Εργαστήριο Δικτυων Υπολογιστών

Εργασια I

Διονύσης Οδυσσέας Σωτηρόπουλος.
A.M. 5661

Μάιος 2017

Server 1: Iterative Server

Ο πίο απλός σέρβερ που καλούμαστε να υλοποιήσουμε μπορεί να εξυπηρετέι έναν client τη φορά. Σημειώνουμε εδώ οτί ο κώδικας για το serv1 θα αποτελέσει βάση για τους υπόλοιπους 3.

Βασικές Ιδέες για την υλοποίηση του.

Ο σέρβερ αφού κάνει τις απαραίτητες λειτουργίες για να στηθεί (βλ. Επεξήγηση κώδικα) θα πρέπει να κρατάει ανοιχτή τη σύνδεση επ'άπειρον, περιμένοντας κάποιον client. Αφού ενας client συνδεθεί θα πρέπει να επεξεργάζεται τις κωδικοποιημένες εντολές, να εκτελεί τις ανάλογες διαδικασίες και να στέλνει πίσω μία απάντηση.

Επεξήγηση Κώδικα

Δομή:

Αρχικά γινονται οι δηλώσεις των βιβλιοθηκών που θα χρησιμοποιήσουμε, στατικές τιμές για το database κλπ. Και στη συνέχεια δηλωσεις των custom συναρτήσεων καθώς και global variables που θα διαχειρίζονται ορισμένες απο τις συναρτήσεις μας.
Μετά τη main συνάρτηση έχουμε τις συναρτήσεις μας.

Υλοποίηση:

1.Αρχικά ο server μας δημιουργεί μια sample database την οποια θα χρησιμοποιούμε για να ελέγξουμε τη λειτουργία του με την κλήση της συνάρτησης create_store()
Επεξήγηση της λειτουργίας της create_store καθώς και σχόλια υπάρχουν στο αρχείο πηγιαιου κώδικα.
2.Δηλωση μεταβλητών.
3.Προστασία κώδικα σε περίπτωση ανεπαρκή input.
4. Ανοίγουμε το socket του server τυπου Ipv4, streaming kai χρήση TCP.
5.Καθαρίζουμε και γεμίζουμε τη δομή του σέρβερ (serv_addr) με port number Ipv4 τύπο internet adress και IN_ADDR_ANY για τη γενική περίπτωση.
6.Συσχετίζουμε το socket με τη διεύθυνση του σέρβερ (bind)
7. Ρυθμίζουμε το σέρβερ να "ακούει" για συνδέσεις

Απο εδώ και μέχρι το τέλος της main ο κώδικας βρίσκεται σε συνεχή λούπα έτσι ωστε να μην τερματίζει όταν εξυπηρετήσει κάποιο client και να συνεχίζει να δέχετε συνδέσεις.

8. δημιουργία χαρακτηριστικής μεταβλητής για socket και δομής για τον client.
9.Accept, δεχόμαστε τη σύνδεση με τον client και συσχετίζουμε τις παραπάνω μεταβλητές με αυτό. Καθαρίζουμε τον buffer που θα δεχθεί το μήνυμα του client.
10. Αποδοχη (receive) του μηνύματος του client και αποθήκευση του στον buffer.
11. Ανάγνωση (read) του μηνύματος από το socket.

12. ΕΠΕΞΕΡΓΑΣΙΑ ΕΝΤΟΛΩΝ ΤΟΥ CLIENT

Διαβάζουμε τον buffer ψάχνοντας το χαρακτηριστικό p ‘η g για τις εντολές put και get.
Σε περίπτωση put, ρυθμίζουμε pointers στο <key> και στο <value>, καλούμε την put συνάρτηση.
Τέλος αυξάνουμε τον pointer να δείχνει στην επόμενη εντολή.
Σε περίπτωση get ρυθμίζουμε τον pointer να δείχνει στο <key> καλούμε την get. Σε αυτό το σημείο χρησιμοποιούμε το global variable exitstat που δηλώνει αν πρέπει να τερματιστεί η σύνδεση λόγω λάθος key (πιο αναλυτικά παρακάτω) και να στείλει πίσω στον client την -μεχρι τωρα επεξεργασμένη- απάντηση.
Αποστολή εντολής με χρήση write συνάρτησης.
Τέλος κάνουμε reset τις global μεταβλητές, καθαρίζουμε τον buffer και κλείνουμε το socket του client πριν ξαναξεκινήσει η λούπα.

ΣΥΝΑΡΤΗΣΕΙΣ:

Θα ξεκινήσουμε απο τις συναρτήσεις put και get διότι περιέχουν κλήσεις σε άλλες συναρτήσεις και είναι και οι 2 βασικές ζητούμενες εξ εκφωνήσεως συναρτήσεις.

Get:

Συνοπτικά η λειτουργία της get ειναι να ελέγχει αν το <key> που παίρνουμε απο τη συνάρτηση getkey είναι ‘n’ (οτι δλδ δεν υπάρχει τέτοιο κλειδι στη βαση) και αν είναι ‘n’ να το γράφει στον buffer απάντησης και να τερματίζει, αλλιώς να γράφει ‘i’ και τη ζητούμενη τιμή αυξάνοντας ένα pointer κατα το μέγεθος του value slot της βάσης μας.
Σημειώνουμε οτι από εκφώνηση μας ζητείτε η get να επιστρέφει ένα δείκτη στο ζητούμενο value το οποίο όμως δεν χρησιμοποιούμε.

Put:

Η put βρίσκει το <key> στη βάση δεδομένων με τη χρήση της find_key και αυξάντοντας ένα pointer κατα το μέγεθος του keyslot της βάσης καθαρίζει το value και μεταφέρει το ζητούμενο εκεί.

find_key:

Η συνάρτηση αυτή ψάχνει τη βάση δεδομένων για ένα συγκεκριμένο <key> και αν το βρεί επιστρέφει ένα δείκτη σ αυτό. Σε περίπτωση που η λούπα ολοκληρωθεί χωρίς να έχει βρεθεί το κλειδι επιστρέφει ενα δείκτη στον κωδικό λάθους κλειδιού ‘n’.

find_ans_next:

“Βρές την επόμενη θέση στο buffer απάντησης”
Μια απλή συνάρτηση που ψάχνει να βρεί το επόμενο άδειο κελί στον buffer απάντησης και δίνει την διεύθηνση του στο global variable curr_ans_cell (current answer cell).
Σημειώνεται οτι κάθε φορά που μεταφέρουμε ένα string στον buffer, αυτό τερματίζει με ‘\0’, η συνάρτηση θα δείχνει σε αυτό έτσι ωστέ να μπορούμε να πετύχουμε το ζητούμενο byte-setup που ζητείται (g<val>\0)
Κάθε φορά που καλείται η συνάρτηση αυτή πάντα θα γραφτεί κάτι στον buffer. Έτσι την επόμενη φορά που θα ξανακληθεί το παρών κελί θα είναι διάφορο του ‘\0’ και θα ανατρέξει στο τέλος της νέας εισαγμένης λέξης.

ans_setup:

(global variables: order + exitstat)
answer_setup, χρησιμοποιείτε για να εισάγει τις ζητούμενες τιμές απο τις get εντολές του client στον buffer απάντησης. Χρησιμοποιεί την find_ans_next για να ξέρει από που θα ξεκινήσιε να γράφει.
Η ans_setup ρυθμίζει τα ‘\0’ μετά από κάθε <val>.
Ελέγχει αν η εντολή είναι η πρώτη που γράφεται στο buffer. Αν οχι(order=0) τότε αφήνει ένα κενό κελί έτσι ωστε το προηγούμενο <val> να είναι null terminated string (αυξάνοντας το curr_ans_cell να

δείχνει στο επόμενο κελι). Επίσης παγιδέυει την περίπτωση λάθος κλειδιού σετάρωντας το exitstat σε 1 έτσι ωστε η σύνδεση να τερματιστεί και να σταλθεί η παρούσα απάντηση χωρίς να επεξεργαστούν οι υπόλοιπες εντολές του client.

send_answer:

χρησιμοποιείται μόνο στην περίπτωση λανθασμένου <key>.
Η περίπτωση αυτή θέλει ειδικό μπάλωμα διότι ξεφεύγει από την ομαλή λειτουργία του σέρβερ αλλά δεν τον τερματίζει.
Αρχικά δηλώνουμε δείκτη στον buffer απάντησης και εναν ακέραιο exit.
Διαβάζοντας ένα ένα τα κελιά του buffer μετράμε '\0' στη σειρά αυξάνοντας το exit με αρχική τιμή -1. Αν ξεπεράσει το 0 (2 στη σειρά) τότε (1. ελέγχουμε αν πρόκειτε για την περίπτωση απάντησης "n " και 2.) γράφουμε την απάντηση στο socket "κουρεύοντας" τα τελευταία '\0'
(αφήνοτας κανενα σε περίοπτωση 'n' λαθος κελιδιού ή ένα σε περπιτωση <val> null terminated string).
Σημειώνουμε οτι η συνάρτηση αυτή διμιουργήθηκε για τη γενική περίπτωση αποστολής απάντησης στον client ανεξάρτητα που τη χρησιμοποιούμε συγκεκριμένα για μία περίπτωση.

get_buffclip:

Παρόμοια με την θεωρία της send_answer αυτή η συνάρτηση επιστρέφει τον αριθμό των ελάχιστων κελιών του γεμάτου buffer απάντησης. Μετράει NULL χαρακτήρες '\0' και όταν ο μετρητής nncnt (null-null counter) φτάσει το 2 τερματίζει τη λούπα και επιστρέφει την διεύθυνση του προηγούμενου κελιού.

create_store:

αυτή η συνάρτηση διμιουργεί μια sample database. Δεν θα επεξηγηθει εδώ η λειτουργία της καθώς δεν είναι ζητούμενο της εργασίας, παρ'όλα αυτα υπάρχει επεξήγηση της στον πηγιαιο κώδικα.

Επιπλέον Σημειώσεις για τον Σέρβερ:

Η αναβάθμιση του σέρβερ έτσι ωστε να μπορει να καλύψει οσο το δυνατο περισσότερες ακραίες περιπτώσεις λειτουργιας καθώς και η προσπάθεια να περάσει τον έλεγχο του grader εχουν οδηγήσει με επιγνωση μου σε μη λειτουργικές λύσεις.
Θα ήθελα να αναφέρω την εξής παρατήρηση. Καθώς η get πρέπει να επιστρέφει δείκτη, στην περίπτωση που έχουμε λάθος κλειδί θα πρεπει να επιστρέφει δείκτη στο 'n'. Οπότε δηλώνοντας
char i = 'i';
char n = 'n';
παρατήρησα οτι η απάντηση περιείχε 'ni' αντί για 'n'. (Υποθέτω οτι βρίσκονται σε γειτωνικές θέσεις και τυπώνονται μαζι εφόσων δεν υπάρχει '/0' αναμεσά τους. Εξού και η αναβαθμισμένη "περίεργη" υλοποίηση.(βλ. Κώδικα)

Client:

Επεξήγηση κώδικα.

1. Σετάρουμε το μέγεθος του buffer που θα στείλουμε στο σέρβερ.
2. Δηλώσεις μεταβλητών(buffer, socket, port, sock_addr struct
3. Δημιουργούμε socket όμοια με του server.
4. Ρυθμίζουμε το ip του σέρβερ, port number, Ipv4
5. Συνδεση (connect) με το σέρβερ
6. ΕΠΕΞΕΡΓΑΣΙΑ ΕΙΣΟΔΟΥ

Σε αυτό σημείο διαβάζουμε ενα ενα τα ορίσματα (input arguments) και τοποθετουμε αναλογως g<key>\0 η' p<key>\0<val>\0 στο buffer. Σημειώνεται οτι εδώ χρησιμοποιούμε ανάλογες συναρτήσεις όπως αυτές του server (findBuffNext→find_ans_next) και global values (cur_cell_num→curr_ans_cell). Επιπλέον σχόλια στον κώδικα.
7. Στέλνουμε αίτημα στο σέρβερ με το ελάχιστο μήκος χρησιμοποιόντας την get_buffclip.
8. Καθαρίζουμε τον buffer.
9. Διαβάζουμε την απάντηση
10. μεταφράζουμε (απο i<val>) και τυπώνουμε τα <val>.
11. Κλείνουμε το socket.

Μέχρι εδώ έχουμε καλύψει το μεγαλύτερο κομμάτι της εργασίας. Η παραπάνω δομή και συναρτήσεις θα χρησιμοποιηθούν για την υλοποίηση των παρακάτω γι αυτό και δεν θα επαναληφθούν

Server 2: On Demand Forking Server

Για την υλοποίηση του 2ου σέρβερ χρησιμοποιήθηκε ο κώδικας του πρώτου σερβερ.
Επιπλέον δηλώνεται μία σημαφόρος που θα κλειδώνει κάθε φορά που η διεργασία διαχειρίζεται τη βάση δεδομένων. Η συνάρτηση accept() εκτελείται σε ατέρμωνη λούπα καλλώντας fork() για κάθε καινούρια σύνδεση. Η διεργασία παιδί εμπεριέχει τον υπόλοιπο κώδικα του σέρβερ 1 ενώ διεργασία γονιός δεν εκτελεί καθόλου κώδικα.
Επίσης χρησιμοποιούνται οι shmget shmat συναρτήσεις στην create_store() προκειμένου η βάση να λειτουργεί ως κοινή μνήμη για τις διεργασίες παιδιά.
Σημειώνεται οτι, παρόλο που έχω δηλώσει σημαφόρο, αυτη δέν έχει χρησιμοποιηθεί διότι (λόγω λάθος χειρισμού της) το πρόγραμμα παγώνει.

Serv 3: Preforking Server

Παρομοίως ο 3ος σέρβερ περιέχει τον κώδικα του 2ου σέρβερ με παραλλαγές.
Στη συγκεκριμένη υλοποίηση η fork καλείται σε λουπα που διατρέχετε οσες φορές όσες το 2ο όρισμα και η συνεχής λούπα εμπεριέχεται στην διεργασία παιδί. Τα υπόλοιπα ομοίως.
Σημειώνεται οτι έχουν ληφθεί μέτρα για το 3ο όρισμα του σέρβερ αυτου σε αντίθεση με τις υπόλοιπες υλοποιήσεις που έχουν μόνο το port number. (π.χ. error()).
Επίσης οι διεργασίες παιδιά ΔΕΝ μπλοκάρωνται από τις διεργασίες γονείς (parent process δεν έχει wait).

Serv 4: Multi-Threaded Server

Ομοίως με τους προηγούμενους servers το βασικό κομμάτι κώδικα και η λογική είναι (από εκφώνηση) ίδια με του 2ου σέρβερ.
Σε αυτή την υλοποίηση το block κώδικα που χρησιμοποιούσαμε ώς child process code (στους servers 2 κ 3) πρέπει να κληθεί ως συνάρτηση-thread (tserv()). Επίσης έχει χρησιμοποιηθεί mutex για συγχρωνισμό των threads (κλείδωμα της βάσης) όταν αυτά γράφουν στην βάση.

Σημειώσεις: Η εργασία έχει εξεταστεί λειτουργικά μονάχα σε Virtual Machine με τοπικό IP. Παρόλα αυτά ο Grader εμφανίζει λάθη όχι μόνο στα tests αλλά και κατά τη μετάφραση πράγμα που δεν συνέβη στο αρχικό σύστημα. Και με όλο το σεβασμο επειδή συνέβη κάτι ανάλογο και σε προηγούμενο μάθημα (εργαστήριο λειτουργικών) παρακαλω πολυ να εξεταστεί ο κωδικας από τον ίδιο τον καθηγητή ή κάποιο μεταπτυχιακό και όχι απλώς να περαστεί το σκορ του grader ως βαθμος...

ffs

Περιορισμοί Εκφώνησης και Θεωρητικές Αναβαθμίσεις Κώδικα

Σύμφωνα με τα αποτελέσματα του grader ζητείται ο κώδικας να προστατεύεται από συγκεκριμένες περιπτώσεις input, παρακάτω περιγράφονται πιθανές λύσεις που απορρίπτονται από την εκφώνηση.

1. Long keys and Values: καθώς οι συναρτήσεις ανάγνωσης και σύνταξης των buffers στους servers αλλα και στον client φαίνεται να μπορουν να ανταπεξέλθουν, για το error ευθύνεται:

α) Η δομή της βάσης δεδομένων: τα κελιά των keys, values έχουν συγκεκριμένο μήκος. Θα μπορούσαμε ίσως να χρησιμοποιήσουμε vectors για την βάση δεδομένων αλλά δηλώθηκε πως δέν θα δωθεί σημασία στη δομή της βασης.

b) Buffer Overflow: επειδή χρησιμοποιούμε την read() συνάρτηση προκειμένου να διαβάσουμε τους buffers των sockets απο client σε server και το αντίθετο πρέπει να ξέρουμε το μέγεθος του buffer εκ των προτέρων. Αποφεύγοντας αυτό, θα υλοποιούσα κώδικα έτσι ώστε να στέλνεται αρχικά ένα socket συγκεκριμένου μήκους γνωστό στον δέκτη με το τελευταίο byte δεσμευμένο για να δηλώνει άν ο δέκτης θα πρέπει να περιμένει επόμενο μήνυμα.

CODE

Client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define BUFF_SIZE 64

int cur_cell_num = 0;//Global Variable of current cell of buffer
int bufflen=0;

void error(char *msg);
int findBuffNext(char buffer[], int size);
void print_buffer(char buffer[], int size);
int get_buffclip(char buffer[]);

//-=-=-=-=-=-=-==-=-=-==-=-=-===-=-=-=-=-=-=-=-==-=-///
                                //MAIN///


int main(int argc, char *argv[])
{
        //---Buffer Length Management---//
        //Bufflen var will be larger than needed socket buffer :(
        int i;

        for(i=0; i<argc; i++)
        {
                bufflen+=strlen(argv[i]);//count chars in args
        }

        char buffer[bufflen];
        int sockfd, portno, n;

        struct sockaddr_in serv_addr;//create structs
        struct hostent *server;
```

```
	//int buf_size = sizeof(buffer)/sizeof(buffer[0]);


	if (argc < 3){		//trap arguements < 3//
	fprintf(stderr, "usage %s hostname port\n", argv[0]);
	exit(1);
		}
	//---set port number---//
	portno = atoi(argv[2]);					//set set Port Number as 2nd arguement
	sockfd = socket(AF_INET, SOCK_STREAM, 0);//Create the default Socket
	if (sockfd < 0)						//trap socket creation Error
		error("ERROR opening socket");

	//---set server ip as 1st arguement---//
	server = gethostbyname(argv[1]);
	if(server == NULL) {		//trap no 1st arguement??
		fprintf(stderr, "ERROR, no suck host\n");
		exit(1);
				}

bzero((char*)&serv_addr,sizeof(serv_addr));	//erease server adress struct
serv_addr.sin_family = AF_INET;		//Start setting struct values. Set Family(v4-v6)
bcopy((char *)server->h_addr,
	(char*)&serv_addr.sin_addr.s_addr,
	server->h_length);
//printf("Checkpoint: Setting Port Number \n");
serv_addr.sin_port = htons(portno);//set port number
printf("Checkpoint: Initializing Connection\n");



				//---Connect---//
if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
	error("ERROR connecting");//trap connection to server fail
printf("Checkpoint: Connection Complete\n");

//Instruction Management II//
//In this section of code we will manage the input and store it in a buffer that will later be sent to the server through the socket.
//The order will be (g<KEY>\0 | p<KEY>\0<VAL>\0)*

bzero(buffer, bufflen);//clear buffer


int order=0;//initial order space: this will change to 1 after the first put/get
i=3;//loop counter. val=3 so it can jump over ip and port
char * buf_ptr, argv_ptr; //pointers to buffer and arguements
buf_ptr = buffer;//initialize buffer pointer to 1st element of buffer

while(i<argc)//parse through rest arguements
	{
		if(strcmp(argv[i], "get")==0)//get order manage
		{
	//inc the global var to jump over the \0 char of the last command. this will be a +0 in case it's the 1st command
		cur_cell_num+=order;
buf_ptr =&buffer[findBuffNext(buffer,bufflen)];//find next terminal char
		*buf_ptr=(char)0x67;			//write get opcode in buffer
buf_ptr=&buffer[findBuffNext(buffer, bufflen)];//move ptr to slot next to g_opcode
		i++;//inc i to get get key
		strcpy(buf_ptr, argv[i]);//move key to buffer
		buf_ptr=&buffer[findBuffNext(buffer,bufflen)];//update buf pointer to point at key's terminal char in buffer

		}else if(strcmp(argv[i], "put")==0)
		{
		cur_cell_num+=order;//<<same as above>>
```

```
//buf_ptr = buffer;//initialize buffer pointer to 1st element of buffer
buf_ptr=&buffer[findBuffNext(buffer, bufflen)];//find terminal char
                        *buf_ptr=(char)0x70;                    //write put opcode in buffer
buf_ptr=&buffer[findBuffNext(buffer, bufflen)];//point to slot next to p_opcode

                        i++;//inc i to get put key
                        strcpy(buf_ptr, argv[i]);//move key 2 buffer directly behind put
                        argv_ptr++;//point to next arguement(value)
buf_ptr=&buffer[findBuffNext(buffer, bufflen)];//point to key's terminal char
                        i++;//inc i to get put value
                        //To pass a \0 between <KEY> and <VAL>
                        cur_cell_num++;//index pointing to -soon2be- 1st letter of <VAL>
                        strcpy(buf_ptr+1, argv[i]);//write p_val to buffer NEXT 2 k_term_char
                        buf_ptr=&buffer[findBuffNext(buffer,bufflen)];//update buf pointer to point at p_value's terminal character
                        }
                        else{printf("invalid command: %s\n exiting\n", argv[i]);exit(1);}//catch invalid command
                order=1;//after 1st loop order is set to one so we can write the next opcode correctly (can be bypassed with another
findBuffNext call)

                i++;//inc i for next opcode or EOF
                }
                //---clip socket tail---//

                printf("sending socket...\n");
                n = send(sockfd, &buffer,get_buffclip(buffer),0);// sizeof(buffer)*buffer[0], 0);
                if(n<0)
                {error("ERROR sending socket");}

                printf("socket has been sent\n");

                bzero(buffer,bufflen);//erease buffer memory
                n = read(sockfd,buffer,bufflen);//read buffer copy we from socket
                if(n<0)
                        error("ERROR reading from socket");

                //---Translate and Print Buffer---//
                char *ptr = &buffer[0];
                ptr++;
                printf("%s\n", ptr);

                for(i=1; i<get_buffclip(buffer);i++)
                {



                        if(buffer[i]=='\0')
                        {
                                ptr=&buffer[i];
                                ptr+=2;
                                printf("%s\n",ptr);
                        }

                }


                close(sockfd);
                return 0;
                }



//=-=-=-=-=-=-=-=-=-=-=-=-=-=FUNCTIONS=-====-=-=-=-=-=-=-=-=-=-//
void error(char *msg)
{
                perror(msg);
```

```
	exit(1);
}
//function that returns the index with the next \0 in buffer. It is updated each time it is being called using global variable cur_cell_num
//Note that if the current cell is \0 the function will return the index to this
int findBuffNext(char buffer[], int size)//pointer to char return
{
	//printf("size of buffer: %ld\n", sizeof(buffer)/sizeof(buffer[0]));
	char *ptr = &buffer[cur_cell_num];
	int i;
	for(i=cur_cell_num; i<size; i++)
		{
		//printf("BufferValue: %c\n",*ptr);
			if(strcmp(ptr,"\0")==0)
				{
				cur_cell_num=i;
	//printf("Cell Num Value is now: %d\n", cur_cell_num);
				return i;
				}ptr++;
		}
}

void print_buffer(char buffer[], int size) //function that prints the buffer
{
	int i;
//printf("Printing Buffer: will stop on 1st \\0\n%s\n",buffer);
	printf("Buffer:\n");
	for(i=0; i<size; i++)
	{
		printf("%c",buffer[i]);
	}
	printf("\n");
}


//This function returns an integer as the minimal size of the buffer.
//It clips aways the extra Null characters at the end of the buffer
int get_buffclip(char buffer[])
{

	int nncnt=0; //NullNullCounter
	int i;
	int buffclip=0;
	for(i=0;i<bufflen;i++)
	{

		if(buffer[i]=='\0')
		{
			nncnt++;
		}
		else nncnt=0;

		if(nncnt>=2)
		{
		buffclip=i;
		i=bufflen;
		}

	}
	return buffclip;
}
```

Serv1.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/wait.h>

#define STORE_DATA 8
#define WORD_L 64
#define BASE_SIZE 64
#define N cmnds[0]
#define I cmnds[2]
#define BLEN 2048

//---global variables declared so that they can be used through custom functions---//
void error(char *msg);
char *find_key(char *key);
void put(char *key, char *val);
char *get(char *key);
char * create_store();
void find_ans_next();
void ans_setup(char *word);
void send_answer(int socket, char ans_buff[], int buff_size);
char answer[256];
int curr_ans_cell=0;//answer buffer index
char cmnds[4]={"n\0i\0"};
int exitstat = 0;
char * ans_key; //pointer to answer buffer
char * global_ptr;//pointer to database
int order;//this variable is used for correct answer setup to evade answer like this: /0i<val1>/0i<val2>...
int get_buffclip(char buffer[]);
int bufflen =BLEN;

int main(int argc, char *argv[])
{

        order = 0;//0 declares that next command to be written in answer is the 1st command
        //---Create a sample database and initialize pointer---//
        global_ptr=create_store();

        //---variable declarations--//
    int sockfd, portno, clilen;
    char buffer[BLEN];
    struct sockaddr_in serv_addr;
    int n;

        //---catch invalid input---//
    if (argc < 2)
        {
       fprintf(stderr,"ERROR, no port provided\n");
       exit(1);
        }
        //---open socket---//
    sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (sockfd < 0)
      error("ERROR opening socket");

        //---server_adress struct setup---//
```

```
  bzero((char *) &serv_addr, sizeof(serv_addr));
  portno = atoi(argv[1]);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(portno);
          //---bind socket to server adress---//
  if (bind(sockfd, (struct sockaddr *) &serv_addr,
         sizeof(serv_addr)) < 0) //Binding socket to port
         error("ERROR on binding");
          //---set listening socket---//
  listen(sockfd,0);//listen for connection requests

          //---recieve and serve infinite loop---//
          while(1){
          struct sockaddr_in cli_addr;
          int newsockfd;
  clilen = sizeof(cli_addr);

          newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);//accept connection request

  if (newsockfd < 0)
      error("ERROR on accept");

           bzero(buffer,BLEN);//clearing buffer to store msg

          //---recieve socket from client---//
          recv(newsockfd, &buffer, sizeof(buffer)*buffer[0],0);
          //---read socket commands---//
          n = read(newsockfd,buffer,BLEN);
          int i;
          if (n < 0) error("ERROR reading from socket");//trap error on read

           //PRINT CONTENTS OF CLIENT BUFFER

          for ( i = 0; i < sizeof(buffer); i++ )
putc( isprint(buffer[i]) ? buffer[i] : '.' , stdout );

          //---process client's command---//

          char *ptr = buffer;
                  for ( i = 0; i < sizeof(buffer); i++ )
                  {

                          if(*ptr == 'p')//process put command//
                          {
                          //---Create Pointer to put key---//
                                  char * val_ptr;
                                  val_ptr=ptr;
                                  ptr++;
                          //---Parse through key and point to value---//
                                  while(*val_ptr!='\0')
                                  {val_ptr++;}
                                  val_ptr++;
                          //---Put value to database---//
                                  put(ptr,val_ptr);
                          //---Set Pointer to next command---//
                                  ptr = val_ptr;
                                  while (*ptr!='\0')
                                  {ptr++;}
                                  ptr++;
                          }
                  else if(*ptr == 'g')//process get command//
                          {
                          //---Point to get key---//
                                  ptr++;
```

```
                                        //---Call get Function---//
                                                get(ptr);
                                                if(exitstat>0)
                                                {
                                                send_answer(newsockfd,answer,get_buffclip(answer));
                                                i=sizeof(buffer);
                                                }
                                        //---Set Pointer to next Command---//
                                                while (*ptr!='\0')
                                                {ptr++;}
                                                ptr++;

                                        }else //---if no get nor put: do nothing---//
                                        {
                                        }



                                }
                        //-----------print data----------------//
                        for(i=0; i<(STORE_DATA); i++)
                        {
                                printf("DATA #%d %s\n",i+1,global_ptr+(i*WORD_L));
                        }
                        //---Print Answer---//
                        for ( i = 0; i < sizeof(answer); i++ )
  putc( isprint(answer[i]) ? answer[i] : '.' , stdout );

                        //---Send Answer---//
    if(global_ptr != '\0')
                         n = write(newsockfd,answer,get_buffclip(answer));
     if (n < 0) error("ERROR writing to socket");
                        curr_ans_cell=0;
                        bzero(answer,BLEN);
                        order = 0;
                        exitstat = 0;
                        close(newsockfd);
                        }//end of infinite while loop
                        close(sockfd);
                        return 0;
}


//=========FUNCTIONS================//

int get_buffclip(char buffer[])
{

                int nncnt=0; //NullNullCounter
                int i;
                int buffclip=0;
                for(i=0;i<bufflen;i++)
                {

                        if(buffer[i]=='\0')
                        {
                                nncnt++;
                        }
                        else nncnt=0;

                        if(nncnt>=2)
                        {
                        buffclip=i;
                        i=bufflen;
```

```
                        }

                }
                return buffclip;
}


void error(char *msg)
{
    perror(msg);
    exit(1);
}

void put(char *key, char *val)
{
                char * ptr = find_key(key);
                ptr+=WORD_L;
                bzero(ptr, WORD_L);
                strcpy(ptr, val);


}
char *get(char *key)
{

                char * getkey;
                getkey = &N;
                if(strcmp(find_key(key),getkey)==0)
                {
                                ans_setup(getkey);
                                return getkey;
                }
                //getkey=&keyfound;//Predict that requested key will be found in datab

                getkey = &I;
                ans_setup(getkey);//write i to answer
                getkey = find_key(key)+WORD_L;
                ans_setup(getkey);

                return find_key(key)+WORD_L;

}
//---This function parses through the database to match they key and returns a pointer to the answer buffer---//
char * find_key(char *key)
{
                char *ptr;
                ptr = global_ptr;//ptr points to 1st cell of database
                int i;
                for(i=0; i<(STORE_DATA); i++)//Parse through all database values
                {
                                if(strcmp(key, ptr)==0)
                                                {
                                                                //printf("Found Key as: %s\n", ptr);
                                                                return ptr;//return pointer to requested key
                                                }
                                else//increment pointer to next data value
                                                ptr+=WORD_L;
                }
                //---this part of code executes only if the requested key was not found in the database
                ptr = &N;
                return ptr;
}
//---This function sets the global variable curr_ans_cell to the next empty cell of the answer---//
void find_ans_next()
{
```

```
		char *empty_ptr;
		empty_ptr=&answer[curr_ans_cell];
		while(*empty_ptr!='\0')
		{
			//printf("search to infinity\n");
			curr_ans_cell++;
			empty_ptr++;
		}

}
//---This function copies the word arguement to the answer buffer---//
void ans_setup(char * word)
{

		char *ptr;

		find_ans_next();//set answer index to the next empty cell
		//---Increase answer index to point to second null char---//
		//---note: not best way to do it but...code developement..---//

		if((order>0) && (strcmp(word, &I))==0)
		{
		 curr_ans_cell+=1;
		}else if((order>0) && (strcmp(word, &N))==0)
		{
			curr_ans_cell+=1;
			exitstat = 1;
		}
		else{order=1;}

		ptr = &answer[curr_ans_cell];

		strcpy(ptr,word);

}

void send_answer(int socket, char ans_buff[], int buff_size)
{
		char *ptr;
		int i;
		int exit = -1;
		ptr = &ans_buff[0];

		for(i=0; i<buff_size; i++)//parse through answer buffer
		{
			if(*ptr=='\0')//after 2nd NULL prepare to exit
			{	exit++;
				if(exit>0)//enter if 2 consecutive NULL chars
				{
					if(i<3)//return if ans=n // trap later decreasing ptr EOF
					{
						if (write(socket,ans_buff,1) < 0) error("ERROR writing to socket");
						curr_ans_cell=0;
						bzero(ans_buff,1);
					}
					//by now our ptr points to the second NULL char of ans
					if(strcmp(ptr,"\0n\0"))//Discern end of <VAL> or not found
					{i-=2;}else{i-=1;}  //choose last cell to send address
						if (write(socket,ans_buff,i) < 0) error("ERROR writing to socket");
						curr_ans_cell=0;
						bzero(ans_buff,BLEN);

				}
			//reset exit status
			exit = -1;
```

```
                }
        }

}
//ABOUT KEY-VALUE DATABASE
//To create the database we will use a custom array made from scratch with
//fixed word bytesize and fixed number of double word slots or <KEY><VALUE>
//To initialize sample keys with values we fill a temporary buffer
//This buffer does not follow the set rules so we fill the database using a
//loop with wordsize incrementing pointer
//Lastly we return a pointer to the first cell of our database
//Comments: As this is not a project about databases the database code is not
//protected by -example- keys/values with larger size than <word_length>
char * create_store()
{
        const int base_bytes = WORD_L*2*BASE_SIZE;
        //allocate and clear database//
        char base[base_bytes];
        bzero(base, base_bytes);
        //fill temporary buffer
        char buffer[256] = {"class\0lock\0race\0midget\0level\09000\0alignment\0LE\0"};
        char *bsptr = base;
        char *bfptr = buffer;

        //fill base//
        int i;
        strcpy(bsptr,bfptr);
        for(i=0; i<(STORE_DATA-1); i++)
        {
                //printf("DATA #%d %s\n",i+1,bsptr);
                bsptr+=WORD_L;
                while(*bfptr!='\0')
                {bfptr++;}
                bfptr++;
                strcpy(bsptr,bfptr);
        }

        bsptr = base;
        return bsptr;
}
```

Serv2.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/wait.h>
#include <ctype.h>
#include <semaphore.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/shm.h>


#define STORE_DATA 8
#define WORD_L 64
#define BASE_SIZE 64
#define N cmnds[0]
#define I cmnds[2]
#define BLEN 1024

#define SHMSZ 27
#define SHMID 785
int shmid;
//---global variables declared so that they can be used through custom functions---//
sem_t semf;//database sema4

void error(char *msg);
char *find_key(char *key);
void put(char *key, char *val);
char *get(char *key);
char * create_store();
void find_ans_next();
void ans_setup(char *word);
void send_answer(int socket, char ans_buff[], int buff_size);
char answer[BLEN];
char buffer[BLEN];
int curr_ans_cell=0;//answer buffer index
char cmnds[4]={"n\0i\0"};
int exitstat = 0;
char * ans_key; //pointer to answer buffer
char * global_ptr;//pointer to database
int order;//this variable is used for correct answer setup to evade answer like this: /0i<val1>/0i<val2>...
int get_buffclip(char buffer[]);
int bufflen =BLEN;

int main(int argc, char *argv[])
{
        int pid;
        //---init semf---//
        if(sem_init(&semf, 1, 1)<0)
                printf("SEMAPHORE CREATION FAILED!");
        sem_t *semf = sem_open("sema4", O_CREAT | O_EXCL, 0644,1);



        order = 0;//0 declares that next command to be written in answer is the 1st command
```

```
        //---Create a sample database and initialize pointer---//
        global_ptr=create_store();

        //---variable declarations--//
int sockfd, portno, clilen;

struct sockaddr_in serv_addr;
int n;

        //---catch invalid input---//
if (argc < 2)
        {
    fprintf(stderr,"ERROR, no port provided\n");
    exit(1);
        }
        //---open socket---//
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

if (sockfd < 0)
  error("ERROR opening socket");

        //---server_adress struct setup---//
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
        //---bind socket to server adress---//
if (bind(sockfd, (struct sockaddr *) &serv_addr,
       sizeof(serv_addr)) < 0) //Binding socket to port
       error("ERROR on binding");
        //---set listening socket---//
listen(sockfd,0);//listen for connection requests
          printf("Server is listening\n");



        struct sockaddr_in cli_addr;
        int newsockfd;
        clilen = sizeof(cli_addr);

        //---recieve and serve infinite loop---//
        while(newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,&clilen))
        {
                if (newsockfd < 0)
            error("ERROR on accept");

                //---Call Fork---//
                if((pid = fork() )<0)
                {return 0;}
                //---Child Process---//
                else if (pid == 0)
                {
                printf("server accepted socket\n");
                 bzero(buffer,BLEN);//clearing buffer to store msg

                //---recieve socket from client---//
                recv(newsockfd, &buffer, sizeof(buffer)*buffer[0],0);
                printf("server recieved socket\n");
                //---read socket commands---//
                n = read(newsockfd,buffer,BLEN);
                int i;
                if (n < 0) error("ERROR reading from socket");//trap error on read
```

```
	 //PRINT CONTENTS OF CLIENT BUFFER
	printf("sizeofbuffer: %ld\n", sizeof(buffer));
	for ( i = 0; i < sizeof(buffer); i++ )
putc( isprint(buffer[i]) ? buffer[i] : '.' , stdout );
	printf("\n");//program gets stuck without this printf
	//---process client's command---//

	char *ptr = buffer;
		//sem_wait(semf);

		for ( i = 0; i < sizeof(buffer); i++ )
		{

			if(*ptr == 'p')//process put command//
			{
			//---Create Pointer to put key---//
				char * val_ptr;
				val_ptr=ptr;
				ptr++;
			//---Parse through key and point to value---//
				while(*val_ptr!='\0')
				{val_ptr++;}
				val_ptr++;
			//---Put value to database---//
				put(ptr,val_ptr);
	//printf("key: %s\nval: %s\n", ptr, val_ptr);
			//---Set Pointer to next command---//
				ptr = val_ptr;
				while (*ptr!='\0')
				{ptr++;}
				ptr++;


			}
		else if(*ptr == 'g')//process get command//
			{
			//---Point to get key---//
				ptr++;
			//---Call get Function---//
				get(ptr);
				if(exitstat>0)
				{
				send_answer(newsockfd,answer,get_buffclip(answer));
				i=sizeof(buffer);
				}
			//---Set Pointer to next Command---//
				while (*ptr!='\0')
				{ptr++;}
				ptr++;


			}else //---if no get nor put: do nothing---//
			{

			}




		}
		//sem_post(semf);
	//-----------print data----------------//
	for(i=0; i<(STORE_DATA); i++)
	{
		printf("DATA #%d %s\n",i+1,global_ptr+(i*WORD_L));
```

```
                }
                //---Print Answer---//
                for ( i = 0; i < sizeof(answer); i++ )
          putc( isprint(answer[i]) ? answer[i] : '.' , stdout );

                //printf("Bufflcip: %d\n", get_buffclip(answer));
                //---Send Answer---//
            if(global_ptr != '\0')
                 n = write(newsockfd,answer,get_buffclip(answer));
             if (n < 0) error("ERROR writing to socket");

                //send_answer(newsockfd, answer, 256);
                curr_ans_cell=0;
                bzero(answer,BLEN);
                order = 0;
                exitstat = 0;
                close(newsockfd);
                close(sockfd);
                printf("Child finished execution\n");

                return 0;}//end of child Process
                //---------------------------------------//



        else{//---parent process---//
        //close(newsockfd);

        wait(NULL);

                }

        }//end of infinite while loop
        //printf("infinity ended, closing socket?\n");
        close(sockfd);
        return 0;
}


//=========FUNCTIONS================//

int get_buffclip(char buffer[])
{

        int nncnt=0; //NullNullCounter
        int i;
        int buffclip=0;
        for(i=0;i<bufflen;i++)
        {

                if(buffer[i]=='\0')
                {
                        nncnt++;
                }
                else nncnt=0;

                if(nncnt>=2)
                {
                buffclip=i;
                i=bufflen;
                }
//              printf("Buffclip Loop: %d : nccnt: %d\n", i, nncnt);

        }
```

```
	return buffclip;
}


void error(char *msg)
{
    perror(msg);
    exit(1);
}

void put(char *key, char *val)
{
	//printf("executing PUT\n");
	char * ptr = find_key(key);
	ptr+=WORD_L;
	bzero(ptr, WORD_L);
	strcpy(ptr, val);


}
char *get(char *key)
{

	char * getkey;
	getkey = &N;
	if(strcmp(find_key(key),getkey)==0)
	{
		//printf("KEY NOT FOUND!!!\n");
		ans_setup(getkey);
		return getkey;
	}
	//getkey=&keyfound;//Predict that requested key will be found in datab

	getkey = &I;
	ans_setup(getkey);//write i to answer

	getkey = find_key(key)+WORD_L;
	ans_setup(getkey);


	return find_key(key)+WORD_L;

}
//---This function parses through the database to match they key and returns a pointer to the answer buffer---//
char * find_key(char *key)
{
	char *ptr;
	ptr = global_ptr;//ptr points to 1st cell of database
	int i;
	for(i=0; i<(STORE_DATA); i++)//Parse through all database values
	{
		if(strcmp(key, ptr)==0)
			{
				//printf("Found Key as: %s\n", ptr);
				return ptr;//return pointer to requested key
			}
		else//increment pointer to next data value
			ptr+=WORD_L;
	}
	//---this part of code executes only if the requested key was not found in the database
	ptr = &N;
	//printf("No such key found in database\nptr %s\n",ptr);
	return ptr;
}
//---This function sets the global variable curr_ans_cell to the next empty cell of the answer---//
```

```
void find_ans_next()
{
	char *empty_ptr;
	empty_ptr=&answer[curr_ans_cell];
	while(*empty_ptr!='\0')
	{

		curr_ans_cell++;
		empty_ptr++;
	}

}
//---This function copies the word arguement to the answer buffer---//
void ans_setup(char * word)
{

	char *ptr;

	find_ans_next();//set answer index to the next empty cell
	//---Increase answer index to point to second null char---//
	//---note: not best way to do it but...code developement..---//

	if((order>0) && (strcmp(word, &I))==0)
	{
	 curr_ans_cell+=1;
	}else if((order>0) && (strcmp(word, &N))==0)
	{
		curr_ans_cell+=1;
		exitstat = 1;
	}
	else{order=1;}

	ptr = &answer[curr_ans_cell];

	strcpy(ptr,word);
}

void send_answer(int socket, char ans_buff[], int buff_size)
{
	char *ptr;
	int i;
	int exit = -1;
	ptr = &ans_buff[0];

	for(i=0; i<buff_size; i++)//parse through answer buffer
	{
		if(*ptr=='\0')//after 2nd NULL prepare to exit
		{	exit++;
			if(exit>0)//enter if 2 consecutive NULL chars
			{
				if(i<3)//return if ans=n // trap later decreasing ptr EOF
				{
					if (write(socket,ans_buff,1) < 0) error("ERROR writing to socket");
					curr_ans_cell=0;
					bzero(ans_buff,1);
				}
				//by now our ptr points to the second NULL char of ans

				if(strcmp(ptr,"\0n\0"))//Discern end of <VAL> or not found
				{i-=2;}else{i-=1;}  //choose last cell to send address
					if (write(socket,ans_buff,i) < 0) error("ERROR writing to socket");
					curr_ans_cell=0;
					bzero(ans_buff,BLEN);

			}
```

```
                        //reset exit status
                        exit = -1;
                        }
                }

}
//ABOUT KEY-VALUE DATABASE
//To create the database we will use a custom array made from scratch with
//fixed word bytesize and fixed number of double word slots or <KEY><VALUE>
//To initialize sample keys with values we fill a temporary buffer
//This buffer does not follow the set rules so we fill the database using a
//loop with wordsize incrementing pointer
//Lastly we return a pointer to the first cell of our database
//Comments: As this is not a project about databases the database code is not
//protected by -example- keys/values with larger size than <word_length>
//EDIT: using

char * create_store()
{
        //declare database variables//
        const int base_bytes = WORD_L*2*BASE_SIZE;

        //--declare shared memory variables--//
        key_t key = SHMID;
        char *shm;
        //--create shared mem segment---//
        if((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
        error("shmget");
        //---bind sh segment to program---//
        if((shm = shmat(shmid,NULL,0))==(char *) -1)
        error("shmat");

        //allocate and clear database//
        //fill temporary buffer
        char buffer[256] = {"class\0lock\0race\0midget\0level\09000\0alignment\0LE\0"};
        char *bsptr;
        bsptr = shm;
        char *bfptr = buffer;
        bzero(shm, base_bytes);
        //fill base//
        int i;
        strcpy(bsptr,bfptr);
        for(i=0; i<(STORE_DATA-1); i++)
        {
                printf("DATA #%d %s\n",i+1,bsptr);
                bsptr+=WORD_L;
                while(*bfptr!='\0')
                {bfptr++;}
                bfptr++;
                strcpy(bsptr,bfptr);
        }
        printf("DATA #%d %s\n",i+1,bsptr);
        bsptr = shm;
        return bsptr;

}
```

Serv3.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/wait.h>
#include <ctype.h>
#include <semaphore.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/shm.h>


#define STORE_DATA 8
#define WORD_L 64
#define BASE_SIZE 64
#define N cmnds[0]
#define I cmnds[2]
#define BLEN 1024

#define SHMSZ 27
#define SHMID 785
int shmid;
//---global variables declared so that they can be used through custom functions---//
//sem_t semf;

void error(char *msg);
char *find_key(char *key);
void put(char *key, char *val);
char *get(char *key);
char * create_store();
void find_ans_next();
void ans_setup(char *word);
void send_answer(int socket, char ans_buff[], int buff_size);
char answer[BLEN];
int curr_ans_cell=0;//answer buffer index
char cmnds[4]={"n\0i\0"};
int exitstat = 0;
char * ans_key; //pointer to answer buffer
char * global_ptr;//pointer to database
int order;//this variable is used for correct answer setup to evade answer like this: /0i<val1>/0i<val2>...
int get_buffclip(char buffer[]);
int bufflen =BLEN;

int main(int argc, char *argv[])
{
        if(argc < 3)error("not enough arguements");
```

```
        int pid, count =0;

        order = 0;//0 declares that next command to be written in answer is the 1st command
        //---Create a sample database and initialize pointer---//
        global_ptr=create_store();

        //---variable declarations--//
    int sockfd, portno, clilen;
    char buffer[BLEN];
    struct sockaddr_in serv_addr;
    int n;

        //---catch invalid input---//
    if (argc < 2)
            {
       fprintf(stderr,"ERROR, no port provided\n");
       exit(1);
            }
        //---open socket---//
    sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (sockfd < 0)
      error("ERROR opening socket");

        //---server_adress struct setup---//
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
        //---bind socket to server adress---//
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0) //Binding socket to port
            error("ERROR on binding");
        //---set listening socket---//
    listen(sockfd,0);//listen for connection requests

        //---recieve and serve infinite loop---//

    printf("Server is listening\n");




        int forkcnt;
        int maxforx=   atoi(argv[2]);
        for(forkcnt=0;forkcnt<maxforx; forkcnt++)
        {
                if((pid = fork() )<0)
                {return 0;}
        else if (pid == 0)
        //---Child Process---//
{
        printf("Child Process\n");
```

```
        while(1){
        struct sockaddr_in cli_addr;
        int newsockfd;
    clilen = sizeof(cli_addr);
        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,                    &clilen);//accept connection
request
        if (newsockfd < 0)
       error("ERROR on accept");


        //---shared memory attachment---//
        //char *shm;
//          if ((shm = shmat(shmid,NULL,0)) == (char *)-1){error("shmat");}


        printf("server accepted socket\n");
         bzero(buffer,BLEN);//clearing buffer to store msg

        //---recieve socket from client---//
        recv(newsockfd, &buffer, sizeof(buffer)*buffer[0],0);
        printf("server recieved socket\n");
        //---read socket commands---//
        n = read(newsockfd,buffer,BLEN);
        int i;
        if (n < 0) error("ERROR reading from socket");//trap error on read

         //PRINT CONTENTS OF CLIENT BUFFER
        printf("sizeofbuffer: %ld\n", sizeof(buffer));
        for ( i = 0; i < sizeof(buffer); i++ )
  putc( isprint(buffer[i]) ? buffer[i] : '.' , stdout );
        printf("\n");//program gets stuck without this printf
        //---process client's command---//

        char *ptr = buffer;

                //sem_wait(semf);
        printf("essential printf\n");//program gets stuck without this printf
                for ( i = 0; i < sizeof(buffer); i++ )
                {
                        //printf("command processing loop %d\n", i);
                        if(*ptr == 'p')//process put command//
                        {
                        //---Create Pointer to put key---//
                                char * val_ptr;
                                val_ptr=ptr;
                                ptr++;
                        //---Parse through key and point to value---//
                                while(*val_ptr!='\0')
                                {val_ptr++;}
                                val_ptr++;
                        //---Put value to database---//
                                put(ptr,val_ptr);
        //printf("key: %s\nval: %s\n", ptr, val_ptr);
                        //---Set Pointer to next command---//
                                ptr = val_ptr;
                                while (*ptr!='\0')
```

```
                        {ptr++;}
                        ptr++;


                }
        else if(*ptr == 'g')//process get command//
                {
                //---Point to get key---//
                        ptr++;
                //---Call get Function---//
                        get(ptr);
                        if(exitstat>0)
                        {
                        send_answer(newsockfd,answer,get_buffclip(answer));
                        i=sizeof(buffer);
                        }
                //---Set Pointer to next Command---//
                        while (*ptr!='\0')
                        {ptr++;}
                        ptr++;


                }else //---if no get nor put: do nothing---//
                {

                }




                }
                //sem_post(semf);
        //-----------print data----------------//
        for(i=0; i<(STORE_DATA); i++)
        {
                printf("DATA #%d %s\n",i+1,global_ptr+(i*WORD_L));
        }
        //---Print Answer---//
        for ( i = 0; i < sizeof(answer); i++ )
putc( isprint(answer[i]) ? answer[i] : '.' , stdout );

        printf("Bufflcip: %d\n", get_buffclip(answer));
        //---Send Answer---//
  if(global_ptr != '\0')
         n = write(newsockfd,answer,get_buffclip(answer));
   if (n < 0) error("ERROR writing to socket");

        curr_ans_cell=0;
        bzero(answer,BLEN);
        order = 0;
        exitstat = 0;
        close(newsockfd);
        printf("Child finished execution\n");
        }//end of infinte loop
        return 0;}//end of child Process
        else{//---parent process---//
```

```
                }

        printf("Forking %d\n", forkcnt);

        }//end of forking loop
        //printf("infinity ended, closing socket?\n");
        close(sockfd);
        return 0;
}


//=========FUNCTIONS================//

int get_buffclip(char buffer[])
{

        int nncnt=0; //NullNullCounter
        int i;
        int buffclip=0;
        for(i=0;i<bufflen;i++)
        {

                if(buffer[i]=='\0')
                {
                        nncnt++;
                }
                else nncnt=0;

                if(nncnt>=2)
                {
                buffclip=i;
                i=bufflen;
                }
//      printf("Buffclip Loop: %d : nccnt: %d\n", i, nncnt);

        }
        return buffclip;
}


void error(char *msg)
{
    perror(msg);
    exit(1);
}

void put(char *key, char *val)
{
        //printf("executing PUT\n");
        char * ptr = find_key(key);
        ptr+=WORD_L;
        bzero(ptr, WORD_L);
        strcpy(ptr, val);
```

```
}
char *get(char *key)
{

	char * getkey;
	getkey = &N;
	if(strcmp(find_key(key),getkey)==0)
	{
		//printf("KEY NOT FOUND!!!\n");
		ans_setup(getkey);
		return getkey;
	}
	//getkey=&keyfound;//Predict that requested key will be found in datab

	getkey = &I;
	ans_setup(getkey);//write i to answer

	getkey = find_key(key)+WORD_L;
	ans_setup(getkey);


	return find_key(key)+WORD_L;

}
//---This function parses through the database to match they key and returns a pointer to the answer buffer---//
char * find_key(char *key)
{
	char *ptr;
	ptr = global_ptr;//ptr points to 1st cell of database
	int i;
	for(i=0; i<(STORE_DATA); i++)//Parse through all database values
	{
		if(strcmp(key, ptr)==0)
			{
				//printf("Found Key as: %s\n", ptr);
				return ptr;//return pointer to requested key
			}
		else//increment pointer to next data value
			ptr+=WORD_L;
	}
	//---this part of code executes only if the requested key was not found in the database
	ptr = &N;
	//printf("No such key found in database\nptr %s\n",ptr);
	return ptr;
}
//---This function sets the global variable curr_ans_cell to the next empty cell of the answer---//
void find_ans_next()
{
	char *empty_ptr;
	empty_ptr=&answer[curr_ans_cell];
	while(*empty_ptr!='\0')
	{
		//printf("search to infinity\n");
		curr_ans_cell++;
		empty_ptr++;
```

```
	}

}
//---This function copies the word arguement to the answer buffer---//
void ans_setup(char * word)
{

	char *ptr;

	find_ans_next();//set answer index to the next empty cell
	//---Increase answer index to point to second null char---//
	//---note: not best way to do it but...code developement..---//


	if((order>0) && (strcmp(word, &I))==0)
	{
	 curr_ans_cell+=1;
	}else if((order>0) && (strcmp(word, &N))==0)
	{
		curr_ans_cell+=1;
		exitstat = 1;
	}
	else{order=1;}

	ptr = &answer[curr_ans_cell];

	strcpy(ptr,word);

}

void send_answer(int socket, char ans_buff[], int buff_size)
{
	char *ptr;
	int i;
	int exit = -1;
	ptr = &ans_buff[0];

	for(i=0; i<buff_size; i++)//parse through answer buffer
	{
		if(*ptr=='\0')//after 2nd NULL prepare to exit
		{	exit++;
			if(exit>0)//enter if 2 consecutive NULL chars
			{
				if(i<3)//return if ans=n // trap later decreasing ptr EOF
				{
					if (write(socket,ans_buff,1) < 0) error("ERROR writing to socket");
					curr_ans_cell=0;
					bzero(ans_buff,1);
				}
				//by now our ptr points to the second NULL char of ans

				if(strcmp(ptr,"\0n\0"))//Discern end of <VAL> or not found
				{i-=2;}else{i-=1;}	//choose last cell to send address
					if (write(socket,ans_buff,i) < 0) error("ERROR writing to socket");
					curr_ans_cell=0;
					bzero(ans_buff,BLEN);
```

```
                        }
                //reset exit status
                exit = -1;
                }
        }

}
//ABOUT KEY-VALUE DATABASE
//To create the database we will use a custom array made from scratch with
//fixed word bytesize and fixed number of double word slots or <KEY><VALUE>
//To initialize sample keys with values we fill a temporary buffer
//This buffer does not follow the set rules so we fill the database using a
//loop with wordsize incrementing pointer
//Lastly we return a pointer to the first cell of our database
//Comments: As this is not a project about databases the database code is not
//protected by -example- keys/values with larger size than <word_length>
//EDIT: using

char * create_store()
{
        //declare database variables//
        const int base_bytes = WORD_L*2*BASE_SIZE;
        key_t key = SHMID;
        char *shm;

        if((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
        error("shmget");
        if((shm = shmat(shmid,NULL,0))==(char *) -1)
        error("shmat");

        //allocate and clear database//
        //fill temporary buffer
        char buffer[256] = {"class\0lock\0race\0midget\0level\09000\0alignment\0LE\0"};
        char *bsptr;
        bsptr = shm;
        char *bfptr = buffer;
        bzero(shm, base_bytes);
        //fill base//
        int i;
        strcpy(bsptr,bfptr);
        for(i=0; i<(STORE_DATA-1); i++)
        {
                printf("DATA #%d %s\n",i+1,bsptr);
                bsptr+=WORD_L;
                while(*bfptr!='\0')
                {bfptr++;}
                bfptr++;
                strcpy(bsptr,bfptr);
        }
        printf("DATA #%d %s\n",i+1,bsptr);
        bsptr = shm;
        return bsptr;
        return bfptr;
}
```

Serv4.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/wait.h>
#include<pthread.h>

#define STORE_DATA 8
#define WORD_L 64
#define BASE_SIZE 64
#define N cmnds[0]
#define I cmnds[2]
#define BLEN 2048

//---global variables declared so that they can be used through custom functions---//
void error(char *msg);
char *find_key(char *key);
void put(char *key, char *val);
char *get(char *key);
char * create_store();
void find_ans_next();
void ans_setup(char *word);
void send_answer(int socket, char ans_buff[], int buff_size);
char answer[BLEN];
int curr_ans_cell=0;//answer buffer index
char cmnds[4]={"n\0i\0"};
int exitstat = 0;
char * ans_key; //pointer to answer buffer
char * global_ptr;//pointer to database
int order;//this variable is used for correct answer setup to evade answer like this: /0i<val1>/0i<val2>...
int get_buffclip(char buffer[]);
int bufflen =BLEN;
char buffer[BLEN];
void *tserv(void  *sockdesc);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_t tid;

int main(int argc, char *argv[])
{

        order = 0;//0 declares that next command to be written in answer is the 1st command
        //---Create a sample database and initialize pointer---//
        global_ptr=create_store();

        //---variable declarations--//
    int sockfd, portno;
    char buffer[BLEN];
```

```
    struct sockaddr_in serv_addr;
    int n;

            //---catch invalid input---//
    if (argc < 2)
            {
        fprintf(stderr,"ERROR, no port provided\n");
        exit(1);
            }
            //---open socket---//
    sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (sockfd < 0)
      error("ERROR opening socket");

            //---server_adress struct setup---//
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
            //---bind socket to server adress---//
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0) //Binding socket to port
            error("ERROR on binding");
            //---set listening socket---//
    listen(sockfd,0);//listen for connection requests

                        struct sockaddr_in cli_addr;
                        int newsockfd;
                        int clilen = sizeof(cli_addr);


            //---recieve and serve infinite loop---//
            while(newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,&clilen))
            {

                if (newsockfd < 0)
                        error("ERROR on accept");

                        printf("server accepted socket\n");


            if(pthread_create(&tid, NULL, &tserv, (void *) &newsockfd)<0)
            error("Thread");

            }//end of infinite while loop
            close(sockfd);
            return 0;
}


//=========FUNCTIONS================//

int get_buffclip(char buffer[])
{
```

```
	int nncnt=0; //NullNullCounter
	int i;
	int buffclip=0;
	for(i=0;i<bufflen;i++)
	{

		if(buffer[i]=='\0')
		{
			nncnt++;
		}
		else nncnt=0;

		if(nncnt>=2)
		{
		buffclip=i;
		i=bufflen;
		}

	}
	printf("made it this far:\n");
	return buffclip;
}


void error(char *msg)
{
   perror(msg);
   exit(1);
}

void put(char *key, char *val)
{
	//printf("executing PUT\n");
	char * ptr = find_key(key);
	ptr+=WORD_L;
	bzero(ptr, WORD_L);
	strcpy(ptr, val);


}
char *get(char *key)
{

	char * getkey;
	getkey = &N;
	if(strcmp(find_key(key),getkey)==0)
	{
		//printf("KEY NOT FOUND!!!\n");
		ans_setup(getkey);
		return getkey;
	}

	getkey = &I;
	ans_setup(getkey);//write i to answer
```

```
	getkey = find_key(key)+WORD_L;
	ans_setup(getkey);


	return find_key(key)+WORD_L;

}
//---This function parses through the database to match they key and returns a pointer to the answer buffer---//
char * find_key(char *key)
{
	char *ptr;
	ptr = global_ptr;//ptr points to 1st cell of database
	int i;
	for(i=0; i<(STORE_DATA); i++)//Parse through all database values
	{
		if(strcmp(key, ptr)==0)
			{
				//printf("Found Key as: %s\n", ptr);
				return ptr;//return pointer to requested key
			}
		else//increment pointer to next data value
			ptr+=WORD_L;
	}
	//---this part of code executes only if the requested key was not found in the database
	ptr = &N;
	//printf("No such key found in database\nptr %s\n",ptr);
	return ptr;
}
//---This function sets the global variable curr_ans_cell to the next empty cell of the answer---//
void find_ans_next()
{
	char *empty_ptr;
	empty_ptr=&answer[curr_ans_cell];
	while(*empty_ptr!='\0')
	{
		//printf("search to infinity\n");
		curr_ans_cell++;
		empty_ptr++;
	}

}
//---This function copies the word arguement to the answer buffer---//
void ans_setup(char * word)
{

	char *ptr;

	find_ans_next();//set answer index to the next empty cell
	//---Increase answer index to point to second null char---//
	//---note: not best way to do it but...code developement..---//

	if((order>0) && (strcmp(word, &I))==0)
	{
	 curr_ans_cell+=1;
	}else if((order>0) && (strcmp(word, &N))==0)
	{
```

```
                curr_ans_cell+=1;
                exitstat = 1;
        }
        else{order=1;}

        ptr = &answer[curr_ans_cell];

        strcpy(ptr,word);

}

void send_answer(int socket, char ans_buff[], int buff_size)
{
        char *ptr;
        int i;
        int exit = -1;
        ptr = &ans_buff[0];

        for(i=0; i<buff_size; i++)//parse through answer buffer
        {
                if(*ptr=='\0')//after 2nd NULL prepare to exit
                {       exit++;
                        if(exit>0)//enter if 2 consecutive NULL chars
                        {
                                if(i<3)//return if ans=n // trap later decreasing ptr EOF
                                {
                                        if (write(socket,ans_buff,1) < 0) error("ERROR writing to socket");
                                        curr_ans_cell=0;
                                        bzero(ans_buff,1);
                                }
                                //by now our ptr points to the second NULL char of ans
                                if(strcmp(ptr,"\0n\0"))//Discern end of <VAL> or not found
                                {i-=2;}else{i-=1;}          //choose last cell to send address
                                        if (write(socket,ans_buff,i) < 0) error("ERROR writing to socket");
                                        curr_ans_cell=0;
                                        bzero(ans_buff,BLEN);

                        }
                //reset exit status
                exit = -1;
                }
        }

}
//ABOUT KEY-VALUE DATABASE
//To create the database we will use a custom array made from scratch with
//fixed word bytesize and fixed number of double word slots or <KEY><VALUE>
//To initialize sample keys with values we fill a temporary buffer
//This buffer does not follow the set rules so we fill the database using a
//loop with wordsize incrementing pointer
//Lastly we return a pointer to the first cell of our database
//Comments: As this is not a project about databases the database code is not
//protected by -example- keys/values with larger size than <word_length>
char * create_store()
{
        //declare database variables//
```

```
	const int base_bytes = WORD_L*2*BASE_SIZE;
	//allocate and clear database//
	char base[base_bytes];
	bzero(base, base_bytes);
	//fill temporary buffer
	char buffer[256] = {"class\0lock\0race\0midget\0level\09000\0alignment\0LE\0"};
	char *bsptr = base;
	char *bfptr = buffer;

	//fill base//
	int i;
	strcpy(bsptr,bfptr);
	for(i=0; i<(STORE_DATA-1); i++)
	{
		//printf("DATA #%d %s\n",i+1,bsptr);
		bsptr+=WORD_L;
		while(*bfptr!='\0')
		{bfptr++;}
		bfptr++;
		strcpy(bsptr,bfptr);
	}
	//printf("DATA #%d %s\n",i+1,bsptr);
	bsptr = base;
	return bsptr;
}


//----------------------------Thread Serv----------------------------//
void *tserv(void *sockdesc)
{
	printf("Executing thread %d\n",(int) tid);
	pthread_t id = pthread_self();
	int newsockfd = *(int*)sockdesc;
	int n;
	if(pthread_equal(id,tid))
	{

	  pthread_mutex_lock(&mutex1);

		bzero(buffer,BLEN);//clearing buffer to store msg

		//---recieve socket from client---//
		recv(newsockfd, &buffer, sizeof(buffer)*buffer[0],0);
		printf("server recieved socket\n");
		//---read socket commands---//
		n = read(newsockfd,buffer,BLEN);
		int i;
		if (n < 0) error("ERROR reading from socket");//trap error on read

		 //PRINT CONTENTS OF CLIENT BUFFER
		printf("sizeofbuffer: %ld\n", sizeof(buffer));
		for ( i = 0; i < sizeof(buffer); i++ )
	  putc( isprint(buffer[i]) ? buffer[i] : '.' , stdout );
		printf("\n");//program gets stuck without this printf
		//---process client's command---//
```

```
char *ptr = buffer;//point to client buffer

        //===Command Processing Loop===//
        for ( i = 0; i < sizeof(buffer); i++ )
        {

                if(*ptr == 'p')//process put command//
                {
                //---Create Pointer to put key---//
                        char * val_ptr;
                        val_ptr=ptr;
                        ptr++;
                //---Parse through key and point to value---//
                        while(*val_ptr!='\0')
                        {val_ptr++;}
                        val_ptr++;
                //---Put value to database---//
                        put(ptr,val_ptr);
//printf("key: %s\nval: %s\n", ptr, val_ptr);
                //---Set Pointer to next command---//
                        ptr = val_ptr;
                        while (*ptr!='\0')
                        {ptr++;}
                        ptr++;


                }
        else if(*ptr == 'g')//process get command//
                {
                //---Point to get key---//
                        ptr++;
                //---Call get Function---//
                        get(ptr);
                        if(exitstat>0)
                        {
                        send_answer(newsockfd,answer,get_buffclip(answer));
                        i=sizeof(buffer);
                        }
                //---Set Pointer to next Command---//
                        while (*ptr!='\0')
                        {ptr++;}
                        ptr++;


                }else //---if no get nor put: do nothing---//
                {

                }



        }
//-----------print data----------------//
for(i=0; i<(STORE_DATA); i++)
{
```

```
                        printf("DATA #%d %s\n",i+1,global_ptr+(i*WORD_L));
                }
                //---Print Answer---//
                for ( i = 0; i < sizeof(answer); i++ )
          putc( isprint(answer[i]) ? answer[i] : '.' , stdout );

                printf("Bufflcip: %d\n", get_buffclip(answer));
                //---Send Answer---//
            if(global_ptr != '\0')
                 n = write(newsockfd,answer,get_buffclip(answer));
             if (n < 0) error("ERROR writing to socket");

                curr_ans_cell=0;
                bzero(answer,BLEN);
                order = 0;
                exitstat = 0;
                close(newsockfd);
                pthread_mutex_unlock(&mutex1);
                printf("Child finished execution\n");
        }//end of thread
}
```