Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Λειτουργικά Συστήματα – Άσκηση 2

Των Προπτυχιακών Φοιτητών
Αρώνη Παναγιώτη Α.Μ. 03109083
Δανασή Παναγιώτη Α.Μ. 03109004
e-mail: el09004@mail.ntua.gr
panaro32@hotmail.com

1 Ασκήσεις:

1.1 Δημιουργία δεδομένου δέντρου εργασιών:

ask2-fork.c

```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"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

// Create this process tree:
//  A-+-B---D
//    `-C

void     fork_procs_A();
void     fork_procs_B();
void     fork_procs_C();
void    fork_procs_D();


int main()
{
    pid_t pid;
    int status;
    // fork root process
    pid=fork();
    if(pid<0)
    {
        perror("main: fork");
        exit(1);
    }
    if(!pid)
    {
        // child process
        fork_procs_A();
        exit(1);
    }
    sleep(SLEEP_TREE_SEC);
    // print the process tree from root
    show_pstree(pid);
    // wait for root process to terminate
    pid=wait(&status);
    explain_wait_status(pid,status);
    return 0;
}
```

```c
void fork_procs_A()
{
    //  initial  process  A
    pid_t     pid,pid1,pid2;
    int status;

    printf("A(%ld): Starting...\n",(long)getpid());
    change_pname("A");
    pid1=fork();
    if(pid1<0)
    {
        perror("A: fork_1");
        exit(1);
    }
    if(!pid1)
    {
        // child process 1
        fork_procs_B();
        exit(1);
    }
    pid2=fork();
    if(pid2<0)
    {
        perror("A: fork_2");
        exit(1);
    }
    if(!pid2)
    {
        // child process 2
        fork_procs_C();
        exit(1);
    }
    printf("A(%ld): Waiting...\n",(long)getpid());
    pid=wait(&status);
    explain_wait_status(pid,status);
    printf("A(%ld): Waiting...\n",(long)getpid());
    pid=wait(&status);
    explain_wait_status(pid,status);
    printf("A(%ld): Exiting...\n",(long)getpid());
    exit(16);
}
```

```c
void fork_procs_B()
{
    // process B
    pid_t pid;
    int status;

    printf("B(%ld): Starting...\n",(long)getpid());
    change_pname("B");
    pid=fork(); if(pid<0)
    {
        perror("B: fork");
        exit(1);
    }
    if(!pid)
    {
        // child process
        fork_procs_D();
        exit(1);
    }
    printf("B(%ld): Waiting...\n",(long)getpid());
    pid=wait(&status);
    explain_wait_status(pid,status);
    printf("B(%ld): Exiting...\n",(long)getpid());
    exit(19);
}



void fork_procs_C()
{
    // process C
    printf("C(%ld): Starting...\n",(long)getpid());
    change_pname("C");
    printf("C(%ld): Sleeping...\n",(long)getpid());
    sleep(SLEEP_PROC_SEC);
    printf("C(%ld): Exiting...\n",(long)getpid());
    exit(17);
}



void fork_procs_D()
{
    // process D
    printf("D(%ld): Starting...\n",(long)getpid());
    change_pname("D");
    printf("D(%ld): Sleeping...\n",(long)getpid());
    sleep(SLEEP_PROC_SEC);
    printf("D(%ld): Exiting...\n",(long)getpid());
    exit(13);
}
```

Έξοδος εκτέλεσης προγράμματος:

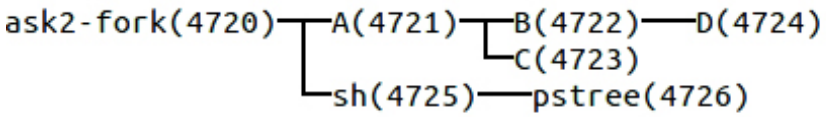
A(2973): Starting...
B(2974): Starting...
B(2974): Waiting...
D(2975): Starting...
D(2975): Sleeping...
A(2973): Waiting...
C(2976): Starting...
C(2976): Sleeping...

A(2973)-T-B(2974)---D(2975)
L-C(2976)

D(2975): Exiting...
C(2976): Exiting...
My PID = 2974: Child PID = 2975 terminated normally, exit status = 13
B(2974): Exiting...
My PID = 2973: Child PID = 2976 terminated normally, exit status = 17
A(2973): Waiting...
My PID = 2973: Child PID = 2974 terminated normally, exit status = 19
A(2973): Exiting...
My PID = 2972: Child PID = 2973 terminated normally, exit status = 16

Ερωτήσεις:

1. Αν τερματίσουμε πρόωρα την διεργασία Α, τότε όσα από τα παιδιά της είναι ακόμα ζωντανά θα υιοθετηθούν από την init.
2. Θα μας δείξει επιπλέον την διεργασία του κώδικα της άσκησής μας, δηλαδή την ask2-fork, η οποία έχει και ένα παιδί παραπάνω που είναι μια διεργασία shell όπου ως παιδί της έχει την διεργασία που εμφανίζει το δέντρο (pstree). Συγκεκριμένα το αποτέλεσμα φαίνεται παρακάτω:



3. Τα όρια σχετικά με το πλήθος των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης θέτονται από τον διαχειριστή για να μην κάνει κατάχρηση κάποιος χρήστης είτε κακόβουλα, είτε από λάθος στον κώδικα που τρέχει και να δημιουργήσει πάρα πολλές διεργασίες, κάτι που θα είχε ως αποτέλεσμα να επιβαρύνει το υπολογιστικό σύστημα και να καταναλώνει άσκοπα πόρους.

1.2 Δημιουργία αυθαίρετου δέντρου εργασιών:

ask2-tree.c

```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

void fork_procs(struct tree_node*);


int main(int argc,char *argv[])
{
    struct tree_node *root;
    pid_t pid;
    int status;

    if(argc!=2)
    {
        fprintf(stderr,"Usage: %s <input_tree_file>\n",argv[0]);
        exit(1);
    }
    // scan tree_nodes
    root=get_tree_from_file(argv[1]);
    // print tree_nodes
    print_tree(root);
    // fork root process
    pid=fork();
    if(pid<0)
    {
        perror("main: fork");
        exit(1);
    }
    if(!pid)
    {
        // child process
        fork_procs(root);
        exit(1);
    }
    sleep(SLEEP_TREE_SEC);
    // print the process tree from root
    show_pstree(pid);
    // wait for root process to terminate
    pid=wait(&status);
    explain_wait_status(pid,status);
    return 0;
}
```

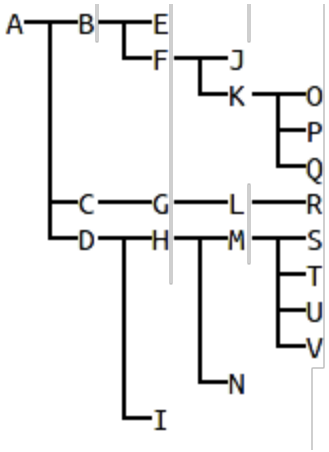
```c
void fork_procs(struct tree_node *node)
{
    pid_t chpid,pid[node->nr_children];
    int cnt,status;

    printf("%s(%ld): Starting...\n",node->name,(long)getpid());
    change_pname(node->name);
    // fork every child of the node
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        pid[cnt]=fork();
        if(pid[cnt]<0)
        {
            perror("fork_procs: fork");
            exit(1);
        }
        if(!pid[cnt])
        {
            // child process
            fork_procs(node->children+cnt);
            exit(1);
        }
    }
    // wait for every child to terminate
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        printf("%s(%ld): Waiting(%d/%d)...\n",node->name,(long)getpid(),cnt+1,node->
        nr_children);
        chpid=wait(&status);
        explain_wait_status(chpid,status);
    }
    // sleep processes sleep before exiting
    if(!(node->nr_children))
    {
        printf("%s(%ld): Sleeping...\n",node->name,(long)getpid());
        sleep(SLEEP_PROC_SEC);
    }
    printf("%s(%ld): Exiting...\n",node->name,(long)getpid());
    exit(0);
}
```

Έξοδος εκτέλεσης προγράμματος:

(με είσοδο το δέντρο που δίνεται expr.tree)

```
+
        10
        *
                +
                        5
                        7
                4
+(3029): Starting...
+(3029): Waiting(1/2)...
*(3031): Starting...
*(3031): Waiting(1/2)...
10(3030): Starting...
10(3030): Sleeping...
4(3033): Starting...
4(3033): Sleeping...
+(3032): Starting...
+(3032): Waiting(1/2)...
7(3035): Starting...
7(3035): Sleeping...
5(3034): Starting...
5(3034): Sleeping...


+(3029)-T-*(3031)-T-+(3032)-T-5(3034)
      ¦       ¦       L-7(3035)
      ¦       L-4(3033)
      L-10(3030)


10(3030): Exiting...
My PID = 3029: Child PID = 3030 terminated normally, exit status = 0
+(3029): Waiting(2/2)...
4(3033): Exiting...
7(3035): Exiting...
My PID = 3031: Child PID = 3033 terminated normally, exit status = 0
*(3031): Waiting(2/2)...
5(3034): Exiting...
My PID = 3032: Child PID = 3035 terminated normally, exit status = 0
+(3032): Waiting(2/2)...
My PID = 3032: Child PID = 3034 terminated normally, exit status = 0
+(3032): Exiting...
My PID = 3031: Child PID = 3032 terminated normally, exit status = 0
*(3031): Exiting...
My PID = 3029: Child PID = 3031 terminated normally, exit status = 0
+(3029): Exiting...
My PID = 3028: Child PID = 3029 terminated normally, exit status = 0
```

(εκτέλεση με είσοδο δέντρο που κατασκευάσαμε εμείς, test.tree)



A
	B
		E
		F
			J
			K
				O
				P
				Q
	C
		G
			L
				R
	D
		H
			M
				S
				T
				U
				V
			N
		I

A(3135): Starting...
A(3135): Waiting(1/3)...
D(3138): Starting...
D(3138): Waiting(1/2)...
C(3137): Starting...
C(3137): Waiting(1/1)...
B(3136): Starting...
B(3136): Waiting(1/2)...
I(3140): Starting...
I(3140): Sleeping...
H(3139): Starting...
H(3139): Waiting(1/2)...
G(3141): Starting...
G(3141): Waiting(1/1)...
F(3143): Starting...
F(3143): Waiting(1/2)...
E(3142): Starting...
E(3142): Sleeping...
L(3146): Starting...
N(3145): Starting...
N(3145): Sleeping...
M(3144): Starting...
M(3144): Waiting(1/4)...
L(3146): Waiting(1/1)...
K(3148): Starting...
K(3148): Waiting(1/3)...
J(3147): Starting...
J(3147): Sleeping...
V(3152): Starting...
V(3152): Sleeping...
R(3153): Starting...
R(3153): Sleeping...

```
T(3150): Starting...
U(3151): Starting...
U(3151): Sleeping...
T(3150): Sleeping...
O(3154): Starting...
O(3154): Sleeping...
Q(3156): Starting...
Q(3156): Sleeping...
P(3155): Starting...
P(3155): Sleeping...
S(3149): Starting...
S(3149): Sleeping...
```

```
A(3135)-T-B(3136)-T-E(3142)
      ¦       L-F(3143)-T-J(3147)
      ¦               L-K(3148)-T-O(3154)
      ¦                       +-P(3155)
      ¦                       L-Q(3156)
      +-C(3137)---G(3141)---L(3146)---R(3153)
      L-D(3138)-T-H(3139)-T-M(3144)-T-S(3149)
              ¦       ¦       +-T(3150)
              ¦       ¦       +-U(3151)
              ¦       ¦       L-V(3152)
              ¦       L-N(3145)
              L-I(3140)
```

```
I(3140): Exiting...
My PID = 3138: Child PID = 3140 terminated normally, exit status = 0
D(3138): Waiting(2/2)...
E(3142): Exiting...
My PID = 3136: Child PID = 3142 terminated normally, exit status = 0
B(3136): Waiting(2/2)...
N(3145): Exiting...
My PID = 3139: Child PID = 3145 terminated normally, exit status = 0
H(3139): Waiting(2/2)...
J(3147): Exiting...
V(3152): Exiting...
R(3153): Exiting...
My PID = 3143: Child PID = 3147 terminated normally, exit status = 0
F(3143): Waiting(2/2)...
My PID = 3144: Child PID = 3152 terminated normally, exit status = 0
M(3144): Waiting(2/4)...
My PID = 3146: Child PID = 3153 terminated normally, exit status = 0
L(3146): Exiting...
My PID = 3141: Child PID = 3146 terminated normally, exit status = 0
G(3141): Exiting...
T(3150): Exiting...
U(3151): Exiting...
O(3154): Exiting...
Q(3156): Exiting...
My PID = 3137: Child PID = 3141 terminated normally, exit status = 0
P(3155): Exiting...
S(3149): Exiting...
C(3137): Exiting...
My PID = 3148: Child PID = 3154 terminated normally, exit status = 0
```

K(3148): Waiting(2/3)...
My PID = 3148: Child PID = 3155 terminated normally, exit status = 0
My PID = 3144: Child PID = 3149 terminated normally, exit status = 0
My PID = 3135: Child PID = 3137 terminated normally, exit status = 0
A(3135): Waiting(2/3)...
K(3148): Waiting(3/3)...
M(3144): Waiting(3/4)...
My PID = 3144: Child PID = 3150 terminated normally, exit status = 0
My PID = 3148: Child PID = 3156 terminated normally, exit status = 0
K(3148): Exiting...
M(3144): Waiting(4/4)...
My PID = 3143: Child PID = 3148 terminated normally, exit status = 0
F(3143): Exiting...
My PID = 3136: Child PID = 3143 terminated normally, exit status = 0
B(3136): Exiting...
My PID = 3135: Child PID = 3136 terminated normally, exit status = 0
A(3135): Waiting(3/3)...
My PID = 3144: Child PID = 3151 terminated normally, exit status = 0
M(3144): Exiting...
My PID = 3139: Child PID = 3144 terminated normally, exit status = 0
H(3139): Exiting...
My PID = 3138: Child PID = 3139 terminated normally, exit status = 0
D(3138): Exiting...
My PID = 3135: Child PID = 3138 terminated normally, exit status = 0
A(3135): Exiting...
My PID = 3134: Child PID = 3135 terminated normally, exit status = 0

Ερωτήσεις:

1. Τα μηνύματα έναρξης εμφανίζονται ανά επίπεδο του δέντρου (κατά πλάτος) καθώς κάθε διεργασία ανά επίπεδο, δημιουργεί τα παιδιά της και έπειτα πάμε στην επόμενη διεργασία.
   Τα μηνύματα τερματισμού εμφανίζονται με την αντίστροφη σειρά, εκτός από τις διεργασίες φύλλα που θα εμφανίσουν πρώτες μήνυμα τερματισμού, αφού δεν έχουν παιδιά να περιμένουν.

1.3 Αποστολή και χειρισμός σημάτων:

ask2-signals.c

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

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

void fork_procs(struct tree_node*);

int main(int argc,char *argv[])
{
    struct tree_node *root;
    pid_t pid;
    int status;

    if(argc!=2)
    {
        fprintf(stderr,"Usage: %s <input_tree_file>\n",argv[0]);
        exit(1);
    }
    // scan tree_nodes
    root=get_tree_from_file(argv[1]);
    // print tree_nodes
    print_tree(root);
    // fork root process
    pid=fork();
    if(pid<0)
    {
        perror("main: fork");
        exit(1);
    }
    if(!pid)
    {
        // child process
        fork_procs(root);
        exit(1);
    }
    // wait for root process to stop
    wait_for_ready_children(1);
    // print the process tree from root
    show_pstree(pid);
    // activate root process
    kill(pid,SIGCONT);
    // wait for root process to terminate
    wait(&status);
    explain_wait_status(pid,status);
    return 0;
}
```

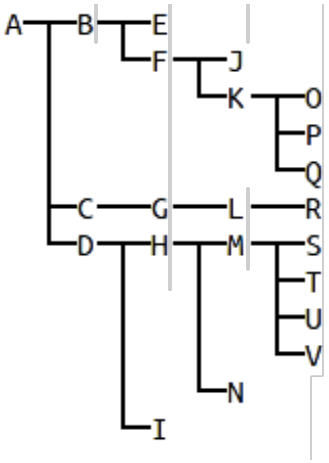
```c
void fork_procs(struct tree_node *node)
{
    pid_t pid[node->nr_children];
    int cnt,status;

    printf("%s(%ld): Starting...\n",node->name,(long)getpid());
    change_pname(node->name);
    // fork every child of the node
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        pid[cnt]=fork();
        if(pid[cnt]<0)
        {
            perror("fork_procs: fork");
            exit(1);
        }
        if(!pid[cnt])
        {
            // child process
            fork_procs(node->children+cnt);
            exit(1);
        }
    }
    // wait for every child to stop
    wait_for_ready_children(node->nr_children);
    // self suspend process
    raise(SIGSTOP);
    printf("%s(%ld): Just woke up!\n",node->name,(long)getpid());
    // wake up every child
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        kill(pid[cnt],SIGCONT);
        // wait for child to terminate (DFS)
        printf("%s(%ld): Waiting(%d/%d)...\n",node->name,(long)getpid(),cnt+1,node->
        nr_children);
        wait(&status);
        explain_wait_status(pid[cnt],status);
    }
    printf("%s(%ld): Exiting...\n",node->name,(long)getpid());
    exit(0);
}
```

Έξοδος εκτέλεσης προγράμματος:

(με είσοδο το δέντρο που δίνεται expr.tree)

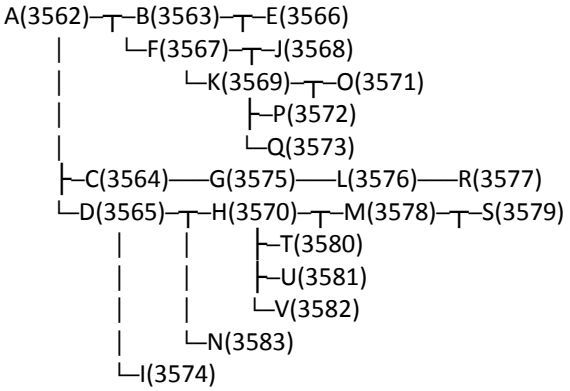
```
+
        10
        *
                +
                        5
                        7
                4
+(3043): Starting...
*(3045): Starting...
10(3044): Starting...
My PID = 3043: Child PID = 3044 has been stopped by a signal, signo = 19
4(3047): Starting...
My PID = 3045: Child PID = 3047 has been stopped by a signal, signo = 19
+(3046): Starting...
7(3049): Starting...
My PID = 3046: Child PID = 3049 has been stopped by a signal, signo = 19
5(3048): Starting...
My PID = 3046: Child PID = 3048 has been stopped by a signal, signo = 19
My PID = 3045: Child PID = 3046 has been stopped by a signal, signo = 19
My PID = 3043: Child PID = 3045 has been stopped by a signal, signo = 19
My PID = 3042: Child PID = 3043 has been stopped by a signal, signo = 19


+(3043)-T-*(3045)-T-+(3046)-T-5(3048)
      ¦       ¦       L-7(3049)
      ¦       L-4(3047)
      L-10(3044)


+(3043): Just woke up!
+(3043): Waiting(1/2)...
10(3044): Just woke up!
10(3044): Exiting...
My PID = 3043: Child PID = 3044 terminated normally, exit status = 0
+(3043): Waiting(2/2)...
*(3045): Just woke up!
*(3045): Waiting(1/2)...
+(3046): Just woke up!
+(3046): Waiting(1/2)...
5(3048): Just woke up!
5(3048): Exiting...
My PID = 3046: Child PID = 3048 terminated normally, exit status = 0
+(3046): Waiting(2/2)...
7(3049): Just woke up!
7(3049): Exiting...
My PID = 3046: Child PID = 3049 terminated normally, exit status = 0
+(3046): Exiting...
My PID = 3045: Child PID = 3046 terminated normally, exit status = 0
*(3045): Waiting(2/2)...
4(3047): Just woke up!
4(3047): Exiting...
My PID = 3045: Child PID = 3047 terminated normally, exit status = 0
*(3045): Exiting...
My PID = 3043: Child PID = 3045 terminated normally, exit status = 0
+(3043): Exiting...
My PID = 3042: Child PID = 3043 terminated normally, exit status = 0
```

(εκτέλεση με είσοδο δέντρο που κατασκευάσαμε εμείς, test.tree)




```
A
        B
                E
                F
                        J
                        K
                                O
                                P
                                Q
        C
                G
                        L
                                R
        D
                H
                        M
                                S
                                T
                                U
                                V
                N
                I
A(3562): Starting...
B(3563): Starting...
F(3567): Starting...
E(3566): Starting...
D(3565): Starting...
My PID = 3563: Child PID = 3566 has been stopped by a signal, signo = 19
K(3569): Starting...
J(3568): Starting...
My PID = 3567: Child PID = 3568 has been stopped by a signal, signo = 19
Q(3573): Starting...
C(3564): Starting...
My PID = 3569: Child PID = 3573 has been stopped by a signal, signo = 19
O(3571): Starting...
P(3572): Starting...
My PID = 3569: Child PID = 3572 has been stopped by a signal, signo = 19
My PID = 3569: Child PID = 3571 has been stopped by a signal, signo = 19
My PID = 3567: Child PID = 3569 has been stopped by a signal, signo = 19
My PID = 3563: Child PID = 3567 has been stopped by a signal, signo = 19
I(3574): Starting...
My PID = 3562: Child PID = 3563 has been stopped by a signal, signo = 19
My PID = 3565: Child PID = 3574 has been stopped by a signal, signo = 19
G(3575): Starting...
L(3576): Starting...
R(3577): Starting...
My PID = 3576: Child PID = 3577 has been stopped by a signal, signo = 19
My PID = 3575: Child PID = 3576 has been stopped by a signal, signo = 19
My PID = 3564: Child PID = 3575 has been stopped by a signal, signo = 19
My PID = 3562: Child PID = 3564 has been stopped by a signal, signo = 19
H(3570): Starting...
M(3578): Starting...
T(3580): Starting...
S(3579): Starting...
V(3582): Starting...
U(3581): Starting...
My PID = 3578: Child PID = 3580 has been stopped by a signal, signo = 19
My PID = 3578: Child PID = 3581 has been stopped by a signal, signo = 19
My PID = 3578: Child PID = 3582 has been stopped by a signal, signo = 19
My PID = 3578: Child PID = 3579 has been stopped by a signal, signo = 19
N(3583): Starting...
```

My PID = 3570: Child PID = 3578 has been stopped by a signal, signo = 19
My PID = 3570: Child PID = 3583 has been stopped by a signal, signo = 19
My PID = 3565: Child PID = 3570 has been stopped by a signal, signo = 19
My PID = 3562: Child PID = 3565 has been stopped by a signal, signo = 19
My PID = 3561: Child PID = 3562 has been stopped by a signal, signo = 19



A(3562): Just woke up!
A(3562): Waiting(1/3)...
B(3563): Just woke up!
B(3563): Waiting(1/2)...
E(3566): Just woke up!
E(3566): Exiting...
My PID = 3563: Child PID = 3566 terminated normally, exit status = 0
B(3563): Waiting(2/2)...
F(3567): Just woke up!
F(3567): Waiting(1/2)...
J(3568): Just woke up!
J(3568): Exiting...
My PID = 3567: Child PID = 3568 terminated normally, exit status = 0
F(3567): Waiting(2/2)...
K(3569): Just woke up!
K(3569): Waiting(1/3)...
O(3571): Just woke up!
O(3571): Exiting...
My PID = 3569: Child PID = 3571 terminated normally, exit status = 0
K(3569): Waiting(2/3)...
P(3572): Just woke up!
P(3572): Exiting...
My PID = 3569: Child PID = 3572 terminated normally, exit status = 0
K(3569): Waiting(3/3)...
Q(3573): Just woke up!
Q(3573): Exiting...
My PID = 3569: Child PID = 3573 terminated normally, exit status = 0
K(3569): Exiting...
My PID = 3567: Child PID = 3569 terminated normally, exit status = 0
F(3567): Exiting...
My PID = 3563: Child PID = 3567 terminated normally, exit status = 0
B(3563): Exiting...
My PID = 3562: Child PID = 3563 terminated normally, exit status = 0
A(3562): Waiting(2/3)...
C(3564): Just woke up!
C(3564): Waiting(1/1)...
G(3575): Just woke up!
G(3575): Waiting(1/1)...
L(3576): Just woke up!
L(3576): Waiting(1/1)...
R(3577): Just woke up!
R(3577): Exiting...

My PID = 3576: Child PID = 3577 terminated normally, exit status = 0
L(3576): Exiting...
My PID = 3575: Child PID = 3576 terminated normally, exit status = 0
G(3575): Exiting...
My PID = 3564: Child PID = 3575 terminated normally, exit status = 0
C(3564): Exiting...
My PID = 3562: Child PID = 3564 terminated normally, exit status = 0
A(3562): Waiting(3/3)...
D(3565): Just woke up!
D(3565): Waiting(1/2)...
H(3570): Just woke up!
H(3570): Waiting(1/2)...
M(3578): Just woke up!
M(3578): Waiting(1/4)...
S(3579): Just woke up!
S(3579): Exiting...
My PID = 3578: Child PID = 3579 terminated normally, exit status = 0
M(3578): Waiting(2/4)...
T(3580): Just woke up!
T(3580): Exiting...
My PID = 3578: Child PID = 3580 terminated normally, exit status = 0
M(3578): Waiting(3/4)...
U(3581): Just woke up!
U(3581): Exiting...
My PID = 3578: Child PID = 3581 terminated normally, exit status = 0
M(3578): Waiting(4/4)...
V(3582): Just woke up!
V(3582): Exiting...
My PID = 3578: Child PID = 3582 terminated normally, exit status = 0
M(3578): Exiting...
My PID = 3570: Child PID = 3578 terminated normally, exit status = 0
H(3570): Waiting(2/2)...
N(3583): Just woke up!
N(3583): Exiting...
My PID = 3570: Child PID = 3583 terminated normally, exit status = 0
H(3570): Exiting...
My PID = 3565: Child PID = 3570 terminated normally, exit status = 0
D(3565): Waiting(2/2)...
I(3574): Just woke up!
I(3574): Exiting...
My PID = 3565: Child PID = 3574 terminated normally, exit status = 0
D(3565): Exiting...
My PID = 3562: Child PID = 3565 terminated normally, exit status = 0
A(3562): Exiting...
My PID = 3561: Child PID = 3562 terminated normally, exit status = 0

Ερωτήσεις:

1. Χρησιμοποιώντας την sleep() για τον συγχρονισμό των διεργασιών πρέπει να περιμένουμε ένα αυθαίρετο χρονικό διάστημα για να ετοιμαστεί το δέντρο των διεργασιών κάτι που προφανώς δεν είναι αποδοτικό αφού το διάστημα αυτό μπορεί να μην είναι αρκετό σε ορισμένες περιπτώσεις και να μην έχει προλάβει να ετοιμαστεί το δέντρο ή μπορεί να είναι πολύ περισσότερο απ ότι χρειαζόμαστε και να περιμένουμε άσκοπα. Σε κάθε περίπτωση επιβαρύνει τον προγραμματιστή να προσπαθεί να υπολογίσει πόσο χρόνο χρειάζεται. Αντίθετα με την χρήση σημάτων, είμαστε σίγουροι πως θα ξυπνήσουμε τις διεργασίες ακριβώς την κατάλληλη στιγμή και δεν επιβαρύνεται και ο προγραμματιστής για να υλοποιήσει σωστά όση καθυστέρηση χρειάζεται.
2. Ο ρόλος της wait_for_ready_children() είναι να ελέγξει κατά πόσο όλα τα παιδιά μιας διεργασίας έχουν αναστείλει την λειτουργία τους. Αυτό μας δίνει την δυνατότητα να τυπώσουμε σωστά το δέντρο των διεργασιών με τον εξής τρόπο: Η main δημιουργεί την ρίζα του δέντρου και εκτελεί την wait_for_ready_children(). Μόλις επιστρέψει από την συγκεκριμένη συνάρτηση γνωρίζει ότι η ρίζα του δέντρου έχει αναστείλει την λειτουργία της. Όμως επειδή την wait_for_ready_children() την τρέχει κάθε κόμβος για τα παιδιά του, στην πραγματικότητα μόλις η main επιστρέψει από την wait_for_ready_children() ξέρει ότι έχουν αναστείλει την λειτουργία τους κάθε κόμβος του δέντρου. Έτσι μπορούμε να τυπώσουμε το δέντρο των διεργασιών. Αν παραλείπαμε την wait_for_ready_children() θα είχαμε πάλι προβλήματα συγχρονισμού καθώς δεν θα ξέραμε αν έχουν δημιουργηθεί οι διάφορες διεργασίες για να τυπώσουμε το δέντρο αλλά και για να τις ξυπνήσουμε μετά.

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης:

ask2-pipes.c

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

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

void fork_procs(struct tree_node*,int*);
int compute(struct tree_node*,int*);


int main(int argc,char *argv[])
{
    struct tree_node *root;
    pid_t pid;
    int pfd[2],status,res;

    if(argc!=2)
    {
        fprintf(stderr,"Usage: %s <input_tree_file>\n",argv[0]);
        exit(1);
    }
    // scan tree_nodes
    root=get_tree_from_file(argv[1]);
    // print tree_nodes
    print_tree(root);
    // create pipe
    printf("main(%ld): Creating pipe...\n",(long)getpid());
    if(pipe(pfd)<0)
    {
        perror("main: pipe");
        exit(1);
    }
    // fork root process
    pid=fork();
    if(pid<0)
    {
        perror("main: fork");
        exit(1);
    }
    if(!pid)
    {
        // child process
        fork_procs(root,pfd);
        exit(1);
    }
```

```c
    // lock write for parent
    if(close(pfd[1])<0)
    {
        perror("main: close");
        exit(1);
    }
    // wait for root process to terminate
    pid=wait(&status);
    explain_wait_status(pid,status);
    // read result from pipe
    if(read(pfd[0],&res,sizeof(res))!=sizeof(res))
    {
        perror("main: read");
        exit(1);
    }
    printf("Result: %d\n",res);
    return 0;
}
```

```c
void fork_procs(struct tree_node *node,int *pfd)
{
    pid_t chpid,pid[node->nr_children];
    int cnt,res,val[node->nr_children],status,pfds[node-
    >nr_children][2];

    printf("%s(%ld): Starting...\n",node->name,(long)getpid());
    change_pname(node->name);
    // lock read for child
    if(close(pfd[0])<0)
    {
        perror("fork_procs: close");
        exit(1);
    }
    // create a pipe for every child
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        printf("%s(%ld): Creating pipes(%d/%d)...\n",node-
        >name,(long)getpid(),cnt+1,node->
        nr_children);
        if(pipe(pfds[cnt])<0)
        {
            perror("fork_procs: pipe");
            exit(1);
        }
    }
    // fork every child of the node
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        pid[cnt]=fork();
        if(pid[cnt]<0)
        {
            perror("fork_procs: fork");
            exit(1);
        }
        if(!pid[cnt])
        {
            // child process
            fork_procs(node->children+cnt,pfds[cnt]);
            exit(1);
        }
    }
    // lock write for parent
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        if(close(pfds[cnt][1])<0)
        {
            perror("fork_procs: close");
            exit(1);
        }
    }
```

```
    // wait for every child to terminate
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        printf("%s(%ld): Waiting(%d/%d)...\n",node->name,(long)getpid(),cnt+1,node->
        nr_children);
        chpid=wait(&status);
        explain_wait_status(chpid,sta
        tus);
    }
    // read result from child pipes for every child
    for(cnt=0;cnt<node->nr_children;cnt++)
    {
        printf("%s(%ld): Reading pipes(%d/%d)...\n",node-
        >name,(long)getpid(),cnt+1,node->
        nr_children);
        if(read(pfds[cnt][0],val+cnt,sizeof(val[cnt]))!=sizeof(val[cnt]))
        {
            perror("fork_procs: read");
            exit(1);
        }
    }

    // compute node
    result
    res=compute(node,
    val);
    // write result to parent pipe
    printf("%s(%ld): Writing pipe...\n",node->name,(long)getpid());
    if(write(pfd[1],&res,sizeof(res))!=sizeof(res))
    {
        perror("fork_procs: write");
        exit(1);
    }
    printf("%s(%ld): Exiting...\n",node->name,(long)getpid());
    exit(0);
}



int compute(struct tree_node *node,int *val)
{
    int res,cnt;
    if(!(node->nr_children)) return atoi(node->name);
    if(!strcmp(node->name,"+"))
    {
        for(res=0,cnt=0;cnt<node->nr_children;cnt++) res+=val[cnt];
        return res;
    }
    if(!strcmp(node->name,"*"))
    {
        for(res=1,cnt=0;cnt<node->nr_children;cnt++) res*=val[cnt];
        return res;
    }
    fprintf(stderr,"Unknown operator: \"%s\"\n",node->name);
    exit(1);
}
```

Έξοδος εκτέλεσης προγράμματος:

(με είσοδο το δέντρο που δίνεται expr.tree)

```
+
        10
        *
                +
                        5
                        7
                4
main(3054): Creating pipe...
+(3055): Starting...
+(3055): Creating pipes(1/2)...
+(3055): Creating pipes(2/2)...
+(3055): Waiting(1/2)...
*(3057): Starting...
*(3057): Creating pipes(1/2)...
*(3057): Creating pipes(2/2)...
*(3057): Waiting(1/2)...
10(3056): Starting...
10(3056): Writing pipe...
10(3056): Exiting...
My PID = 3055: Child PID = 3056 terminated normally, exit status = 0
+(3055): Waiting(2/2)...
4(3059): Starting...
4(3059): Writing pipe...
4(3059): Exiting...
My PID = 3057: Child PID = 3059 terminated normally, exit status = 0
*(3057): Waiting(2/2)...
+(3058): Starting...
+(3058): Creating pipes(1/2)...
+(3058): Creating pipes(2/2)...
+(3058): Waiting(1/2)...
7(3061): Starting...
7(3061): Writing pipe...
7(3061): Exiting...
My PID = 3058: Child PID = 3061 terminated normally, exit status = 0
+(3058): Waiting(2/2)...
5(3060): Starting...
5(3060): Writing pipe...
5(3060): Exiting...
My PID = 3058: Child PID = 3060 terminated normally, exit status = 0
+(3058): Reading pipes(1/2)...
+(3058): Reading pipes(2/2)...
+(3058): Writing pipe...
+(3058): Exiting...
My PID = 3057: Child PID = 3058 terminated normally, exit status = 0
*(3057): Reading pipes(1/2)...
*(3057): Reading pipes(2/2)...
*(3057): Writing pipe...
*(3057): Exiting...
My PID = 3055: Child PID = 3057 terminated normally, exit status = 0
+(3055): Reading pipes(1/2)...
+(3055): Reading pipes(2/2)...
+(3055): Writing pipe...
+(3055): Exiting...
My PID = 3054: Child PID = 3055 terminated normally, exit status = 0
Result: 58
```

Ερωτήσεις:

1. Στην συγκεκριμένη περίπτωση χρειαζόμαστε δύο σωλήνες ανά γονική διεργασία (γενικά χρειαζόμαστε τόσους σωλήνες όσα και τα παιδιά της κάθε διεργασίας). Θα μπορούσαμε να χρησιμοποιήσουμε και μόνο ένα σωλήνα αλλά θα έπρεπε με κάποιον τρόπο (πχ με την χρήση σημάτων) να συγχρονίσουμε σωστά την ανάγνωση και την εγγραφή σε αυτόν. Γενικά με ένα σωλήνα θα μπορούσε να γίνει σε κάθε τελεστή, με λίγη περισσότερη προσοχή ως προς τον συγχρονισμό της εγγραφής και της ανάγνωσης στους τελεστές που δεν είναι αντιμεταθετικοί.
2. Το πλεονέκτημα που θα είχαμε σε ένα σύστημα πολλαπλών επεξεργαστών αν αποτιμούσαμε την αριθμητική έκφραση με χρήση δέντρου διεργασιών είναι ότι οι διεργασίες θα εκτελούνταν παράλληλα στους επεξεργαστές του συστήματος και συνεπώς κάθε επεξεργαστής θα αναλάμβανε ένα επιμέρους υπολογισμό. Με αυτόν τον τρόπο θα αξιοποιούσαμε περισσότερους από έναν επεξεργαστές και θα οδηγούμασταν ταχύτερα στο αποτέλεσμα, αντίθετα με την περίπτωση όπου ένας επεξεργαστής μόνο θα αναλάμβανε την αποτίμηση της έκφρασης εκτελώντας μία μόνο διεργασία.