Σκοπός

Ανάπτυξη μιας multithreaded network εφαρμογής η οποία θα αναλαμβάνει την εκτέλεση εργασιών (jobs) που θα δίνονται μέσω δικτύου από το χρήστη. Η δουλειά της εφαρμογής είναι να βάζει worker threads να αναλαμβάνουν την εκτέλεση εργασιών (jobs) και την επιστροφή των εξόδων τους στο χρήστη, αλλά με έλεγχο του ρυθμού ροής τους (flow control). Για το λόγο αυτό, βασική έννοια είναι ο τρέχων βαθμός παραλληλίας (concurrency). Για παράδειγμα, ας θεωρήσουμε ότι ο βαθμός παραλληλίας είναι 4. Αυτό σημαίνει ότι 4 worker threads θα δημιουργηθούν για να τρέξουν τις 4 πρώτες διαθέσιμες εργασίες που έχουν υποβληθεί στον εξυπηρέτη (ή όσες υπάρχουν, αν έχουν υποβληθεί λιγότερες από 4). Οι υπόλοιπες εργασίες που έχουν υποβληθεί θα μπαίνουν σε εναν ενταμιευτή (buffer). Αν μία εργασία από τις 4 τερματίσει, τότε θα πρέπει ένα worker thread να αρχίσει να εκτελεί την πρώτη από τον ενταμιευτή. Αν ο βαθμός παραλληλίας αλλάξει, η εφαρμογή θα προσαρμοστεί αντίστοιχα, χωρίς να σταματήσει εργασίες που ήδη τρέχουν, σε περίπτωση μείωσης του βαθμού παραλληλίας.

Η εφαρμογή θα προσφέρει τη λειτουργικότητά της στο χρήστη μέσω δύο διεργασιών,

- του jobCommander μέσω του οποίου ο χρήστης αλληλεπιδρά με την εφαρμογή και
- του jobExecutorServer που αναλαμβάνει την διαχείριση και εκτέλεση των εντολών

jobCommander

Ο jobCommander δίνει τη δυνατότητα στον χρήστη να αλληλεπιδράσει με τον jobExecutorServer μέσω απλών εντολών (commands). Οι εντολές δίνονται ως ορίσματα κατά την κλήση του jobCommander και στέλνονται μέσω διαδικτύου στον jobExecutorServer. Συγκεκριμένα, ο jobCommander θα παίρνει τα ακόλουθα ορίσματα:

```
prompt> ./jobCommmander [serverName] [portNum] [jobCommanderInputCommand]
```

όπου:

- **serverName:** το όνομα μηχανήματος που τρέχει ο jobExecutorServer στον οποίο θα συνδεθεί. Θα πρέπει να γίνεται resolve το name, οχι IPs.
- **portNum:** αριθμός θύρας που ακούει ο jobExecutorServer
- **jobCommanderInputCommand:** μια από τις εντολές που δέχεται ο jobCommander με τη μορφή που περιγράφεται παρακάτω.

Ο κύκλος ζωής του jobCommander είναι μία μόνο εντολή (command). Δηλαδή, η εκτέλεσή του ολοκληρώνεται όταν μεταβιβάσει με επιτυχία την εντολή στον jobExecutorServer και λάβει το μήνυμα απάντησης (το οποίο συμπεριλαμβάνει πιθανόν την εξοδο της εργασίας στην περίπτωση της issueJob) από τον jobExecutorServer.

issueJob <job>:

Μέσω αυτής της εντολής εισάγονται εργασίες (jobs) στο σύστημα που πρόκειται να εκτελεστούν. Η εργασία είναι μια συνηθισμένη γραμμή εντολών Unix. Ο jobExecutorServer αναθέτει ένα μοναδικό αναγνωριστικό <jobID, job> στην εργασία. Το jobID θα πρέπει να ακολουθεί το πρότυπο job_XX όπου XX ένας αύξων μοναδικός αριθμός (χωρίς leading zeros) που αυξάνεται για κάθε νέα εργασία που δέχεται ο jobExecutorServer. Επιστρέφεται μήνυμα στον jobCommander ως εξής:

JOB <jobID, job> SUBMITTED

και στη συνέχεια, επιστρέφεται η έξοδος της εντολής όταν τρέξει και τερματίσει.

Για τις ομάδες νημάτων, το μήνυμα "JOB <jobID, job> SUBMITTED" και η έξοδος της εργασίας που έχει υποβληθεί για εκτέλεση θα ερθουν ασύγχρονα στον πελάτη (αλλωστε αποστέλλονται από διαφορετικά νήματα). Ο πελάτης job Commander θα περιμένει και τα δύο για να να τερματίσει. Ο jobCommander εκτυπώνει το μήνυμα και την έξοδο της εργασίας στην κονσόλα με τη σειρά και τον χρόνο που του έρχονται σε αυτόν.

setConcurrency <N>:

Η παράμετρος αυτή θέτει το βαθμό παραλληλίας, δηλαδή το μέγιστο αριθμό ενεργών νημάτων (worker threads) που εκτελούν εργασίες ανά πάσα χρονική στιγμή. Η προκαθορισμένη τιμή είναι 1. Η εντολή μπορεί να αποσταλεί και κατά τη διάρκεια εκτέλεσης εργασιών και θα αλλάξει τη συμπεριφορά του εξυπηρέτη από την λήψη της και μετά. Επιστρέφεται μήνυμα στον jobCommander ως εξής

CONCURRENCY SET AT N

Ο jobCommander εκτυπώνει το μήνυμα στην κονσόλα.

stop <jobID>:

Αφαιρείται από τον ενταμιευτή των υπό εκτέλεση εργασιών, η εργασία με το συγκεκριμένο αναγνωριστικό jobID. (Σε αντίθεση με την πρώτη εργασία, η εντολή stop είναι μόνο για εργασίες στον ενταμιευτή. Εργασίες που ήδη εκτελούνται θα τις αφήνουμε να τερματίζουν κανονικά). Επιστρέφεται μήνυμα στον jobCommander ως εξής:

JOB <jobID> REMOVED

ή

JOB <jobID> NOTFOUND

αν δε βρέθηκε στον ενταμιευτή (δεν ενδιαφέρει αν είναι λάθος, τρέχει ή έχει ήδη τρέξει.) Ο jobCommander εκτυπώνει το μήνυμα στην κονσόλα.

poll:

Για κάθε εργασία που είναι σε κατάσταση αναμονής (queued) δηλαδή στον ενταμιευτή, επιστρέφει το ζεύγος <jobID,job> το οποίο ο jobCommander εκτυπώνει στην κονσόλα. (Όπως με την εντολή stop, η εντολή poll είναι μόνο για εργασίες στον ενταμιευτή).

Ο jobCommander εκτυπώνει τα ζεύγη, ένα ανά γραμμή, στην κονσόλα.

exit:

Τερματίζεται η λειτουργία του jobExecutorServer. Ο jobExecutorServer πριν τερματίσει επιστρέφει στον jobCommander το μήνυμα:

SERVER TERMINATED

Ο jobCommander εκτυπώνει το μήνυμα στην κονσόλα.

Σενάριο εκτέλεσης

Ακολουθεί ένα σενάριο εκτέλεσης του jobCommander:

jobCommander linux01.di.uoa.gr 2035 issueJob ls -l /path/to/directory1
jobCommander linux01.di.uoa.gr 2035 setConcurrency 2
jobCommander linux01.di.uoa.gr 2035 issueJob wget aUrl
jobCommander linux01.di.uoa.gr 2035 issueJob grep "keyword" /path/to/file1 jobCommander linux01.di.uoa.gr 2035 issueJob cat /path/to/file2
jobCommander linux01.di.uoa.gr 2035 issueJob ./progDelay 20
jobCommander linux01.di.uoa.gr 2035 issueJob ./executable arg1 arg2

jobCommander linux01.di.uoa.gr 2035 poll
jobCommander linux01.di.uoa.gr 2035 stop job_1
jobCommander linux01.di.uoa.gr 2035 stop job_15
jobCommander linux01.di.uoa.gr 2035 exit

Multi-threaded network jobExecutorServer

Ο πολυνηματικός εξυπηρετητής δικτύου(σε γλώσσα C) που θα ονομάσετε jobExecutorServer θα εκτελείται από τη γραμμή εντολής ως εξής:

```
prompt> ./jobExecutorServer  [portNum] [bufferSise] [threadPoolSize]
```

όπου τα ορίσματα είναι:

- portnum: ο αριθμός θύρας που ακούει ο εξυπηρετητής.
- bufferSize: το μέγεθος ενός ενταμιευτή που θα κρατάει τα στοιχεία των πελατών που περιμένουν να εξυπηρετηθούν. Πρέπει να είναι > 0.
- threadPoolSize: Ο συνολικός αριθμός των worker threads στο thread pool.

Για παράδειγμα, αν τρέξετε το πρόγραμμα σας με τα ακόλουθα ορίσματα:

```
prompt> ./jobExecutorServer 7856 8 5
```

τότε ο εξυπηρετητής θα ακούει στη θύρα 7856 και θα κατασκευάσει/χρησιμοποιήσει έναν ενταμιευτή που κρατάει μέχρι 8 συνδέσεις/εργασίες που περιμένουν εξυπηρέτηση. Επίσης θα δημιουργηθούν 5 worker threads στο thread pool

Ο jobExecutorServer θα έχει τρία είδη νημάτων, το αρχικό

- main thread
- controller threads και

● worker threads Main thread

Το αρχικό main thread θα δημιουργεί threadPoolSize worker threads. Θα ακούει για συνδέσεις από jobCommander πελάτες, δηλαδή θα δέχεται συνδέσεις με την accept κλήση συστήματος. Όταν συνδέεται ένας jobCommander πελάτης το main thread θα δημιουργεί ένα controller thread.

Controller threads

Τα controller threads μπορεί να είναι παραπάνω από ένα. Κάθε φορά που πραγματοποιείται μία σύνδεση ξεκινά ένα νέο Controller thread το οποίο κάποια στιγμή - όταν ολοκληρώσει την δράση του - θα τελειώσει. Το κάθε controller thread θα διαβάζει από τη σύνδεση την εντολή του πελάτη jobCommander και θα εκτελεί την εντολή ή θα εισάγει στον κοινόχρηστο buffer την εργασία για εκτέλεση, όπως περιγράφεται παρακάτω για την κάθε τύπο εντολής:

Λειτουργία Controller thread για εντολή issueJob

Το controller thread θα δημιουργεί ένα μοναδικό jobID για την εργασία που ζητάει ο πελάτης να τρέξει και θα τοποθετεί σε έναν ενταμιευτή συγκεκριμένου μεγέθους (που ορίζεται από το bufferSize) την τριπλετα <jobID, job, clientSocket>, όπου το jobID είναι το μοναδικό αναγνωριστικό για την εργασια που ζητάει ο πελάτης, το job είναι η συγκεκριμένη εργασία, και το clientSocket είναι ο περιγραφέας αρχείου (file descriptor) για το socket που επιστρέφει η accept() για επικοινωνία με τον συγκεκριμένο πελάτη. Στη συνέχεια το controller thread επιστρέφει στον πελάτη το μήνυμα:

JOB <jobID, job> SUBMITTED

Λειτουργία Controller thread για εντολή setConcurrency

Το controller thread ενημερώνει την τιμή μιας κοινόχρηστης μεταβλητής concurrencyLevel που αρχικά έχει τιμή 1.

Στέλνει στον πελάτη το μήνυμα **CONCURRENCY SET AT N**

Λειτουργία Controller thread για εντολή stop

Αν το αίτημα είναι stop <jobID>, το controller thread αφαιρεί το αντίστοιχο job από την κοινόχρηστη ουρά και στέλνει στον πελάτη το μήνυμα

JOB <jobID> REMOVED

ή

JOB <jobID> NOTFOUND οπως περιγράφηκε ανωτέρω.

Λειτουργία Controller thread για εντολή poll

Το controller thread διατρέχει τον κοινόχρηστο ενταμιευτή, μαζεύει τα ζεύγη <jobID, job> των εντολών του συγκεκριμένου πελάτη, και τα επιστρέφει στον πελάτη σε μήνυμα.

Λειτουργία Controller thread για εντολή exit

Το controller thread τερματίζει τον jobExecutorServer. Ο τερματισμός του jobExecutorServer θα πραγματοποιηθεί μετά την ολοκλήρωση όλων των εργασιών που τρέχουν και την επιστροφή των εξόδων τους στον αντίστοιχο πελάτη Ο ενταμιευτής αδειάζει χωρίς να τρέξουν οι εργασίες που περιέχει και οι πελάτες που έχουν υποβαλει τις εργασίες θα ενημερώνονται με μήνυμα

SERVER TERMINATED BEFORE EXECUTION

Worker threads

Η δουλειά των worker threads είναι:

- να διαβάζουν τις εργασίες από τον κοινόχρηστο ενταμιευτή (που είχαν τοποθετηθεί εκεί από controller thread),
- να τρέχουν τις εργασίες
- να στέλνουν την έξοδο των εντολών στους πελάτες.

Ένα worker thread ξυπνά όταν υπάρχει τουλάχιστον μια τριπλέτα <jobID, job, clientSocket> στον ενταμιευτή. (δηλαδή τουλάχιστον ένας πελάτης έχει συνδεθεί στον εξυπηρετητή και έχει στείλει issueJob αίτημα).

Όταν ξυπνήσει το worker thread, ελέγχει αν το τρέχον concurrency level του επιτρέπει να αφαιρέσει και να εκτελέσει μια εργασία από τον ενταμιευτή. Αν του το επιτρέπει, αφαιρεί

μια τριπλέτα από τον ενταμιευτή και την εκτελεί. Καλεί την fork() κλήση συστήματος (Σημείωση: στο child process που δημιουργείται, θα υπάρχουν αντίγραφο μόνο του νήματος που κάλεσε την fork()).

Το child process αναλαμβάνει δύο πράγματα:

1. την εκτέλεση της εργασίας μέσω κάποιας εκ των συναρτήσεων της οικογένειας exec*() (execv(),execvp κλπ.) και

2. την ανακατεύθυνση της εξόδου της εντολής σε ένα αρχείο.

Πρώτα δημιουργεί ένα νέο αρχείο με όνομα “pid.output” όπου το pid είναι το process id του παιδιού.Ύστερα, θα πρέπει να κάνει χρήση της dup2() κλήσης ώστε ο STDOUT file descriptor(1) να δείχνει στο νέο αρχείο. Τέλος, καλεί την exec*() call. Με αυτόν τον τρόπο, εκτελείται η εντολή και η έξοδος θα αποθηκεύεται στο αρχείο pid.output.

Το parent process θα περιμένει να τερματίσει το child process. Στη συνέχεια, θα διαβάζει το αρχείο pid.out και θα στέλνει τα περιεχόμενα στον πελάτη μέσω του clientSocket file descriptor Για λόγους ευκρίνειας των αποτελεσμάτων θα προσθέσει πριν και μετα την εξοδο το αναγνωριστικό jobId (start/end) όπως φαίνεται παρακάτω.

-----jobID output start-----

..εξοδος εργασίας..

-----jobID output end------

Αφού τελειώσει η μεταφορά του αρχείου στον πελάτη, κλείνει το clientSocket και αφαιρεί το αρχείο pid.output.

Σχέση controller/worker threads

Τα controller threads και τα worker threads έχουν σχέση παραγωγού-καταναλωτή και έτσι στην υλοποίησή σας θα πρέπει οι προσβάσεις τους στο κοινό ενταμιευτή να συγχρονίζονται. Συγκεκριμένα, ένα controller thread πρέπει να μπλοκάρεται και να περιμένει όταν ο

ενταμιευτής είναι γεμάτος ενώ ένα worker thread πρέπει να περιμένει όταν ο ενταμιευτής είναι άδειος. Με αυτή τη προσέγγιση, αν υπάρχουν περισσότερα worker threads από εργασίες στον ενταμιευτή, τότε κάποια από τα worker threads θα μπλοκάρονται, περιμένοντας νέες issueJob εντολές να φτάσουν στον εξυπηρετητή και να τοποθετηθούν στον ενταμιευτή νέες εργασίες, ενώ αν υπάρχουν περισσότερα νέες issueJob εντολές από ότι worker threads, τότε θα τοποθετούνται στον ενταμιευτή.