ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

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

Τμήμα Β, Ομάδα 3η

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:

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void fork_procs(struct tree_node *me)
{
    /*
     * initial process is A.
     */
    int i;
    pid_t pid;
    int status;

    change_pname(me->name);
    /* loop to fork for all my children */

    for (i=0;i<me->nr_children;i++)
    {
        pid = fork();
        if (pid < 0) {
            perror("main: fork");
            exit(1);
        }
        if (pid == 0) {
            /* Child */
            me=me->children+i;
            fork_procs(me);
            exit(1);
        }

    }
    if(me->nr_children==0)
    {
        printf("%s: Sleeping...\n",me->name);
        sleep(SLEEP_PROC_SEC);
    }

    /* ... */
    for(i=0;i<me->nr_children;i++)
    {
        pid = wait(&status);
        explain_wait_status(pid, status);
    }

    printf("%s: Exiting...\n",me->name);
    switch(*me->name)
    {
        case 'A':
            exit(16);
            break;
        case 'B':
            exit(19);
        case 'C':
            exit(17);
        case 'D':
```

```
            exit(13);
        default:
            exit(1);
    }
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(void)
{
    pid_t pid;
    int status;
    struct tree_node * root = get_tree_from_file("init.tree");

    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        fork_procs(root);
        exit(1);
    }

    /*
     * Father
     */
    /* for ask2-signals */
    /* wait_for_ready_children(1); */

    /* for ask2-{fork, tree} */
    sleep(SLEEP_TREE_SEC);

    /* Print the process tree root at pid */
    show_pstree(pid);

    /* for ask2-signals */
    /* kill(pid, SIGCONT); */

    /* Wait for the root of the process tree to terminate */
    pid = wait(&status);
    explain_wait_status(pid, status);

    return 0;
}
```

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

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void fork_procs(struct tree_node *me)
{
    /*
     * initial process is A.
     */
    int i;
    pid_t pid;
    int status;

    change_pname(me->name);
    /* loop to fork for all my children */

    for (i=0;i<me->nr_children;i++)
    {
        pid = fork();
        if (pid < 0) {
            perror("main: fork");
            exit(1);
        }
        if (pid == 0) {
            /* Child */
            me=me->children+i;
            fork_procs(me);
            exit(1);
        }

    }
    printf("%s: Sleeping...\n",me->name);
    sleep(SLEEP_PROC_SEC);

    /* ... */
    if (me->nr_children>0)
    {
        pid = wait(&status);
        printf("%s said:\n",me->name);
        explain_wait_status(pid, status);
    }

    printf("%s: Exiting...\n",me->name);
    exit(16);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc,char **argv)
{
    if(argc!=2)
```

```
    {
        printf("Usage:%s <input.tree> \n",argv[0]);
        exit(1);
    }
    pid_t pid;
    int status;
    struct tree_node * root = get_tree_from_file(argv[1]);

    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        fork_procs(root);
        exit(1);
    }

    /*
     * Father
     */
    /* for ask2-signals */
    /* wait_for_ready_children(1); */

    /* for ask2-{fork, tree} */
    sleep(SLEEP_TREE_SEC);

    /* Print the process tree root at pid */
    show_pstree(pid);

    /* for ask2-signals */
    /* kill(pid, SIGCONT); */

    /* Wait for the root of the process tree to terminate */
    pid = wait(&status);
    explain_wait_status(pid, status);

    return 0;
}
```

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

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void fork_procs(struct tree_node *me)
{
    /*
     * initial process is A.
```

```c
     */
    int i;
    int status;
    pid_t pid;
    pid_t *children_pids;
    children_pids=(pid_t *)calloc(me->nr_children,sizeof(pid_t));

    change_pname(me->name);
    /* loop to fork for all my children */

    for (i=0;i<me->nr_children;i++)
    {
        pid = fork();
        if (pid < 0) {
            perror("fork_procs: fork");
            exit(1);
        }
        if (pid == 0) {
            /* Child */
            me = me->children+i;
            fork_procs(me);
            exit(1);
        }
        *(children_pids+i)=pid;

    }
    wait_for_ready_children(me->nr_children);
    printf("%s: pausing...\n",me->name);
    raise(SIGSTOP);
    for (i=0;i<(me->nr_children);i++)
    {
        pid = *(children_pids+i);
        kill(pid,SIGCONT);
        waitpid(pid,&status,WUNTRACED);
        explain_wait_status(pid,status);
    }

    /* ... */
    //if (me->nr_children>0)
    //{
    //    pid = wait(&status);
    //    printf("%s said:\n",me->name);
    //    explain_wait_status(pid, status);
    //}

    printf("%s: Exiting...\n",me->name);
    exit(0);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc,char **argv)
{
    if(argc!=2)
    {
        printf("Usage:%s <input.tree> \n",argv[0]);
        exit(1);
    }
    pid_t pid;
    int status;
    struct tree_node * root = get_tree_from_file(argv[1]);

    /* Fork root of process tree */
    pid = fork();
    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 καύτερη 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 έφρασης

```
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];
```

```
    int answers[2];
    children_pids=(pid_t *)calloc(me->nr_children,sizeof(pid_t));
    change_pname(me->name);
    /* loop to fork for all my children */


    for (i=0;i<me->nr_children;i++)
    {
        if(pipe(pipes+2*i) == -1)
        {
            perror("pipe");
            exit(EXIT_FAILURE);
        }
        pid = fork();
        if (pid < 0)
        {
            perror("fork_procs: fork");
            exit(1);
        }
        if (pid == 0)
        {
            /* Child */
            close(*(pipes + 2*i)); //child closes read
            me = me->children+i;
            fork_procs(me);
            exit(1);
        }
        close(*(pipes + 2*i + 1)); //father closes write
        *(children_pids+i)=pid;

    }
    wait_for_ready_children(me->nr_children);
    //should the father wait for ready children?
    for (i=0;i<me->nr_children;i++)
    {
        pid = *(children_pids+i);
        while(read(pipes[2*i],answers+i,1)>0)
        {//just read!!!
        }
        //need to add sanity checks
        waitpid(pid,&status,WUNTRACED);
        explain_wait_status(pid,status);
    }

    if(me->nr_children>0)
    {
        switch(*(me->name))
            case '+'
                while(write(


    }




    /* ... */
    //if (me->nr_children>0)
    //{
    //    pid = wait(&status);
    //    printf("%s said:\n",me->name);
    //    explain_wait_status(pid, status);
    //}

    printf("%s: Exiting...\n",me->name);
    exit(0);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
```

```
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc,char **argv)
{
    if(argc!=2)
    {
        printf("Usage:%s <input.tree> \n",argv[0]);
        exit(1);
    }
    pid_t pid;
    int status;
    struct tree_node * root = get_tree_from_file(argv[1]);

    /* Fork root of process tree */
    pid = fork();
    if (pid < 0)
    {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0)
    {
        /* Child */
        fork_procs(root);
        exit(1);
    }

    /*
     * Father
     */
    /* for ask2-signals */
    /* wait_for_ready_children(1); */

    /* for ask2-{fork, tree} */
    wait_for_ready_children(1);
    show_pstree(pid);
    kill(pid,SIGCONT);
    waitpid(pid,&status,WUNTRACED);
    explain_wait_status(pid,status);

    /* Print the process tree root at pid */

    /* for ask2-signals */
    /* kill(pid, SIGCONT); */

    /* Wait for the root of the process tree to terminate */
    pid = wait(&status);
    explain_wait_status(pid, status);

    return 0;
}
```