Εργασία 1: Πολυνηματική μηχανή αναζήτησης.

Βλαχογκουντής Θεόφιλος (Theo Goudis)

ΣΥΝΟΨΗ

Η μηχανή αναζήτησης που δημιουργήσαμε λειτουργεί με προσέγγυση client-server. Ο client αναμένει είσοδο από το χρήστη. Όταν λάβει μία είσοδο ελέγχει εάν πρέπει να αναζητηθούν μία ή περισσότερες λέξεις. Οι λέξεις, αφού εκτελεστούν οι κατάλληλες ενέργειες σε περίπτωση πολλαπλής αναζήτησης, μέσω ενός network socket αποστέλονται στο server . Ο server λαμβάνει αυτές τις πληροφορίες και τις τοποθετεί σε μία στοίβα. Τώρα μέσω νημάτων γίνεται η αναζήτηση (παράλληλα) τον λέξεων αυτων σε ένα ορισμένο πλήθος αρχείων. Όταν η αναζήτηση ολοκληρωθεί ο server αποστέλει στον client, μέσω του network socket, τα αποτελέσματα. Ο client εκτυπώνει τα αρχεία και το πλήθος των εμφανίσεων της λέξης, στο καθένα. Επίσης κρατούνται πληροφορίες για το πόσες λέξεις έχουν αναζητηθεί, ο συνολικός χρόνος για το πόση ώρα “περίμεναν” οι εντολές αναζήτησης της κάθε λέξης μέχρι να εκτελεσθούν και ο συνολικός χρόνος που σπαταλήθηκε για την αναζήτηση των λέξεων.

Client-ΛΕΙΤΟΥΡΓΙΕΣ

Ο client λαμβάνει κάποια δεδομένα τα οποία βρίσκονται στην μεταβλητή argv της main. Εάν το πλήθος των λέξεων προς αναζήτηση είναι 1 (argc==3) τότε δημιουργεί ένα socket, συνδέεται με το server και αποστέλει τη λέξη προς αναζήτηση στο server. Εάν οι λέξεις είναι περισσότερες από 1 (έχουμε ορίσει ανότατο όριο MAXCOMMANDS 10) τότε μέσω της συνάρτησης void initCommand(int argc, char** argv) οι λέξεις προς αναζήτηση τοποθετούντε στη στοίβα char *Command_stack[MAXCOMMANDS]. Μετά δημιουργούνται πολλάπλές διεργασίες (που τερματίζουν ασύγχρονα) μέσω της συνάρτησης void func_exec(char **argv, int pid_num) / fork() και με χρήση της execvp() δημιουργούνται πολλαπλές διεργασίες client που έχουνε η κάθε μία ως είσοδο και από μία διαφορετική λέξη προς αναζήτηση (στο argv[2]). Τώρα ο κάθε ένας από αυτούς τους clients δημιουργεί ένα νέο socket μέσω του οποίου αποστέλει την λέξη στον server και λαμβάνει τα αποτελέσματα τα οποία εκτυπώνει. Στο τέλος της μητρικής διεργασίας client εκτελείται και wait() για καθαρισμό των διεργασιών παιδιών.

Client-ΣΥΝΑΡΤΗΣΕΙΣ & ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Αρχικά γίνονται define οι εξής μεταβλητές:
#define SERVER_PORT (το port μέσω του οποίου ο client επικοινωνεί με το server)
#define BUF_SIZE (μέγιστο μέγεθος του buffer στον οποίο αποθηκεύεται η απάντηση του server)
#define MAXHOSTNAMELEN
#define MAXCOMMANDS (μέγιστο μέγεθος λέξεων προς αναζήτηση σε μία εντολή (χωρισμένες με κενό)

Μετά ορίζονται οι μεταβλητές:
char *Command_stack[MAXCOMMANDS] (στοίβα αποθήκευσης των λέξεων)
int Cur_command_stack=-1 (δείκτης στη στοίβα των λέξεων)
pid_t pid[MAXCOMMANDS] (στοίβα αποθήκευσης των process id των διεργασιων παιδιών)

ΣΥΝΑΡΤΗΣΗ: void initCommand(int argc, char** argv)

Τοπικές μεταβλητές: int i=0; int temp_length=0;

Λειτουργίες: Η συνάρτηση ελέγχει εάν έχουν εισαχθει περισσότερες από τις αποδεκτές λέξεις προς αναζήτηση, αν ναι επιστρέφει error. Τέλος τοποθετεί τις λέξεις στη στοίβα Command_stack.

ΣΥΝΑΡΤΗΣΗ: void func_exec(char **argv, int pid_num)

Τοπικές μεταβλητές: (καμία)

Λειτουργίες: Πέρνει ένα argv[4] και στη θέση 2 αποθηκεύει την λέξη που βρίσκεται στο Command_stack[Cur_command_stack], Μετά δημιουργεί μία διεργασία παιδί. Το παιδί με την execvp εκτελεί την client με είσοδο το argv[4] (argc==3)

ΣΥΝΑΡΤΗΣΗ: main

Τοπικές μεταβλητές:
struct sockaddr_in server_addr;
struct hostent *host_info;
int socket_fd, numbytes;
char buffer[BUF_SIZE];
char *argvT[4];
int temp_size=0;
int pid_num=-1;
int i=0;

Λειτουργίες: Αρχικά ελέγχεται το μέγεθος της εισαγωγής argc.
Άν argc<3 δεν επαρκούν τα ορίσματα οπότε και γίνεται τερματισμός.
Εάν argc==3 τότε υπάρχει μόνο μία λέξη προς αναζήτηση. Δημιουργείται το socket και γίνεται η σύνδεση με το server. Μέσω του socket αποστέλεται στο server η λέξη προς αναζήτηση. Ύστερα λαμβάνεται το αποτέλεσμα από τον server και εκτυπώνεται.
Εάν argc>3 τότε πρέπει να γίνει αναζήτηση πλήθους λέξεων μεγαλύτερου του ενός. Καλείται η συνάρτηση initCommand. Ύστερα στη μεταβλητή argvT[4], στη θέση 0 και 1 αποθηκεύονται τα αντίστοιχα ορίσματα της argv, η argvT[3] γίνεται NULL (για χρήση στην execvp). Μετά, για κάθε λέξη στη στοίβα Command_stack καλείται η συνάρτηση func_exec με ορίσματα argvT και pid_num (η θέση στη στοίβα pid[MAXCOMMANDS] στην οποία θα αποθηκευτεί το pid της διεργασίας παιδιού). Στο τέλος εκτελείται waitpid() για τον καθαρισμό των παιδιών.

Server-ΛΕΙΤΟΥΡΓΙΕΣ

Ο server λαμβάνει πληροφορίες (λέξεις προν αναζήτηση) μέσω ενός socket. Όταν λαμβάνει λέξεις τις τοποθετεί σε μία στοίβα εισόδων (Server_Con_Stack[MAX_PENDING_CONNECTIONS]). Ύστερα, με διάβασμα της κάθε λέξης από τη στοίβα, γίνεται αναζήτηση στα αρχεία και βρίσκεται το πλήθος των εμφανίσεων της λέξης στο εκαστο. Μετά επιστρέφονται τα αποτελέσματα στον client.
Οι λειτουργίες τοποθέτησης λέξεων στη στοίβα και αναζήτησης γίνεται με χρήση νημάτων
1x producer Thread & C_THREADS x Consumer thread.
Το νήμα producer αποθηκεύει νέες λέξεις προς αναζήτηση στη στοίβα Server_Con_Stack και αυξάνει το “δείκτη” Cur_Server_Con στη θέση της τρέχουσας λέξης προς αναζήτηση (εαν ειναι -1 τότε η στοίβα είναι κενή). Εάν η στοίβα είναι γεμάτη τότε περιμένει μέχρι να αδειάσει κάποια θέση και να τοποθετήσει κάποια νέα λέξη. Εάν η στοίβα ήταν κενή πρίν την εισαγωγή της λέξης, τότε αφού την εισάγει, ενημερώνει τα νήματα consumers πως υπάρχουν διαθέσιμες λέξεις προς αναζήτηση.
Το κάθε νήμα consumer διαβάζει από τη στοίβα μία λέξη, της αφαιρεί απο τη στοίβα και μετακινεί το δείκτη στοίβας, και την αναζητά στο σύνολο των αρχείων. Όταν διαβάσει μία λέξη ελέγχει αν υπάρχει και άλλη λέξη στη στοίβα, εάν υπάρχει την αναζητεί και αυτή. Όσο υπάρχουν λέξεις προς αναζήτηση το νήμα την εκτελεί. Εάν η στοίβα είναι κενή τότε το νήμα περιμένει μέχρι να επανεισαχθεί κάποια λέξη. Εάν η στοίβα ήταν γεμάτη πρίν την ανάγνωση της λέξης τότε το νήμα ενημερώνει τον producer πως υπάρχει διαθέσιμος χώρος για νέες εισαγωγές. Κατά την εγγραφή ή την ανάγνωση από την στοίβα Server_Con_Stack εφαρμόζεται αμοιβαίος αποκλεισμός μεταξύ των νημάτων ώστε μόνο ένα να έχει πρόσβαση.
(διευκρινίζουμε πως όταν λέμε πως γίνεται ανάγνωση ή προσθήκη λέξης στη στοίβα, αναγνώσκεται ή προστίθεται ο file descriptor μίας σύνδεσης η οποία αποστέλει τη λέξη)

Server-ΣΥΝΑΡΤΗΣΕΙΣ & ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Αρχικά γίνονται define τα εξής:
#define C_THREADS (Πλήθος Consumer threads)
#define MY_PORT (port στο οποίο επικοινωνούν οι clients με τον server)
#define BUF_SIZE (μέγιστο μέγεθος των buffers)
#define MAX_PENDING_CONNECTIONS (μέγιστο μέγεθος συνδέσεων την κάθε στιγμή)

Μετά ορίζονται οι μεταβλητές:
int completed_requests = 0 (πλήθος εκτελεσμένων αναζητήσεων)
struct timeval total_waiting_time (διάρκεια αναμονής της κάθε σύνδεσης client μέχρι να εξυπηρετηθεί)
struct timeval total_service_time (διάρκεια εκτέλεσης αναζητήσεων)

int exit_status = 0; (exit flag για έλεγζο εξόδου σε περίπτωση λήψης κάποιου σήματος που γίνεται hadle από το server)
typedef struct dcon {
int con;
struct timeval stime;
}dcon; (τύπος δεδομένων dcon ο οποίος μορφοποιεί τους κόμβους της στοίβας μας)
dcon Server_Con_Stack[MAX_PENDING_CONNECTIONS] (στοίβα αποθήκευσης των fd των συνδέσεων)
int Cur_Server_Con=-1 (δείκτης στη στοίβα συνδέσεων)
pthread_t Producer_Thread (το thread producer)
pthread_t Consumer_Thread[C_THREADS] (τα consumer threads)
pthread_mutex_t Mexcl_mutex=PTHREAD_MUTEX_INITIALIZER (mutex το οποίο κλειδώνει και ξεκλειδώνει την πρόσβαση στη στοίβα εξασφαλίζοντας με αμοιβαίο αποκλεισμό την πρόσβαση ενός μόνο thread κάθε φορά)
int socket_fd=-1; (socket στο οποίο λαμβάνει νέες client συνδέσεις ο server)

pthread_mutex_t Texcl_mutex=PTHREAD_MUTEX_INITIALIZER; (mutex το οποίο κλειδώνει και ξεκλειδώνει την πρόσβαση στις μεταβλητές total_waiting_time, total_service_time editing και completed_requests εξασφαλίζοντας με αμοιβαίο αποκλεισμό την πρόσβαση ενός μόνο thread κάθε φορά)
pthread_cond_t Non_full_stack=PTHREAD_COND_INITIALIZER (μεταβλητή condition που περιγράφει την κατάσταση FULL της στοίβας)
pthread_cond_t Non_empty_stack=PTHREAD_COND_INITIALIZER (μεταβλητή condition που περιγράφει την κατάσταση EMPTY της στοίβας)
pthread_attr_t attr (μεταβλητή attribute που καθορίζει αν το νήμα θα δημιουργηθεί στην κατάσταση detached ή joinable)
(οι μεταβλητές *_INITIALIZER αρχικοποιούνται στις default τιμές)

ΣΥΝΑΡΤΗΣΗ: void process_request(int socket_fd)

Τοπικές μεταβλητές:
char buffer[BUF_SIZE]="\n";
char filebuf[BUF_SIZE]; buffer for word (for fscanf)
char word_to_search[BUF_SIZE]; the word that has to be searched in the files
int word_size;
int numbytes, i, occurrences;
FILE *search_fd; he file descriptor for the opened file

Λειτουργίες: Η συνάρτηση διαβάζει από μία σύνδεση (socket fd) την λέξη προς αναζήτηση και την αναζητεί στα αρχεία που υπάρχουν στον πίνακα filenames μετρώντας, με χρήση της occurences, το πόσες φορές αυτή εμφανίζεται σε κάθε ένα από τα αρχεία. Μετά στέλνει τα αποτελέσματα στο socket.

ΣΥΝΑΡΤΗΣΗ: void *myfunc_Consumer(void* arg)

Τοπικές μεταβλητές:
int c_socket_fd, i;
struct timeval tv, tv2;

Λειτουργίες: Αυτή είναι η συνάρτηση που εκτελούν τα νήματα καταναλωτή (consumers). Το νήμα εκτελεί ατέρμονα (μέχρι το κλείσιμο της διεργασίας server) τις ίδιες λειτουργίες.
Κλειδώνει το mutex ώστε να έχει μόνο αυτή πρόσβαση στη στοίβα συνδέσεων.
Εάν η στοίβα είναι άδεια περιμένει μέχρι να πάψει να είναι άδεια (Non_empty_stack condition). Όσο το νήμα περιμένει το mutex ξεκλειδώνεται, όταν η στοίβα πάψει να είναι άδεια και δεν έχει προλάβει κάποιο άλλο νήμα consumer να κλειδώσει το mutex, τότε κλειδώνει το mutex ξανά. Μετά αφαιρεί από τη στοίβα ένα στοιχείο (μία σύνδεση με socket) και χρησιμοποιώντας το mutex υπεύθυνο για την πρόσβαση στις μεταβλητες που υπολογίζουν τους χρόνους εκτέλεσης και αναμονής ενημερώνει τη συνολική διάρκεια αναμονής της εκάστοτε σύνδεσης client στη στοίβα. Μετά ξεκλειδώνει το mutex, μιας και πλέον δεν χρειάζεται να έχει πρόσβαση στη στοίνα. Μετά, εάν η στοίβα ήταν γεμάτη πριν την αφαίρεση του στοιχείου, ειδοποιεί τον producer ότι πλέον υπάρχει διαθέσιμος χώρος. Τέλος εκτελεί την process_request με γνώρισμα το socket fd που πήρε από τη στοίβα ενημερώνοντας και τον συνολικό χρόνο εκτέλεσης των αναζητήσεων.

ΣΥΝΑΡΤΗΣΗ: void *myfunc_Producer(void* arg)

Τοπικές μεταβλητές:
int new_fd μεταβλητή που χρησιμοποιείται για την τοποθέτηση νέων connections στο stack
socklen_t clen;
struct sockaddr_in server_addr, my address information
client_addr; connector's address information
struct timeval tv;
int i;
struct pollfd fds[2]; μεταβλητή που χρησιμοποιείται από την poll
int fds_items=2; >>

Λειτουργίες:
Αυτή είναι η συνάρτηση που εκτελεί το νήμα παραγωγού (producer).
Δημιουργεί το socket στο οποίο ο server θα δέχεται συνδέσεις από νέους clients και τοποθετεί τις νέες συνδέσεις στο stack των νέων συνδέσεων προς εξυπηρέτηση από τους consumers.

Ακολουθιακά:

-Δημιουργεί ένα socket στο οποίο θα δέχεται συνδέσεις από clients.
-Θέτει το socket σε NONBLOCK για να μπορούμε να κάνουμε ελέγχους με την poll
-Θέτει το socket σε SO_REUSEADDR ώστε να μπορεί μία νέα διεργασία να επαναχρησιμοποιήσει (bind) το Port το οποίο βρίσκεται ακόμα σε TIME_WAIT (μετά από τερματισμό προηγούμενου bind)
-Γίνεται bind του socket στο Port
-Γίνεται listen ώστε να αρχίσουμε να “ακούμε” για εισερχόμενες συνδέσεις
-while loop που ελέγχει για τερματισμό προγράμματος
Μέσα στο loop η poll ελέγχει αν υπάρχουν νέα εισερχόμενα (POLL_IN) δεδομένα στο socket (άρα νέα σύνδεση από client)
Αν υπάρχει νέα σύνδεση, κλειδώνει το mutex ώστε μόνο αυτό να έχει πρόσβαση στη στοίβα συνδέσεων. Ελέγχει εάν η στοίβα είναι γεμάτη. Εάν είναι γεμάτη τότε περιμένει μέχρι υπάρξει χώρος (wait για την κατάσταση Non_full_stack). Όσο περιμένει το mutex ξεκλειδώνεται, εάν υπάρξει διαθέσιμος χώρος στη στοίβα και δεν έχει προλάβει κάποιο νήμα καταναλωτή να κλειδώσει το mutex, τότε κλειδώνει το mutex. Προσθέτει ένα νέο στοιχείο (νέο socket fd) στη στοίβα και ξεκλειδώνει το mutex. Μετά, εάν η στοίβα ήταν κενή, ενημερώνει όλες τις διεργασίες consumer (broadcast) πως υπάρχει νέα καταχώρηση και τερματίζει.

ΣΥΝΑΡΤΗΣΕΙΣ: stop_occured, exit_occured
Συναρτήσεις που καλούνται για την διαχείρηση κάποιου σήματος.
Αν ληφθεί SIGINT η exit_occured ξεκινά διαδικασίες τερματισμού προγράμματος καλώντας την exit_handle
Αν ληφθεί SIGSTP η stop_occured εκτυπώνει τους χρόνους εξυπηρέτησης και αναμονής των requests ξεκινά διαδικασίες τερματισμού προγράμματος καλώντας την exit_handle
Η exit_handle θέτει το exit flag exit_status=1 και στέλνει το σήμα pthread_cond_signal(&Non_full_stack) στο producer thread για να ξεμπλοκάρει σε περίπτωση που, όταν λήφθηκε το σήμα, ανέμενε για διαθέσιμο χώρο στη στοίβα (wait για την κατάσταση Non_full_stack).

ΣΥΝΑΡΤΗΣΗ: main

Τοπικές μεταβλητές:
int i;
struct sigaction a,b; signal handlers για SIGSTP και SIGINT

Λειτουργίες:
Στη main αρχικά δημιουργούνται οι signal handlers για την εκτέλεση εργασιών σε περίπτωση λήψης κάποιοι σήματος (SIGINT, SIGSTP)

Βρίσκει ποιά είναι τα αρχεία στα οποία θα γίνονται οι αναζητήσεις και αποθηκεύει τα ονόματά τους στον πίνακα filenames.

Ακολουθεί η αρχικοποίηση των στοιχείων της στοίβας Server_Con_Stack σε -1.

Ορίζεται η τιμή DETACHED (και όχι JOINABLE) στο attr ώστε τα νήματα καταναλωτή να καταστρέφονται αυτόματα με την επιστροφή τους.

Ορίζεται η τιμή JOINABLE (και όχι DETACHED) στο attr ώστε η main να περιμένει για τερματισμό του νήματος παραγωγού
Αφού δημιουργηθεί το νήμα παραγωγού γίνεται join και ξεκινά η αναμονή για τον τερματισμό του
Όταν τερματίσει το νήμα παραγωγού κλείνει το socket_fd και τερματίζουμε

Άλλα αρχεία

utils.h : Η βιβλιοθήκη που συμπεριλαμβάνεται σε όλα τα υπόλοιπα αρχεία. Περιέχει βιβλιοθήκες οι οποίες χρειάζονται για την εκτέλεση των διεργασιών του προγράμματος. Επίσης ορίζει τις συναρτήσεις void ERROR(char *msg); , int write_str_to_socket(int socket_fd, char *buf, int numbytes); , int read_str_from_socket(int socket_fd, char *buf, int bufsize);
προσθέσαμμε τις βιβλιοθήκες :
#include <sys/types.h>
#include <sys/signal.h>
#include <unistd.h>
#include <sys/time.h>

utils.c : Εδώ υπάρχει ο κώδικας των συναρτήσεων που ορίζονται στο utils.h

group : Τα ονόματά μας.

README.pdf : Εξηγεί και αναλύει το σκεπτικό και τις λειτουργίες του προγράμματος.

f*.txt : Τα αρχεία που περιέχουν τα κείμενα στα οποία αναζητούνται οι λέξεις.

Makefile : Περιέχει τις εντολές για το compilation του προγράμματος.

test.ph: Bash script το οποίο δημιουργεί το directory samples και τοποθετεί τα αρχεία αναζήτησησης σε αυτό. Μετα εκτελεί make all και δίνει ένα menu με επιλογές για την εκτέλεση διαφόρων εντολών με σκοπό τον έλεγχο του προγράμματος.

ΣΗΜΕΙΩΣΕΙΣ

1. Το πρόγραμμά μας βασίζεται στο δοθέν samplecode. Εμπλουτίσαμε τις λειτουργίες ώστε να υποστηρίζονται νήματα και ταυτόχρονη αναζήτηση. Επίσης προσθέσαμε την υποστήριξη εισόδου πολλών λέξεων προς αναζήτηση σε μία μόνο κλήση της client.

2. Ορίσαμε το πλήθος των νημάτων consumer του server να είναι 6. Το λογικά μεγιστο δυνατό θα ήταν να βάλουμε 10, όσο και το μέγεθος της στοίβας, πράγμα που όμως δεν θα ανέβαζε την απόδοση επειδή ο επεξεργαστής θα έπρεπε να εκτελέσει 10 αντί για 6 νήματα με αποτέλεσμα την καθυστέρηση της εκτέλεσης του εκάστοτε νήματος διότι θα έπρεπε να εξυπηρετηθούν και τα υπόλοιπα 9. Επίσης με τον ορισμό 6 consumers, ο producer μπορεί να εγγράψει ποιό γρήγορα μία νέα σύνδεση στη στοίβα. Λόγω του αμοιβαίου αποκλεισμού μόνο ένα νήμα έχει πρόσβαση στη στοίβα, έτσι λοιπόν στη χείρηστη περίπτωση που 10 νήματα καταναλωτή περιμένουν να αναγνώσουν απο τη στοίβα ένα νήμα producer θέλει να εγγράψει. Ο porducer θα μπορέσει να έχει πρόσβαση στη στοίβα μετά απο 10 αναγνώσεις. Αντιθέτως στην περίπτωση των 6 νημάτων consumer θα μπορεί να έχει πρόσβαση μετά από max6 αναγνώσεις εξασφαλίζοντας μία καλύτερη ροή εγγραφών/αναγνώσεων στη στοίβα. Συγκρίναμε τους χρόνους αναμονής χρησιμοποιώντας διαφορα πλήθη από consumer threads για να καταλήξουμε στο άνωθεν συμπέρασμα.
   Για παράδειγμα σας παρουσιάζουμε έναν από τους ελέγχους που κάναμε. Είναι τα αποτελέσματα πολλών αναζητήσεων η κάθε μία από τις οποίες αναζητεί 30 λέξεις χρησιμοπώντας 3(αρχικούς) clients (κάθε client έχει είσοδο 10 λέξεις). Δίνουμε το πλήθος των consumer threads και από 2 αποτελέσματα των αναζητήσεων για το κάθε πλήθος.

   3 consumer threads
   Requests serviced: 30 .
   Total waiting time (microsec): 16560112 .
   Total service time (microsec): 14569786 .

   4 consumer threads
   Requests serviced: 30 .
   Total waiting time (microsec): 12667376 .
   Total service time (microsec): 11481739 .

   5 consumer threads
   Requests serviced: 30 .
   Total waiting time (microsec): 7910828 .
   Total service time (microsec): 19720875 .

   6 consumer threads
   Requests serviced: 30 .
   Total waiting time (microsec): 7827602 .
   Total service time (microsec): 20655685 .

   7 consumer threads
   Requests serviced: 30 .
   Total waiting time (microsec): 9559862 .
   Total service time (microsec): 33553185 .

   8 consumer threads

Requests serviced: 30 .
Total waiting time (microsec): 16707151 .
Total service time (microsec): 13667385 .

3. Μετά την αρχική αποσφαλμάτωση κάναμε ελέγχους με διάφορες εισόδους (με λιγότερες εισόδους από τις απαιτούμενες και με εισόδους που υπερέβεναν τα όρια του προγράμματος) και είδαμε ότι το πρόγραμμά μας αποδίδει σωστά ακόμα και στις εξαιρετικές περιπτώσεις.

4. Σε περίπτωση που χρησιμοποιήσουμε ένα τέρμιναλ ο server πρέπει να εκτελεσθεί στο παρασκήνιο με την εντολή “./server &”, για την αποστολή του SIGINT πρέπει να εκτελεσθεί “kill -INT (server pid)” (βρίσκουμε το pid μέσω ps -all) και όχι ctrl+c.
   Σε περίπτωση που θέλουμε να στείλουμε σήμα SIGINT στο server με χρήση ctrl+c πρέπει να εκτελέσουμε σε ένα terminal τον server στο προσκήνιο “./server” και σε ένα άλλο terminal τους clients. Έτσι μπορούμε να δούμε τα αποτελέσματα για το πλήθος των λέξεων που έχουν αναζητηθεί, τους χρόνους αναμονής και εκτέλεσης και να τερματίσουμε τον server με ctrl+c στο terminal που τρέχει Αναλόγως για σήμα SIGSTP (kill -SIGTSTP (server pid)”