Κωνσταντίνος Σκορδούλης

AM : 1115 2016 00155

Η εργασία έχει υλοποιηθεί σε C++ (and compiled with g++), βέβαια τα περισσότερα I/O γίνονται με συναρτήσεις/βιβλιοθήκες της C . Για την επικοινωνία μεταξύ διεργασιών χρησιμοποιήθηκαν **unnamed pipes** ή απλά pipes. Χρησιμοποιήθηκε κυρίως **blocking I/O** , αλλά έγινε και χρήση **non-blocking I/O**, (με τη χρήση poll(), more on this later).

Πολύ χρήσιμη ήταν η **snprintf()** για μετατροπή οποιουδήποτε σε char* και ακριβή δέσμευση μνήμης με length = snprintf(NULL,0, "%typeofdata", data) +1; (για το '/0') .

Από δομές δεδομένων, χρησιμοποίησα μια απλή συνδεδεμένη λίστα (single linked list σε επίπεδο leaf) για να μαζεύω τα records (called them BRecs = BinaryRecords) και 2 structs, BRecs and SRecs (SRecs = Statistical Records) . Πέρα από αυτά , χρησιμοποίησα πίνακες , όπου το μέγεθος αυτών ορίζεται κατά τη διάρκεια της εκτέλεσης του προγράμματος (more on that later).

Source Files: **myfind.cpp(root), Splitter.cpp, Leaf.cpp, Functions.cpp**

* Γρήγορη αναφορά, το **Functions.cpp,** περιέχει τις **συναρτήσεις μέλη της συνδεδεμένης λίστας**, καθώς και global συναρτήσεις που αφορούν I/O → **read_all()** and **write_all(),** οι οποίες λύνουν πρόβλημα τύπου : να διαβάσει/γράψει λιγότερα bytes από αυτά που του έχουμε πει. Εμπνευσμένο από http://www.linux-mag.com/id/308/ .

Root:

1) Καταρχάς, διαβάζουμε το input (char* argv[]) και κάνουμε τις κατάλληλες αναθέσεις.
2) Ανοίγουμε το αρχείο Records.bin για να δούμε πόσες είναι οι εγγραφές συνολικά.
3) Μετατροπή δεδομένων σε char * με την snprintf()
4) Δημιουργία **pipe() → p[2]** → (sm0->p[2]->root)
5) Δημιουργία νέας διαδικασίας με χρήση **fork()**.
    a. Η υπό-διεργασία sp1/sm0 (subprocess1) , κλείνει το Read End του p[2], γιατί χρειάζεται μόνο το write end για να γράψει τα αποτελέσματα στο root (οι file descriptors διατηρούνται μετά την execlp()).
    b. Ανάλογα με το αν υπήρξε **–s flag**, το process εκτελεί , μία από τις δύο execlp() , οι οποίες έχουν διαφορετικά ορίσματα(ίδιο εκτελέσιμο **splitter**). Τα ορίσματα περιγράφονται και μέσα στο πρόγραμμα(σχόλια).
    c. Σημαντικό, περνάμε σαν όρισμα το p[1] (**Write End** of pipe) (σε char*) για να ξέρει σε ποιον fd να γράψει ο sm0.
6) Η root συνεχίζει, **κλείνοντας το Write End** του p[2], αφού θέλει να διαβάσει από την sm0.
7) Στη συνέχεια αρχίζει να διαβάζει **read_all()** :

a. Πρώτα διαβάζει ένα **int records_found** , πόσα records βρήκαμε➔ **Blocked**. Αν records_found = 0, τότε δεν θα δημιουργήσει κάποιο πίνακα, και **θα τερματίσει,** χωρίς να εκτυπώσει τα στατιστικά.
b. Δημιουργεί πίνακα **BRec matrix[records_found]**, στον οποίο αποθηκεύει τα records του pipe(τα οποία διαβάζει σε μια for).

8) Έχοντας διαβάσει όλα τα records, **waitpid(sm0)**, περιμένει να τελειώσει ο sm0. Και αρχίζει το **Sorting** με την κλήση μιας fork()**:**
   a. Το child(**sort node**) μετατρέπει τα BRecs σε char* (με την χρήση της snprintf)
   b. Γράφει τα BRecs (σε μορφή char*) σε ένα **αρχείο output.txt** (αταξινόμητα αποτελέσματα)
   c. Καλείται η **sort**, με τη βοήθεια της execlp(), και παίρνει σαν όρισμα το output.txt
   d. Τα αποτελέσματα εκτυπώνονται στο tty.
9) Περιμένει να τερματίσει το **sort node** , waitpid(sort node).
10) Διαβάζει το SRec από το pipe, και **κλείνει το p[READ] = p[0]**
11) Τέλος, εκτυπώνει τα στατιστικά που απαιτούνται.

** Για την χρονομέτρηση των προγραμμάτων, χρησιμοποιείται η **gettimeofday()** (end – start) με ακρίβεια **double.**

Splitter: Τα argv[] περιγράφονται στα σχόλια.

Α) Αν υπάρχει **–s flag,** τότε παίζουμε με το **range των παιδιών**, όπως περιγράφεται και στο @87 του Piazza(start_child, end child). Δηλαδή κάθε leaf , γνωρίζει ποιος searcher είναι ($1^{ος}$ $2^{ος}$ $3^{ος}$ κλπ), και στη leaf υπολογίζεται εν τέλει το range of Records.

Β) Αν δεν υπάρχει **–s flag**, τότε παίζουμε απευθείας με το **range των Records,** και όταν φτάσουμε στο leaf γνωρίζουμε κατευθείαν το εύρος αναζήτησης.

Γ) Σε κάθε περίπτωση έχουμε τα εξής διαστήματα: [start, end1] και [start2, end] ,για τα οποία ισχύει : end1 = start + (end-start)/2 και start2 = end1+1 (end-start γιατί αρχίζω να μετράω από το 1)

Δ) Το πρόγραμμα έχει αναδρομική υλοποίηση

1) Ξεκινάω τσεκάροντας αν **h==0**, δηλαδή είμαστε σε leaf_node, όπου καλείται η leaf με τη βοήθεια της execlp() και ανάλογα με την ύπαρξη, ή μη, του **–s flag** , παίρνει και τα ανάλογα ορίσματα.
2) Στη συνέχεια επαναορίζει τα 2 παραπάνω διαστήματα.
3) Φτιάχνει δύο pipes **p1[2]** και **p2[2]**. Left Child ->P1->father, Right Child->P2->father
   a. Άρα ο **father**, κλείνει το p1[WRITE] και το p2[WRITE], κρατώντας τα άλλα δύο, για να μπορεί να διαβάσει τα δεδομένα των παιδιών(p1[READ],p2[READ], WtoGparent = p[WRITE] **Ανοιχτά**) (READ = 0, WRITE = 1 **defined**)

b. Τα **αριστερό παιδί**, αντίστοιχα παίρνει σαν όρισμα το p1[WRITE] για να μπορεί να γράψει στον **father** και ξανά-εκτελείται η splitter, με τις updated τιμές των ορισμάτων.
c. Τα **δεξί παιδί**, αντίστοιχα παίρνει σαν όρισμα το p2[WRITE] για να μπορεί να γράψει στον **father** και ξανά-εκτελείται η splitter, με τις updated τιμές των ορισμάτων.

4) Στη συνέχεια, ο **father** περιμένει να διαβάσει από τον p1[2] , **int records_found1 MONO**→Blocked**.** Στη συνέχεια, διαβάζει και το **int records_found2 MONO.** Αν είναι !=0, τότε δημιουργόυμε τ Εδώ μπορεί να υπάρξει ένα μικρό διάστημα (waiting time) , γιατί μπορεί το p2 να φτάσει πρώτο , αλλά δεν θέτει σε κίνδυνο κανένα write block(pipe overflow), μιας και το read των records_found γίνεται το 1 μετά το άλλο.
5) Τσεκάρουμε αν records_found1 == 0 ή records_found2 == 0.
   a. Αν 1 από τα 2 **είναι 0**, τότε διαβάζουμε τα records του pipe που δεν είναι 0 και τα αποθηκεύουμε στον αντίστοιχο πίνακα(δεν υπάρχει κάποιος κίνδυνος).
   b. Αν και τα 2 **είναι 0**, τότε προσπερνάμε την διαδικασία διαβάσματος records(δεν υπάρχει κάποιος κίνδυνος).
   c. Αν και τα 2 **!= 0**. Για να αποφύγουμε την περίπτωση όπου, διαβάζουμε p1[2]➔ **Blocked**, ενώ παράλληλα γράφονται records στο p2[2] , και αν δεν προλάβουμε θα έχουμε **Writing Block**, υπάρχει μια απλή λύση. Διαβάζουμε ένα record από κάθε pipe **each time** (First Come First Served):
      i. Δύο counters, **i=0, j=0** η οποίοι δείχνουν πόσα records έχουμε διαβάσει μέχρι τώρα (**i<=records_found1** and **j<=records_found2**)
      ii. Αυτά βρίσκονται μέσα σε μια while loop: while((i<records_found1) || (j<records_found2)) → και τα 2 πρέπει να είναι ψευδή δηλαδή (i = records_found1 and j=records_found2)
      iii. Αν έχουμε ήδη διαβάσει τα records_found1 ή records_found2 , απλά διαβάζουμε από το άλλο pipe.
      iv. Το σημαντικό της υπόθεσης , χρήση της **poll(),** βλέπουμε κάθε φορά αν υπάρχουν διαθέσιμα δεδομένα στο αντίστοιχο pipe.
      Το λαμβάνουμε και αυτό σαν παράμετρο για να αποφύγουμε τα παραπάνω προβλήματα.
6) Αφού τα διαβάσαμε όλα τα BRecs, περιμένουμε να τερματίσουν τα παιδιά (κλείνοντας τα αντίστοιχα write ends των pipes).
7) Αν έχουμε διαβάσει BRecs γενικά, τότε γράφουμε στο **WtoGparent fd** (Write to Grand Parent) , τα BRecs.
8) Έχοντας τελειώσει με τα BRecs, μας μένουν τα στατιστικά(Δημιουργία καινούργιου SRec (Statistical Record):
   a. Αν height == 1 , απλά περιμένει τον χρόνο εκτέλεσης των searchers (γονείς των leaf nodes). Διαβάζει 2 double χρόνους.
   b. Αν height > 1 , περιμένει 2 **SRecs** τα οποία συγκρίνει, και μαζί με τον χρόνο εκτέλεσης αυτού του κόμβου(splitter) βγάζει το καινούργιο average των τριών.
9) Το γράφει στο **grandparent** και κλείνει το fd ➔ close(WtoGparent); exit(0);

10) Εν κατακλείδι :
    a. Διαβάζει 2 integers (records_found1, records_found2)
    b. Διαβάζει τα αντίστοιχα pipes για τα BRecs (αν τα παραπάνω είναι !=0)
    c. Γράφει τα BRecs στο **grandparent**.
    d. Διαβάζει τα SRecs από τα pipes (ή τους αντίστοιχους χρόνους εκτέλεσης).
    e. Γράφει το καινούργιο SRec στο **grandparent**

Leaf:

1) Ορίζεται το εύρος αναζήτησης που θα ψάξει(ανάλογα με την ύπαρξη ή μη του –s flag)
2) Γίνεται αναζήτηση στο αρχείο , μετατρέποντας κάθε πεδίο σε char*(**snprintf()**) και χρησιμοποιείται η **strstr()** για την εύρεση του pattern.
    a. Κάθε BRec που έχει το pattern, προστίθεται σε μία συνδεδεμένη λίστα.
3) Γράφουμε όλα τα BRecs του **list** στον **parent fd (WritetoParent)**, καθώς και το **χρόνο εκτέλεσης** .
4) Τέλος κλείνουμε τον WritetoParent.