**Λειτουργικά Συστήματα**

**Άσκηση 4: Χρονοδρομολόγηση**

**1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη**

1 #include **<**errno**.**h**>**

2 #include **<**unistd**.**h**>**

3 #include **<**stdlib**.**h**>**

4 #include **<**stdio**.**h**>**

5 #include **<**signal**.**h**>**

6 #include **<**string**.**h**>**

7 #include **<**assert**.**h**>**

8

9 #include **<**sys**/**wait**.**h**>**

10 #include **<**sys**/**types**.**h**>**

11

12 #include "proc-common.h"

13 #include "request.h"

14

15 /\* Compile-time parameters. \*/

16 #define SCHED\_TQ\_SEC 2 /\* time quantum \*/

17 #define TASK\_NAME\_SZ 60 /\* maximum size for a task's name \*/

18

19 **typedef** struct PCBstruct **{** //structure of the processes

20 int id**;**

21 int pid**;**

22 char name**[**TASK\_NAME\_SZ**];**

23 struct PCBstruct **\***next**;**

24 **}** pcb\_t**;**

25

26 int process**;** //how many processes i have

27 pcb\_t **\***current**,\***previous**;** //pointers to the list

28

29 /\*

30 \* SIGALRM handler

31 \*/

32 static void

33 sigalrm\_handler**(**int signum**)**

34 **{**

35 **if** **(**kill**(**current**->**pid**,** SIGSTOP**)** **<** 0**)** **{** //kill current process with sigstop

36 perror**(**"SIGSTOP"**);** exit**(**1**);** //check for errors

37 **}**

38 printf**(**"Stopping procress with ID = %d \n"**,**current**->**id**);**

39 **}**

40

41 /\*

42 \* SIGCHLD handler

43 \*/

44 static void

45 sigchld\_handler**(**int signum**)**

46 **{**

47 **for** **(;;)** **{**

48 int p**,**status**;**

49 p **=** waitpid**(-**1**,** **&**status**,** WUNTRACED **|** WNOHANG**);**

50 //whohang => return immediately if no child has exited

51 //wuntraced =>return if a child has stopped

52 //waitpid returns the process id of the child whose state has changed

53 **if** **(**p **<** 0**)** **{**

54 perror**(**"waitpid"**);**

55 exit**(**1**);**

56 **}**

57 **if** **(**p **==** 0**)** **break;** //has not changed state yet

58

59 // explain\_wait\_status(p, status);

60

61 **if** **(**WIFEXITED**(**status**)** **||** WIFSIGNALED**(**status**)){** /\* A child has died \*/

62 alarm**(**0**);** // midenizoume ton xrono tou alarm

63 printf**(**"Died process with ID = %d.\n"**,**current**->**id**);**

64 previous**->**next**=**current**->**next**;** // delete current node xwris na sp azw ti lista

65 free**(**current**);**

66 process **--;** //meiwnw arithmo diergasiwn

67 **if** **(**process **!=** 0**){** //exw kai alles diergasies

68 current **=** previous **->** next**;** //dixnw stin epomeni

69 kill**(**current**->**pid**,**SIGCONT**);** //tin 3ekinw

70 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

71 **if(**process **!=**1**)** alarm**(**SCHED\_TQ\_SEC**);** // unless a single p rocess is left initiate new alarm

72 **}**

73 **else{** // case : no more processes left to execute

74 printf**(**"No more processes left.\n"**);**

75 exit**(**0**);**

76 **}**

77 **}**

78

79 **if** **(**WIFSTOPPED**(**status**))** **{** /\* A child has stopped due to SIGSTOP/SI GTSTP(alarm went off), etc \*/

80 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

81 previous **=** current**;** // the next one

82 current **=** current**->**next**;**

83 kill**(**current**->**pid**,**SIGCONT**);** //sinexizei i epomeni

84 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

85 alarm**(**SCHED\_TQ\_SEC**);** //3anarxizw to alarm

86 **}**

87

88 **}**

89 **}**

90

91 /\* Install two signal handlers.

92 \* One for SIGCHLD, one for SIGALRM.

93 \* Make sure both signals are masked when one of them is running.

94 \*/

95 static void

96 install\_signal\_handlers**(**void**)**

97 **{**

98 sigset\_t sigset**;**

99 struct sigaction sa**;**

100

101 sa**.**sa\_handler **=** sigchld\_handler**;** //sa\_handler -> pointer to a signal-catching function

102 sa**.**sa\_flags **=** SA\_RESTART**;** //flags affect the signals b ehavior. if set the function can be interrupted by the signal and the functi on restarts.

103 sigemptyset**(&**sigset**);**

104 sigaddset**(&**sigset**,** SIGCHLD**);**

105 sigaddset**(&**sigset**,** SIGALRM**);**

106 sa**.**sa\_mask **=** sigset**;** //additional set of signals are blocked during execution of signal-catching function

107 **if** **(**sigaction**(**SIGCHLD**,** **&**sa**,** **NULL)** **<** 0**)** **{**

108 perror**(**"sigaction: sigchld"**);**

109 exit(1);

110 }

111

112 sa.sa\_handler = sigalrm\_handler;

113 if (sigaction(SIGALRM, &sa, NULL) < 0) {

114 perror("sigaction: sigalrm");

115 exit(1);

116 }

117

118 /\*

119 \* Ignore SIGPIPE, so that write()s to pipes

120 \* with no reader do not result in us being killed,

121 \* and write() returns EPIPE instead.

122 \*/

123 if (signal(SIGPIPE, SIG\_IGN) < 0) {

124 perror("signal: sigpipe");

125 exit(1);

126 }

127 }

128

129 void child(char \*process\_name) {

130 char \*newargv[] = {process\_name, NULL, NULL, NULL };

131 char \*newenviron[] = { NULL };

132

133 raise(SIGSTOP); // arxika tis stamatame oles

134

135

136 // replace myself with the process\_name, when receives SIGCONT it continues as a "prog"

137 execve(process\_name, newargv, newenviron);

138

139 /\* execve() only returns on error \*/

140 perror("execve");

141 exit(1);

142 }

143

144

145

146

147 int main(int argc, char \*argv[])

148 {

149 int nproc,i; //nproc=number of given processes

150 /\*

151 \* For each of argv[1] to argv[argc - 1],

152 \* create a new child process, add it to the process list.

153 \*/

154

155 nproc = argc -1; /\* number of proccesses goes here \*/

156 pcb\_t \*pcb = NULL; //pointer to the list arxika NULL

157

158 for(i=0; i < nproc; i++)

159 {

160 pid\_t pidc;

161

162

163 pidc = fork();

164 if (pidc < 0) { // error

165 perror("parent: fork");

166 exit(1);

167 }

168 if (pidc == 0) { // child

169 child(argv[i+1]);

170 exit(1);

171 }

172

173 if (pcb == NULL){

174 pcb = malloc(sizeof(pcb\_t));

175 if (pcb == NULL) return(1);

176

177 pcb->id=i; //dinw id

178 pcb->pid=pidc; //dinw pid

179 strcpy(pcb->name,argv[i+1]); //dinw onoma

180 pcb->next=NULL; //kanw to next NULL

181 current = pcb;

182 }

183 else {

184 current->next = malloc(sizeof(pcb\_t)); //dimiourgw ti lista

185 current = current->next;

186 current->id=i;

187 current->pid=pidc;

188 strcpy(current->name,argv[i+1]);

189 if(i == (nproc-1)) current->next = pcb;

190 }

191 }

192

193 /\* Wait for all children to raise SIGSTOP before exec()ing. \*/

194 wait\_for\_ready\_children(nproc);

195

196 /\* Install SIGALRM and SIGCHLD handlers. \*/

197 install\_signal\_handlers();

198

199 if (nproc == 0) {

200 fprintf(stderr, "Scheduler: No tasks. Exiting...\n");

201 exit(1);

202 }

203 previous = current;

204 current = pcb; //dinxw sti kefali tis listas

205 process = nproc; //poses exw pou perimenoun

206 kill(current->pid,SIGCONT); //initiate

207 alarm(SCHED\_TQ\_SEC); //initate alarm

208

209 /\* loop forever until we exit from inside a signal handler. \*/

210 while (pause());

211

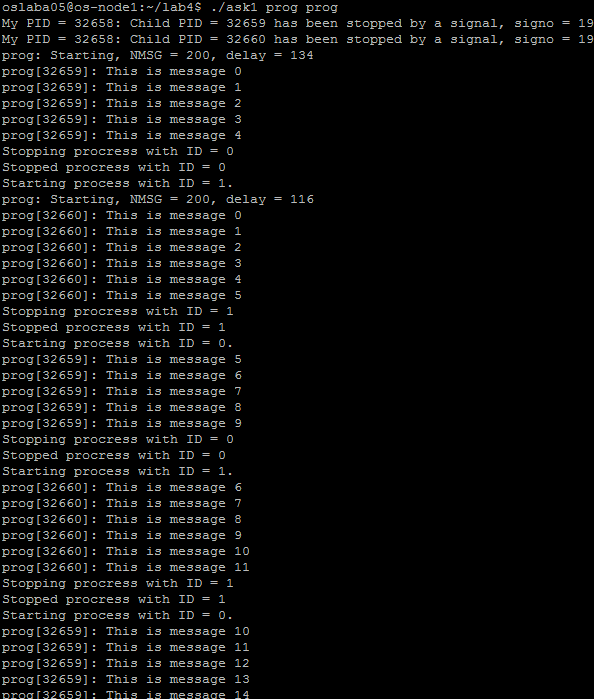
212 /\* Unreachable \*/

213 fprintf(stderr, "Internal error: Reached unreachable point\n");

214 return 1;

215 }

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου (συνεχίζεται επ 'αόριστον):



**Απαντήσεις**:

1. Μέσα από τη συνάρτηση install\_signal\_handlers() ορίσαμε handlers που μπλοκάρουν άλλα σήματα κάθε φορά που λαμβάνουμε σήμα SIGARLM και SIGCHLD. Έτσι όταν λάβουμε ένα από αυτά τα δύο σήματα θα εκτελεστεί ο αντίστοιχος handler και τυχόν άλλο σήμα που καταυθάνει θα μπλοκάρει να τελειώσει ο handler του πρώτου. Εξασφαλίζουμε δηλαδή τη σειριακή τους εκτέλεση. Με αυτό το τρόπο αποφεύγουμε τυχών ασυνέπειες που μπορεί να δημιουργηθούν αφού πλέον ο χρονοδρομολογητής θα χειρίζεται μόνο ένα σήμα κάθε φορά. Σε ένα πραγματικό χρονοδρομολογητή χώρου πυρήνα (kernel) γίνεται κάτι αντίστοιχο αλλά για interrupts.

2. Κάθε φορά που ο πατέρας λαμβάνει SIGCHLD αναμένουμε ότι θα τερματιστεί η τρέχουσα διεργασία. Σε περίπτωση που η τρέχουσα διεργασία λάβει σήμα KILL τότε θα στείλει SIGCHLD στο χρονοδρομολογητή για να συνεχίσει με μια νέα διεργασία. Αν λάβει όμως σήμα KILL μια οποιαδήποτε άλλη διεργασία που είναι απλά στη λίστα, θα δημιουργηθεί πρόβλημα αφού δεν έχουμε διαχειριστεί μια τέτοια περίπτωση. Με τον υπάρχον κώδικα, αν ληφθεί σήμα KILL θα γίνει free ο curent και όχι η διεργασία που έλαβε το KILL με αποτέλεσμα να δημιουργηθεί ασυνέπεια στη δομή. Όταν μετά ο χρονοδρομολογητής προσπαθήσει να τρέξει τη διεργασία που είχε λάβει το KILL θα εμφανίζεται σφάλμα αφού έχει σκοτωθεί και δεν μπορεί να τρέξει. Δηλαδή σκοτώνουμε τη σωστή διεργασία αλλά αφαιρείται από τη λίστα διεργασιών μια άλλη διεργασία αντί αυτή. Θα μπορούσαμε να προβλέψουμε αυτή τη περίπτωση, κάνοντας έλεγχο με βάσει το PID. Δηλαδή όταν λαμβάναμε εξωτερικό σήμα KILL, να παίρναμε το PID και να το συγκρίναμε με αυτό της τρέχουσας διεργασίας. Αν δεν θα ήταν το ίδιο τότε θα ψάχναμε στη λίστα να βρούμε τη διεργασία που έχει αυτό το PID και θα την κάναμε αυτή free και θα ανανεώναμε σωστά τη δομή τη λίστας.

3. Χρειαζόμαστε δύο σήματα ούτως ώστε να είμαστε σίγουροι ότι η διεργασία έχει σταματήσει. Με τη χρήση και των δύο σημάτων, λαμβάνουμε πληροφορία από τη γονική διεργασία ότι το παιδί της έχει σίγουρα σταματήσει. Αν δεν εξασφαλίζαμε αυτή τη λειτουργία (δηλαδή χρησιμοποιούσαμε ένα σήμα μόνο) θα υπήρχε περίπτωση το παιδί να μην τερμάτιζε ασχέτως του ότι του στείλαμε σήμα KILL και έτσι θα ξεκινούσαμε την επόμενη διεργασία χωρίς να είμαστε σίγουροι ότι η πρώτη έχει σταματήσει.

**1.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού**

1 #include **<**errno**.**h**>**

2 #include **<**unistd**.**h**>**

3 #include **<**stdlib**.**h**>**

4 #include **<**stdio**.**h**>**

5 #include **<**signal**.**h**>**

6 #include **<**string**.**h**>**

7 #include **<**assert**.**h**>**

8

9 #include **<**sys**/**wait**.**h**>**

10 #include **<**sys**/**types**.**h**>**

11

12 #include "proc-common.h"

13 #include "request.h"

14

15 /\* Compile-time parameters. \*/

16 #define SCHED\_TQ\_SEC 2 /\* time quantum \*/

17 #define TASK\_NAME\_SZ 60 /\* maximum size for a task's name \*/

18 #define SHELL\_EXECUTABLE\_NAME "shell" /\* executable for shell \*/

19

20 **typedef** struct PCBstruct **{**

21 int id**;**

22 int pid**;**

23 char name**[**TASK\_NAME\_SZ**];**

24 struct PCBstruct **\***next**;**

25 **}** pcb\_t**;**

26

27 int help**=**0**;**

28 int process**,**shellpid**,**idc**;**

29 pcb\_t **\***current**,\***previous**,\***head**;**

30

31

32

33 void child**(**char **\***process\_name**)** **{**

34 char **\***newargv**[]** **=** **{**process\_name**,** **NULL,** **NULL,** **NULL** **};**

35 char **\***newenviron**[]** **=** **{** **NULL** **};**

36

37 raise**(**SIGSTOP**);** // arxika tis stamatame oles

38

39

40 // replace myself with the process\_name

41 execve**(**process\_name**,** newargv**,** newenviron**);**

42

43 /\* execve() only returns on error \*/

44 perror**(**"execve"**);**

45 exit**(**1**);**

46 **}**

47

48

49

50 /\* Print a list of all tasks currently being scheduled. \*/

51 static void

52 sched\_print\_tasks**(**void**)**

53 **{**

54 pcb\_t **\***i **=** current**;**

55 printf**(**"\nRUNNING PROCESS : name: %s ,pid: %d ,id: %d\n"**,** i**->**name**,** i**->**pid**,** i**->**id**);**

56 **while** **(**i**->**next **!=** current**)** **{**

57 i **=** i**->** next**;**

58 printf**(**"name: %s ,pid: %d ,id: %d\n"**,** i**->**name**,** i**->**pid**,** i**->**id**);**

59 **}**

60 printf**(**"\n"**);**

61 **}**

62

63 /\* Send SIGKILL to a task determined by the value of its

64 \* scheduler-specific id.

65 \*/

66 static void

67 sched\_kill\_task\_by\_id**(**int id**)**

68 **{**

69 pcb\_t **\***i **=** head**;**

70 **if** **(**head**->**id **==** id**)**

71 **{**

72 printf**(**"Killing process with id : %d\n"**,**i**->**id**);**

73 kill**(** i**->**pid**,** SIGTERM**);**

74 **}**

75 **else** **{**

76 **while** **(**i**->**next **!=** head**){**

77 i **=** i**->** next**;**

78 **if** **(**i**->**id **==** id**){**

79 printf**(**"Killing process with id : %d\n"**,**i**->**id**);**

80 kill**(** i**->**pid**,** SIGTERM**);**

81 **return;**

82 **}**

83

84 **}**

85

86 printf**(**"Not found process with id : %d\n"**,** id**);**

87 **}**

88 **}**

89

90

91 /\* Create a new task. \*/

92 static void

93 sched\_create\_task**(**char **\***executable**)**

94 **{**

95 pid\_t p**;**

96 pcb\_t **\***i**;**

97

98 p **=** fork**();**

99

100 **if** **(**p **<** 0**)** **{** // error

101 perror**(**"parent: fork"**);**

102 exit**(**1**);**

103 **}**

104 **if** **(**p **==** 0**)** **{** // child

105 child**(**executable**);**

106 exit**(**1**);**

107 **}**

108 help**++;**

109 i **=** malloc**(sizeof(**pcb\_t**));**

110 i**->**id**=++**idc**;**

111 i**->**pid**=**p**;**

112 strcpy**(**i**->**name**,**executable**);**

113 process**++;**

114 previous**->**next **=** i**;**

115 i **->** next **=** current**;**

116

117 printf**(**"Added %s to the processes list.\n\n"**,** executable**);**

118 **}**

119

120 /\* Process requests by the shell. \*/

121 static int

122 process\_request**(**struct request\_struct **\***rq**)**

123 **{**

124 **switch** **(**rq**->**request\_no**)** **{**

125 **case** REQ\_PRINT\_TASKS**:**

126 sched\_print\_tasks**();**

127 **return** 0**;**

128

129 **case** REQ\_KILL\_TASK**:**

131 **return** 0**;**

132

133 **case** REQ\_EXEC\_TASK**:**

134 sched\_create\_task**(**rq**->**exec\_task\_arg**);**

135 **return** 0**;**

136

137 **default:**

138 **return** **-**ENOSYS**;**

139 **}**

140 **}**

141

142 /\*

143 \* SIGALRM handler

144 \*/

145 static void

146 sigalrm\_handler**(**int signum**)**

147 **{**

148 **if** **(**kill**(**current**->**pid**,** SIGSTOP**)** **<** 0**)** **{**

149 perror**(**"SIGSTOP"**);** exit**(**1**);**

150 **}**

151 printf**(**"Stopping procress with ID = %d \n"**,**current**->**id**);**

152 **}**

153

154 /\*

155 \* SIGCHLD handler

156 \*/

157 static void

158 sigchld\_handler**(**int signum**)**

159 **{**

160 **for** **(;;)** **{**

161 int p**,**status**;**

162 p **=** waitpid**(-**1**,** **&**status**,** WUNTRACED **|** WNOHANG**);**

163 **if** **(**p **<** 0**)** **{**

164 perror**(**"waitpid"**);**

165 exit**(**1**);**

166 **}**

167 **if** **(**p **==** 0**)** **break;**

168 **if(**help **>** 0**)** **{**help**--;** **return;}**

169 // explain\_wait\_status(p, status);

170

171 **if** **(**WIFEXITED**(**status**)** **||** WIFSIGNALED**(**status**)){** /\* A child has died \*/

172 alarm**(**0**);** // midenizoume ton xrono tou alarm

173 printf**(**"Died process with ID = %d.\n"**,**current**->**id**);**

175 **if(**current**==**head**)** head **=** current**->**next**;**

176 free**(**current**);**

177 process **--;**

178 **if** **(**process **!=** 0**){**

179 current **=** previous **->** next**;**

180 kill**(**current**->**pid**,**SIGCONT**);**

181 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

182 alarm**(**SCHED\_TQ\_SEC**);**

183 //if(process !=1) alarm(SCHED\_TQ\_SEC); // unless a single process is left

184 **}**

185 **else{** // case : no more processes left to execute

186 printf**(**"No more processes left.\n"**);**

187 exit**(**0**);**

188 **}**

189 **}**

190

191 **if** **(**WIFSTOPPED**(**status**))** **{** /\* A child has stopped due to SIGSTOP/SIGTSTP, etc \*/

192 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

193 previous **=** current**;** // the next one

194 current **=** current**->**next**;**

195 kill**(**current**->**pid**,**SIGCONT**);**

196 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

197 alarm**(**SCHED\_TQ\_SEC**);**

198 **}**

199

200 **}**

201 **}**

202

203 /\* Disable delivery of SIGALRM and SIGCHLD. \*/

204 static void

205 signals\_disable**(**void**)**

206 **{**

207 sigset\_t sigset**;**

208

209 sigemptyset**(&**sigset**);**

210 sigaddset**(&**sigset**,** SIGALRM**);**

211 sigaddset**(&**sigset**,** SIGCHLD**);**

212 **if** **(**sigprocmask**(**SIG\_BLOCK**,** **&**sigset**,** **NULL)** **<** 0**)** **{**

213 perror**(**"signals\_disable: sigprocmask"**);**

214 exit**(**1**);**

215 **}**

216 **}**

217

218 /\* Enable delivery of SIGALRM and SIGCHLD. \*/

219 static void

220 signals\_enable**(**void**)**

221 **{**

222 sigset\_t sigset**;**

223

224 sigemptyset**(&**sigset**);**

225 sigaddset**(&**sigset**,** SIGALRM**);**

226 sigaddset**(&**sigset**,** SIGCHLD**);**

227 **if** **(**sigprocmask**(**SIG\_UNBLOCK**,** **&**sigset**,** **NULL)** **<** 0**)** **{**

228 perror**(**"signals\_enable: sigprocmask"**);**

229 exit**(**1**);**

230 **}**

231 **}**

232

233

234 /\* Install two signal handlers.

235 \* One for SIGCHLD, one for SIGALRM.

236 \* Make sure both signals are masked when one of them is running.

237 \*/

238 static void

239 install\_signal\_handlers**(**void**)**

240 **{**

241 sigset\_t sigset**;**

242 struct sigaction sa**;**

243

244 sa**.**sa\_handler **=** sigchld\_handler**;**

245 sa**.**sa\_flags **=** SA\_RESTART**;**

246 sigemptyset**(&**sigset**);**

247 sigaddset**(&**sigset**,** SIGCHLD**);**

248 sigaddset**(&**sigset**,** SIGALRM**);**

249 sa**.**sa\_mask **=** sigset**;**

250 **if** **(**sigaction**(**SIGCHLD**,** **&**sa**,** **NULL)** **<** 0**)** **{**

251 perror**(**"sigaction: sigchld"**);**

252 exit**(**1**);**

253 **}**

254

255 sa**.**sa\_handler **=** sigalrm\_handler**;**

256 **if** **(**sigaction**(**SIGALRM**,** **&**sa**,** **NULL)** **<** 0**)** **{**

257 perror**(**"sigaction: sigalrm"**);**

258 exit**(**1**);**

259 **}**

260

261 /\*

262 \* Ignore SIGPIPE, so that write()s to pipes

263 \* with no reader do not result in us being killed,

264 \* and write() returns EPIPE instead.

265 \*/

266 **if** **(**signal**(**SIGPIPE**,** SIG\_IGN**)** **<** 0**)** **{**

267 perror**(**"signal: sigpipe"**);**

268 exit**(**1**);**

269 **}**

270 **}**

271

272 static void

273 do\_shell**(**char **\***executable**,** int wfd**,** int rfd**)**

274 **{**

275 char arg1**[**10**],** arg2**[**10**];**

276 char **\***newargv**[]** **=** **{** executable**,** **NULL,** **NULL,** **NULL** **};**

277 char **\***newenviron**[]** **=** **{** **NULL** **};**

278

279 sprintf**(**arg1**,** "%05d"**,** wfd**);**

280 sprintf**(**arg2**,** "%05d"**,** rfd**);**

281 newargv**[**1**]** **=** arg1**;**

282 newargv**[**2**]** **=** arg2**;**

283

284 raise**(**SIGSTOP**);**

285 execve**(**executable**,** newargv**,** newenviron**);**

286

287 /\* execve() only returns on error \*/

288 perror**(**"scheduler: child: execve"**);**

289 exit**(**1**);**

290 **}**

291

292 /\* Create a new shell task.

293 \*

294 \* The shell gets special treatment:

295 \* two pipes are created for communication and passed

296 \* as command-line arguments to the executable.

297 \*/

298 static void

299 sched\_create\_shell**(**char **\***executable**,** int **\***request\_fd**,** int **\***return\_fd**)**

300 **{**

301 pid\_t p**;**

302 int pfds\_rq**[**2**],** pfds\_ret**[**2**];**

303

304 **if** **(**pipe**(**pfds\_rq**)** **<** 0 **||** pipe**(**pfds\_ret**)** **<** 0**)** **{**

305 perror**(**"pipe"**);**

306 exit**(**1**);**

307 **}**

308

309 p **=** fork**();**

310 **if** **(**p **<** 0**)** **{**

311 perror**(**"scheduler: fork"**);**

312 exit**(**1**);**

313 **}**

314

315 **if** **(**p **==** 0**)** **{**

316 /\* Child \*/

317 close**(**pfds\_rq**[**0**]);**

318 close**(**pfds\_ret**[**1**]);**

319 do\_shell**(**executable**,** pfds\_rq**[**1**],** pfds\_ret**[**0**]);**

320 assert**(**0**);**

321 **}**

322 /\* Parent \*/

323 shellpid **=** p**;**

324 close**(**pfds\_rq**[**1**]);**

325 close**(**pfds\_ret**[**0**]);**

326 **\***request\_fd **=** pfds\_rq**[**0**];**

327 **\***return\_fd **=** pfds\_ret**[**1**];**

328 **}**

329

330 static void

331 shell\_request\_loop**(**int request\_fd**,** int return\_fd**)**

332 **{**

333 int ret**;**

334 struct request\_struct rq**;**

335

336 /\*

337 \* Keep receiving requests from the shell.

338 \*/

339 **for** **(;;)** **{**

340 **if** **(**read**(**request\_fd**,** **&**rq**,** **sizeof(**rq**))** **!=** **sizeof(**rq**))** **{**

341 perror**(**"scheduler: read from shell"**);**

342 fprintf**(**stderr**,** "Scheduler: giving up on shell request processing.\n"**);**

343 **break;**

344 **}**

345

346 signals\_disable**();**

347 ret **=** process\_request**(&**rq**);**

348 signals\_enable**();**

349

350 **if** **(**write**(**return\_fd**,** **&**ret**,** **sizeof(**ret**))** **!=** **sizeof(**ret**))** **{**

351 perror**(**"scheduler: write to shell"**);**

352 fprintf**(**stderr**,** "Scheduler: giving up on shell request processing.\n"**);**

353 **break;**

354 **}**

355 **}**

356 **}**

357

358

359

360

361

362 int main**(**int argc**,** char **\***argv**[])**

363 **{**

364 int nproc**;**

365 /\* Two file descriptors for communication with the shell \*/

366 static int request\_fd**,** return\_fd**;**

367

368 /\* Create the shell. \*/

369 sched\_create\_shell**(**SHELL\_EXECUTABLE\_NAME**,** **&**request\_fd**,** **&**return\_fd**);**

370 /\* TODO: add the shell to the scheduler's tasks \*/

371

372 /\*

373 \* For each of argv[1] to argv[argc - 1],

374 \* create a new child process, add it to the process list.

375 \*/

376 head **=** **NULL;**

377 nproc **=** argc**-**1**;** /\* number of proccesses goes here \*/

378

379 int i**;**

380 **for(** i**=**0**;** i **<** nproc**;** i**++)**

381 **{**

382 pid\_t pidc**;**

383

384

385 pidc **=** fork**();**

386 **if** **(**pidc **<** 0**)** **{** // error

387 perror**(**"parent: fork"**);**

388 exit**(**1**);**

389 **}**

390 **if** **(**pidc **==** 0**)** **{** // child

391 child**(**argv**[**i**+**1**]);**

392 exit**(**1**);**

393 **}**

394

395 **if** **(**head **==** **NULL){**

396 head **=** malloc**(sizeof(**pcb\_t**));**

397 **if** **(**head **==** **NULL)** **return(**1**);**

398

399 head**->**id**=**i**;**

400 head**->**pid**=**pidc**;**

401 strcpy**(**head**->**name**,**argv**[**i**+**1**]);**

402 head**->**next**=NULL;**

403 current **=** head**;**

404 **}**

405 **else** **{**

406 current**->**next **=** malloc**(sizeof(**pcb\_t**));**

407 current **=** current**->**next**;**

408 current**->**id**=**i**;**

409 current**->**pid**=**pidc**;**

410 strcpy**(**current**->**name**,**argv**[**i**+**1**]);**

411 **}**

412 **}**

413 idc **=** nproc**;**

414 current**->**next **=** malloc**(sizeof(**pcb\_t**));**

415 current **=** current**->**next**;**

416 current**->**id**=**idc**;**

417 current**->**pid**=**shellpid**;**

418 strcpy**(**current**->**name**,**SHELL\_EXECUTABLE\_NAME**);**

419 current**->**next **=** head**;**

420

421

422 /\* Wait for all children to raise SIGSTOP before exec()ing. \*/

423 wait\_for\_ready\_children**(**nproc**+**1**);**

424

425 /\* Install SIGALRM and SIGCHLD handlers. \*/

426 install\_signal\_handlers**();**

427

428 **if** **(**nproc **==** 0**)** **{**

429 fprintf**(**stderr**,** "Scheduler: No tasks. Exiting...\n"**);**

430 exit**(**1**);**

431 **}**

432

433 previous **=** current**;**

434 current **=** head**;**

435 process **=** nproc **+** 1**;**

436 kill**(**current**->**pid**,**SIGCONT**);**

437 alarm**(**SCHED\_TQ\_SEC**);**

438

439 shell\_request\_loop**(**request\_fd**,** return\_fd**);**

440

441 /\* Now that the shell is gone, just loop forever

442 \* until we exit from inside a signal handler.

443 \*/

444 **while** **(**pause**());**

445

446 /\* Unreachable \*/

447 fprintf**(**stderr**,** "Internal error: Reached unreachable point\n"**);**

448 **return** 1**;**

449 **}**

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου (συνεχίζεται επ 'αόριστον):

oslaba05@os-node1:~/lab4$ ./ask2 prog prog prog

My PID = 32700: Child PID = 32701 has been stopped by a signal, signo = 19

My PID = 32700: Child PID = 32702 has been stopped by a signal, signo = 19

My PID = 32700: Child PID = 32703 has been stopped by a signal, signo = 19

My PID = 32700: Child PID = 32704 has been stopped by a signal, signo = 19

prog: Starting, NMSG = 200, delay = 57

prog[32702]: This is message 0

prog[32702]: This is message 1

prog[32702]: This is message 2

pprog[32702]: This is message 3

prog[32702]: This is message 4

prog[32702]: This is message 5

prog[32702]: This is message 6

prog[32702]: This is message 7

prog[32702]: This is message 8

prog[32702]: This is message 9

prog[32702]: This is message 10

prog[32702]: This is message 11

Stopping procress with ID = 0

Stopped procress with ID = 0

Starting process with ID = 1.

prog: Starting, NMSG = 200, delay = 39

prog[32703]: This is message 0

prog[32703]: This is message 1

prog[32703]: This is message 2

k prog[32703]: This is message 3

prog[32703]: This is message 4

prog[32703]: This is message 5

2

prog[32703]: This is message 6

prog[32703]: This is message 7

prog[32703]: This is message 8

prog[32703]: This is message 9

prog[32703]: This is message 10

prog[32703]: This is message 11

prog[32703]: This is message 12

prog[32703]: This is message 13

prog[32703]: This is message 14

prog[32703]: This is message 15

prog[32703]: This is message 16

prog[32703]: This is message 17

Stopping procress with ID = 1

Stopped procress with ID = 1

Starting process with ID = 2.

prog: Starting, NMSG = 200, delay = 150

prog[32704]: This is message 0

prog[32704]: This is message 1

prog[32704]: This is message 2

prog[32704]: This is message 3

prog[32704]: This is message 4

Stopping procress with ID = 2

Stopped procress with ID = 2

Starting process with ID = 3.

This is the Shell. Welcome.

Shell> Shell: issuing request...

Shell: receiving request return value...

RUNNING PROCESS : name: shell ,pid: 32701 ,id: 3

name: prog ,pid: 32702 ,id: 0

name: prog ,pid: 32703 ,id: 1

name: prog ,pid: 32704 ,id: 2

Shell> Shell: issuing request...

Shell: receiving request return value...

Killing process with id : 2

Shell> e prog

Shell: issuing request...

Shell: receiving request return value...

Added prog to the processes list.

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

Starting process with ID = 0.

prog[32702]: This is message 12

prog[32702]: This is message 13

prog[32702]: This is message 14

prog[32702]: This is message 15

prog[32702]: This is message 16

prog[32702]: This is message 17

prog[32702]: This is message 18

prog[32702]: This is message 19

prog[32702]: This is message 20

prog[32702]: This is message 21

pprog[32702]: This is message 22

prog[32702]: This is message 23

Stopping procress with ID = 0

Stopped procress with ID = 0

Starting process with ID = 1.

prog[32703]: This is message 18

prog[32703]: This is message 19

prog[32703]: This is message 20

prog[32703]: This is message 21

prog[32703]: This is message 22

prog[32703]: This is message 23

prog[32703]: This is message 24

prog[32703]: This is message 25

prog[32703]: This is message 26

prog[32703]: This is message 27

prog[32703]: This is message 28

prog[32703]: This is message 29

prog[32703]: This is message 30

prog[32703]: This is message 31

prog[32703]: This is message 32

prog[32703]: This is message 33

prog[32703]: This is message 34

Stopping procress with ID = 1

Stopped procress with ID = 1

Starting process with ID = 2.

Died process with ID = 2.

Starting process with ID = 4.

prog: Starting, NMSG = 200, delay = 67

prog[32705]: This is message 0

prog[32705]: This is message 1

prog[32705]: This is message 2

prog[32705]: This is message 3

prog[32705]: This is message 4

prog[32705]: This is message 5

prog[32705]: This is message 6

prog[32705]: This is message 7

prog[32705]: This is message 8

prog[32705]: This is message 9

Stopping procress with ID = 4

Stopped procress with ID = 4

Starting process with ID = 3.

Shell: issuing request...

Shell: receiving request return value...

RUNNING PROCESS : name: shell ,pid: 32701 ,id: 3

name: prog ,pid: 32702 ,id: 0

name: prog ,pid: 32703 ,id: 1

name: prog ,pid: 32705 ,id: 4

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

Starting process with ID = 0.

prog[32702]: This is message 24

prog[32702]: This is message 25

prog[32702]: This is message 26

prog[32702]: This is message 27

prog[32702]: This is message 28

prog[32702]: This is message 29

prog[32702]: This is message 30

prog[32702]: This is message 31

prog[32702]: This is message 32

prog[32702]: This is message 33

prog[32702]: This is message 34

prog[32702]: This is message 35

Stopping procress with ID = 0

Stopped procress with ID = 0

Starting process with ID = 1.

prog[32703]: This is message 35

prog[32703]: This is message 36

prog[32703]: This is message 37

prog[32703]: This is message 38

prog[32703]: This is message 39

prog[32703]: This is message 40

prog[32703]: This is message 41

prog[32703]: This is message 42

prog[32703]: This is message 43

prog[32703]: This is message 44

prog[32703]: This is message 45

prog[32703]: This is message 46

prog[32703]: This is message 47

prog[32703]: This is message 48

prog[32703]: This is message 49

prog[32703]: This is message 50

prog[32703]: This is message 51

Stopping procress with ID = 1

Stopped procress with ID = 1

Starting process with ID = 4.

prog[32705]: This is message 10

prog[32705]: This is message 11

**Απαντήσεις**:

1. Ως τρέχουσα διεργασία θα εμφανίζεται πάντα ο φλοιός. Ο χρήστης έχει επαφή μόνο με τον φλοιό άρα κατά την εκτέλεση της εντολής ‘p’, η διεργασία που θα την εκτελέσει είναι ο φλοιός άρα αυτή θα είναι η τρέχουσα. Βάση του κώδικα μας δεν θα μπορούσε άλλη διεργασία να εμφανίζεται ως τρέχουσα αφού καμία άλλη διεργασία δεν αλληλεπιδρά με τον χρήστη. Έτσι δεν θα μπορούσε να διαβάσει την εντολή που έδωσε και δεν θα μπορούσε να την εκτελέσει. Όταν διαγραφεί φλοιός τότε όλες οι διεργασίες θα εκτελούνταν με την σειρά που είχαν πριν τη διαγραφή του φλοιού και ο χρήστης δεν θα μπορεί να επέμβει.

2. Όταν δίνεται μία εντολή στο φλοιό, υπάρχει τροποποίηση στη λίστα έτσι ώστε να αλλάζει η σειρά εκτέλεσης των διεργασιών. Κατά την τροποποίηση αυτή πρέπει να εξασφαλίσουμε ότι δεν θα γίνει κάποια διακοπή ώστε να μην έχουμε λάθος αποτελέσματα. Δηλαδή η εκτέλεση των εντολών πρέπει να γίνεται χωρίς διακοπές. Αν δεν είχαμε κάνει αυτό τότε θα είχαμε πρόβλημα αν όταν κατά την εκτέλεση μίας εντολής ο scheduler διακοπτόταν την στιγμή που ανέθετε ποια θα είναι η επόμενη. Έτσι θα είχαμε λάθος στην σειρά εκτέλεσης διεργασιών και ίσως κάποιες διεργασίες να μην εκτελούνταν ποτέ.

**1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή**

1 #include **<**errno**.**h**>**

2 #include **<**unistd**.**h**>**

3 #include **<**stdlib**.**h**>**

4 #include **<**stdio**.**h**>**

5 #include **<**signal**.**h**>**

6 #include **<**string**.**h**>**

7 #include **<**assert**.**h**>**

8

9 #include **<**sys**/**wait**.**h**>**

10 #include **<**sys**/**types**.**h**>**

11

12 #include "proc-common.h"

13 #include "request.h"

14

15 /\* Compile-time parameters. \*/

16 #define SCHED\_TQ\_SEC 2 /\* time quantum \*/

17 #define TASK\_NAME\_SZ 60 /\* maximum size for a task's name \*/

18 #define SHELL\_EXECUTABLE\_NAME "shell" /\* executable for shell \*/

19

20 **typedef** struct PCBstruct **{**

21 int id**;**

22 int pid**;**

23 char name**[**TASK\_NAME\_SZ**];**

24 int prio**;**

25 struct PCBstruct **\***next**;**

26 **}** pcb\_t**;**

27 int help **=**0**;**

28 int processl**,**shellpid**,**idc**,**processh**;**

29 pcb\_t **\***current**,\***previous**,\***headh**,\***start**,**headl**,\***head**;**

30

31

32

33

34 void child**(**char **\***process\_name**)** **{**

35 char **\***newargv**[]** **=** **{**process\_name**,** **NULL,** **NULL,** **NULL** **};**

36 char **\***newenviron**[]** **=** **{** **NULL** **};**

37

38 raise**(**SIGSTOP**);** // arxika tis stamatame oles

39

40

41 // replace myself with the process\_name

42 execve**(**process\_name**,** newargv**,** newenviron**);**

43

44 /\* execve() only returns on error \*/

45 perror**(**"execve"**);**

46 exit**(**1**);**

47 **}**

48

49

50 /\* Print a list of all tasks currently being scheduled. \*/

51

52 static void

53 sched\_print\_tasks**(**void**)**

54 **{**

55 pcb\_t **\***i **=** current**;**

56 printf**(**"\nRUNNING PROCESS : name: %s ,pid: %d ,id: %d, prio: %d\n"**,** i**->**name**,** i**->**pid**,** i**->**id**,**i**->**prio**);**

57 **while** **(**i**->**next **!=** current**)** **{**

58 i **=** i**->** next**;**

59 printf**(**"name: %s ,pid: %d ,id: %d, prio: %d\n"**,** i**->**name**,** i**->**pid**,** i**->**id**,**i**->**prio**);**

60 **}**

61 printf**(**"\n"**);**

62 **}**

63

64 /\* Send SIGKILL to a task determined by the value of its

65 \* scheduler-specific id.

66 \*/

67 static void

68 sched\_kill\_task\_by\_id**(**int id**)**

69 **{**

70 pcb\_t **\***i **=** head**;**

71 **if** **(**head**->**id **==** id**)**

72 **{**

73 printf**(**"Killing process with id : %d\n"**,**i**->**id**);**

74 kill**(** i**->**pid**,** SIGTERM**);**

75 **}**

76 **else** **{**

77 **while** **(**i**->**next **!=** head**){**

78 i **=** i**->** next**;**

79 **if** **(**i**->**id **==** id**){**

80 printf**(**"Killing process with id : %d\n"**,**i**->**id**);**

81 kill**(** i**->**pid**,** SIGTERM**);**

82 **return;**

83 **}**

84 **}**

85 printf**(**"Not found process with id: %d\n"**,**id**);**

86

87 **}**

88 **}**

89

90 /\* Create a new task. \*/

91 void print **(){**

92 pcb\_t **\***i **=** head**;**

93 printf**(**"%d -> %d"**,**i**->**id**,**i**->**next**->**id**);**

94 i **=** i**->**next**;**

95 **while** **(**i **!=** head**)** **{**printf**(**"%d -> %d "**,**i**->**id**,**i**->**next**->**id**);** i **=** i**->**next**;}**

96 printf**(**"\n"**);**

97 **}**

98

99 static void

100 sched\_create\_task**(**char **\***executable**)**

101 **{**

102 pid\_t p**;**

103 pcb\_t **\***i**;**

104

105 p **=** fork**();**

106

107 **if** **(**p **<** 0**)** **{** // error

108 perror**(**"parent: fork"**);**

109 exit**(**1**);**

110 **}**

111 help **++;**

112 **if** **(**p **==** 0**)** **{** // child

113 child**(**executable**);**

114 exit**(**1**);**

115 **}**

116

117 i **=** malloc**(sizeof(**pcb\_t**));**

118 i**->**id**=++**idc**;**

119 i**->**pid**=**p**;**

120 i**->**prio **=**0**;**

121 strcpy**(**i**->**name**,**executable**);**

122

123 **if(**processh **==** 0**){**

124 pcb\_t **\***q **=** head**;**

125 **while(**q**->**next **!=** head**)** q **=** q**->**next**;**

126 i**->**next **=** head**;**

127 head **=** i**;**

128 q**->**next **=** head**;**

129

130 **}**

131 **else** **if(**processl **==** 0**){**

132 pcb\_t **\***q **=** head**->**next**;**

133 **while** **(**q**->**next **!=** head**)** q **=** q**->**next**;**

134 q**->**next **=** i**;**

135 i**->** next **=** head**;**

136 **}**

137 **else{**

138 pcb\_t **\***q **=** head**->**next**,\***j**=** head**;**

139 **while** **(**q**->**prio **==** 1**)** **{**j **=** q**;** q **=** q**->**next**;}**

140 j**->**next **=** i**;**

141 i**->**next **=** q**;**

142 processl**++;**

143 **}**

144

145 printf**(**"Added %s to the processes list.\n\n"**,** executable**);**

146 **}**

147

148 static void

149 sched\_low**(**int id**)**

150 **{**

151 pcb\_t **\***curr **=** head**,\***prev **=** **NULL;**

152

153 **if(**head **==** **NULL)**

154 **return;**

155

156 **while** **(**curr**->**id **!=** id**)**

157 **{**

158 **if** **(**curr**->**next **==** head**)**

159 **{**

160 printf**(**"\nGiven node is not found"

161 " in the list!!!"**);**

162 **break;**

163 **}**

164 prev **=** curr**;**

165 curr **=** curr**->**next**;**

166 **}**

167 **if(**curr**->**prio **==** 0**){** printf**(**"\nGiven node is already LOW prio\n"**);** **return;** **}**

169 **if** **(**curr **==** head**)** **{**

170 **if** **(**processh **==** 1**){**

171 curr**->**prio **=** 0**;**

172 processh**--;**

173 processl**++;**

174 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

175 **}**

176 **else** **if** **(**processl **!=** 0**){**

177 pcb\_t **\***q **=** head**->**next**,\***i**=** head**,\***z**=**head**;**

178 **while** **(**q**->**prio **==** 1**)** **{**i **=** q**;** q **=** q**->**next**;}**

179 **while(**z**->**next **!=** head**)**z **=** z**->**next**;**

180 i**->** next **=** curr**;**

181 head **=** curr**->**next**;**

182 curr**->**next **=** q**;**

183 curr**->**prio **=** 0**;**

184 z**->**next **=** head**;**

185

186 processh**--;**

187 processl**++;**

188 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

189 **}**

190 **else{**

191 pcb\_t **\***q **=** head**;**

192 **while** **(**q**->**next **!=** head**)** q **=** q**->**next**;**

193 q**->**next **=** curr**;**

194 head **=** curr**->**next**;**

195 curr**->**next **=** head**;**

196 curr**->**prio **=** 0**;**

197 processh**--;**

198 processl**++;**

199 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

200 **}**

201 **}**

202 **else** **if** **(**curr**->**next **==** head**)**

203 **{**

204 curr**->**prio **=** 0**;**

205 processh**--;**

206 processl**++;**

207 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

208 **}**

209 **else{**

210 **if(**processl **!=** 0**){**

211 pcb\_t **\***q **=** head**->**next**,\***i**=** head**;**

212 **while** **(**q**->**prio **==** 1**)** **{**i **=** q**;** q **=** q**->**next**;}**

213 prev**->**next **=** curr**->**next**;**

214 i**->** next **=** curr**;**

215 curr**->**next **=** q**;**

216 curr**->**prio **=** 0**;**

217 processh**--;**

218 processl**++;**

219 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

220 **}**

221 **else{**

222 pcb\_t **\***q **=** head**;**

223 **while** **(**q**->**next **!=** head**)** q **=** q**->**next**;**

224 q**->**next **=** curr**;**

225 prev**->**next **=** curr**->**next**;**

226 curr**->**next **=** head**;**

227 curr**->**prio **=** 0**;**

228 processh**--;**

229 processl**++;**

230 printf**(**"Process with ID %d to LOW priority.\n\n"**,** curr**->**id**);**

231 **}**

232

233 **}**

234 // print();

235 **}**

236

237 static void

238 sched\_high**(**int id**)**

239 **{**

240 pcb\_t **\***curr **=** head**,** **\***prev **=** **NULL;**

241

242 **if(**head **==** **NULL)**

243 **return;**

244

245 **while** **(**curr**->**id **!=** id**)**

246 **{**

247 **if** **(**curr**->**next **==** head**)**

248 **{**

249 printf**(**"\nGiven node is not found"

250 " in the list!!!"**);**

251 **return;**

252 **}**

253 prev **=** curr**;**

254 curr **=** curr**->**next**;**

255 **}**

256 **if(**curr**->**prio **==** 1**)** **{** printf**(**"\nGiven node is already HIGH prio\n"**);** **return;** **}**

257

258 **if** **(**curr **==** head**)** **{**

259 curr**->**prio **=**1**;**

260 processh**++;**

261 processl**--;**

262 printf**(**"Process with ID %d to HIGH priority.\n\n"**,** curr**->**id**);**

263 **}**

264 **else** **if** **(**curr**->**next **==** head**)**

265 **{**

266 curr**->**prio **=**1**;**

267 processh**++;**

268 processl**--;**

269 printf**(**"Process with ID %d to HIGH priority.\n\n"**,** curr**->**id**);**

270 head **=** curr**;**

271 prev**->**next **=** head**;**

272 **}**

273 **else{**

274 printf**(**"Process with ID %d to HIGH priority.\n\n"**,** curr**->**id**);**

275 curr**->**prio **=**1**;**

276 processh**++;**

277 processl**--;**

278 prev**->**next **=** curr**->**next**;**

279 prev **=** head**;**

280 **while** **(**prev**->**next **!=** head**)** prev **=** prev**->**next**;**

281 curr**->**next **=** head**;**

282 prev**->**next **=** curr**;**

283 head **=** curr**;**

284

285 **}**

286 // print();

287 **}**

288

289

290

291

292 /\* Process requests by the shell. \*/

293 static int

294 process\_request**(**struct request\_struct **\***rq**)**

295 **{**

296 **switch** **(**rq**->**request\_no**)** **{**

297 **case** REQ\_PRINT\_TASKS**:**

298 sched\_print\_tasks**();**

299 **return** 0**;**

300

301 **case** REQ\_KILL\_TASK**:**

302 sched\_kill\_task\_by\_id**(**rq**->**task\_arg**);**

303 **return** 0**;**

304

305 **case** REQ\_EXEC\_TASK**:**

306 sched\_create\_task**(**rq**->**exec\_task\_arg**);**

307 **return** 0**;**

308

309 **case** REQ\_HIGH\_TASK**:**

310 sched\_high**(**rq**->**task\_arg**);**

311 **return** 0**;**

312

313 **case** REQ\_LOW\_TASK**:**

314 sched\_low**(**rq**->**task\_arg**);**

315 **return** 0**;**

316

317 **default:**

318 **return** **-**ENOSYS**;**

319 **}**

320 **}**

321

322 /\*

323 \* SIGALRM handler

324 \*/

325 static void

326 sigalrm\_handler**(**int signum**)**

327 **{**

328 **if** **(**kill**(**current**->**pid**,** SIGSTOP**)** **<** 0**)** **{**

329 perror**(**"SIGSTOP"**);** exit**(**1**);**

330 **}**

331 printf**(**"Stopping procress with ID = %d \n"**,**current**->**id**);**

332 **}**

333

334 /\*

335 \* SIGCHLD handler

336 \*/

337 static void

338 sigchld\_handler**(**int signum**)**

339 **{**

340 **for** **(;;)** **{**

341 int p**,**status**;**

342 p **=** waitpid**(-**1**,** **&**status**,** WUNTRACED **|** WNOHANG**);**

343 **if** **(**p **<** 0**)** **{**

344 perror**(**"waitpid"**);**

345 exit**(**1**);**

346 **}**

347 **if** **(**p **==** 0**)** **break;**

348

349 // explain\_wait\_status(p, status);

350 **if(**help **>** 0**)** **{**help**--;** **return;}**

351 **if** **(**WIFEXITED**(**status**)** **||** WIFSIGNALED**(**status**)){** /\* A child has died \*/

352 alarm**(**0**);** // midenizoume ton xrono tou alarm

353 printf**(**"Died process with ID = %d.\n"**,**current**->**id**);**

354 **if(**current **==** head**){**

355 pcb\_t **\***q **=** head**;**

356 **while(**q**->**next **!=** head**)** q**=** q**->**next**;**

357 head **=** current**->**next**;**

358 q**->**next **=** head**;**

359 **}**

360 **else** previous**->**next**=**current**->**next**;** // delete current node

361 **if(**current**->**prio **==** 1**)** processh **--;**

362 **else** processl **--;**

363

364

365 **if** **(**current**->**next**->**prio **==** 1**){**

366 free**(**current**);**

367 current **=** current **->** next**;**

368 kill**(**current**->**pid**,**SIGCONT**);**

369 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

370 alarm**(**SCHED\_TQ\_SEC**);**

371 // if(processh !=1) alarm(SCHED\_TQ\_SEC); // unless a single process is left

372 **}**

373 **else** **if** **((**processh **!=** 0**)** **&&** **(**previous**->**next**->**prio **==** 0**)){**

374 free**(**current**);**

375 current **=** head**;**

376 kill**(**current**->**pid**,**SIGCONT**);**

377 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

378 alarm**(**SCHED\_TQ\_SEC**);**

379 // if(processh !=1) alarm(SCHED\_TQ\_SEC);

380 **}**

381 **else** **if** **(**processl **!=** 0**){**

382 free**(**current**);**

383 current **=** current **->** next**;**

384 kill**(**current**->**pid**,**SIGCONT**);**

385 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

386 alarm**(**SCHED\_TQ\_SEC**);**

387 // if(processl !=1) alarm(SCHED\_TQ\_SEC); // unless a single process is left

388 **}**

389

390 **else{** // case : no more processes left to execute

391 printf**(**"No more processes left.\n"**);**

392 exit**(**0**);**

393 **}**

394 **}**

395

396 **if** **(**WIFSTOPPED**(**status**))** **{** /\* A child has stopped due to SIGSTOP/SIGTSTP, etc \*/

397 **if(**processh **==** 0**){**

398 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

399 previous **=** current**;** // the next one

400 current **=** current**->**next**;**

401 kill**(**current**->**pid**,**SIGCONT**);**

402 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

403 alarm**(**SCHED\_TQ\_SEC**);**

404 **}**

405 **else** **{**

406 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

407 **if** **(**current**->**next**->**prio **==** 1**){**

408 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

409 previous **=** current**;** // the next one

410 current **=** current**->**next**;**

411 kill**(**current**->**pid**,**SIGCONT**);**

412 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

413 alarm**(**SCHED\_TQ\_SEC**);**

414 **}**

415 **else{**

416 printf**(**"Stopped procress with ID = %d \n"**,**current**->**id**);**

417 previous **=** current**;** // the next one

418 current **=** head**;**

419 kill**(**current**->**pid**,**SIGCONT**);**

420 printf**(**"Starting process with ID = %d.\n"**,**current**->**id**);**

421 alarm**(**SCHED\_TQ\_SEC**);**

422 **}**

423 **}**

424

425 **}**

426 **}**

427 **}**

428

429

430

431

432 /\* Disable delivery of SIGALRM and SIGCHLD. \*/

433 static void

434 signals\_disable**(**void**)**

435 **{**

436 sigset\_t sigset**;**

437

438 sigemptyset**(&**sigset**);**

439 sigaddset**(&**sigset**,** SIGALRM**);**

440 sigaddset**(&**sigset**,** SIGCHLD**);**

441 **if** **(**sigprocmask**(**SIG\_BLOCK**,** **&**sigset**,** **NULL)** **<** 0**)** **{**

442 perror**(**"signals\_disable: sigprocmask"**);**

443 exit**(**1**);**

444 **}**

445 **}**

446

447 /\* Enable delivery of SIGALRM and SIGCHLD. \*/

448 static void

449 signals\_enable**(**void**)**

450 **{**

451 sigset\_t sigset**;**

452

453 sigemptyset**(&**sigset**);**

454 sigaddset**(&**sigset**,** SIGALRM**);**

455 sigaddset**(&**sigset**,** SIGCHLD**);**

456 **if** **(**sigprocmask**(**SIG\_UNBLOCK**,** **&**sigset**,** **NULL)** **<** 0**)** **{**

457 perror**(**"signals\_enable: sigprocmask"**);**

458 exit**(**1**);**

459 **}**

460 **}**

461

462

463 /\* Install two signal handlers.

464 \* One for SIGCHLD, one for SIGALRM.

465 \* Make sure both signals are masked when one of them is running.

466 \*/

467 static void

468 install\_signal\_handlers**(**void**)**

469 **{**

470 sigset\_t sigset**;**

471 struct sigaction sa**;**

472

473 sa**.**sa\_handler **=** sigchld\_handler**;**

474 sa**.**sa\_flags **=** SA\_RESTART**;**

475 sigemptyset**(&**sigset**);**

476 sigaddset**(&**sigset**,** SIGCHLD**);**

477 sigaddset**(&**sigset**,** SIGALRM**);**

478 sa**.**sa\_mask **=** sigset**;**

479 **if** **(**sigaction**(**SIGCHLD**,** **&**sa**,** **NULL)** **<** 0**)** **{**

480 perror**(**"sigaction: sigchld"**);**

481 exit**(**1**);**

482 **}**

483

484 sa**.**sa\_handler **=** sigalrm\_handler**;**

485 **if** **(**sigaction**(**SIGALRM**,** **&**sa**,** **NULL)** **<** 0**)** **{**

486 perror**(**"sigaction: sigalrm"**);**

487 exit**(**1**);**

488 **}**

489

490 /\*

491 \* Ignore SIGPIPE, so that write()s to pipes

492 \* with no reader do not result in us being killed,

493 \* and write() returns EPIPE instead.

494 \*/

495 **if** **(**signal**(**SIGPIPE**,** SIG\_IGN**)** **<** 0**)** **{**

496 perror**(**"signal: sigpipe"**);**

497 exit**(**1**);**

499 **}**

500

501

502 static void

503 do\_shell**(**char **\***executable**,** int wfd**,** int rfd**)**

504 **{**

505 char arg1**[**10**],** arg2**[**10**];**

506 char **\***newargv**[]** **=** **{** executable**,** **NULL,** **NULL,** **NULL** **};**

507 char **\***newenviron**[]** **=** **{** **NULL** **};**

508

509 sprintf**(**arg1**,** "%05d"**,** wfd**);**

510 sprintf**(**arg2**,** "%05d"**,** rfd**);**

511 newargv**[**1**]** **=** arg1**;**

512 newargv**[**2**]** **=** arg2**;**

513

514 raise**(**SIGSTOP**);**

515 execve**(**executable**,** newargv**,** newenviron**);**

516

517 /\* execve() only returns on error \*/

518 perror**(**"scheduler: child: execve"**);**

519 exit**(**1**);**

520 **}**

521

522 /\* Create a new shell task.

523 \*

524 \* The shell gets special treatment:

525 \* two pipes are created for communication and passed

526 \* as command-line arguments to the executable.

527 \*/

528 static void

529 sched\_create\_shell**(**char **\***executable**,** int **\***request\_fd**,** int **\***return\_fd**)**

530 **{**

531 pid\_t p**;**

532 int pfds\_rq**[**2**],** pfds\_ret**[**2**];**

533

534 **if** **(**pipe**(**pfds\_rq**)** **<** 0 **||** pipe**(**pfds\_ret**)** **<** 0**)** **{**

535 perror**(**"pipe"**);**

536 exit**(**1**);**

537 **}**

538

539 p **=** fork**();**

540 **if** **(**p **<** 0**)** **{**

541 perror**(**"scheduler: fork"**);**

542 exit**(**1**);**

543 **}**

544

545 **if** **(**p **==** 0**)** **{**

546 /\* Child \*/

547 close**(**pfds\_rq**[**0**]);**

548 close**(**pfds\_ret**[**1**]);**

549 do\_shell**(**executable**,** pfds\_rq**[**1**],** pfds\_ret**[**0**]);**

550 assert**(**0**);**

551 **}**

552 /\* Parent \*/

553 shellpid **=** p**;**

554 close**(**pfds\_rq**[**1**]);**

555 close**(**pfds\_ret**[**0**]);**

556 **\***request\_fd **=** pfds\_rq**[**0**];**

557 **\***return\_fd **=** pfds\_ret**[**1**];**

558 **}**

559

560 static void

561 shell\_request\_loop**(**int request\_fd**,** int return\_fd**)**

562 **{**

563 int ret**;**

564 struct request\_struct rq**;**

565

566 /\*

567 \* Keep receiving requests from the shell.

568 \*/

569 **for** **(;;)** **{**

570 **if** **(**read**(**request\_fd**,** **&**rq**,** **sizeof(**rq**))** **!=** **sizeof(**rq**))** **{**

571 perror**(**"scheduler: read from shell"**);**

572 fprintf**(**stderr**,** "Scheduler: giving up on shell request processing.\n"**);**

573 **break;**

574 **}**

575

576 signals\_disable**();**

577 ret **=** process\_request**(&**rq**);**

578 signals\_enable**();**

579

580 **if** **(**write**(**return\_fd**,** **&**ret**,** **sizeof(**ret**))** **!=** **sizeof(**ret**))** **{**

581 perror**(**"scheduler: write to shell"**);**

582 fprintf**(**stderr**,** "Scheduler: giving up on shell request processing.\n"**);**

583 **break;**

584 **}**

585 **}**

586 **}**

587

588

589

590

591

592 int main**(**int argc**,** char **\***argv**[])**

593 **{**

594 int nproc**;**

595 /\* Two file descriptors for communication with the shell \*/

596 static int request\_fd**,** return\_fd**;**

597

598 /\* Create the shell. \*/

599 sched\_create\_shell**(**SHELL\_EXECUTABLE\_NAME**,** **&**request\_fd**,** **&**return\_fd**);**

600 /\* TODO: add the shell to the scheduler's tasks \*/

601

602 /\*

603 \* For each of argv[1] to argv[argc - 1],

604 \* create a new child process, add it to the process list.

605 \*/

606 head **=** **NULL;**

607 nproc **=** argc**-**1**;** /\* number of proccesses goes here \*/

608

609 int i**;**

610 **for(** i**=**0**;** i **<** nproc**;** i**++)**

611 **{**

612 pid\_t pidc**;**

613

614

615 pidc **=** fork**();**

616 **if** **(**pidc **<** 0**)** **{** // error

617 perror**(**"parent: fork"**);**

618 exit**(**1**);**

619 **}**

620 **if** **(**pidc **==** 0**)** **{** // child

621 child**(**argv**[**i**+**1**]);**

622 exit**(**1**);**

623 **}**

624

625 **if** **(**head **==** **NULL){**

626 head **=** malloc**(sizeof(**pcb\_t**));**

627 **if** **(**head **==** **NULL)** **return(**1**);**

628

629 head**->**id**=**i**;**

630 head**->**pid**=**pidc**;**

631 head**->**prio**=**0**;**

632 strcpy**(**head**->**name**,**argv**[**i**+**1**]);**

633 head**->**next**=NULL;**

634 current **=** head**;**

635 **}**

636 **else** **{**

637 current**->**next **=** malloc**(sizeof(**pcb\_t**));**

638 current **=** current**->**next**;**

639 current**->**id**=**i**;**

640 current**->**pid**=**pidc**;**

641 current**->**prio **=** 0**;**

642 strcpy**(**current**->**name**,**argv**[**i**+**1**]);**

643 **}**

644 **}**

645 idc **=** nproc**;**

646 current**->**next **=** malloc**(sizeof(**pcb\_t**));**

647 current **=** current**->**next**;**

648 current**->**id**=**idc**;**

649 current**->**prio **=** 0**;**

650 current**->**pid**=**shellpid**;**

651 strcpy**(**current**->**name**,**SHELL\_EXECUTABLE\_NAME**);**

652 current**->**next **=** head**;**

653

654

655 /\* Wait for all children to raise SIGSTOP before exec()ing. \*/

656 wait\_for\_ready\_children**(**nproc**+**1**);**

657

658 /\* Install SIGALRM and SIGCHLD handlers. \*/

659 install\_signal\_handlers**();**

660

661 **if** **(**nproc **==** 0**)** **{**

662 fprintf**(**stderr**,** "Scheduler: No tasks. Exiting...\n"**);**

663 exit**(**1**);**

664 **}**

665

666 previous **=** current**;**

667 current **=** head**;**

668 start **=** head**;**

669 processl **=** nproc **+** 1**;**

670 kill**(**current**->**pid**,**SIGCONT**);**

671 alarm**(**SCHED\_TQ\_SEC**);**

672

673 shell\_request\_loop**(**request\_fd**,** return\_fd**);**

674

675 /\* Now that the shell is gone, just loop forever

676 \* until we exit from inside a signal handler.

677 \*/

678 **while** **(**pause**());**

679

680 /\* Unreachable \*/

681 fprintf**(**stderr**,** "Internal error: Reached unreachable point\n"**);**

682 **return** 1**;**

683 **}**

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου (συνεχίζεται επ 'αόριστον):

oslaba05@os-node1:~/lab4$ ./ask3 prog prog prog

My PID = 32766: Child PID = 32767 has been stopped by a signal, signo = 19

My PID = 32766: Child PID = 300 has been stopped by a signal, signo = 19

My PID = 32766: Child PID = 301 has been stopped by a signal, signo = 19

My PID = 32766: Child PID = 302 has been stopped by a signal, signo = 19

prog: Starting, NMSG = 200, delay = 94

prog[300]: This is message 0

prog[300]: This is message 1

prog[300]: This is message 2

h prog[300]: This is message 3

4

prog[300]: This is message 4

prog[300]: This is message 5

hprog[300]: This is message 6

prog[300]: This is message 7

Stopping procress with ID = 0

Stopped procress with ID = 0

Starting process with ID = 1.

prog: Starting, NMSG = 200, delay = 141

prog[301]: This is message 0

3

prog[301]: This is message 1

prog[301]: This is message 2

prog[301]: This is message 3

hprog[301]: This is message 4

Stopping procress with ID = 1

Stopped procress with ID = 1

Starting process with ID = 2.

prog: Starting, NMSG = 200, delay = 122

prog[302]: This is message 0

1

prog[302]: This is message 1

prog[302]: This is message 2

prog[302]: This is message 3

prog[302]: This is message 4

prog[302]: This is message 5

Stopping procress with ID = 2

Stopped procress with ID = 2

Starting process with ID = 3.

This is the Shell. Welcome.

Shell> Shell: issuing request...

Shell: receiving request return value...

Shell> Shell: issuing request...

Shell: receiving request return value...

Given node is not found in the list!!!Process with ID 3 to HIGH priority.

Shell> Shell: issuing request...

Shell: receiving request return value...

Process with ID 1 to HIGH priority.

Shell> p

Shell: issuing request...

Shell: receiving request return value...

RUNNING PROCESS : name: shell ,pid: 32767 ,id: 3, prio: 1

name: prog ,pid: 300 ,id: 0, prio: 0

name: prog ,pid: 302 ,id: 2, prio: 0

name: prog ,pid: 301 ,id: 1, prio: 1

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

Stopped procress with ID = 3

Starting process with ID = 1.

prog[301]: This is message 5

prog[301]: This is message 6

prog[301]: This is message 7

prog[301]: This is message 8

prog[301]: This is message 9

Stopping procress with ID = 1

Stopped procress with ID = 1

Stopped procress with ID = 1

Starting process with ID = 3.

l 1

Shell: issuing request...

Shell: receiving request return value...

Process with ID 1 to LOW priority.

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

Stopped procress with ID = 3

Starting process with ID = 3.

h 2

Shell: issuing request...

Shell: receiving request return value...

Process with ID 2 to HIGH priority.

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

Stopped procress with ID = 3

Starting process with ID = 2.

prog[302]: This is message 6

prog[302]: This is message 7

prog[302]: This is message 8

prog[302]: This is message 9

kprog[302]: This is message 10

Stopping procress with ID = 2

Stopped procress with ID = 2

Stopped procress with ID = 2

Starting process with ID = 3.

2

Shell: issuing request...

Shell: receiving request return value...

Killing process with id : 2

Shell> p

Shell: issuing request...

Shell: receiving request return value...

RUNNING PROCESS : name: shell ,pid: 32767 ,id: 3, prio: 1

name: prog ,pid: 301 ,id: 1, prio: 0

name: prog ,pid: 300 ,id: 0, prio: 0

name: prog ,pid: 302 ,id: 2, prio: 1

Shell> Stopping procress with ID = 3

Stopped procress with ID = 3

**Απαντήσεις**:

1. Σενάριο λιμοκτονίας θα είχαμε εάν η εκτέλεση μιας low priority διεργασίας εξαρτιόταν από μια διεργασία με high priority. Έτσι η διεργασία με το low priority δεν θα εκτελούταν ποτέ. Για να αντιμετωπίζαμε το πρόβλημα αυτό θα μπορούσαμε σε μια τέτοια περίπτωση το ζεύγος των εξαρτώμενων διεργασιών να έχει πάντα το ίδιο priority είτε το priority μιας διεργασίας να πέφτει με το πέρασμα του χρόνου για να δίνεται η ευκαιρία και σε άλλες διεργασίες να εκτελεστούν. Φαινόμενο λιμοκτονίας θα μπορούσε επίσης να υπήρχε εάν θέταμε κάθε καινούργια διεργασία σε high priority. Έτσι σε περίπτωση που υπήρχαν low priority διεργασίες δεν θα εκτελούνταν ποτέ .Τέλος πρόβλημα θα αποτελούσε αν ο shell βρισκόταν πάντα σε high priority αφού θα εκτελούνταν όλες οι high διεργασίες και μετά ο shell αγνοώντας τις low priority διεργασίες.