Εισαγωγή

Στόχος αυτής της εργασίας είναι να εξοικειωθείτε με τη δημιουργία νημάτων (threads) και τον δικτυακό προγραμματισμό (sockets).

Στα πλαίσια αυτής της εργασίας θα αλλάξετε το κατανεμημένο εργαλείο `travelMonitor` που φτιάξατε στην Άσκηση 2 έτσι ώστε να χρησιμοποιεί threads, ενώ η επικοινωνία με το κεντρικό (parent) process και τα monitors θα γίνεται πάνω από sockets. Η γενική ιδέα και λειτουργικότητα της εργασίας είναι παρόμοια με την Άσκηση 2, δηλαδή και εδώ ένα κεντρικό (parent) process θα δέχεται αιτήματα από πολίτες που θέλουν να ταξιδέψουν σε άλλες χώρες, θα ελέγχει αν έχουν κάνει τον κατάλληλο εμβολιασμό, και θα εγκρίνει αν επιτρέπεται η είσοδος σε μια χώρα από έναν ταξιδιώτη. Συγκεκριμένα, θα υλοποιήσετε την εφαρμογή `travelMonitorClient` η οποία θα δημιουργεί μια σειρά από monitor διεργασίες που, μαζί με την εφαρμογή, θα απαντούν σε ερωτήματα του χρήστη. Σε πολύ γενικές γραμμές (οι λεπτομέρειες πιο κάτω), σε σχέση με τη 2η άσκηση οι διαφορές είναι πως η επικοινωνία του parent process με τα monitors γίνεται μέσω sockets αντί για pipes, και πως τα αρχεία με τα δεδομένα των χωρών διαβάζονται από έναν αριθμό από threads.

Η εφαρμογές travelMonitorClient (50%) και monitorServer (50%)

Η εφαρμογή `travelMonitorClient` θα χρησιμοποιείται ως εξής:

```
./travelMonitorClient –m numMonitors -b socketBufferSize –c cyclicBufferSize –s
  sizeOfBloom –i input_dir –t numThreads
```

όπου:

- Η παράμετρος `numMonitors` είναι ο αριθμός monitorServer διεργασιών που θα δημιουργήσει η εφαρμογή.
- Η παράμετρος `socketBufferSize`: είναι το μέγεθος του buffer σε bytes για διάβασμα πάνω από τα sockets.
- Η παράμετρος `cyclicBufferSize`: είναι το μέγεθος του buffer σε entries (π.χ. 10 file names) για το κυκλικό buffer του `monitorServer` (δείτε πιο κάτω)
- Η παράμετρος `sizeOfBloom` καθορίζει το μέγεθος των bloom filter σε *bytes*. Ενδεικτικό μέγεθος του bloom filter για τα δεδομένα της άσκησης θα είναι της τάξης των 100Kbytes. Η παράμετρος αυτή είναι της ίδιας λογικής ακριβώς με την αντίστοιχη παράμετρο της δεύτερης εργασίας.
- Η παράμετρος `numThreads` καθορίζει τον αριθμό των threads μέσα σε κάθε `monitorServer` process (λεπτομέρειες πιο κάτω)
- Η παράμετρος `input_dir`: είναι ένα directory το οποίο περιέχει subdirectories με τα αρχεία που θα επεξεργάζονται οι monitorServer processes. Κάθε subdirectory θα έχει το όνομα μιας χώρας και θα περιέχει ένα ή περισσότερα αρχεία. Για παράδειγμα, το `input_dir` θα μπορούσε να περιέχει subdirectories `China/` `Italy/` και `France/` τα οποία έχουν τα εξής αρχεία:
-

```
/input_dir/China/China-1.txt
/input_dir/China/China-2.txt
/input_dir/China/China-3.txt
…
/input_dir/Italy/Italy-1.txt
/input_dir/Italy/Italy-2.txt
…
```

```
/input_dir/France/France-1.txt
/input_dir/France/France-2.txt
/input_dir/France/France-3.txt
…
```

- Κάθε αρχείο περιέχει μια σειρά από εγγραφές πολίτων όπου κάθε γραμμή θα περιγράφει την κατάσταση εμβολιασμού ενός πολίτη για κάποια συγκεκριμένη ίωση. Για παράδειγμα, αν τα περιεχόμενα στο αρχείο /input_dir/France/France-1.txt .txt είναι:

```
889 John Papadopoulos France 52 COVID-19 YES 27-12-2020
889 John Papadopoulos France 52 H1N1 NO
776 Maria Tortellini  France 36 SARS-1 NO
125 Jon Dupont France 76 H1N1 YES 30-10-2020
```

σημαίνει πως στη Γαλλία έχουμε έναν πολίτη (John Papadopoulos) που έχει εμβολιαστεί για COVID-19 στις 27-12-2020 αλλά όχι για H1N1, την Maria Tortellini που δεν έχει εμβολιαστεί ακόμα για το SARS-1, και τον Jon Dupont ο οποίος εμβολιάστηκε για το H1N1. Σε αυτή την άσκηση μπορείτε να υποθέσετε πως δε θα υπάρχουν γραμμές που θα δημιουργούν inconsistency. Αν θέλετε να κρατήσετε την ίδια λειτουργικότητα με την άσκηση 2 (οτι δηλαδή τις contradicting εγγραφές τις πετάτε) μπορείτε.Συγκεκριμένα, μια εγγραφή είναι μια γραμμή ASCII κειμένου ίδιου με τη 2η εργασία.

Ξεκινώντας, η εφαρμογή `travelMonitorClient` θα πρέπει να κάνει fork `numMonitors` child processes. Κάθε child process, θα καλεί την exec με εκτελέσιμο αρχείο ένα πρόγραμμα που ονομάζεται `monitorServer` που θα γράψετε και που θα παίρνει ως ορίσματα:

```
monitorServer -p port -t numThreads -b socketBufferSize -c cyclicBufferSize -s sizeOfBloom path1 path2 ... pathn
```

όπου
- `port` είναι το port στο οποίο θα ακούει για επικοινωνία με το parent process. Και το parent process (δηλαδη το travelMonitorClient) και οι monitorServers θα τρέχουν στην ίδια IP του μηχανήματος που ξεκίνησαν την οποία θα πρέπει να βρουν αυτόματα (**προσοχή** θα πρέπει να βρείτε την IP, όχι να χρησιμοποιήσετε την 127.0.0.1).
- `t numThreads` είναι ο αριθμός νημάτων που θα δημιουργήσει το αρχικό νήμα για επεξεργασία των input files.
- Η παράμετρος `socketBufferSize`: είναι το μέγεθος του buffer σε bytes για διάβασμα πάνω από τα sockets.
- Η παράμετρος `cyclicBufferSize`: είναι το μέγεθος του buffer σε entries για το κυκλικό buffer του `monitorServer`. Είναι το μέγεθος ενός **κυκλικού** buffer που θα μοιράζεται ανάμεσα στα νήματα που δημιουργούνται από το monitorServer process. Αναπαριστά τον αριθμό των paths που μπορούν να αποθηκευτούν σε αυτόν (π.χ. 10, σημαίνει 10 file names).
- Η παράμετρος `sizeOfBloom` καθορίζει το μέγεθος των bloom filter σε *bytes*.
- `path1 … pathn` είναι τα μονοπάτια από τις χώρες τις οποίες αναλαμβάνει να εξυπηρετήσει το συγκεριμένο `monitorServer`. Το `travelMonitorClient` θα μοιράζει ομοιόμορφα (με round-robin alphabetically όπως και στη 2η άσκηση) τα subdirectories με τις χώρες που βρίσκονται στο input_dir στα `monitorServer` processes.

Όταν η εφαρμογή (το parent process) τελειώσει τις ενέργειες αρχικοποίησης, θα περιμένει μια σειρά από bloom filters από τα `monitorServer` processes (όπως και στη 2η άσκηση) μέσω των ανοιχτών sockets και

όταν λάβει όλες τις πληροφορίες, θα είναι έτοιμη να δεχθεί είσοδο (εντολές) από τον χρήστη από το πληκτρολόγιο (πιο κάτω για τις εντολές).

Το αρχικό thread σε κάθε `monitorServer` process θα δημιουργεί έναν buffer οπου τα στοιχεία θα είναι τα ονόματα **αρχείων μέσα στα μονοπάτια** `path1...pathn`. Στη συνέχεια, το αρχικό thread θα ξεκινάει `numThreads` threads και θα ακούει στο `port` για συνδέσεις από τον travelMonitorClient για να λάβει ερωτήματα. Κάθε ένα από τα `numThreads` threads θα αφαιρεί από τον buffer ένα αρχείο, θα διαβάζει το αντίστοιχο αρχείο και θα ενημερώνει τις κοινόχρηστες δομές δεδομένων που θα χρησιμοποιούνται για να απαντάμε τα ερωτήματα που θα προωθεί το parent (travelMonitorClient) process. Οι δομές αυτές είναι ίδιες με τη 2η άσκηση, δηλαδή μια δομή θα είναι το bloom filter που θα χρησιμοποιείται για γρήγορο έλεγχο για το αν έχει εμβολιαστεί ένας πολίτης (με αναγνωριστικό `citizenID)` για τη συγκεκριμένη ίωση. Μπορείτε να υποθέσετε πως δε θα υπάρχουν inconsistencies στα input files. Κάθε ένα από τα `numThreads` νήματα ξυπνάει όταν υπάρχει τουλάχιστον ένα μονοπάτι στον buffer. Αν ο αριθμός των αρχείων είναι μεγαλύτερος από το μέγεθος του buffer, τότε θα πρέπει το parent thread να βάλει, όταν υπάρχει χώρος στο buffer, και τα υπόλοιπα αρχεία, έτσι ώστε να διαβαστούν όλα.

Το κάθε `monitorServer` process, αφού έχει τελειώσει την ανάγνωση των input files, θα στέλνει μέσω ενος socket στο parent process ένα bloom filter για κάθε ίωση που θα αναπαριστά το σύνολο των εμβολιασμένων πολιτών των χωρών που διαχειρίζεται το monitorSever process. Όταν το `monitorServer` process τελειώσει την ανάγνωση των αρχείων του και έχει στείλει όλα τα Bloom filters στο parent, ειδοποιεί το parent process μέσω του socket πως είναι έτοιμο να δεχτεί αιτήματα.

Σε αυτή την άσκηση δε χρειάζεται να υλοποιήσετε κάποια λειτουργικότητα για τα signals, δηλαδή δε χρειάζεται ο αντίστοιχος κώδικας για τα signals της Άσκησης 2. Όταν ένα monitorServer process πρόκειται να τερματίσει (μέσω της /exit - δείτε πιο κάτω), τότε τυπώνει σε ένα αρχείο με ονομασία log_file.xxx (όπου το xxx είναι το process ID του) το όνομα των χωρών (των subdirectories) που διαχειρίζεται, το συνολικό αριθμό αιτημάτων που δέχθηκε για είσοδο στις χώρες που διαχειρίζεται, και το συνολικό αριθμό αιτημάτων που εγκρίθηκαν και απορρίφθηκαν. Όπως ακριβώς και στη 2η άσκηση.

Αν το parent process λάβει την εντολή /exit, θα στέλνει μία αντίστοιχη εντολή πάνω από το socket στα `monitorServers`, και θα τυπώνει σε ένα αρχείο με ονομασία log_file.xxx όπου το xxx είναι το process ID του, το όνομα όλων των χωρών (των subdirectories) που συμμετείχαν στην εφαρμογή με δεδομένα, το συνολικό αριθμό αιτημάτων που δέχθηκε για είσοδο στις χώρες, και το συνολικό αριθμό αιτημάτων που εγκρίθηκαν και απορρίφθηκαν. Όπως ακριβώς και στη 2η άσκηση.

Ο χρήστης θα μπορεί να δίνει τις ακόλουθες εντολές στην travelMonitorClient εφαρμογή:

- `/travelRequest citizenID date countryFrom countryTo virusName`
  Η λειτουργία είναι ακριβώς ίδια με τη 2η άσκηση. Η επικοινωνία γίνεται πάνω από socket.

- `/travelStats virusName date1 date2 [country]`
  Η λειτουργία είναι ακριβώς ίδια με τη 2η άσκηση. Η επικοινωνία γίνεται πάνω από socket.

- `/addVaccinationRecords country`
  Με αυτό το αίτημα ο χρήστης έχει τοποθετήσει στο `input_dir/country` ένα ή περισσότερα αρχεία για επεξεργασία. Το parent (`travelMonitorClient`) process στέλνει ειδοποίηση μέσω socket στο `monitorServer` process που διαχειρίζεται τη χώρα `country` ότι υπάρχουν input files για ανάγνωση στον κατάλογο. Το `monitorServer` process διαβάζει ό,τι νέο αρχείο βρει, ενημερώνει τις δομές δεδομένων, και στέλνει πίσω στο parent process, μέσω socket, τα ενημερωμένα του bloom filters που αναπαριστούν το σύνολο πολιτών που έχουν εμβολιαστεί. Πιο συγκεκριμένα, το αρχικό (listening) thread βάζει στον κυκλικό

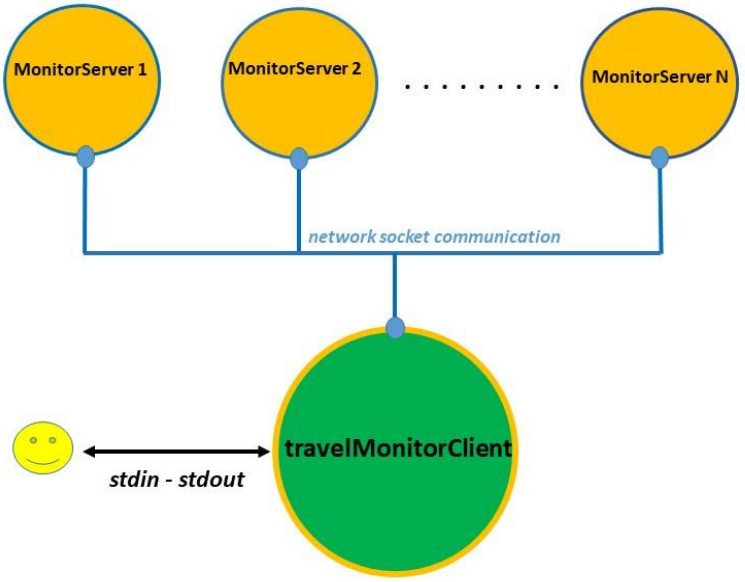
buffer ό,τι νέο αρχείο βρει για διάβασμα και επεξεργασία από τα `numThreads` threads τα οποία θα ενημερώνουν τις δομές δεδομένων. Αφού τελειώσει η επεξεργασία των νέων input files, κάποιο thread θα στέλνει πίσω στο parent process τα ενημερωμένα bloom filters που αναπαριστούν το σύνολο πολιτών που έχουν εμβολιαστεί στις χώρες που διαχειρίζεται το `monitorServer` process.

- `/searchVaccinationStatus citizenID`

Η λειτουργία είναι ακριβώς ίδια με τη 2η άσκηση. Η επικοινωνία γίνεται πάνω από socket.

- `/exit`

Έξοδος από την εφαρμογή. Το parent process στέλνει μία αντίστοιχη εντολή exit (δική σας επιλογή πώς ακριβώς θα είναι) στους `monitorServers`, μέσω socket και τυπώνει σε ένα αρχείο με ονομασία log_file.xxx όπου το xxx είναι το process ID του, το όνομα όλων των χωρών (των subdirectories) που συμμετείχαν στην εφαρμογή με δεδομένα, το συνολικό αριθμό αιτημάτων που δέχθηκε για είσοδο στις χώρες, και το συνολικό αριθμό αιτημάτων που εγκρίθηκαν και απορρίφθηκαν. Πριν τερματίσει, θα απελευθερώνει σωστά όλη τη δεσμευμένη μνήμη. Το format είναι ίδιο όπως και στη 2η άσκηση.



Η επικοινωνία ανάμεσα στο parent process και κάθε MonitorServer λαμβάνει χώρα μέσω sockets (δείτε εικόνα). Το πόσα sockets και το τι πρωτόκολλο επικοινωνίας θα έχετε μεταξύ `travelMonitorClient` και `monitorServers` είναι δική σας σχεδιαστική επιλογή.

Όποιες σχεδιαστικές επιλογές κάνετε, θα πρέπει να τις περιγράψετε σε ένα README αρχείο που θα υποβάλλετε μαζί με τον κώδικά σας.

Παρατηρήσεις

- Η συγκεκριμένη εργασία απαιτεί αρκετή σκέψη και καλό σχεδιασμό όσον αφορά κατανεμημένους πόρους, ειδικά στην περίπτωση των threads που μπορεί να προσπελαύνουν τις ίδιες δομές. Κοινές μεταβλητές που μοιράζονται ανάμεσα σε πολλαπλά νήματα θα πρέπει να προστατεύονται με τη χρήση mutexes. Τονίζεται πως το busy-waiting δεν είναι αποδεκτή λύση για τη παραμονή πρόσβασης σε κάποιον κοινό buffer ανάμεσα στα νήματα των προγραμμάτων σας.
- Η άσκηση δεν περιγράφει όλες τις λεπτομέρειες και τις δομές για τον απλό λόγο πως οι σχεδιαστικές επιλογές είναι αποκλειστικά δικές σας (βεβαιωθείτε φυσικά πως τις περιγράφετε αναλυτικά στο README). Αν έχετε διάφορες επιλογές για κάποιο σημείο της άσκησης σκεφτείτε τα υπέρ και τα κατά, τεκμηριώστε τα στο README, επιλέξτε αυτό που θεωρείτε σωστό και λογικό και περιγράψτε γιατί το επιλέξατε στο README.

Παραδοτέα

- Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματός σας. 1-2 σελίδες ASCII κειμένου είναι αρκετές. Συμπεριλάβετε την εξήγηση και τις οδηγίες για το compilation και την εκτέλεση του προγράμματός σας σε ένα αρχείο README μαζί με τον κώδικα που θα υποβάλετε.
- Ο κώδικας που θα υποβάλετε θα πρέπει να είναι δικός σας. Απαγορεύεται η χρήση κώδικα που δεν έχει γραφεί από εσάς (**αυτό συμπεριλαμβάνει και κώδικα από το Διαδίκτυο!)** .
- Όλη η δουλειά σας (πηγαίος κώδικας, Makefile και README) σε ένα tar.gz file με ονομασία OnomaEponymoProject3.tar.gz. Προσοχή να υποβάλετε μόνο κώδικα, Makefile, README και όχι τα binaries.
- Καλό θα είναι να έχετε ένα backup .tar της άσκησής σας όπως ακριβώς αυτή υποβλήθηκε σε κάποιο εύκολα προσπελάσιμο μηχάνημα (server του τμήματος, private github repository, private cloud).
- Η σωστή υποβολή ενός σωστού tar.gz που περιέχει τον κώδικα της άσκησής σας και ό,τι αρχεία χρειάζονται είναι αποκλειστικά ευθύνη σας. **Άδεια tar/tar.gz ή tar/tar.gz που έχουν λάθος και δεν γίνονται extract δεν βαθμολογούνται**.

Διαδικαστικά

- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com. Η πλήρης διεύθυνση είναι https://piazza.com/uoa.gr/spring2021/k24/home. Η παρακολούθηση του φόρουμ στο Piazza είναι υποχρεωτική.
- Το πρόγραμμά σας θα πρέπει να γραφεί σε C (ή C++). Στην περίπτωσή που χρησιμοποιήσετε C++ δεν μπορείτε να χρησιμοποιήσετε τις έτοιμες δομές της Standard Template Library (STL). Σε κάθε περίπτωση το πρόγραμμά σας θα πρέπει να τρέχει στα Linux workstations του Τμήματος.
- Το πρόγραμμά σας θα πρέπει να κάνει compile στο εκτελέσιμο (`travelMonitorClient`) και (`monitorServer`) να έχει τα ίδια ονόματα για τις παραμέτρους (`-m, -b, -s, -i, -p`) όπως ακριβώς περιγράφεται στην εκφώνηση.
- Ο κώδικάς σας θα πρέπει να αποτελείται από τουλάχιστον δύο (και κατά προτίμηση περισσότερα) διαφορετικά αρχεία. Η χρήση του separate compilation είναι επιτακτική και ο κώδικάς σας θα πρέπει να έχει ένα Makefile.
- Βεβαιωθείτε πως ακολουθείτε καλές πρακτικές software engineering κατά την υλοποίηση της άσκησης. Η οργάνωση, η αναγνωσιμότητα και η ύπαρξη σχολίων στον κώδικα αποτελούν κομμάτι της βαθμολογίας σας.
- Η υποβολή θα γίνει μέσω eclass.

Άλλες σημαντικές παρατηρήσεις

- Οι εργασίες είναι ατομικές.
- Όποιος υποβάλει / παρουσιάσει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο μηδενίζεται στο μάθημα.

- Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ. Τονίζουμε πως θα πρέπει να λάβετε τα κατάλληλα μέτρα ώστε να είναι προστατευμένος ο κώδικάς σας και να μην αποθηκεύεται κάπου που να έχει πρόσβαση άλλος χρήστης (π.χ., η δικαιολογία «Το είχα βάλει σε ένα github repo και μάλλον μου το πήρε από εκεί», δεν είναι δεκτή.)
- Οι ασκήσεις προγραμματισμού μπορούν να δοθούν με καθυστέρηση το πολύ 3 ημερών και με ποινή 5% για κάθε μέρα αργοπορίας. Πέραν των 3 αυτών ημερών, δεν μπορούν να κατατεθούν ασκήσεις.