Εργαλεία Ανάπτυξης Λογισμικού και Προγραμματισμός Συστημάτων

Τμήματος ΗΜΜΥ

Πολυτεχνείου Κρήτης

Αναφορά

Απομακρυσμένη Εκτέλεση Εντολών

Ομάδα LAB21142346

Παντουράκης Μιχαήλ ΑΜ 2015030185

Ημερομηνία παράδοσης: 13/1/2020

1.1 Εισαγωγικά

Η παρούσα αναφορά περιγράφει την συνοδευόμενη υλοποίηση των προγραμμάτων remoteClient-remoteServer για την υλοποίηση εφαρμογής απομακρυσμένης εκτέλεσης εντολών. Εκτός από τα ζητούμενα παραδοτέα της εργασίας, η παρούσα εφαρμογή μαζί με μερικά παραδείγματα testing θα παραμείνει για οποιοδήποτε περαιτέρω έλεγχο στο remote repository https://github.com/padoura/comp211_remoteCommands (για ευνόητους λόγους, θα γίνει public λίγες μέρες μετά το πέρας της προθεσμίας της εργασίας).

1.2 Πρόγραμμα remoteServer: Parent Process

Το πρόγραμμα εκκινεί δημιουργώντας τα απαραίτητα resources για την εκτέλεση των λειτουργιών του server. Σε αυτά συμπεριλαμβάνονται: 1) η δημιουργία του TCP socket (με τη συνάρτηση make_socket), 2) η δημιουργία 2 pipes ανά παιδί, 3) η αρχικοποίηση των file descriptor sets που θα παρακολουθούν τα select της διεργασίας πατέρα, και 4) τη δημιουργία των διεργασιών παιδιών (συνάρτηση create_children).

Η διεργασία πατέρας είναι υπεύθυνη για τρεις σημαντικές λειτουργίες: 1) τον συγχρονισμό της επικοινωνίας με τις διεργασίες παιδιά για το διαμοιρασμό των εισερχόμενων εντολών σε αυτά, 2) την υποδοχή συνδέσεων και ανάγνωση των εντολών που στέλνουν πολλαπλά instances του προγράμματος πελάτη, και 3) την ανταλλαγή σημάτων με τις διεργασίες παιδιά για την αποδέσμευση των πόρων και τον (μερικό ή ολικό) τερματισμό του server. Στη συνέχεια, περιγράφω αναλυτικότερα τις λειτουργίες αυτές.

1.2.1 Επικοινωνία με Χρήση Σωλήνων και TCP

Η επικοινωνία για τον διαμοιρασμό των εντολών στα παιδιά πραγματοποιείται αποκλειστικά μέσω pipes. Συγκεκριμένα, ο πατέρας χρησιμοποιεί δύο σωλήνες για κάθε παιδί: ένα με κατεύθυνση μηνυμάτων από το παιδί στον πατέρα, και ένα από πατέρα προς παιδί.

Η διεργασία πατέρα αναμένει μηνύματα (τόσο από τα παιδιά, όσο και από πελάτες) μέσα σε επαναληπτική διαδικασία while. Η διαδικασία ξεκινά υποθέτοντας ότι κανένα παιδί δεν είναι έτοιμο να δεχτεί νέα εντολή. Χρησιμοποιώντας τη συνάρτηση select με είσοδο τους file descriptors των εισερχόμενων σε αυτόν pipes, ο πατέρας αναμένει έως ότου ένα παιδί επικοινωνήσει (με την εγγραφή ενός byte στο σωλήνα) ότι είναι έτοιμο.

Έχοντας εξασφαλίσει την ετοιμότητα ενός παιδιού, η γονεϊκή διεργασία αναμένει TCP συνδέσεις στον αρχικοποιημένο υποδοχέα. Ξανά μία συνάρτηση select βοηθάει στο να διαπιστώνουμε πότε ο πατέρας δέχτηκε αίτημα για μία νέα σύνδεση (όπου η επικοινωνία γίνεται στον αρχικό περιγραφητή του socket), και πότε είναι έτοιμος ο πελάτης να στείλει μήνυμα (παίρνοντας νέο descriptor από την εντολή accept). Αφού συμβεί το δεύτερο, τότε προχωράμε στην ανάγνωση της εισερχόμενης εντολής, και στην ανάθεσή της στη διαθέσιμη διεργασία-παιδί.

Η ανάγνωση της εισερχόμενης εντολής από τον πελάτη πραγματοποιείται με τη συνάρτηση read_from_client. Η συνάρτηση αυτή, διαβάζοντας ένα χαρακτήρα τη φορά επαναληπτικά, χρησιμοποιεί δομή (struct) Command για την αποθήκευση μιας εντολής αλλά και συνοδευτικών πληροφοριών (δείτε την ενότητα Δομή Μηνυμάτων). Η ανάγνωση μέσω της TCP σύνδεσης τερματίζεται προσωρινά είτε όταν διαβαστεί ένας newline character (δηλαδή το τέλος μίας εντολής), είτε τερματίζεται οριστικά όταν ο client φτάσει σε EOF και τερματίσει την αποστολή δεδομένων. Στην τελευταία περίπτωση, η διεργασία πατέρα φροντίζει να κλείσει τη συγκεκριμένη σύνδεση, και να την αφαιρέσει από το σύνολο των περιγραφητών που αναμένει μηνύματα (στη select).

Τέλος, η μεταφορά της εντολής στο διαθέσιμο παιδί πραγματοποιείται με την συνάρτηση allocate_to_children. Σε αυτή συντίθεται το μήνυμα που περιέχει την εντολή και όλη την απαραίτητη πληροφορία για το παιδί (δείτε Δομή Μηνυμάτων), ενώ πλέον το παιδί σηματοδοτείται ξανά σαν απασχολημένο.

Αν και η εκφώνηση της άσκησης αναφέρεται στην καταγραφή των ερωτημάτων σε ένα μόνο σωλήνα, στην παραδοτέα υλοποίηση προχώρησα στη χρήση ξεχωριστού σωλήνα για κάθε παιδί. Η επιλογή αυτή έγινε για τους εξής λόγους: 1) όπως αναφέρει η άσκηση, για να λαμβάνει υπόψιν του ο γονέας το φορτίο κάθε παιδιού, και με προβλέψιμο τρόπο να δίνει την προς εκτέλεση εντολή σε ένα διαθέσιμο παιδί, 2) εκμεταλλευόμενοι το γεγονός ότι υπάρχει ένα pipe ανά παιδί και εντολές άνω των 100 γραμμών αγνοούνται, αυτομάτως εξασφαλίζεται η ατομικότητα. Αρχικά η λύση μου είχε δημιουργηθεί με 1 pipe για κάθε παιδί (δείτε Tag SinglePipeWithoutTerminationCommands στο remote repository της άσκησης), χωρίς όμως να εξασφαλίζει με κάποιο τρόπο την ατομικότητα (π.χ. με μη μετάδοση πάνω από PIPE_BUF bytes στο σωλήνα τη φορά) και πριν τη υλοποίηση των εντολών τερματισμού. Τελικά, λόγω του χρονικού περιορισμού για την παράδοση της άσκησης δεν προσάρμοσα τη λογική αυτή στο τελικό παραδοτέο.

1.3 Πρόγραμμα remoteServer: Child Process

Οι διεργασίες παιδιά ξεκινούν, όπως προαναφέραμε, ενημερώνοντας τον γονέα ότι είναι έτοιμα να εκτελέσουν κάποια εντολή (γράφοντας στον κατάλληλο σωλήνα). Στη συνέχεια, μέσα σε βρόχο while

κάθε παιδί περιμένει (με blocked read) να διαβάσει την εντολή που του ανέθεσε ο γονέας. Αφότου διαβάσει το εισερχόμενο μήνυμα και με διαδοχικά splits διαχωρίσει τα διάφορα μέρη του μηνύματος, ελέγχει την εντολή που έστειλε ο πελάτης.

Αρχικά ελέγχεται αν η εντολή έχει μέγεθος 101 χαρακτήρων, κάτι που σημαίνει ότι δεν θα εκτελεστεί. Στη συνέχεια, αφού αφαιρεθούν εντολές μετά από unquoted χαρακτήρα ‘;’, leading, και trailing spaces (με αντίστοιχες τρεις συναρτήσεις), ελέγχεται αν είναι η εντολή “end”, “timeToStop” (δείτε επόμενη ενότητα για τη λειτουργία τους), αν είναι άδεια (δηλαδή ήταν γραμμή που αποτελούνταν μόνο από κενά), και αν η πρώτη εντολή ανήκει στις πέντε υποστηριζόμενες εντολές UNIX.

Προτού οι υποστηριζόμενες εντολές εκτελεστούν, ακολουθείται επιπλέον επεξεργασία για την εύρεση μη υποστηριζόμενων εντολών σε σωληνώσεις. Τέλος, στην εντολή UNIX γίνεται append του τμήματος “ 2>/dev/null” ώστε να μην τυπώνονται πουθενά τυχόν σφάλματα (stderr) των εντολών. Εδώ αξίζει να σημειωθεί ότι θα ήταν ορθό (και ασφαλέστερο για τον server) να αποκλείονται command substitutions μέσα στις υποστηριζόμενες εντολές, όμως λόγω του χρονικού περιορισμού της άσκησης δεν υλοποιήθηκε.

Οι υποστηριζόμενες εντολές που τελικά πέρασαν τους παραπάνω ελέγχους, εν τέλει εκτελούνται με χρήση της popen, που προσφέρει έναν file pointer για εύκολη ανάγνωση του (stdout) αποτελέσματος. Μιας και η αποστολή των αποτελεσμάτων στον πελάτη απαιτείται να γίνει με σύνδεση UDP με 512 bytes τη φορά, η ανάγνωση από την popen γίνεται ανά επαναληπτικά ανά 512 bytes μείον όσα χρειάζονται για την υπόλοιπη μεταπληροφορία του πελάτη (ξανά, δείτε Δομή Μηνυμάτων). Τα αποτελέσματα αποθηκεύονται στη δυναμική θέση μνήμης result, χωρισμένα μεταξύ τους ανά 513 bytes (για εξασφάλιση nul termination).

Η result τέλος χρησιμοποιείται στη send_result_with_UDP, όπου με χρήση UDP σύνδεσης στέλνονται στον πελάτη, πακέτα των 512 bytes έως ότου αποσταλεί και το τελευταίο πακέτο του αποτελέσματος. Στην ενότητα “Πρόγραμμα remoteClient: Child Process” εξηγώ περαιτέρω το πώς διαχειρίζεται τα εισερχόμενα αποτελέσματα η εφαρμογή πελάτη, αλλά και τους περιορισμούς του παραδοτέου λόγω της UDP σύνδεσης.

1.4 Επικοινωνία μέσω Σημάτων

1.4.1 Εντολή end

Το πρόγραμμα του εξυπηρετητή συνεχίζει την λειτουργία του, έως ότου λάβει τις τερματικές εντολές “end” ή/και “timeToStop” από κάποιον πελάτη. Για να πετύχουν τη σκοπό τους, η απαίτηση της άσκησης ήταν η χρήση σημάτων μεταξύ γονέα και παιδιών.

Συγκεκριμένα, όταν ένα παιδί λάβει την εντολή “end”, στέλνει στον γονέα (με kill) ένα σήμα SIGUSR1. Η διεργασία γονέα διαχειρίζεται αυτό το σήμα με τη συνάρτηση handle_end, η οποία ως sa_sigaction περέχει την κατάλληλη πληροφορία για τον προσδιορισμό του παιδιού (process id) που πρόκειται να τερματίσει τη λειτουργία του. Αυτό είναι απαραίτητο για να μπορέσει ο γονέας να κλείσει τους περιγραφητές σωλήνων που το αφορούν και δεν χρειάζονται πλέον. Επιπλέον, ο γονέας τους αφαιρεί και από το σετ των περιγραφητών σωλήνων της select, ώστε να εξασφαλιστεί ότι δεν θα επιλεχθούν λανθασμένα σαν pipe file descriptors (π.χ. αν ξαναχρησιμοποιηθεί ως new connection αργότερα).

Για επιτυχημένο συγχρονισμό μεταξύ τερματισμού παιδιού και κλεισίματος πόρων γονέα, ο γονέας αναμένει το παιδί να τερματίσει (με τη waitpid), ενώ πριν την επιστροφή από τον handler δίνεται sleep εντός δευτερολέπτου (για την αποφυγή επιστροφής σε λανθασμένο σημείο του κύριου while loop του parent server).

Επιπλέον, για τη σηματοδότηση του πότε είναι ασφαλές να συνεχίσει μέσα στο while loop ο πατέρας, χρησιμοποιείται η boolean μεταβλητή g_continue (αποκλειστική χρήση από τον πατέρα). Πριν ξεκινήσει την ανάγνωση εντολής από πελάτη ή/και όταν λάβει SIGUSR1, ο πατέρας σηματοδοτεί ότι δεν μπορεί να ξαναλειτουργήσει προτού έρθει ένα σήμα SIGCONT από παιδί. Τα παιδιά στέλνουν SIGCONT είτε όταν φτάσουν ένα βήμα πριν τερματίσουν τη λειτουργία τους, είτε όταν εξασφαλίσουν ότι η εντολή του πελάτη δεν είναι τερματική.

Τέλος, η διεργασία παιδί, μόλις στείλει το σήμα SIGUSR1, αποστέλλει (άδειο) αποτέλεσμα στον πελάτη, εξέρχεται από το βρόχο while, απελευθερώνει δυναμική μνήμη, κλείνει τους περιγραφητές που χρησιμοποιούσε και τυπώνει το μήνυμα τερματισμού “process {pid} terminated” στην stderr του εξυπηρετητή.

1.4.2 Εντολή timeToStop

Η εντολή “timeToStop”, η οποία σηματοδοτείται στον γονέα με το σήμα SIGUSR2 από το παιδί, επαναχρησιμοποιεί τις λειτουργίες που αναφέρθηκαν για την “end”, τόσο του γονέα, όσο και του

παιδιού. Για να εξασφαλίσουμε ότι ο γονέας θα τερματίσει χωρίς να επηρρεαστεί από το SIGCONT σήμα άλλων παιδιών, είναι masked για τον handler του γονέα και αυτό το σήμα.

Ο γονέας με τη σειρά του στέλνει επίσης ένα σήμα SIGUSR2 σε κάθε παιδί (το οποίο και αλλάζει την τιμή της Boolean childStop) και μιας και επαναχρησιμοποιούμε τη λειτουργικότητα της end, η waitpid εξασφαλίζει ότι ο γονέας θα τερματίσει τη λειτουργία αφότου τερματίσουν όλα τα παιδιά του. Τερματίζοντας ο γονέας κάνει shutdown (μπλοκάροντας νέες συνδέσεις προς όλες τις κατευθύνσεις) και close του TCP socket. Σημειωτέων ότι ο ίδιος τερματισμός του γονέα μπορεί να συμβεί αν με πολλαπλές εντολές “end” τερματίσουν όλα τα παιδιά του, οπότε δεν μένει παρά να τερματίσει και ο ίδιος ο γονέας τη λειτουργία του.

1.5 Πρόγραμμα remoteClient: Parent Process

Το πρόγραμμα πελάτη είναι απλούστερο στη λειτουργία του, όπως αναμενόταν και από τα requirements της άσκησης. Αρχικά διαβάζει προκαταρκτικά το αρχείο εισόδου, ώστε να δώσει την απαραίτητη πληροφορία του αριθμού εντολών στην διεργασία γονέα και παιδί (για άδειο αρχείο τερματίζεται η λειτουργία του). Στη συνέχεια, δημιουργείται μία διεργασία παιδί, η οποία και θα αναμένει τα εισερχόμενα αποτελέσματα των εντολών αυτών μέσω UDP (επόμενη ενότητα).

Αφού συνδεθεί μέσω TCP, η διεργασία γονέα αποστέλλει τις εντολές σειρά προς σειρά (μαζί με επιπλέον πληροφορία) στον εξυπηρετητή, κάνοντας μία παύση πέντε δευτερολέπτων για κάθε 10 εντολές που αποστέλλει. Τέλος, αφού κλείνει την TCP σύνδεση, αναμένει απλά τον τερματισμό του παιδιού του για να ολοκληρώσει και εκείνος τη λειτουργία του.

1.6 Πρόγραμμα remoteClient: Child Process

Η receive_results περιέχει όλη τη λειτουργικότητα της διεργασίας παιδί του πελάτη. Η δημιουργία του παιδιού αυτού εξασφαλίζει ότι όσο αποστέλλει ο πατέρας εντολές με TCP, το παιδί θα μπορεί παράλληλα να παραλαμβάνει πακέτα αποτελεσμάτων μέσω UDP.

Για να διαβάσει τα εισερχόμενα πακέτα, το παιδί ελέγχει επαναληπτικά για μηνύματα προς ανάγνωση με τη recvfrom. Το κάθε πακέτο αποθηκεύεται ξεχωριστά σε προσωρινό αρχείο με μορφή ονόματος “output.receive{clientPort}.{cmdNumber}.{partNum}”, όπου partNum είναι το μέρος του αποτελέσματος (αύξων αριθμός, με concatenated “f” αν είναι το τελικό μέρος). Η επαναληπτική αυτή διαδικασία τερματίζεται είτε όταν ληφθεί το τελικό πακέτα κάθε αποτελέσματος, είτε όταν φτάσουμε το ορισμένο timeout της non-blocking recvfrom (στο παραδοτέο 20 δευτερόλεπτα). Φυσικά λόγω της UDP σύνδεσης, η παραπάνω διαδικασία έχει το μειονέκτημα ότι κάποια πακέτα μπορεί να μη ληφθούν πριν την έξοδο

από το while, ενώ λόγω περιορισμένου χρόνου δεν υλοποιήθηκε λογική ACK και επαναποστολής χαμένων πακέτων.

Τέλος, η συνάρτηση merge_files εξασφαλίζει τη σύνθεση, με σωστή σειρά, του τελικού αποτελέσματος κάθε εντολής στα ζητούμενα αρχεία με όνομα “output.receive{clientPort}.{cmdNumber}”, διαγράφοντας τα προαναφερθέντα προσωρινά αρχεία. Αν μία εντολή δεν έλαβε αποτελέσματα (επειδή χάθηκε το πακέτο), τότε το αντίστοιχο αρχείο αποθηκεύεται άδειο.

1.7 Δομή Μηνυμάτων

Η δομή των μηνυμάτων είναι η παρακάτω:

1) Αποστολή εντολής client -> parent server: “cmdNumber;clientPort;cmd”
   Με τα παραπάνω πεδία ο πελάτης περνάει την πληροφορία που χρειάζεται ο εξυπηρετητής για να γνωρίζει τη σειρά της εντολής, την ίδια την εντολή, αλλά και τη θύρα που αναμένει τη σύνδεση UDP.

2) Αποστολή εντολής σε pipe parent -> child server: “clientPort;clientIP;cmdNumber;cmd”.
   Με τα παραπάνω πεδία εξασφαλίζεται ότι το παιδί γνωρίζει που θα στείλε μέσω UDP το αποτέλεσμα, ποια εντολή (σειρά) είναι από αυτές που έστειλε ο πελάτης, και την ίδια την εντολή.

3) Αποστολή πακέτου αποτελέσματος child server -> client: “cmdNumber;partNum;cmdResult”.
   Με αυτά τα πεδιά ο πελάτης γνωρίζει σε ποια εντολή αντιστοιχεί το πακέτο αποτελέσματος που παρέλαβε, και τη σειρά του μέρους που παρέλαβε για την επανασύνθεση του ολικού αποτελέσματος.