

Project 2: Unix Shell

Määritelmä

Projektin tarkoituksena oli luoda oma C-kielinen versio unix shellistä, nimeltä Wisconsin shell, eli wish. Ohjelmalle syötetään käskyjä, jotka se toteuttaa luomalla lapsiprosessin, joka suorittaa shellille annetun käskyn. Tämän jälkeen shell jää odottamaan lisää käskyjä.

Shellä voi käyttää interaktiivisessa tai batch moodissa. Interaktiivisessa moodissa shell ottaa käyttäjän syötteen komento riviltä ja batch moodissa käskyt luetaan ohjelmalle annetusta tiedostosta. Shell ohjelma sisältää kolme sisään rakennettua käskyä: exit: lopettaa ohjelman, cd: vaihtaa hakemistoa, path: muokkaa shellin haku polkua.

Shell voi myös ohjata ohjelman tulosteen tiedostoon ruudun sijaan. Lisäksi shellille voi antaa useamman käskyn kerrallaan rinnan suoritettavaksi, jolloin shell odottaa, että ne kaikki on suoritettu ennen kuin jatkaa.

Shell tulostaa useimpien virheiden jälkeen standardi virheviestin ja jatkaa toimintaansa. Virheviestejä on useampi ja useimmissa tapauksissa tulostetaan tarkempi viesti.

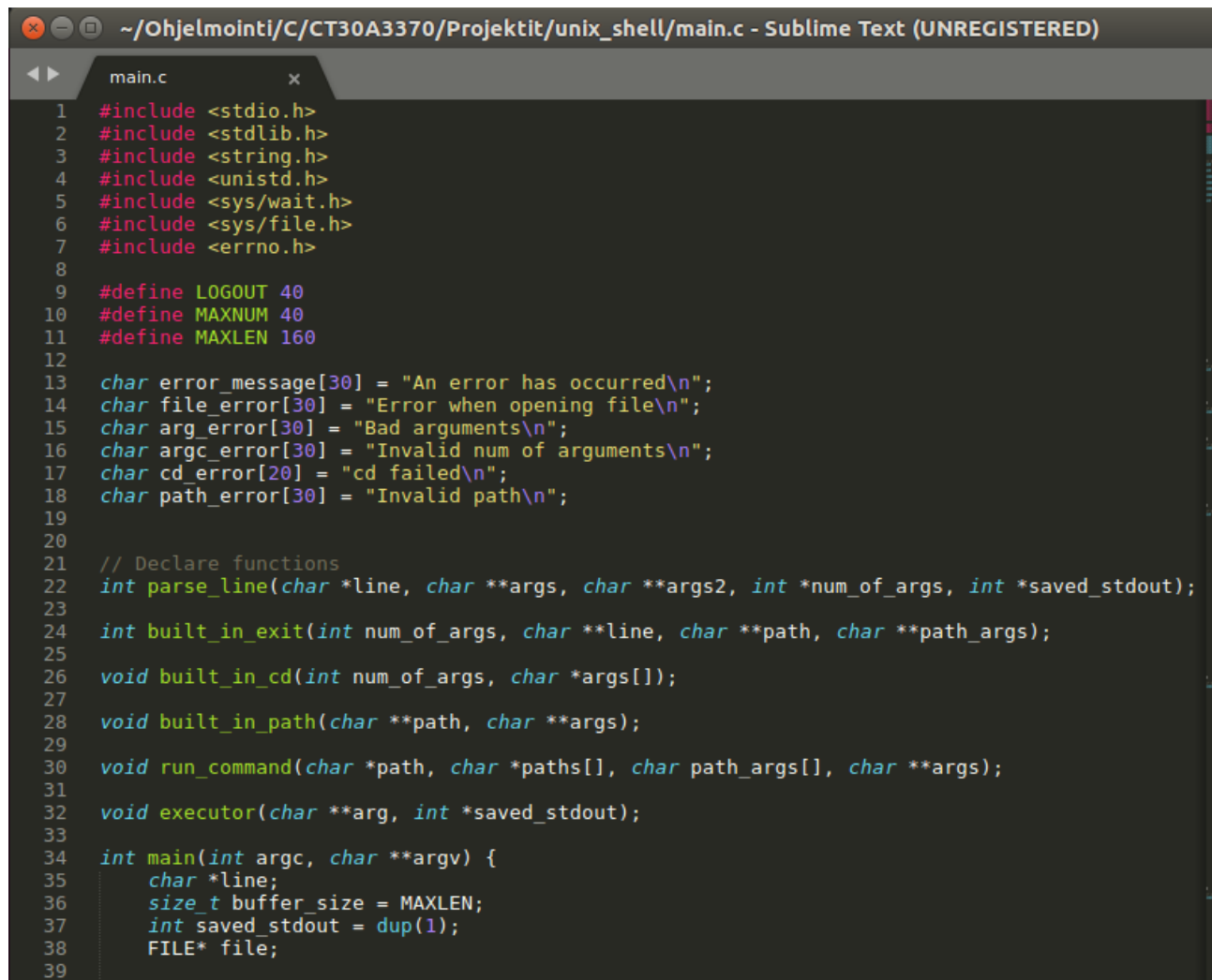
Ohjelman lähdekoodi on yhdellä C tiedostolla, sen olisi hyvinkin voinut jakaa useampaan tiedostoon. Ohjelmassa on kuusi funktiota mainin lisäksi. Käyn ohjelman toiminnan läpi aloittaen mainista, jokaiseen osioon on liitetty siihen liittyvä kuva/kuvia koodista.

Main

Ohjelman alussa on määritelty kuusi virheviestiä ja julistettu funktiot. Mainin tehtävänä on tarkastaa annettujen parametrien määrä ja ohjata sen perusteella shell interaktiiviseen tai batch moodiin. Jos parametreja on enemmän kuin yksi eli on esimerkiksi annettu enemmän kuin yksi batch tiedosto, ohjelma keskeyttää toimintansa. Jos niitä on nolla tai yksi, ohjelma aloittaa while loopin, joka kestää, kunnes batch tiedosto on käyty läpi tai kunnes interaktiivisessa moodissa annetaan exit käsky. Jos parametreja ei ole annettu mennään

interaktiiviseen moodiin, jossa tulostetaan *wish*> prompti ja otetaan käyttäjän syöte *getline*:lla ja kutsutaan *executor* funktiota. Kun parametreja on annettu yksi, eli batch tiedosto, avataan se ja luetaan sitä rivi kerrallaan while loopissa *getline*:lla ja jokaisella rivillä kutsutaan *executor* funktiota. Jos *getline* aiheuttaa virheen, ohjelma keskeyttää toimintansa.

Kuvat: main



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/wait.h>
6  #include <sys/file.h>
7  #include <errno.h>
8
9  #define LOGOUT 40
10 #define MAXNUM 40
11 #define MAXLEN 160
12
13 char error_message[30] = "An error has occurred\n";
14 char file_error[30] = "Error when opening file\n";
15 char arg_error[30] = "Bad arguments\n";
16 char argc_error[30] = "Invalid num of arguments\n";
17 char cd_error[20] = "cd failed\n";
18 char path_error[30] = "Invalid path\n";
19
20
21 // Declare functions
22 int parse_line(char *line, char **args, char **args2, int *num_of_args, int *saved_stdout);
23
24 int built_in_exit(int num_of_args, char **line, char **path, char **path_args);
25
26 void built_in_cd(int num_of_args, char *args[]);
27
28 void built_in_path(char **path, char **args);
29
30 void run_command(char *path, char *paths[], char path_args[], char **args);
31
32 void executor(char **arg, int *saved_stdout);
33
34 int main(int argc, char **argv) {
35     char *line;
36     size_t buffer_size = MAXLEN;
37     int saved_stdout = dup(1);
38     FILE* file;
39 }
```

```
~/Ohjelmointi/C/CT30A3370/Projektit/unix_shell/main.c - Sublime Text (UNREGISTERED)
main.c x
33
34 int main(int argc, char **argv) {
35     char *line;
36     size_t buffer_size = MAXLEN;
37     int saved_stdout = dup(1);
38     FILE* file;
39
40     line = (char *)malloc(buffer_size * sizeof(char));
41     if (line == NULL) {
42         write(STDERR_FILENO, error_message, strlen(error_message));
43         exit(1);
44     }
45
46     if (argc == 1 || argc == 2) {
47         while (1) {
48             /* Restore stdout */
49             // Source: https://stackoverflow.com/questions/11042218/c-restore-stdout-to-term
50             dup2(saved_stdout, 1);
51             close(saved_stdout);
52
53             if (argc == 1) { /* Run in interactive mode if no argument given */
54                 /* Print the prompt */
55                 printf("wish> ");
56
57                 /* Read the users command */
58                 getline(&line, &buffer_size, stdin);
59                 /* Execute command */
60                 executor(&line, &saved_stdout);
61                 continue;
62             } else { /* Run in batch mode if argument given */
63                 if ((file = fopen(argv[1], "r")) == NULL) {
64                     write(STDERR_FILENO, file_error, strlen(file_error));
65                     exit(1);
66                 }
67                 /* Read batch file for commands */
68                 while (getline(&line, &buffer_size, file) != -1) {
69                     if (errno != 0) {
70                         exit(1);
71                     }
72                     /* Execute command */
73                     executor(&line, &saved_stdout);
74                 }
75                 free(line);
76                 fclose(file);
77                 exit(0);
78             }
79         }
80     } else { /* Terminate program if shell invoked with more than one argument */
81         write(STDERR_FILENO, argc_error, strlen(argc_error));
82         exit(1);
83     }
84
85     return 0;
86 }
87
88
```

Executor-funktio

Executor-funktio ottaa parametreina, joko käyttäjän syöttämän rivin tai batch tiedostosta luetun rivin ja tarkistaa *strpbrk* funktiolla sisältääkö rivi "&" merkkiä rinnakkais komentojen merkiksi. Jos tämä merkki löytyy, jaetaan rivi taulukkoon erillisiksi komennoiksi while loopissa käyttäen *strsep* funktiota "&" erittely merkillä. Sen jälkeen jokaiselle taulukon erilliselle komennolle toistetaan for loopissa erittely komennoksi ja argumenteiksi kutsumalla *parse_line* funktiota, jonka jälkeen katsotaan *strcmp*:lla vastaako komento jotain shellin

sisäänrakennetuista komennoista (exit, cd, path) ja jos vastaa niin kutsutaan vastaavaa funktiota. Jos ei niin kutsutaan funktiota *run_command*. Jolle annetaan parametreina hakupolku/polut ja komento ja sen argumentit. Lopuksi palautetaan tuloste normaaliksi, jos se on ohjattu tiedostoon. Jos "&" merkkiä ei ole annettu tehdään käytännössä sama kuin ylempänä, mutta riviä ei jaotella osiin.

Kuvat: executor

```
~/Ohjelmointi/C/CT30A3370/Projektit/unix_shell/main.c - Sublime Text (UNREGISTERED)
main.c
263
264 /* Checks conditions and executes code and calls functions appropriately */
265 void executor(char **arg, int *saved_stdout) {
266     char *paths[MAXNUM], *args[MAXNUM], *args2[MAXNUM], *lines[MAXNUM];
267     char *cmd, *path = "/bin";
268     char *path_args = NULL;
269     int i, num_of_args;
270     char *line = *arg;
271
272     if (line == NULL) {
273         printf("\nlogout\n");
274         exit(0);
275     }
276
277     if (strlen(line) == 0) {
278         return;
279     }
280
281     line[strlen(line) - 1] = '\0';
282
283     if (strpbrk(line, "&")!=0) { /* Run multiple commands in parallel if "&" in input */
284         i = 0;
285         cmd = line;
286         /* Split input into separate commands */
287         while ((lines[i] = strsep(&cmd, "&")) != NULL) {
288             i++;
289         }
290         /* Parse and run each separate command */
291         for (int x = 0; x<i; x++) {
292             if (parse_line(lines[x], args, args2, &num_of_args, saved_stdout)==1) {
293                 return;
294             }
295             /* Skip command if NULL */
296             if (args[0]==NULL) {
297                 continue;
298             }
299             /* Check commands for built-ins */
300             if (strcmp(args[0], "exit")==0) {
301                 if (built_in_exit(num_of_args, &line, &path, &path_args)==1) {
302                     return;
303                 }
304             } else if (strcmp(args[0], "cd")==0) {
305                 built_in_cd(num_of_args, args);
306                 return;
307             } else if (strcmp(args[0], "path")==0) {
308                 built_in_path(&path, args);
309                 return;
310             } else { /* If no built-in command given, run with execv */
311                 /* Run command with given arguments */
312                 run_command(path, paths, path_args, args);
313
314                 /* Restore stdout */
315                 dup2(*saved_stdout, 1);
316                 close(*saved_stdout);
317             }
318         }
319     }
320     return;
}
```

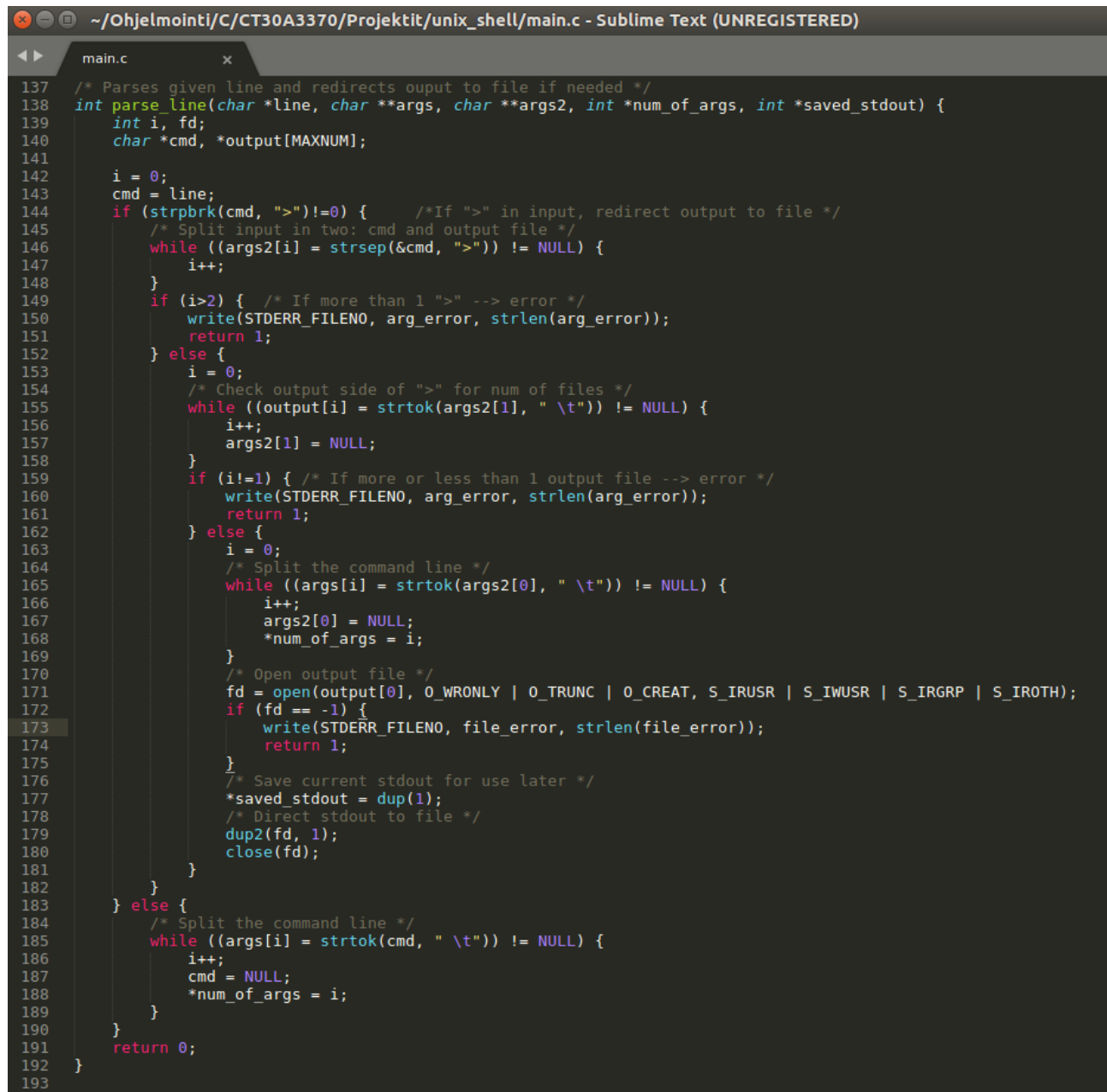
```
~/Ohjelmointi/C/CT30A3370/Projektit/unix_shell/main.c - Sublime Text (UNREGISTERED)
main.c
302         return;
303     }
304     } else if (strcmp(args[0], "cd")==0) {
305         built_in_cd(num_of_args, args);
306         return;
307     } else if (strcmp(args[0], "path")==0) {
308         built_in_path(&path, args);
309         return;
310     } else { /* If no built-in command given, run with execv */
311         /* Run command with given arguments */
312         run_command(path, paths, path_args, args);
313
314         /* Restore stdout */
315         dup2(*saved_stdout, 1);
316         close(*saved_stdout);
317     }
318 }
319 return;
320 } else { /* Run single command if no "&" in input */
321     if (parse_line(line, args, args2, &num_of_args, saved_stdout)==1) {
322         return;
323     }
324     /* Skip command if NULL */
325     if (args[0]==NULL) {
326         return;
327     }
328     /* Check commands for built-ins */
329     if (strcmp(args[0], "exit")==0) {
330         if (built_in_exit(num_of_args, &line, &path, &path_args)==1) {
331             return;
332         }
333     } else if (strcmp(args[0], "cd")==0) {
334         built_in_cd(num_of_args, args);
335         return;
336     } else if (strcmp(args[0], "path")==0) {
337         built_in_path(&path, args);
338         return;
339     } else { /* If no built-in command given, run with execv */
340         /* Run command with given arguments */
341         run_command(path, paths, path_args, args);
342         return;
343     }
344 }
345 }
```

Parse_line-funktio

Parse_line ottaa parametreina syöte rivin ja kaksi taulukkoa, joihin talletetaan komennot ja niiden argumentit. Ensin tarkastetaan *strpbrk* funktiolla onko syötteessä ">" merkki tulosteen uudelleen ohjauksen merkiksi ja jos on, syöte jaetaan kahteen osaan: komennoksi ja kohde tiedostoksi. Jos uudelleenohjaus merkkejä on useampi kuin yksi tulostetaan virheviesti ja palataan shelliin, muussa tapauksessa tarkastetaan annettujen tiedostojen lukumäärä jakamalla *strtok* funktiolla while loopissa syötteen kohde tiedosto puoli ja laskemalla kierrosten määrä, jos kierrosten määrä on muu kuin yksi, tulostetaan virheviesti ja palataan shelliin. Jos kierroksia on yksi, jaetaan syötteen komento puoli osiin, eli itse komentoon ja sen argumentteihin ja koska tuloste haluttiin ohjata tiedostoon, avataan tiedosto ja luodaan

se tarvittaessa ja *dup2* funktiolla siirretään tuloste tiedostoon. Jos uudelleen ohjaus merkkiä ei ole syötteessä tarvitsee vain jakaa se *strtok*:lla komennoksi ja sen argumenteiksi.

Kuva: parse_line



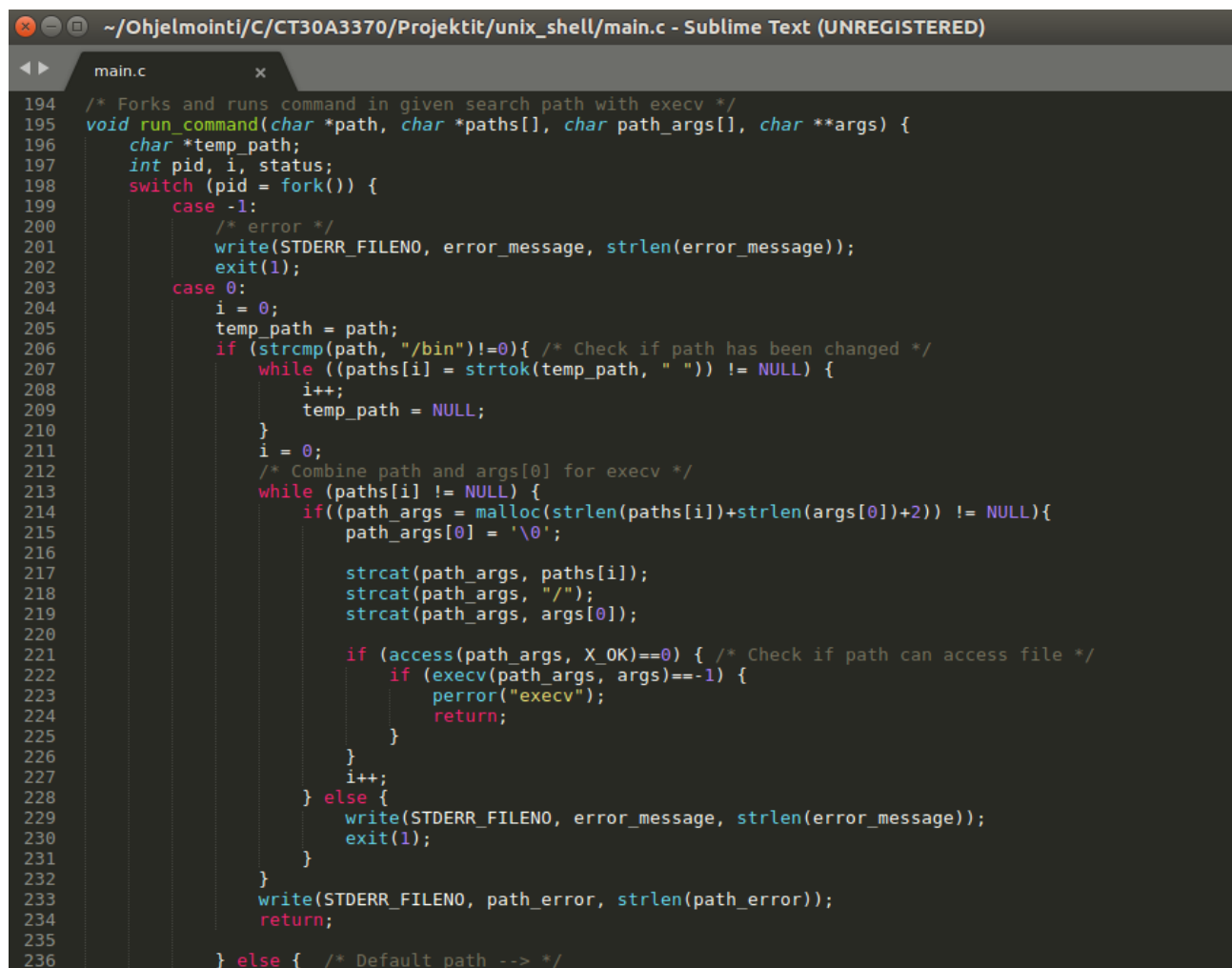
```
137 /* Parses given line and redirects output to file if needed */
138 int parse_line(char *line, char **args, char **args2, int *num_of_args, int *saved_stdout) {
139     int i, fd;
140     char *cmd, *output[MAXNUM];
141
142     i = 0;
143     cmd = line;
144     if (strpbrk(cmd, ">")!=0) { /*If ">" in input, redirect output to file */
145         /* Split input in two: cmd and output file */
146         while ((args2[i] = strsep(&cmd, ">")) != NULL) {
147             i++;
148         }
149         if (i>2) { /* If more than 1 ">" --> error */
150             write(STDERR_FILENO, arg_error, strlen(arg_error));
151             return 1;
152         } else {
153             i = 0;
154             /* Check output side of ">" for num of files */
155             while ((output[i] = strtok(args2[1], " \t")) != NULL) {
156                 i++;
157                 args2[1] = NULL;
158             }
159             if (i!=1) { /* If more or less than 1 output file --> error */
160                 write(STDERR_FILENO, arg_error, strlen(arg_error));
161                 return 1;
162             } else {
163                 i = 0;
164                 /* Split the command line */
165                 while ((args[i] = strtok(args2[0], " \t")) != NULL) {
166                     i++;
167                     args2[0] = NULL;
168                     *num_of_args = i;
169                 }
170                 /* Open output file */
171                 fd = open(output[0], O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
172                 if (fd == -1) {
173                     write(STDERR_FILENO, file_error, strlen(file_error));
174                     return 1;
175                 }
176                 /* Save current stdout for use later */
177                 *saved_stdout = dup(1);
178                 /* Direct stdout to file */
179                 dup2(fd, 1);
180                 close(fd);
181             }
182         }
183     } else {
184         /* Split the command line */
185         while ((args[i] = strtok(cmd, " \t")) != NULL) {
186             i++;
187             cmd = NULL;
188             *num_of_args = i;
189         }
190     }
191     return 0;
192 }
193
```

Run_command-funktio

Run_command luo lapsiprosessin kutsumalla *fork()*:ia ja tässä lapsiprosessissa tarkastetaan onko shellin hakupolkua muutettu, jos ei niin yhdistetään hakupolku ja

suoritettava komento yhdeksi merkkijonoksi käyttäen *strcat* funktiota, jolloin merkkijono voidaan antaa parametriksi *execv*:lle, joka etsii hakupolusta komento vastaavan ohjelman ja suorittaa sen annetuilla argumenteilla. Tämän jälkeen palataan shelliin. Jos hakupolkua on muutettu oletusarvosta jaetaan hakupolku osiin *strtok*:lla ja kullekin osalle toistetaan ylempänä mainittu merkkijonojen yhdistäminen *execv*:tä varten, lisäksi ennen kuin suoritetaan *execv* kutsu, testataan *access*():lla onko hakupolku toimiva.

Kuvat: run_command



```
194 /* Forks and runs command in given search path with execv */
195 void run_command(char *path, char *paths[], char path_args[], char **args) {
196     char *temp_path;
197     int pid, i, status;
198     switch (pid = fork()) {
199         case -1:
200             /* error */
201             write(STDERR_FILENO, error_message, strlen(error_message));
202             exit(1);
203         case 0:
204             i = 0;
205             temp_path = path;
206             if (strcmp(path, "/bin")!=0){ /* Check if path has been changed */
207                 while ((paths[i] = strtok(temp_path, " ")) != NULL) {
208                     i++;
209                     temp_path = NULL;
210                 }
211                 i = 0;
212                 /* Combine path and args[0] for execv */
213                 while (paths[i] != NULL) {
214                     if((path_args = malloc(strlen(paths[i])+strlen(args[0])+2)) != NULL){
215                         path_args[0] = '\0';
216
217                         strcat(path_args, paths[i]);
218                         strcat(path_args, "/");
219                         strcat(path_args, args[0]);
220
221                         if (access(path_args, X_OK)==0) { /* Check if path can access file */
222                             if (execv(path_args, args)==-1) {
223                                 perror("execv");
224                                 return;
225                             }
226                         }
227                         i++;
228                     } else {
229                         write(STDERR_FILENO, error_message, strlen(error_message));
230                         exit(1);
231                     }
232                 }
233                 write(STDERR_FILENO, path_error, strlen(path_error));
234                 return;
235             }
236     } else { /* Default path --> */
```

```
~/Ohjelmointi/C/CT30A3370/Projektit/unix_shell/main.c - Sublime Text (UNREGISTERED)
main.c
230         exit(1);
231     }
232 }
233 write(STDERR_FILENO, path_error, strlen(path_error));
234 return;
235
236 } else { /* Default path --> */
237     /* Combine path and args[0] for execv */
238     if((path_args = malloc(strlen(path)+strlen(args[0])+2)) != NULL){
239         path_args[0] = '\0';
240         strcat(path_args, path);
241         strcat(path_args, "/");
242         strcat(path_args, args[0]);
243     } else {
244         write(STDERR_FILENO, error_message, strlen(error_message));
245         exit(1);
246     }
247     if (execv(path_args, args)==-1) {
248         perror("execv");
249         return;
250     }
251 }
252 break;
253 default:
254     /* parent (shell) */
255     if (wait(&status) == -1) {
256         write(STDERR_FILENO, error_message, strlen(error_message));
257         exit(1);
258     }
259     break;
260 }
261 return;
262 }
263 }
```

Sisäänrakennetut funktiot: exit, cd, path

Exit on yksinkertainen sillä se vain tarkistaa, että sille on annettu oikea määrä argumentteja, jos ei niin se palaa shelliin ja tulostaa virheviestin, jos on niin se vapauttaa muistin ja kutsuu `exit()`.

Myös `cd` tarkastaa argumenttien määrän ja tulostaa tarvittaessa virheviestin. Funktio vaihtaa hakemistoa kutsumalla `chdir()` ja jos `chdir` palauttaa -1 tulostetaan virheviesti.

Path muokkaa shellin hakupolkua ylikirjoittamalla vanhan hakupolun ja korvaamalla sen käyttäjän antaman syötteen perusteella. Tämä tapahtuu while loopissa käyttäen `strcpy` ja `strcat`.

Kuva: exit, cd ja path

```
~ /Ohjelmointi/C/CT30A3370/Projektit/unix_shell/main.c - Sublime Text (UNREGISTERED)
main.c
87
88
89 /* Built-in 'exit' command */
90 int built_in_exit(int num_of_args, char **line, char **path, char **path_args) {
91     if (num_of_args > 1) { /* Check input for correct num of args */
92         write(STDERR_FILENO, argc_error, strlen(argc_error));
93         return 1;
94     } else {
95         free(*line);
96         free(*path_args);
97         exit(0);
98     }
99     return 0;
100 }
101
102 /* Built-in 'cd' command */
103 void built_in_cd(int num_of_args, char *args[]) {
104     if (num_of_args == 2) { /* Check input for correct num of args */
105         if (chdir(args[1]) == -1) {
106             write(STDERR_FILENO, cd_error, strlen(cd_error));
107         }
108     } else {
109         write(STDERR_FILENO, argc_error, strlen(argc_error));
110     }
111     return;
112 }
113
114 /* Built-in 'path' command: Overwrites shell search path */
115 void built_in_path(char **path, char **args) {
116     int i = 1;
117     *path = NULL;
118
119     while (args[i] != NULL) {
120         if (*path == NULL) {
121             *path = (char *)malloc((strlen(args[i])+2));
122             strcpy(*path, args[i]);
123             strcat(*path, " ");
124             i++;
125         } else {
126             *path = (char *)realloc(*path, (strlen(*path)+strlen(args[i])+2));
127             strcat(*path, args[i]);
128             strcat(*path, " ");
129             i++;
130         }
131     }
132     strcat(*path, "\0");
133     return;
134 }
135
```