε διαβάσει όλους τους primes που έστειλε όποτε δεν έχουμε να διαβάσουμε κάτι άλλο από το παιδί αυτό. Επομένως κρατάμε για κάθε παιδί την πληροφορία για το αν έχει διαβαστεί το totalTime του. Αν έχουμε διαβάσει τα totalTime όλων των παιδιών σημαίνει ότι έχουμε διαβάσει όλα τα δεδομένα που γράφτηκαν από όλα τα παιδιά. Επέλεξα να χρησιμοποιώ αυτόν τον επιπλέον τρόπο για έλεγχο, πέρα από την poll, έτσι ώστε να λειτουργεί το πρόγραμμα και για μεγάλα διαστήματα όπως [1,1.000.000]. Καθώς σε τέτοιες περιπτώσεις με την poll δεν μπορούμε να ανιχνεύσουμε αν έχουμε τελειώσει την ανάγνωση ή αν απλά αργούν οι συναρτήσεις εύρεσης πρώτων, όσο μεγάλο waiting time και αν βάλουμε στην poll αφού οι αλγόριθμοι εύρεσης πρώτων αριθμών αργούν πολύ.
Αφού διαβάσουμε όλους τους πρώτους αριθμούς (και τα totalTime) από όλους τους κόμβους-φύλλα που δημιούργησε ο συγκεκριμένος middleNode χρειάζεται να τα στείλουμε ταξινομημένα στον κόμβο ρίζα (δηλαδή να τα γράφουμε στο pipe που έχουμε πάρει σαν όρισμα τον συγκεκριμένο middleNode). Τα δεδομένα είναι ήδη ταξινομημένα οπότε μας μένει απλά η εγγραφή τους στο pipe. Η εγγραφή των δεδομένων γίνεται σε batches. Πιο συγκριμένα τοποθετούμε τα δεδομένα σε batches και στέλνουμε ένα-ένα τα batches. Το μέγεθος των batches αυτών γίνεται defined στο αρχείο types.h και μπορεί να αλλαχθεί ανάλογα με την προτίμηση μας και το εύρος που θέλουμε να τρέξουμε το πρόγραμμα. **ΠΡΟΣΟΧΗ!** Η σταθερά που καθορίζει το μέγεθος των batches είναι η TRANSFERBUFFSIZE και όχι η BUFFSIZE που επίσης υπάρχει στο ίδιο αρχείο (εξηγείται και στα αντίστοιχα comment που υπάρχουν στο αρχείο). Σε περίπτωση που θέλουμε τα δεδομένα να μεταφέρονται ένα προς ένα στην ρίζα (δηλαδή όχι σε batches) ορίζουμε το TRANSFERBUFFSIZE = 2. Κάθε batch είναι ένας στατικός πίνακας από nodes μεγέθους TRANSFERBUFFSIZE. Επίσης στην πρώτη θέση του πίνακα σε κάθε batch υπάρχει ένα node που περιέχει ένα count (δηλαδή έχει flag isCount=1) το οποίο μας δείχνει πόσα στοιχεία έχει πραγματικά το batch (καθώς ενδεχομένως να μην είναι γεμάτο) έτσι ώστε να μπορούμε να κάνουμε σωστά την ανάγνωση των δεδομένων του batch στην ρίζα.
Με το που φτιάξουμε ένα batch το γράφουμε στο pipe και συνεχίζουμε με την “κατασκευή” του επόμενου έως ότου βάλουμε και τον τελευταίο prime σε ένα batch. Αφού στείλουμε και το τελευταίο batch, έχουμε τελειώσει όλες τις λειτουργίες του ενδιάμεσου κόμβου οπότε κλείνουμε το pipe που γράφαμε και τερματίζεται η εκτέλεση.

*Όσον αφορά το signal handing των USR1 εξηγείται σε ξεχωριστή παράγραφο παρακάτω.

Κόμβος ρίζα (root.c)

Το εκτελέσιμο του κόμβου-ρίζα δέχεται από την γραμμή εντολών (κατά την εκτέλεση του) τις παραμέτρους που αναφέρονται στην εκφώνηση, δηλαδή τα low bound και upper bound του συνολικού διαστήματος που θα ψάξουμε και τον παράγοντα διακλάδωσής (numOfChildren). Αφού διαβαστούν οι παράμετροι αυτοί ο κόμβος ρίζα αρχικά ακολουθεί παρόμοια διαδικασία με τους ενδιάμεσους κόμβους. Δηλαδή, μέσα σε μία επαναληπτική διαδικασία δημιουργεί ένα ένα τα παιδιά. Πριν δημιουργηθεί το κάθε παιδί, δημιουργείται το pipe μέσω του οποίου θα επικοινωνεί ο κόμβος-ρίζα με το συγκεκριμένο παιδί και υπολογίζεται το διάστημα στο οποίο θα εργαστεί το συγκεκριμένο παιδί. Έπειτα δημιουργείται η νέα διεργασία-παιδί μέσω της fork. Μετά την fork, στην διεργασία-παιδί μετατρέπονται σε string όσες παράμετροι χρειάζονται να δοθούν στο εκτελέσιμο που θα κληθεί για το παιδί (εκτελέσιμο middleNodes), γίνεται duplicate το pipe και και τελικά καλείται το εκτελέσιμο του ενδιάμεσου κόμβου μέσω της execl(). Αντίθετα ο πατέρας συνεχίζει με την δημιουργία του επόμενου παιδιού. Δηλαδή, ακολουθείται ακριβώς η ίδια διαδικασία που εξηγήθηκε και παραπάνω για τους middleNodes.
Αφού ο πατέρας δημιουργήσει όλα τα παιδιά/ενδιάμεσους-κόμβους ξεκινάει να διαβάζει από τα pipes τα δεδομένα που του γράφουν οι ενδιάμεσοι κόμβοι. Ο κόμβος ρίζα είναι μονίμως stand-by ελέγχοντας αν έχει γραφτεί κάτι σε κάποιο pipe. Με το που γραφτεί κάτι σε ένα pipe κατευθείαν ο κόμβος-ρίζα διαβάζει από αυτό το pipe. Αυτό έχει επιτευχθεί μέσω της χρήσης της poll με timeout 0, και έτσι οι διεργασίες είναι non-blocking.
Η διαδικασία που ακολουθείται για την ανάγνωση των δεδομένων από τα pipes και η χρήση της poll είναι η ίδια με την διαδικασία που ακολουθούν οι ενδιάμεσοι κόμβοι και εξηγήθηκε παραπάνω. Η μόνη διαφορά είναι ότι ο κόμβος ρίζα διαβάζει batches από nodes και όχι ένα-ένα τα nodes (όπως κάνουν οι ενδιάμεσοι κόμβοι). Το μέγεθος των batches αυτών είναι σταθερό και ίσο με TRANSFERBUFFSIZE όπως εξηγήθηκε παραπάνω. Έτσι γνωρίζουμε ακριβώς το μέγεθός του πίνακα που θα διαβάσουμε. Επειδή όμως ο πίνακας ενδεχομένως να μην είναι γεμάτος στην πρώτη θέση ενός batch υπάρχει πάντα ένα count που μας υποδεικνύει το πόσα στοιχεία πραγματικά έχει. Με την βοήθεια του count προσπελαύνουμε ένα-ένα τα nodes του batch και τα βάζουμε σε μία ταξινομημένη λίστα.
Μέσα στα batches μεταφέρεται και η πληροφορία για το totalTime του κάθε worker. Οι πληροφορίες αυτές είναι πάντα τα τελευταία δεδομένα που στέλνονται από έναν ενδιάμεσο κόμβο. Αυτό συμβαίνει καθώς στα nodes των totalTime ορίζουμε ως value τον μέγιστο αριθμό που μπορεί να αποθηκεύσει η μεταβλητή τύπου long int. Άρα αφού τα δεδομένα στέλνονται ταξινομημένα, τα values αυτά θα είναι πάντα μεγαλύτερα από όλους τους primes και επομένως θα βρίσκονται πάντα στο τέλος. Μέσω αυτής της πληροφορίας κάνουμε τον επιπλέον έλεγχο για τον αν έχουμε τελειώσει την ανάγνωση από ένα pipe (η οποία εξηγήθηκε στους ενδιάμεσους κόμβους).
Επίσης κάθε φορά που διαβάζουμε ένα totalTime το αποθηκεύουμε σε έναν πίνακα μεγέθους ίσο με τον αριθμό των workers έτσι ώστε να έχουμε την πληροφορία για το πόσο χρόνο χρειάστηκε συνολικά ο κάθε worker.
Αφού τελειώσει η παραπάνω διαδικασία ανάγνωσης έχουμε σε μία ταξινομημένη λίστα όλους τους πρώτους αριθμούς που βρέθηκαν από τους workers και διαβάστηκαν από τα pipes. Άρα, εκτυπώνουμε τα περιεχόμενα της λίστας (δηλαδή τους πρώτους αριθμούς μαζί με τον αντίστοιχο χρόνο τους) και στην συνέχεια αποδεσμεύουμε την λίστα αυτή.
Τέλος, αφού έχουμε τον συνολικό χρόνο για κάθε worker, υπολογίζουμε τον μέγιστο και τον ελάχιστο και τους εκτυπώνουμε. Επίσης εκτυπώνουμε και τον χρόνο του κάθε worker ξεχωριστά (όπως ζητείται από τους κανόνες μορφοποίησης). Οι χρόνοι που τυπώνονται είναι σε milliseconds (όπως υποδεικνύεται από τους κανόνες μορφοποίησης). Επείδη οι χρόνοι που υπολογίζουμε είναι σε seconds, πριν εκτυπωθούν πολλαπλασιάζονται με το 1000 έτσι ώστε να μετατραπούν σε milliseconds.
Τέλος εκτυπώνονται το πόσα σήματα USR1 δέχθηκε ο κόμβος ρίζα (εξηγείται στην επόμενη παράγραφο) αλλά και το πόσοι πρώτοι αριθμοί βρέθηκαν (δεν ζητείται από την εκφώνηση αλλά το πρόσθεσα για ευκολότερο έλεγχο των αποτελεσμάτων).

Signal handling

Για την διαχείριση των σημάτων χρησιμοποιώ την sigaction. Ορίζω την δικιά μου συνάρτηση ως handler μόνο για τα σήματα USR1.
Η διαδικασία που ακολουθώ για να στείλω ένα σήμα από τον κόμβο φύλο στην ρίζα είναι:
- Ο κόμβος-φύλλο στέλνει ένα σήμα USR1 στην στον πατέρα του (ενδιάμεσο κόμβο) μέσω της kill και βρίσκοντας το pid του πατέρα μέσω της getppid()
- Στον ενδιάμεσος κόμβος όταν ληφθεί ένα σήμα USR1 από έναν κόμβο φύλλο εκτελείτε ο hander που έχουμε ορίσει, ο οποίος στέλνει ένα σήμα USR1 στον πατέρα του (με τον ίδιο τρόπο που έστειλε και ο worker) δηλαδή στην ρίζα.
- Τέλος, ο κόμβος ρίζα κάθε φορά που λαμβάνει ένα σήμα USR1 (από έναν ενδιάμεσο κόμβο), εκτελείται ο handler του ο οποίο αυξάνει απλά έναν global μετρητή έτσι ώστε να γνωρίζουμε πόσα σήματα λήφθηκαν συνολικά.

Επέλεξα, να χρησιμοποιήσω την sigaction για να κάνω το signal handling όσο πιο reliable μπορώ. Βέβαια εξακολουθούν να χάνονται κάποια σήματα ορισμένες φορές. Αυτό συμβαίνει κυρίως όταν τρέχουμε το πρόγραμμα σε μικρά διαστήματα όπου τυχαίνει να σταλθούν πολλά σήματα ταυτόχρονα από πολλούς workers και το λειτουργικό τα διαχειρίζεται σαν ένα. Σε εκτελέσεις για μεγάλα διαστήματα τιμών παρατήρησα ελάχιστες (και συνήθως μηδενικές) απώλειες σημάτων.

Αλγόριθμοι εύρεσης πρώτων αριθμών

Οι συναρτήσεις prime1 και prime2 είναι ίδιες με αυτές που μας δόθηκαν με τις εξής 2 διαφορές:
- Κάθε φορά επιστρέφουν τον πρώτο αριθμοί που βρίσκουν στο διάστημα που τους δόθηκε (όπως εξηγήθηκε στην παράγραφο των κόμβων-φύλλων) και πρέπει να ξανακληθούν για το υπόλοιπο διάστημα.
- Για κάθε prime που βρίσκουν επιστρέφουν και τον χρόνο που χρειάστηκε (real time) για να βρεθεί ο αριθμός αυτός μέσω ενός pointer ορίσματος που δέχονται. Ο χρόνος υπολογίζεται μέσω της <sys/times.h> , όπως υποδεικνύεται στα βοηθητικά αρχεία που δόθηκαν

Ο αλγόριθμος prime3 είναι μια μέθοδος εύρεσης πρώτων την οποία υλοποιήσαμε στο μάθημα Εισαγωγή στον προγραμματισμό το έτος 2018-2019 (προφανώς παραλλαγμένη στις απαιτήσεις της εργασίας).
Ο αλγόριθμος έχει τα εξής 2 βασικά χαρακτηρίστηκα:
1. Αποφεύγει να ελέγξει εάν ένας αριθμός είναι πρώτος σε περίπτωση που διαιρείται από το 2 ή από το 3
2. Εάν για έναν αριθμό δεν ισχύει το (1) τότε ψάχνουμε να βρούμε εάν κάποιον αριθμό ο οποίος διαιρεί τον αριθμό που ελέγχουμε. Αποφεύγουμε να ελέγξουμε όλα τα πολλαπλάσια του 2 και του 3 ως πιθανούς διαιρετές καθώς αφού ο αριθμός που ελέγχουμε δεν ανήκει στην κατηγορία (1) οπότε αποκλείεται να διαιρείται από κάποιο πολλαπλάσιο του 2 ή του 3. Επίσης, ψάχνουμε για πιθανούς διαιρέτες μέχρι και την ρίζα του αριθμού που εξετάζουμε (όπως και στον prime2).

Επίσης, για τον prime3 ισχύουν και τα 2 bullets που αναφέρθηκαν παραπάνω για τους prime1 και prime2.

Ταξινομημένη λίστα

Στο αρχείο sortlist.c έχει γίνει μια πολύ απλή υλοποίηση μιας ταξινομημένης λίστας, Η λίστα αυτή περιέχει nodes. Οι λεπτομέρειες της υλοποίησης εξηγούνται στα σχόλια που υπάρχουν στο αρχείο.
Παρέχονται οι εξής συναρτήσεις:
- createList : για την δημιουργία και αρχικοποίηση της λίστας
- insertToList : για να προσθέσουμε ένα νέο node στην λίστα. Το node αυτό τοποθετείται στην σωστή θέση με βάση την τιμή x του node έτσι ώστε η λίστα να παραμείνει ταξινομημένη μετά την εισαγωγή.
- printList : εκτυπώνει για κάθε node που περιέχει η λίστα τον prime που περιέχει το node αυτό και τον αντίστοιχο χρόνο. Η συνάρτηση αυτή χρησιμοποιείται για την τελική εκτύπωση των αποτελεσμάτων, οπότε εκτυπώνει τα δεδομένα (prime και χρόνους) σύμφωνα με τους κάνονες μορφοποίησης που έχουν δοθεί.
- deleteList : αποδεσμεύει την σταδιακά ολόκληρη την λίστα

Το αρχείο sortlist.h περιέχει τις δηλώσεις των παραπάνω συναρτήσεων και γίνεται include στα αρχεία root και middleNodes τα οποία χρησιμοποιούν ταξινομημένη λίστα.