



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ
Λειτουργικά Συστήματα 2^η Άσκηση
Ακ. έτος 2011-2012

Τμήμα Β, Ομάδα 3^η

Γερακάρης Βασίλης Α.Μ.: 03108092
Λύρας Γρηγόρης Α.Μ.: 03109687

9 Δεκεμβρίου 2011

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Ο πηγαίος κώδικας της main.c που κληθήκαμε να γράψουμε ήταν ο εξής:

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  #include "proc-common.h"
9  #include "tree.h"
10
11 #define SLEEP_PROC_SEC  10
12 #define SLEEP_TREE_SEC  3
13
14 /*
15  * Create this process tree:
16  * A--B---D
17  *   `--C
18  */
19 void fork_procs(struct tree_node *me)
20 {
21     /*
22      * initial process is A.
23      */
24     int i;
25     pid_t pid;
26     int status;
27
28     change_pname(me->name);
29     /* loop to fork for all my children */
30
31     for (i=0;i<me->nr_children;i++)
32     {
33         pid = fork();
34         if (pid < 0) {
35             perror("main: fork");
36             exit(1);
37         }
38         if (pid == 0) {
39             /* Child */
40             me=me->children+i;
41             fork_procs(me);
42             exit(1);
43         }
44     }
45
46     if(me->nr_children==0)
47     {
48         printf("%s: Sleeping...\n",me->name);
49         sleep(SLEEP_PROC_SEC);
50     }
51
52     /* ... */
53     for(i=0;i<me->nr_children;i++)
54     {
55         pid = wait(&status);
56         explain_wait_status(pid, status);
57     }
58
59     printf("%s: Exiting...\n",me->name);
60     switch(*me->name)
61     {
62         case 'A':
63             exit(16);
64             break;
65         case 'B':
66             exit(19);
67         case 'C':
68             exit(17);
69         case 'D':
```

```

70         exit(13);
71     default:
72         exit(1);
73     }
74 }
75
76 /*
77  * The initial process forks the root of the process tree,
78  * waits for the process tree to be completely created,
79  * then takes a photo of it using show_pstree().
80  *
81  * How to wait for the process tree to be ready?
82  * In ask2-{fork, tree}:
83  *     wait for a few seconds, hope for the best.
84  * In ask2-signals:
85  *     use wait_for_ready_children() to wait until
86  *     the first process raises SIGSTOP.
87  */
88 int main(void)
89 {
90     pid_t pid;
91     int status;
92     struct tree_node * root = get_tree_from_file("init.tree");
93
94     /* Fork root of process tree */
95     pid = fork();
96     if (pid < 0) {
97         perror("main: fork");
98         exit(1);
99     }
100    if (pid == 0) {
101        /* Child */
102        fork_procs(root);
103        exit(1);
104    }
105
106    /*
107     * Father
108     */
109    /* for ask2-signals */
110    /* wait_for_ready_children(1); */
111
112    /* for ask2-{fork, tree} */
113    sleep(SLEEP_TREE_SEC);
114
115    /* Print the process tree root at pid */
116    show_pstree(pid);
117
118    /* for ask2-signals */
119    /* kill(pid, SIGCONT); */
120
121    /* Wait for the root of the process tree to terminate */
122    pid = wait(&status);
123    explain_wait_status(pid, status);
124
125    return 0;
126 }

```

Ερωτήσεις

1. Αν η διεργασία A τερματίσει πρόωρα τα παιδιά της (B,C) θα συνεχίσουν να εκτελούνται και υιοθετούνται από την INIT. Όταν αυτά τερματίσουν η INIT θα κάνει wait και θα φύγουν από τη μνήμη.
2. Φαίνεται μία ακόμη διεργασία. Αυτή είναι η διεργασία που ξεκίνησε τα forks δημιουργώντας την A.
3. Σε διαφορετική περίπτωση ένας χρήστης θα μπορούσε να εκτελέσει απεριόριστα διεργασίες. Κάτι τέτοιο δεν επιτρέπεται μιας και η ανεξέλεγκτη κατανάλωση πόρων συστήματος μπορεί να καταστήσει το σύστημά μας μη λειτουργικό, κάτι που ο διαχειριστής πρέπει να αποφύγει.

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Ο πηγαίος κώδικας της main.c που

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  #include "proc-common.h"
9  #include "tree.h"
10
11 #define SLEEP_PROC_SEC 10
12 #define SLEEP_TREE_SEC 3
13
14 /*
15  * Create this process tree:
16  * A+-B---D
17  *   |-C
18  */
19 void fork_procs(struct tree_node *me)
20 {
21     /*
22      * initial process is A.
23      */
24     int i;
25     pid_t pid;
26     int status;
27
28     change_pname(me->name);
29     /* loop to fork for all my children */
30
31     for (i=0;i<me->nr_children;i++)
32     {
33         pid = fork();
34         if (pid < 0) {
35             perror("main: fork");
36             exit(1);
37         }
38         if (pid == 0) {
39             /* Child */
40             me=me->children+i;
41             fork_procs(me);
42             exit(1);
43         }
44     }
45
46     printf("%s: Sleeping...\n",me->name);
47     sleep(SLEEP_PROC_SEC);
48
49     /* ... */
50     if (me->nr_children>0)
51     {
52         pid = wait(&status);
53         printf("%s said:\n",me->name);
54         explain_wait_status(pid, status);
55     }
56
57     printf("%s: Exiting...\n",me->name);
58     exit(16);
59 }
60
61 /*
62  * The initial process forks the root of the process tree,
63  * waits for the process tree to be completely created,
64  * then takes a photo of it using show_pstree().
65  *
66  * How to wait for the process tree to be ready?
67  * In ask2-{fork, tree}:
68  *     wait for a few seconds, hope for the best.
69  * In ask2-signals:
70  *     use wait_for_ready_children() to wait until
71  *     the first process raises SIGSTOP.
72  */
73 int main(int argc, char **argv)
74 {
75     if(argc!=2)

```

```

76     {
77         printf("Usage:%s <input.tree> \n",argv[0]);
78         exit(1);
79     }
80     pid_t pid;
81     int status;
82     struct tree_node * root = get_tree_from_file(argv[1]);
83
84     /* Fork root of process tree */
85     pid = fork();
86     if (pid < 0) {
87         perror("main: fork");
88         exit(1);
89     }
90     if (pid == 0) {
91         /* Child */
92         fork_procs(root);
93         exit(1);
94     }
95
96     /*
97      * Father
98      */
99     /* for ask2-signals */
100    /* wait_for_ready_children(1); */
101
102    /* for ask2-{fork, tree} */
103    sleep(SLEEP_TREE_SEC);
104
105    /* Print the process tree root at pid */
106    show_pstree(pid);
107
108    /* for ask2-signals */
109    /* kill(pid, SIGCONT); */
110
111    /* Wait for the root of the process tree to terminate */
112    pid = wait(&status);
113    explain_wait_status(pid, status);
114
115    return 0;
116 }

```

Ερωτήσεις

1. Οι διεργασίες γεννιούνται κατά επίπεδο. Παρόλα αυτά, δεν είναι εξασφαλισμένη η σειρά που θα εκτελεστούν οι διεργασίες. Έχει να κάνει με τη χρονοδρομολόγηση (δηλαδή το D του προηγούμενου ερωτήματος θα μπορούσε να εκτελεστεί πριν από το C αν η χρονοδρομολόγηση το ευνοούσε. Ωστόσο το κβάντο χρόνου είναι τέτοιο η εμφάνιση των μηνυμάτων δημιουργίας μοιάζει σχεδόν σειριακή.

1.3 Αποστολή και χειρισμός σημάτων

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  #include "proc-common.h"
9  #include "tree.h"
10
11 #define SLEEP_PROC_SEC  10
12 #define SLEEP_TREE_SEC  3
13
14 /*
15  * Create this process tree:
16  * A--+B---D
17  *   `--C
18  */
19 void fork_procs(struct tree_node *me)
20 {
21     /*
22      * initial process is A.

```

```

23     */
24     int i;
25     int status;
26     pid_t pid;
27     pid_t *children_pids;
28     children_pids=(pid_t *)calloc(me->nr_children,sizeof(pid_t));
29
30     change_pname(me->name);
31     /* loop to fork for all my children */
32
33     for (i=0;i<me->nr_children;i++)
34     {
35         pid = fork();
36         if (pid < 0) {
37             perror("fork_procs: fork");
38             exit(1);
39         }
40         if (pid == 0) {
41             /* Child */
42             me = me->children+i;
43             fork_procs(me);
44             exit(1);
45         }
46         *(children_pids+i)=pid;
47
48     }
49     wait_for_ready_children(me->nr_children);
50     printf("%s: pausing...\n",me->name);
51     raise(SIGSTOP);
52     for (i=0;i<(me->nr_children);i++)
53     {
54         pid = *(children_pids+i);
55         kill(pid,SIGCONT);
56         waitpid(pid,&status,WUNTRACED);
57         explain_wait_status(pid,status);
58     }
59
60     /* ... */
61     //if (me->nr_children>0)
62     //{
63     //    pid = wait(&status);
64     //    printf("%s said:\n",me->name);
65     //    explain_wait_status(pid, status);
66     //}
67
68     printf("%s: Exiting...\n",me->name);
69     exit(0);
70 }
71
72 /*
73  * The initial process forks the root of the process tree,
74  * waits for the process tree to be completely created,
75  * then takes a photo of it using show_pstree().
76  *
77  * How to wait for the process tree to be ready?
78  * In ask2-{fork, tree}:
79  *     wait for a few seconds, hope for the best.
80  * In ask2-signals:
81  *     use wait_for_ready_children() to wait until
82  *     the first process raises SIGSTOP.
83  */
84 int main(int argc,char **argv)
85 {
86     if(argc!=2)
87     {
88         printf("Usage:%s <input.tree> \n",argv[0]);
89         exit(1);
90     }
91     pid_t pid;
92     int status;
93     struct tree_node * root = get_tree_from_file(argv[1]);
94
95     /* Fork root of process tree */
96     pid = fork();
97     if (pid < 0) {

```

```

98     perror("main: fork");
99     exit(1);
100 }
101 if (pid == 0) {
102     /* Child */
103     fork_procs(root);
104     exit(1);
105 }
106
107 /*
108  * Father
109  */
110 /* for ask2-signals */
111 /* wait_for_ready_children(1); */
112
113 /* for ask2-{fork, tree} */
114 wait_for_ready_children(1);
115 show_pstree(pid);
116 kill(pid, SIGCONT);
117 waitpid(pid, &status, WUNTRACED);
118 explain_wait_status(pid, status);
119
120 /* Print the process tree root at pid */
121
122 /* for ask2-signals */
123 /* kill(pid, SIGCONT); */
124
125 /* Wait for the root of the process tree to terminate */
126 //pid = wait(&status);
127 //explain_wait_status(pid, status);
128
129 return 0;
130 }

```

Ερωτήσεις

1. Είναι σαφώς καλύτερη και πιο αξιόπιστη καθώς δεν στηρίζεται στη χρονοδρομολόγηση των διεργασιών.
2. Προτού στείλουμε σήματα στα παιδιά πρέπει να έχουμε βεβαιωθεί πως αυτά είναι σε ready state ώστε να δεχτούν σήμα. Αν το κάνουμε νωρίτερα αυτό θα αγνοηθεί. Συνεπώς περιμένουμε μέχρι όλα τα παιδιά να κάνουν raise(SIGSTOP) ώστε να δεχτούν το SIGCONT.

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <assert.h>
5  #include <sys/types.h>
6  #include <sys/wait.h>
7
8  #include "proc-common.h"
9  #include "tree.h"
10
11 #define SLEEP_PROC_SEC 10
12 #define SLEEP_TREE_SEC 3
13
14 /*
15  * Create this process tree:
16  * A-+-B---D
17  *   `--C
18  */
19 void fork_procs(struct tree_node *me)
20 {
21     /*
22      * initial process is A.
23      */
24     int i;
25     int status;
26     pid_t pid;
27     pid_t *children_pids;
28     int pipes[4];

```

```

29     int answers[2];
30     children_pids=(pid_t *)calloc(me->nr_children,sizeof(pid_t));
31     change_pname(me->name);
32     /* loop to fork for all my children */
33
34
35     for (i=0;i<me->nr_children;i++)
36     {
37         if(pipe(pipes+2*i) == -1)
38         {
39             perror("pipe");
40             exit(EXIT_FAILURE);
41         }
42         pid = fork();
43         if (pid < 0)
44         {
45             perror("fork_procs: fork");
46             exit(1);
47         }
48         if (pid == 0)
49         {
50             /* Child */
51             close(*(pipes + 2*i)); //child closes read
52             me = me->children+i;
53             fork_procs(me);
54             exit(1);
55         }
56         close(*(pipes + 2*i + 1)); //father closes write
57         *(children_pids+i)=pid;
58
59     }
60     wait_for_ready_children(me->nr_children);
61     //should the father wait for ready children?
62     for (i=0;i<me->nr_children;i++)
63     {
64         pid = *(children_pids+i);
65         while(read(pipes[2*i],answers+i,1)>0)
66             {//just read!!!
67             }
68         //need to add sanity checks
69         waitpid(pid,&status,WUNTRACED);
70         explain_wait_status(pid,status);
71     }
72
73     if(me->nr_children>0)
74     {
75         switch(*(me->name))
76         case '+'
77             while(write(
78
79
80     }
81
82
83
84
85     /* ... */
86     //if (me->nr_children>0)
87     //{
88     //     pid = wait(&status);
89     //     printf("%s said:\n",me->name);
90     //     explain_wait_status(pid, status);
91     //}
92
93     printf("%s: Exiting...\n",me->name);
94     exit(0);
95 }
96
97 /*
98 * The initial process forks the root of the process tree,
99 * waits for the process tree to be completely created,
100 * then takes a photo of it using show_pstree().
101 *
102 * How to wait for the process tree to be ready?
103 * In ask2-{fork, tree}:

```



```

104  *      wait for a few seconds, hope for the best.
105  * In ask2-signals:
106  *      use wait_for_ready_children() to wait until
107  *      the first process raises SIGSTOP.
108  */
109  int main(int argc, char **argv)
110  {
111      if(argc!=2)
112      {
113          printf("Usage:%s <input.tree> \n",argv[0]);
114          exit(1);
115      }
116      pid_t pid;
117      int status;
118      struct tree_node * root = get_tree_from_file(argv[1]);
119
120      /* Fork root of process tree */
121      pid = fork();
122      if (pid < 0)
123      {
124          perror("main: fork");
125          exit(1);
126      }
127      if (pid == 0)
128      {
129          /* Child */
130          fork_procs(root);
131          exit(1);
132      }
133
134      /*
135       * Father
136       */
137      /* for ask2-signals */
138      /* wait_for_ready_children(1); */
139
140      /* for ask2-{fork, tree} */
141      wait_for_ready_children(1);
142      show_pstree(pid);
143      kill(pid,SIGCONT);
144      waitpid(pid,&status,WUNTRACED);
145      explain_wait_status(pid,status);
146
147      /* Print the process tree root at pid */
148
149      /* for ask2-signals */
150      /* kill(pid, SIGCONT); */
151
152      /* Wait for the root of the process tree to terminate */
153      pid = wait(&status);
154      explain_wait_status(pid, status);
155
156      return 0;
157  }

```