Αναφορά 3ης Άσκησης

Δελατόλας Θάνος 2016030074

Μοίρασμα εργασίας

Η εργασία είναι ατομική, το άλλο μέλος της ομάδας δεν συμμετείχε.

Remote Server

Αρχικά αρχικοποιούμε τα σήματα του σήματα του signal handler και δημιουργούμε τον σωλήνα (pipe). Ο σωλήνας και ο signal handler θα κληρονομηθούν από τις διεργασίες παιδιά. Στη συνέχεια δημιουργούνται NUMCHILDREN διεργασίες παιδιά όπου το κάθε παιδί καλεί τη συνάρτηση child_function που δεν τερματίζει. Εφόσον ο πατέρας γεννήσει τις διεργασίες-παιδία καλεί την συνάρτηση server_function που επίσης δεν τερματίζει. Οι δυο παραπάνω συναρτήσεις τερματίζουν μόνο όταν διαβαστεί end ή timeToStop(λεπτομέρειες παρακάτω).

Πριν την ανάλυση των παραπάνω συναρτήσεων είναι πολύ σημαντικό να επισημάνουμε τις global δομές δεδομένων που χρησιμοποιούνται.

- pid_t* workers_array: Πίνακας με τα process id της κάθε διεργασίας παιδιού.
- pid_t parent: Process id της διεργασίας πατέρα.
- int timeToStop: Αρχικά έχει μηδενική τιμή. Βασιζόμενοι στο γεγονός πως κάθε διεργασία έχει την δική της μνήμη η μεταβλητή θα πάρει την τιμή 1 για κάθε παιδί, όταν το κάθε παιδί ειδοποιηθεί από το σήμα SIGUSR2.
- typedef struct server_worker_msg: struct που περιέχει την εντολή, το port που θα σταλεί η απάντηση και τον αριθμό της εντολής στο αρχείο(1η,2η κτλ.). Η struct αυτή γίνεται serialize και γράφεται στο pipe για να διαβαστεί από ένα παιδί.

- typedef struct connectlist_node: struct που περιέχει τον file descriptor του socket που στέλνει τις εντολές, το port που θα σταλεί η απάντηση και έναν int command_code που δείχνει πόσες εντολές έχει στείλει, δηλαδή το νούμερο της γραμμής στο αρχείο της εκάστοτε εντολής (αρχικά έχει τη τιμή 0).
- connectlist_node connection_list[max_clients]: Πίνακας για κάθε connection, δηλαδή για κάθε client.
- int server_fd: File descriptor του tcp socket που λαμβάνει τις εντολές.
- int pipe_fds[2]: File descriptors του pipe για την επικοινωνία των διεργασιών.

server_function

Με την συνάρτηση create_TCP_SOCKET δημιουργούμε ένα socket που δέχεται tcp συνδέσεις. Η παραπάνω συνάρτηση είναι βασισμένη στις διαλέξεις του μαθήματος.

Αρχικά δηλώνεται το fd_set socks. Στην συνέχεια αρχικοποιούμε τους file descriptors της connectionlist στο 0 και ο server μπαίνει στο main loop το οποίο τελειώνει μόνο όταν ληφθεί η timeToStop ή η end και υπάρχει ένα μόνο παιδί. Σε κάθε επανάληψη αδειάζουμε το set, προσθέτουμε το server_fd και όσους file descriptors δεν είναι 0 από την connection_list. Στη συνέχεια καλείται η select με όρισμα το fd_set socks και ο server περιμένει μέχρι ένας ή περισσότεροι file descriptor να είναι έτοιμη για ανάγνωση. Πρώτα ελέγχεται ο server_fd και αν είναι έτοιμος καλείται η συνάρτηση accept. Αφού γίνει δεκτή η νέα connection ελέγχεται αν υπάρχει χώρος στην connection_list για την αποθήκευση του file descriptor που επιστέφει η accept. Αν δεν υπάρχει διαθέσιμος χώρος ο server κλείνει τον νέο file descriptor. Στην συνέχεια ελέγχεται όλη η connection_list με ένα loop, και όποιος file descriptor είναι στο fd_set socks με το οποίο κλήθηκε η select, έχει διαθέσιμα δεδομένα για ανάγνωση.

Οι clients πριν στείλουν τις εντολές στέλνουν το receivePort με format (“receivePort:”receivePort πχ. “receivePort:”4000)[1] που δέχονται udp μνήματα. Το receivePort αποθηκεύεται στην εκάστοτε connectlist_node στο πεδίο port.

Αν τα νέα δεδομένα δεν περιέχουν το receivePort αυξάνεται command_code του εκάστοτε connectlist_node, εφόσον διαβάστηκε νέα εντολή από το συγκεκριμένο socket. Με το command_code,το receivePort και την εντολή δημιουργείται η struct server_worker_msg γίνεται serialize και γράφεται στον σωλήνα (pipe) για να διαβαστεί από ένα παιδί. Τα παραπάνω αποτελούν το main loop του server.

Στο σημείο αυτό είναι σημαντικό να αναλυθεί η βιβλιοθήκη (header file) remote.h η οποία χρησιμοποιείται και από τον server και από τον client.

Περιέχει:

- PACKET_SIZE 512 : σταθερά που δείχνει το μέγιστο αριθμό bytes του πακέτου που στέλνεται με udp στον client.
- UPD_CMD_SIZE PACKET_SIZE - 2*sizeof(int): σταθερά που δείχνει το μέγιστο αριθμό bytes του string (απάντηση στην εντολή) που περιέχει το πακέτο που στέλνεται με udp στον client.
- SERVER_CLOSED: σταθερά που στέλνεται σ ’όλους τους client όταν κλείνει ο server.
- END: σταθερά που περιέχει την εντολή end.
- TIME_TO_STOP: σταθερά που περιέχει την εντολή timeToStop.
- typedef struct udp_msg: struct μεγέθους PACKET_SIZE που γίνεται serialize και στέλνεται με udp στον client ως απάντηση στον εκάστοτε εντολή. Αν η απάντηση είναι μεγαλύτερη από UPD_CMD_SIZE τότε

[1] Ο client ακόμη και το αρχείο να περιέχει την εντολή receivePort:4000 δεν θα τη στείλει, θα την αλλάξει σε invalid.

στέλνονται πολλές struct (πακέτα) στον client. Περιέχει δυο ακέραιους και ένα string όπως φαίνεται από το UPD_CMD_SIZE. Ο πρώτος ακέραιος είναι ο αριθμός της εντολής στο αρχείο με τον οποίο γνωρίζει ο client σε ποια εντολή αναφέρεται το πακέτο. Ο δεύτερος είναι ένας ακέραιος που όταν πάρει τη τιμή 1 ο client γνωρίζει πως είναι το τελευταίο πακέτο απάντησης της εντολής. Τέλος περιέχει το string με το μέρος της απάντησης της εντολής.

- trim(char* str): συνάρτηση που κάνει trim ένα string.

child_function

Η συνάρτηση αυτή εκτελείται από όλες τις διεργασίες παιδιά. Όλα τα παιδιά κλείνουν το pipe_fds[1] εφόσον δεν γράφουν στο pipe. Δημιουργούν ένα datagram socket με το οποίο θα στέλνουν τις απαντήσεις και αρχικοποιούν τη μεταβλητή working στο 0. Η μεταβλητή αυτή δείχνει αν έχουν διαβάσει από το pipe ένα server_worker_msg και το εκτελούν. Στην συνέχεια εισέρχονται σ' ένα loop που τελειώνει αν το συγκεκριμένο παιδί λάβει την εντολή end ή κάποιο άλλο παιδί ή το ίδιο λάβει την εντολή timeToStop.

Όποιο παιδί έχει τη μεταβλητή working 0 διαβάζει στο pipe_fds[0] δηλαδή διαβάζει από το pipe. Αν το pipe δεν έχει δεδομένα τότε τα παιδιά μπλοκάρουν στη read. Όταν γραφτούν νέα δεδομένα στο pipe ,σύμφωνα με το documentation του POSIX, ο kernel αποφασίζει ποιο απ' όλες τις διεργασίες παιδιά θα διαβάσει τα νέα δεδομένα. Έτσι κάθε παιδί διαβάζει μια εντολή και αφού την εκτελέσει περιμένει για την επόμενη.

Εφόσον το εκάστοτε παιδί έχει διαβάσει ένα serialized server_worker_msg κάνει deserialize και έχει πλέον ένα server_worker_msg που περιέχει την εντολή, τον αριθμό της εντολής στο αρχείο και το port που θα στείλει την απάντηση.

Ήρθε η ώρα για τον έλεγχο της εντολής. Αν η εντολή δεν είναι ούτε **end** ούτε **timeToStop**, δημιουργείται ένας πίνακας με όλες τις pipelined εντολές δηλαδή αν διαβαστεί η εντολή ls | tr "[a-z]" "[A-Z]" θα δημιουργηθεί ο πίνακας {ls, tr "[a-z]" "[A-Z]" }. Στη συνέχεια ελέγχεται κάθε εντολή για την εγκυρότητα της ως προς το όνομα. Αν η πρώτη εντολή δεν είναι έγκυρη η pipelined εντολή θεωρείται λανθασμένη, αν δεν είναι έγκυρη οποιαδήποτε

άλλη αφαιρείται από την pipelined εντολή. Στη συνέχεια εκτελείται η εντολή με την συνάρτηση execute του server που χρησιμοποιεί το system call popen για την εκτέλεση της pipelined εντολής. Αν το αρχείο που επιστέφει η popen είναι NULL, η εντολή δεν μπορεί να εκτελεστεί λόγω των ορισμάτων της.

Αν το αρχείο δεν είναι NULL δημιουργείται μια struct udp_msg που έχει τον αριθμό της γραμμής που βρίσκεται στο αρχείο των εντολών, που το παίρνουμε server_worker_msg που διαβάστηκε από το pipe. Διαβάζεται το αρχείο που επέστρεψε η popen και προσθέτουμε έναν-έναν χαρακτήρα σ' ένα string, αν το string φτάσει να έχει μέγεθος UPD_CMD_SIZE -1, προσθέτουμε τον τερματικό χαρακτήρα και η μεταβλητή last του struct udp_msg γίνεται 0, εφόσον δεν είναι το τελευταίο πακέτο που θα σταλεί για την απάντηση της εντολής. Η struct udp_msg γίνεται serialize και στέλνεται στον client (θυμίζουμε έχουμε το receivePort από το server_worker_msg το οποίο με τη σειρά του το έχει από το connectlist_node). Συνεχίζεται να διαβάζεται το αρχείο με την απάντηση της εντολής και όταν το μήκος του string είναι μικρότερο UPD_CMD_SIZE -1 η μεταβλητή last της struct udp_msg γίνεται 1 για να ξέρει ο client πως αυτό είναι το τελευταίο πακέτο για την εκάστοτε εντολή.

Αν το αρχείο που επιστρέφει η popen είναι NULL ή εντολή είναι λανθασμένη στέλνεται στο client μια struct udp_msg με τον αριθμό της γραμμής στο αρχείο εντολών, με last 1 και το κενό string.

Αξίζει να σημειωθεί πως τα παιδιά μετά από κάθε sendto κάνουν sleep(0.005) για να μην χάνονται τα πακέτα που στέλνονται.

Αν η εντολή είναι **end** τότε δημιουργείται ένα string με το pid της διεργασίας παιδιού αρχικά στέλνεται μια struct udp_msg με τον αριθμό της γραμμής στο αρχείο εντολών, με last 1 και το string με το pid της εντολής. Το παιδί στέλνει στο πατέρα το σήμα SIGUSR1, κλείνει pipe_fds[0] και sockfd(datagram socket) και κάνει exit. Η διεργασία του πατέρα μειώνει τη μεταβλητή activeChildren η οποία αρχικά είχε τη τιμή NUMCHILDREN, αν τα activeChildren είναι 0 τότε ο server κλείνει και ειδοποιεί[2] όλους τους client. Αν όχι περιμένει τη διεργασία παιδί να τερματίσει.

[2] Πως ειδοποιεί ο server τους client θα αναλυθεί στη συνέχεια.

Πριν αναλύσουμε την εντολή timeToStop είναι σημαντικό να επισημάνουμε πως όλες οι διεργασίες παιδιά έχουν κληρονομήσει από τον πατέρα τον ακέραιο timeToStop με τιμή 0.

Αν η εντολή είναι **timeToStop** το παιδί που την διάβασε τυπώνει στο stderr το process id του στέλνει ένα πακέτο με το string "timeToStop command" στον client, στέλνει SIGUSR2 στη διεργασία πατέρα και τερματίζει.

Ο πατέρας όταν λάβει SIGUSR2

- στέλνει SIGUSR2 σ' όλες τις διεργασίες παιδιά.
- περιμένει να τελειώσουν όλες οι διεργασίες παιδιά.
- Αφού τελειώσουν τυπώνει ο πατέρας το pid.
- Ειδοποιεί όλους τους client.
- Κλείνει όλους τους file descriptors του.(connectionlist, server_fd , pipe_fds[1])
- τερματίζει

Τα παιδιά όταν λάβουν SIGUSR2 κάνουν -το καθένα το δικό του- ακέραιο timeToStop 1. Στο main loop το εκάστοτε παιδί αν ο ακέραιος timeToStop έχει τη τιμή 1 η διεργασία παιδί :

- κλείνει pipe_fds[0]
- κλείνει το datagram socket
- τυπώνει στην stderr το pid του
- τερματίζει

Τέλος θα αναλύσουμε το πώς ειδοποιεί ο server όλους τους συνδεδεμένους client ότι πρόκειται να κλείσει. Απλά στέλνεται ένα πακέτο με χρήση datagram socket σε κάθε client που βρίσκεται στην connectionlist που περιέχει το string "serverClosed". Όταν το διαβάσει ο client τερματίζει

Remote Client

Αρχικά ελέγχεται αν υπάρχει και μπορεί να διαβαστεί το αρχείο με τις εντολές που έδωσε ως όρισμα ο χρήστης. Αν δεν υπάρχει ή δεν μπορεί να διαβαστεί το πρόγραμμα τερματίζει. Στη συνέχεια ορίζεται ο signal_handler για το σήμα SIGUSR1, όταν μια διεργασία λάβει το SIGUSR1 τερματίζει μέσω του signal_handler. Μέσω της συνάρτησης create_commands_array δημιουργείται ένας πίνακας με τις εντολές του αρχείου. Αν υπάρχει εντολή στο αρχείο που να είναι της μορφής receivePort:4000 αποθηκεύεται στον πίνακα ως invalid. Για να είναι εφικτό να στέλνονται εντολές και ταυτόχρονα να διαβάζονται απαντήσεις δημιουργείται μια διεργασία παιδί η οποία στέλνει τις εντολές με το μοτίβο που περιγράφεται στην εκφώνηση. Πριν στείλει τις εντολές στέλνει στον server το receivePort.

Ο πατέρας είναι υπεύθυνος για την παραλαβή των πακέτων και τη δημιουργία των αρχείων. Όπως έχουμε αναφέρει στέλνεται από τον server η serialized struct udp_msg η οποία περιέχει το νούμερο της εντολής, με βάση το νούμερο της εντολής και το receivePort ανοίγουμε το αρχείο της απάντησης (output.receivePort.1) και κάνουμε append. Αφού γίνει append κλείνουμε το αρχείο γιατί το επόμενο πακέτο που θα ληφθεί δεν αναφέρεται αναγκαστικά στην ίδια εντολή. Αν η μεταβλητή last της struct udp_msg είναι 1 αυξάνεται ένας counter που δείχνει πόσες εντολές έχουν απαντηθεί. Όταν απαντηθούν όλες ο πατέρας τερματίζει.

Τέλος, στέλνεται SIGUSR1 από τη διεργασία παιδί στη διεργασία πατέρα αν για κάποιο λόγο δεν μπορεί να στείλει τις εντολές. Με αυτόν τον τρόπο η διεργασία πατέρα δεν περιμένει απαντήσεις και το πρόγραμμα τερματίζει. Ο πατέρας στέλνει στο παιδί SIGUSR1 αν ειδοποιηθεί πως ο server κλείνει. Για να τερματίσει και το παιδί.