ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

1η ΕΡΓΑΣΙΑ ΧΕΙΜΕΡΙΝΟΥ ΕΞΑΜΗΝΟΥ 2019-2020

ΕΠΩΝΥΜΟ : ΣΟΛΔΑΤΟΣ

ΟΝΟΜΑ : ΔΗΜΗΤΡΙΟΣ

Α.Μ : 1115201700143

coordinator.c :

Το πρόγραμμα δέχεται ως είσοδο τα εξής :

- Peers : Πλήθος διεργασιών -> argv[1]
- Entries: Πλήθος entry -> argv[2]
- Ratio: Αναλογία readers/writers -> argv[3]
- Iterations: Πλήθος επαναλήψεων -> argv[4]

ShmCreate() : Δημιουργεί shared memory μεγέθους entries*sizeof(Struct entry).

ShmAttach() : Κάνει attach το δημιουργηθεί shared memory στο virtual memory της πατρικής διεργασίας και κατ' επέκτασιν και στα δημιουργηθεντα παιδιά-διεργασίες.

SemInit(): Αρχικοποιεί τους σημαφόρους του shared memory.

Η σχεδιαστική μου επιλογή, για την τυχαία λειτουργία της διεργασίας σε κάθε επανάληψη είναι η εξής.

Αρχικά, αρχικοποιώ την γεννήτρια srand με time(NULL)*getpid() επειδή τα δευτερόλεπτα κατά τα οποία εκτελούνται οι διεργασίες είναι μηδαμινά. Η αρχικοποίηση της με απλό time(NULL) θα επέστρεφε συνεχώς τους ίδιους αριθμούς. Οπότε,

πολλαπλασιάζοντας το με getpid() που κάθε φορά αλλάζει, επιτυγχάνεται η συνεχής τυχαιότητα στο εύρος των αριθμών.

Operation = 1(write) σημαίνει ότι η διεργασία στην εκάστοτε επανάληψη θα κάνει write και **Operation = 0 read.**

Το πόσα read και πόσα write θα κάνει η διεργασία δρομολογείται απ' το πλήθος των iteration καθώς και το ratio.

Για παράδειγμα αν δώσουμε ratio = 2 και iteration = 6 τότε αυτό θα δώσει 4 read και 2 write για κάθε διεργασία με τυχαία σειρά.

Αναλόγως αν δώσουμε ratio 1 και 10 iteration θα δώσει 5write και 5 read. **Επίσης στην υλοποίηση αυτή είναι δυνατόν να προκύψουν 0 read ή 0 write.**

Αφού γίνει η τυχαία επιλογή entry και operation γίνεται και το ανάλογο write/read**. Δεν υπάρχει κάποια ιδιαίτερη υλοποίηση σε αυτό το κομμάτι γιατί ο αλγόριθμος είναι γνωστός ως reader/writer problem.**

ShmDetatch(): Γίνεται Detatch του shared memory.

ShmDelete(): Γίνεται Delete του shared memory.

distribution.c :

Exponential(): Γίνεται τυχαία επιλογή αριθμού με βάση την εκθετική κατανομή. Ο κώδικας δεν προέκυψε από δικιά μου ιδέα αλλά από το διαδίκτυο στο stackoverflow.

Uniform(): Ομοίως αλλά για ομοιόμορφη κατανομή. Χρησιμοποιείται για την επιλογή τυχαίου entry.

Εντολή εκτέλεσης:

-make coordinator

- ./build/coordinator x1 x2 x3 x4 (όπου χ1…χ4 τα peers,entries,ratio,iterations αντίστοιχα)

ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ

- **./build/coordinator 2 1 2 2** .: Σ ε αυτήν την περίπτωση αναμένουμε **μηδενικό μέσο χρόνο** αναμονής καθώς υπάρχουν μόνο readers. Πράγμα που συμβαίνει κατά την εκτέλεση.

- **./build/coordinator 2 1 0 1** : Σε αυτήν την περίπτωση, δηλαδή στην περίπτωση που δώσουμε **0 το ratio οι readers είναι 0**. Και για ένα iteration θα έχουμε 1 write κάθε διεργασία. **Εδώ λοιπόν το λογικό είναι πως οι δύο διεργασίες θα προσπαθήσουν παράλληλα να μπουν στο μοναδικό entry. Άρα αναμένουμε ο χρόνος αναμονής του entry που θα έρθει δεύτερο να είναι ο χρόνος που κάνει sleep() ο πρώτος που μπαίνει. Πράγμα που συμβαίνει κατά την εκτέλεση.**

```
sdi1700143@DIMITRIS-PC:/mnt/c/Users/Dimitris/Desktop/OS$ ./build/coordinator 2 1  0  1
Peers = 2, Entries = 1, ratio = 0.000000 , iterations 1
PID 293   writes 1 reads 0  Avg waiting =  0.0000 sec
PID 294   writes 1 reads 0  Avg waiting =  2.0005 sec
```

- **./build/coordinator 4 5 1 6 :** Εδώ έχουμε 5 entry 1 ratio(δηλαδή writers = readers και 6 επαναλήψεις. Οπότε αναμένουμε κάθε διεργασία 3 read και 3 write. Επίσης επειδή η κατανομή της επιλογής των entries είναι ομοιόμορφη σημαίνει ότι θα έχουμε μικρούς χρόνους αναμονής αφού το entry θα ναι σχετικά διαφορετικό κάθε φορά, με αποτέλεσμα να μην περιμένουν πολύ οι διεργασίες**. Λογικό, επειδή οι διεργασίες είναι 4 και τα entries 5. Σπάνιο να πέσουν δύο διεργασίες στο ίδιο entry σε συγκεκριμένο iteration. ΠΡΑΓΜΑΤΙ :**

```
sdi1700143@DIMITRIS-PC:/mnt/c/Users/Dimitris/Desktop/OS$ ./build/coordinator 4 5  1 6
Peers = 4, Entries = 5, ratio = 1.000000 , iterations 6
PID 326   writes 3 reads 3  Avg waiting =  0.0000 sec
PID 329   writes 3 reads 3  Avg waiting =  0.5001 sec
PID 327   writes 3 reads 3  Avg waiting =  0.0000 sec
PID 328   writes 3 reads 3  Avg waiting =  0.6668 sec


Entry 1 has red 5 times and has been written 0 times
Entry 2 has red 0 times and has been written 3 times
Entry 3 has red 1 times and has been written 3 times
Entry 4 has red 2 times and has been written 3 times
Entry 5 has red 4 times and has been written 3 times
```

- **<u>./build/coordinator 10 3 0.25 10 :</u>** Εδώ περιμένουμε 4απλασιο αριθμό writer. Επίσης αναμένουμε σχετικά μεγάλο χρόνο αναμονής αφού οι peer είναι 10 και τα entries 3. **Λογικό, αφού τα entries θα είναι απασχολημένα για αρκετή ώρα απ' τις πολλές διεργασίες, με αποτέλεσμα να περιμένουν οι διεργασίες που ζητούν είσοδο στο entry.**