Soluzione esercitazione 4

Gruppo 23

* Marco Bertoni
* Michele Iftimie
* Paolo Di Simone
* Giulia Vivarelli

Client TCP:

Il client è un filtro:

- Riceve da stdin il nome di un direttorio
- Restituisce la lista di file nei direttori di secondo livello del direttorio specificato

Client UDP:

Il client è un filtro:

- Riceve da stdin il nome di un file e una parola da eliminare
- Restituisce il numero di occorrenze trovate nel file

SERVER:

Il server è un demone espone due servizi multiplexati attraverso select:

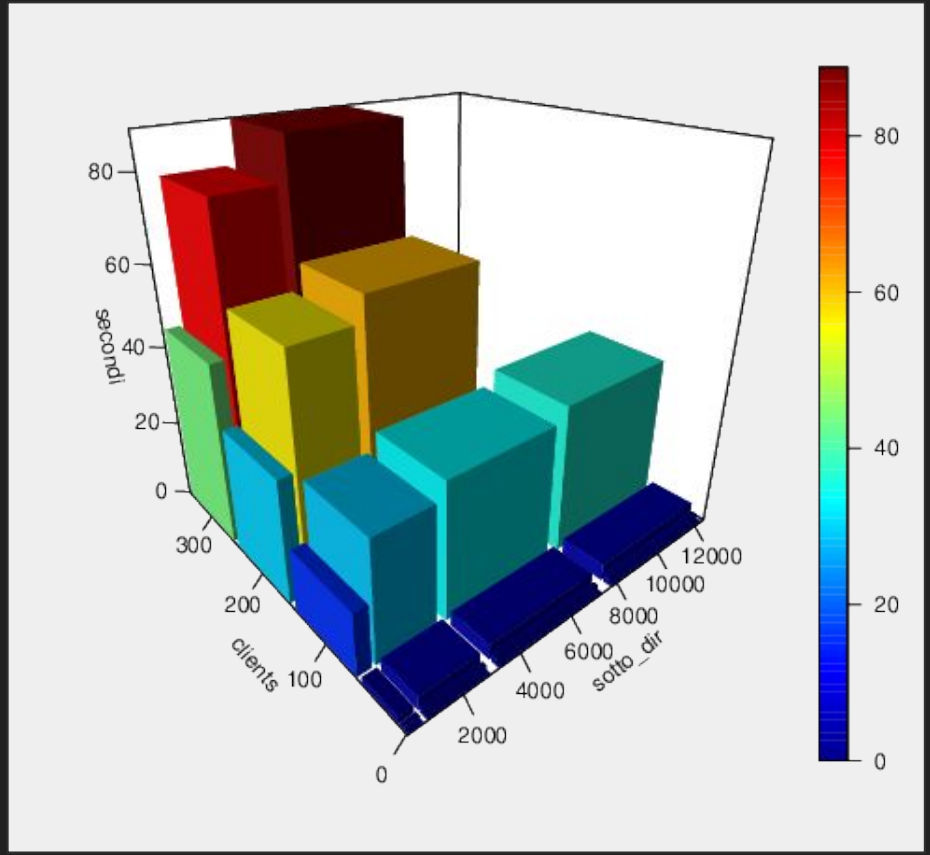
- Servizio UDP sequenziale
- Servizio TCP concorrente

Analisi di performance:

- Abbiamo registrato il tempo tra invio della richiesta e terminazione della risposta
- Abbiamo sfruttato pesantemente il preprocessore per definire versioni diverse del codice
- Per aumentare la leggibilità del codice abbiamo scritto diverse funzioni, ma sfruttando la direttiva “always inline” di gcc questo non ha alcun overhead

Analisi di performance:

Client TCP



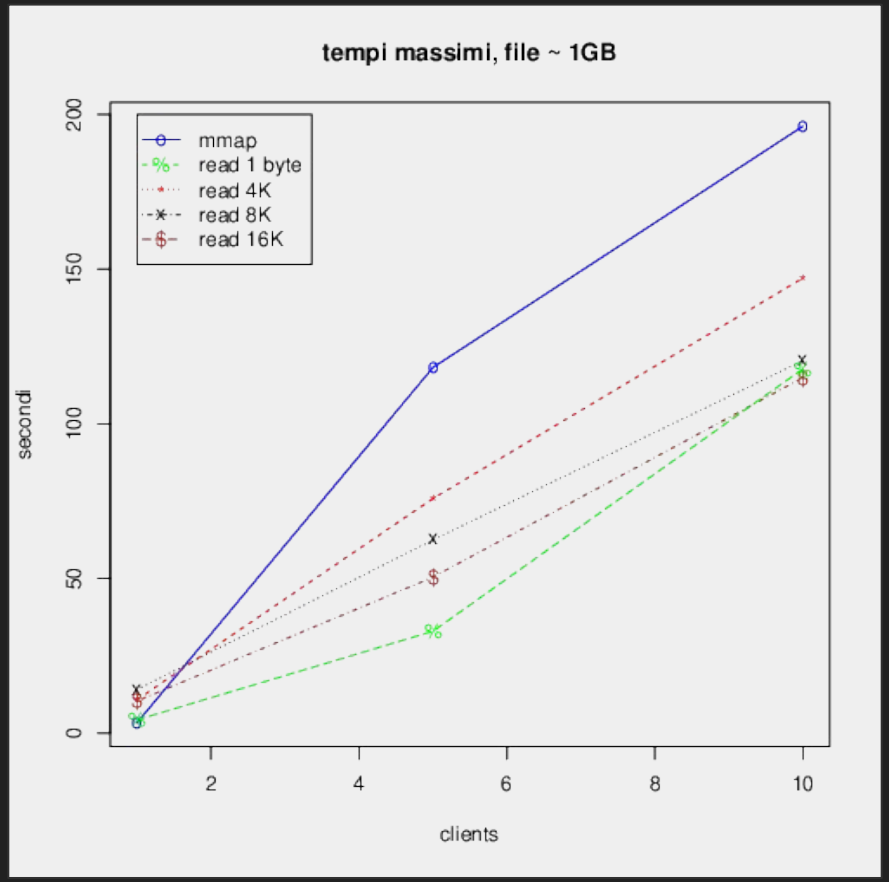
Analisi di performance:

Client UDP

File 100’000’000 parole

Tempi massimi

(~ ultimo client)



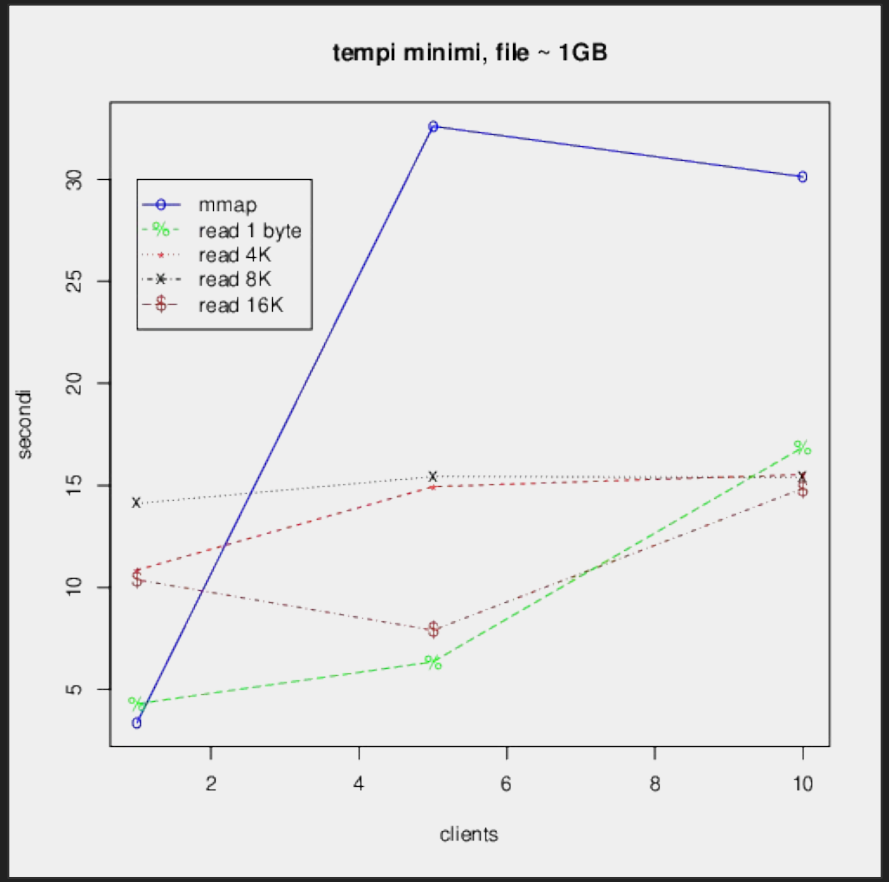
Analisi di performance:

Client UDP

File 100’000’000 parole

Tempi minimi

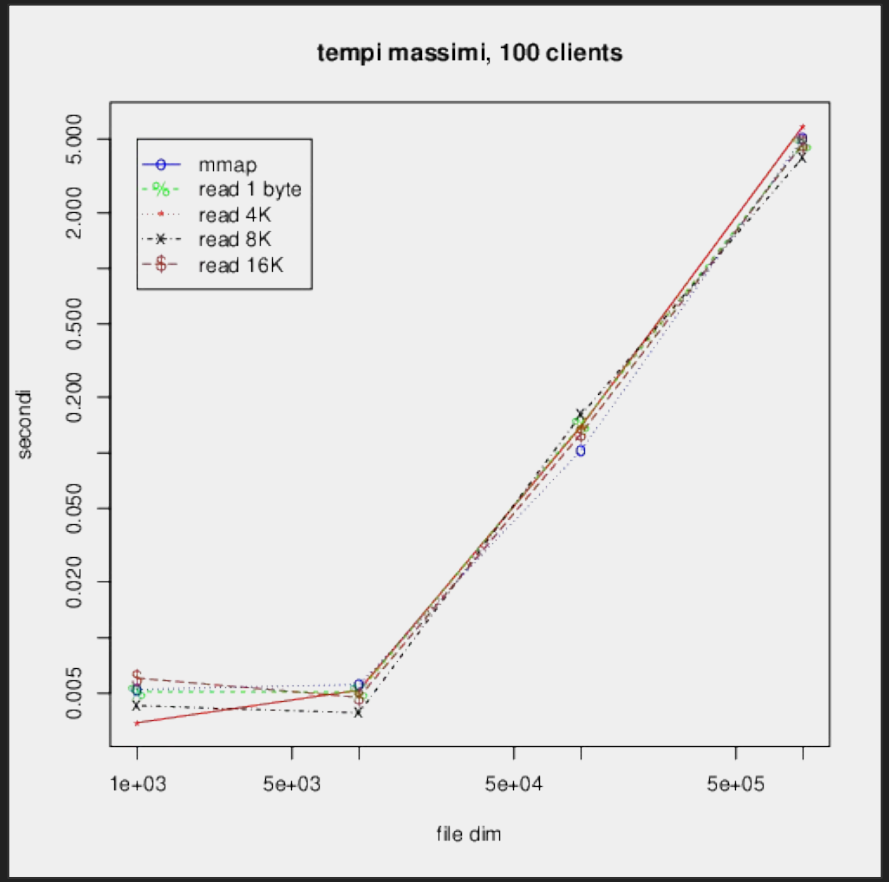
(~ primo client)



Analisi di performance:

Client UDP

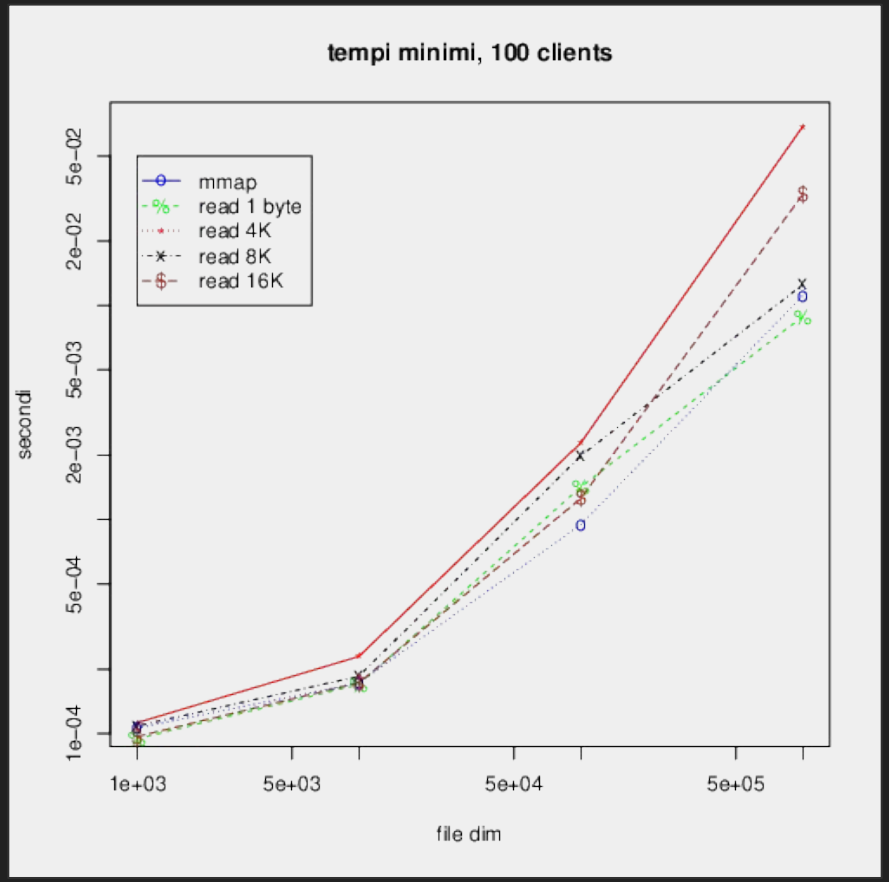
100 client
Tempi massimi
(~ ultimo client)
Scala logaritmica



Analisi di performance:

Client UDP

100 client
Tempi minimi
(~ primo client)
Scala logaritmica



SERVER - Ricezione richiesta udp

```
if (FD_ISSET(socket_udp, &rset)){
    LOGD("udp is set\n");
    recvfrom(socket_udp, buf, BUF_SIZE * 2, 0, (struct sockaddr *)
    &client_addr, &client_addr_len);
    file = buf;
    word = buf + strlen(buf) + 1;

    #if defined REP_STR_MMAP
    ris = replace_string_mmap(file, word);
    #elif defined REP_STR_READ
    ris = replace_string_read(file, word);
    #else
    ris = deleteOccurences(file, word);
    #endif
```

SERVER - Controllo stringa:

```
inline int replace_string_read(char* file, char* word)
{
    orig_fd = open(file, O_RDONLY);
    temp_fd = open(temp_file, O_WRONLY|O_CREAT|O_TRUNC, 0600);
    if (orig_fd < 0){
        ...
        return -1;
    }
    if (temp_fd < 0){
        ...
        return -1;
    }
    word_len = strlen(word);
    word_len >= READ_BUF_SIZE && die("buffer read troppo piccolo",
-100);
```

SERVER - Controllo stringa:

```
while ((buf_len = read(orig_fd, buf, READ_BUF_SIZE)) > 0) {
    multiple_strstr(buf, buf_len, word, word_len, temp_fd, &count);

    edge_buf = buf + buf_len + 1 - word_len;
    for (i = 0; i < word_len - 1 && same == false; i++){
        if (edge_buf[i] == word[0]){
            same = true;
            start_from_edge = i;
            for (j = 0; i < word_len - 1 && same == true; j++){
                if (edge_buf[i + j] != word[j]){
                    same = false;
}}}}
```

SERVER - Controllo stringa:

```
    if (same == true){
        remaining_len = start_from_edge + 1;
        alredy_checked_len = word_len - remaining_len;
        buf_len = read(orig_fd, buf, remaining_len);
        if (buf_len == remaining_len && !memcmp(buf, word +
alredy_checked_len, remaining_len)){
            lseek(temp_fd, -alredy_checked_len, SEEK_CUR);
        } else {
            write(temp_fd, buf, buf_len);
        }}}
```

SERVER - Controllo stringa:

```
void multiple_strstr(char * haystack, int haylen, char * needle, int
needlen, int outfd, int * count)
{
    char * start_strstr;
    start_strstr = strstr(haystack, needle);
    if (start_strstr){
        (*count)++;
        write(outfd, haystack, start_strstr - haystack);
        multiple_strstr(start_strstr + needlen, haylen -
((start_strstr - haystack) + needlen), needle, needlen, outfd,
count);
    } else {
        write(outfd, haystack, haylen);
    }}
```

Tail recursion optimization (GCC)

```
00000000000013d0 <multiple_strstr>:
;; inizializzazione funzione, entry point (jmp)
    1400: 49 89 c5                mov    r13,rax
    1403: 4c 89 fe                mov    rsi,r15
    1406: 44 89 e7                mov    edi,r12d
    1409: 4d 29 fd                sub    r13,r15
    140c: 4c 89 ea                mov    rdx,r13
    140f: e8 1c fc ff ff          call   1030 <write@plt>
    1414: 48 8b 44 24 08          mov    rax,QWORD PTR [rsp+0x8]
    1419: 2b 6c 24 04             sub    ebp,DWORD PTR [rsp+0x4]
    141d: 44 29 ed                sub    ebp,r13d
    1420: 4c 8d 3c 03             lea    r15,[rbx+rax*1]
    1424: 4c 89 f6                mov    rsi,r14
    1427: 4c 89 ff                mov    rdi,r15
    142a: e8 a1 fc ff ff          call   10d0 <strstr@plt>
    142f: 48 89 c3                mov    rbx,rax
    1432: 48 85 c0                test   rax,rax
;; recursione senza utilizzare call
    1435: 75 c9                   jne    1400 <multiple_strstr+0x30>
```

SERVER - Inizializzazione connessione tcp:

```
if (FD_ISSET(socket_tcp, &rset)){
      int socket_conn;
      LOGD("tcp is set\n");
      if ((socket_conn = accept(socket_tcp, (struct sockaddr *) &client_addr, &client_addr_len)) < 0){
            …
      if (fork() == 0){
            uint32_t msg_len, msg_len_net;
            size_t tmp_sizet;
            char msg[BUF_SIZE];
            DIR * dir1, * dir2;
            char * entry1_name, * entry2_name;
            struct dirent * entry1, * entry2;
            struct hostent * host;
            const int zero = 0;

            close(socket_tcp);
            host = gethostbyaddr((char *) &client_addr.sin_addr, sizeof(client_addr.sin_addr), AF_INET);
```

Perchè dichiarare le variabili dentro il fork:

La syscall fork ha due modi per separare lo spazio di indirizzi del padre da quello del figlio:

- Copiare i dati da padre a figlio
- Settare il sistema copy on write

Entrambi hanno un costo ben maggiore rispetto a sottrarre il valore desiderato da rsp

Perchè dichiarare le variabili dentro il fork:

La syscall fork ha due modi per separare lo spazio di indirizzi del padre da quello del figlio:

- Copiare i dati da padre a figlio
- Settare il sistema copy on write

Entrambi hanno un costo ben maggiore rispetto a sottrarre il valore desiderato da rsp

Questo in teoria, visto che controllando il decompilato lo spazio sullo stack viene allocato alla creazione del main.

Possibili cause:

- La mia tesi è sbagliata
- Gcc non riconosce di essere all'interno del child di una chiamata fork

SERVER - Lettura richiesta cliente tcp:

```
LOGD("child starting\n");
read(socket_conn, &msg_len_net, sizeof(uint32_t));
msg_len = ntohl(msg_len_net);
LOGD("msg_len: %d\n", msg_len);
read(socket_conn, msg, msg_len);
LOGD("msg: %s\n", msg);
```

SERVER - Lettura direttori:

```
dir1 = opendir(msg);

tmp_sizet = msg_len;
if (dir1) {
    while ((entry1 = readdir(dir1))) {
        entry1_name = entry1->d_name;
        /* inline strcmp*/
        if (entry1_name[0] == '.' && (entry1_name[1] == '\0'
    || (entry1_name[1] == '.' && entry1_name[2] == '\0')))
            continue;
```

SERVER - Lettura direttori:

```
if (entry1->d_type == DT_DIR){
    /* riutilizzo il buffer */
    msg_len = strlen(entry1_name) + strlen(msg) + 1;
    if (msg_len > BUF_SIZE){
        LOGD("Implement heap allocation or increase
BUF_SIZE\n");
        continue;
    }
    msg_len = strlen(entry1_name) + 1;
    /* keeping \0 */
    memcpy(msg + tmp_sizet, entry1_name, msg_len);
    msg[tmp_sizet - 1] = '/';
    … open(dir2) e invio dei file al client
```

SERVER - Lettura direttori:

```
write(socket_conn, &msg_len_net, sizeof(uint32_t));
write(socket_conn, entry2_name, msg_len);
```

Grazie per l’attenzione

```
#! /bin/bash

TEST_PORT=65111
FILES_TESTS="500 1000 5000 10000"
CLIENT_TESTS="1 5 10"
LOC_TESTS="100000000"
REMBUF="stdbuf -i0 -o0 -e0"
PAROLA="abaca"
TCP_DIR_TEST="files/tcp"

TEMPFILE=$(mktemp)
for i in `seq 15000`; do
    cat /usr/share/dict/words >> $TEMPFILE
done
for loc in $LOC_TESTS; do
    head -n $loc $TEMPFILE > files/$loc-orig.txt
done
rm $TEMPFILE

mkdir -p $TCP_DIR_TEST
for dir_i in $FILES_TESTS; do
    mkdir $TCP_DIR_TEST/$dir_i
    for i in `seq $dir_i`; do
        mkdir $TCP_DIR_TEST/$dir_i/$i
        for j in `seq 100`; do
            touch $TCP_DIR_TEST/$dir_i/$i/$j
        done
    done
done 2>/dev/null

# udp
for CCARGS in '-DREP_STR_MMAP' '-DREP_STR_READ' '-DREP_STR_READ -DMANUAL_READ_BUF_SIZE -DREAD_BUF_SIZE=8192' '-DREP_STR_READ -DMANUAL_READ_BUF_SIZE -DREAD_BUF_SIZE=16384' ''; do
    CCARGS="$CCARGS -DTEST" make purge > /dev/null 2>&1
    CCARGS="$CCARGS -DTEST" make > /dev/null 2> logs/compilation_"$CCARGS".log
    if test $? -eq 0; then
        echo '[+] compilazione terminata con flag ' "$CCARGS"
    else
        echo '[-] qualcosa è andato storto, controlla il log di compilazione'
        echo 'waiting per continuare...'
        read
    fi
    CCARGS=$(echo $CCARGS | tr ' ' '_')
    for loc in $LOC_TESTS; do
        for n_cli in $CLIENT_TESTS; do
            #start server
            logprefix="logs/server-$CCARGS-$loc-$n_cli"
            $REMBUF ./server $TEST_PORT 2> $logprefix-timings.log > $logprefix-stdout.log &
            echo $! > logs/server-$loc-$n_cli.pid
            echo '[+] server in funzione' $loc $n_cli $CCARGS

            logprefix="logs/client_udp-$CCARGS-$loc-$n_cli"
            rm -f $logprefix-{timings,stdout}.log
            for i in `seq 1 $n_cli`; do
                cp files/$loc-orig.txt files/$loc-$i.txt
```