Λειτουργικά Συστήματα – 2η Εργαστηριακή Άσκηση

Ομάδα εργαστηρίου: **oslabb30**

Φοιτητές: **Γεώργιος Λαγός 03117034, Παναγιώτης Ζευγολατάκος 03117804**

Στην αναφορά χρησιμοποιείται παντού Α’ πληθυντικός γιατί τόσο η αναφορά όσο και η ίδια η εργασία έγιναν με πλήρη επικοινωνία μεταξύ των φοιτητών.

Ασκήσεις

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

```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(void)
{
        /*
         *   * initial process is A.
         *       */

        pid_t b,c,d;   //THA XREIASTEI NA KATASKEUASOUME TA B,C,D
        int status;

        change_pname("A");
        printf("A: Created!\n");
        printf("A: Creating child B...\n");
        b = fork(); // KATASKEUAZETE TO B
        if (b < 0) {
                perror("B: fork"); //ERROR
                exit(1);
        }
        if (b == 0) { // AN TO B EINAI 0 TOTE EIMASTE STO PROCCESS TOU CHILD
                /* Child B */
                change_pname("B");
                printf("B: Created!\n");
                printf("B: Creating child D...\n");
                d = fork(); // TO B EXEI PAIDI TO D OPOTE KATASKEUAZOUME TO D
                if (d < 0) {
                        perror("D: fork"); // ERROR
                        exit(1);
                }
                if (d == 0) { // AN TO D EINAI 0 TOTE EIMASTE STO PROCCES TOU CHILD
                        /* Child D */
                        change_pname("D");
                        printf("D: Created!\n");
                        printf("D: Sleeping...\n");
                        sleep(SLEEP_PROC_SEC); // KOMVOI XWRIS PEDIA THELOUME NA EKTELOUN TIN SLEEP
                        printf("D: Exiting...\n");
                        exit(13);
                }
                printf("B: Waiting...\n");
                d = wait(&status); // PRWTOU KLEISOUME TO B PREPEI NA E3ASFALISOUME OTI TO PAIDI TOU EKLEISE SWSTA
```

```
                explain_wait_status(d,status);  // EPE3ERGASIA TOU TI EPESTREPSE H KLHSH TOU WAIT
                printf("B: Exiting...\n");
                exit(19);

        }
        // EXONTAS KLEISEI TO D KAI META TO B, GURNAME STO A OPOU TWRA KATASKEUAZOUME TO C
        printf("A: Creating child C...\n");
        c = fork();
        if (c < 0) {
                perror("C: fork");
                exit(1);
        }
        if (c == 0) { //AN TO C EINAI 0 TOTE EIMASTE STO PROCCESS TOU CHILD
                /* Child C */
                change_pname("C");
                printf("C: Created!\n");
                printf("C: Sleeping...\n");
                sleep(SLEEP_PROC_SEC); // EINAI FULLO ARA PREPEI NA EKTELESEI SLEEP
                printf("C: Exiting...\n");
                exit(17);
        }

        /* Parent  A */
        printf("A: Waiting...\n");
        b = wait(&status);
        explain_wait_status(b,status); // ELEGXOUME AN TA PAIDIA TOU A (B,C) EKLEISAN SWSTA
        c = wait(&status);
        explain_wait_status(c,status);
        printf("A: Exiting...\n");
        exit(16);
}

int main () {
        pid_t pid;
        int status;
        pid = fork();
        if (pid < 0) {
            perror("main: fork");
            exit(1);
        }
        if (pid == 0) {
            fork_procs();
            exit(1);
        }
        sleep(SLEEP_TREE_SEC);
        sleep(SLEEP_TREE_SEC);
        /* Print the process tree root at pid */
        show_pstree(pid);
        /* Wait for the root of the process tree to terminate */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

Αφού μεταγλωτίσσαμε το πρόγραμμα μας και ενώσαμε τα δύο object files “proc-common.o” και “ask2-fork.o”, τρέξαμε το εκτελέσιμο μας με όνομα main1 και λάβαμε το ακόλουθο αποτέλεσμα:

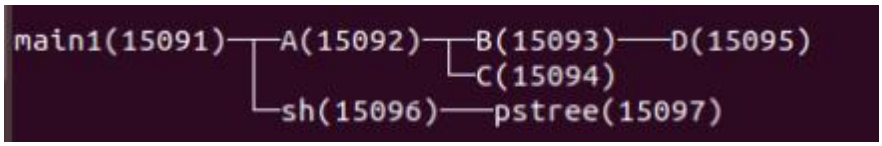
```
oslabb30@orion:~/giorgos/exercise2$ gcc -c ask2-fork.c
oslabb30@orion:~/giorgos/exercise2$ gcc ask2-fork.o proc-common.o -o main1
oslabb30@orion:~/giorgos/exercise2$ ./main1
A: Created!
A: Creating child B...
A: Creating child C...
A: Waiting...
B: Created!
B: Creating child D...
B: Waiting...
D: Created!
D: Sleeping...
C: Created!
C: Sleeping...


A(15107)─┬─B(15108)──D(15110)
         └─C(15109)


D: Exiting...
C: Exiting...
My PID = 15108: Child PID = 15110 terminated normally, exit status = 13
B: Exiting...
My PID = 15107: Child PID = 15109 terminated normally, exit status = 17
My PID = 15107: Child PID = 15108 terminated normally, exit status = 19
A: Exiting...
My PID = 15106: Child PID = 15107 terminated normally, exit status = 16
oslabb30@orion:~/giorgos/exercise2$
```

Ερωτήσεις:

1. Αν τερματίσουμε πρόωρα την διεργασία Α με την δοθείσα εντολή, ενώ θα «σκοτωθεί» η διεργασία Α, τα παιδιά της δεν θα τερματιστούν και θα συνεχίσουν να εκτελούνται. Διεργασίες των οποίων ο «parent» έχει πεθάνει ενώ αυτές ακόμα εκτελούνται ονομάζονται «zombie» και «υιοθετούνται» από την init. Η διεργασία init είναι η πρώτη διεργασία κατά την εκκίνηση του συστήματος και εκτελείται μέχρι τον τερματισμό του. Η init αναλαμβάνει τον τερματισμό των διεργασιών «zombie» κάνοντας απανωτά wait().
2. Γνωρίζουμε ότι η κλήση συστήματος getpid() επιστρέφει το Process ID της καλούμενης διεργασίας και έτσι κάνοντας show_ps_tree(getpid()) αντί για show_ps_tree(pid) θα εμφανιστεί το δέντρο συμπεριλαμβάνοντας την διεργασία του προγράμματος μας. Πιο συγκεκριμένα, το πρόγραμμα μας κάνει fork() μία διεργασία shell, η οποία με την σειρά της καλεί την pstree. Το αποτέλεσμα που προκύπτει είναι το ακόλουθο:



3. Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης, διότι μόνο έτσι μπορεί να προστατεύσει το σύστημα από κακόβουλη χρήση/επίθεση. Για παράδειγμα, θα μπορούσε κάποιος μέσω του forkbomb (https://en.wikipedia.org/wiki/Fork_bomb) να εξαντλήσει τους πόρους του συστήματος καθιστώντας το αδύνατο να λειτουργήσει σωστά.

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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

#include "tree.h"

void create (struct tree_node *root) {
        change_pname(root->name); // allazoume to onoma tis proccess
        int status;

        printf("Proccess %s created! \n" , root->name);
        if (root->nr_children == 0){
                printf("%s: Sleeping\n",root->name);
                sleep(SLEEP_PROC_SEC); // an einai fullo theloume na koimatai
                printf("%s: Exiting\n",root->name);
                exit(1);
        }

        else {
                unsigned i;
                pid_t next;
                for (i=0;i<root->nr_children;i++){
                        next=fork();
                        if (next<0){
                                perror("PROVLIMA\n");
                                exit(1);
                        }
                        if (next==0){
                                create(root->children+i);
                                exit(1);
                        }
                }
                while (i--) {
                        printf("%s: Waiting\n",root->name);
                        next=wait(&status);
                        explain_wait_status(next,status);
                }
        }
        printf("Exiting: %s\n",root->name);
}
```

```c
int main(int argc, char *argv[])
{
        struct tree_node *root;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
                exit(1);
        }
        root = get_tree_from_file(argv[1]);
        print_tree(root);
        int status;

        if (root==NULL) {
                printf("%s\n","Tree is empty");
        }

        else {

                pid_t start = fork();
                if (start==0) { // an eimai sto root
                        create(root);
                        exit(1);
                }
                // AN eimai ektos root
                sleep(SLEEP_TREE_SEC);
                show_pstree(start);
                start = wait(&status);
                explain_wait_status(start, status);

        }

        return 0;
}
```

```
oslabb30@orion:~/giorgos/exercise2$ ./main2 proc.tree
A
        B
                E
                F
        C
        D
Proccess A created!
Proccess B created!
A: Waiting
Proccess C created!
B: Waiting
Proccess E created!
E: Sleeping
C: Sleeping
Proccess D created!
Proccess F created!
F: Sleeping
D: Sleeping


A(12866)─┬─B(12867)─┬─E(12870)
         │          └─F(12871)
         ├─C(12868)
         └─D(12869)


E: Exiting
C: Exiting
D: Exiting
F: Exiting
My PID = 12867: Child PID = 12870 terminated normally, exit status = 1
B: Waiting
My PID = 12867: Child PID = 12871 terminated normally, exit status = 1
Exiting: B
My PID = 12866: Child PID = 12867 terminated normally, exit status = 1
A: Waiting
My PID = 12866: Child PID = 12868 terminated normally, exit status = 1
A: Waiting
My PID = 12866: Child PID = 12869 terminated normally, exit status = 1
Exiting: A
My PID = 12865: Child PID = 12866 terminated normally, exit status = 1
```

Ερωτήσεις:

1. Τα μηνύματα έναρξης των διεργασιών εμφανίζονται όπως βλέπουμε παραπάνω ως εξής: A,B,C,E,D,F, δηλαδή δεν είναι κάποια από τις γνωστές DFS/BFS. Αυτ;ο συμβαίνει, διότι όλες οι εργασίες συμβαίνουν ταυτόχρονα, δεν υπάρχουν σήματα για αναμονή και επομένως τηρείται μόνο το αυτονόητο δηλαδή κάθε παιδί

φτιάχνεται μετά από τον γονέα του. Ομοίως κάθε γονέας τερματίζει αφότου έχουν τερματίσει όλα τα παιδία του.

1.3 Αποστολή και χειρισμός σημάτων

```c
#include <sys/wait.h>

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

void fork_procs(struct tree_node *root)
{
        int status;
        unsigned i;
        pid_t *p;
        printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
        change_pname(root->name);
        int number=root->nr_children;
        if (number==0) {
                printf("%s: %s\n",root->name,"is a leaf");
                raise(SIGSTOP);
                printf("PID = %ld, name = %s is awake\n",(long)getpid(), root->name);
                exit(0);
        }
        else {
                p= calloc(number,sizeof(pid_t));
                // kanoume pinaka pou tha apothikeusoume ta PID twn children
                for (i=0;i<root->nr_children;i++) {
                        pid_t next;
                        next=fork();
                        if (next<0) perror("provlima");
                        if (next==0) {
                                fork_procs(root->children+i);
                                exit(0);
                        }
                        else { //edw theloume na sunexisei mono o goneas gia na kanei save ta PID twn paidiwn
                                p[i]=next;
                        }
                        wait_for_ready_children(1);
                }
                raise(SIGSTOP);

                printf("PID = %ld, name = %s is awake\n",(long)getpid(), root->name);
                for (i=0;i<number;i++) {
                        kill(p[i],SIGCONT);
                        wait(&status);
                        explain_wait_status(p[i],status);
                }
                wait(&status);
                explain_wait_status(getpid(),status);
                exit(0);
        }
}
```

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

        if (argc < 2){
                fprintf(stderr, "Usage: %s <tree_file>\n", argv[0]);
                exit(1);
        }

        /* Read tree into memory */
        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(0);
        }

        wait_for_ready_children(1);
        show_pstree(pid);
        /* for ask2-signals */
        kill(pid, SIGCONT);

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


        return 0;
```

```
oslabb30@orion:~/giorgos/exercise2$ ./main3 proc.tree
PID = 12834, name A, starting...
PID = 12835, name B, starting...
PID = 12836, name E, starting...
E: is a leaf
My PID = 12835: Child PID = 12836 has been stopped by a signal, signo = 19
PID = 12837, name F, starting...
F: is a leaf
My PID = 12835: Child PID = 12837 has been stopped by a signal, signo = 19
My PID = 12834: Child PID = 12835 has been stopped by a signal, signo = 19
PID = 12838, name C, starting...
C: is a leaf
My PID = 12834: Child PID = 12838 has been stopped by a signal, signo = 19
PID = 12839, name D, starting...
D: is a leaf
My PID = 12834: Child PID = 12839 has been stopped by a signal, signo = 19
My PID = 12833: Child PID = 12834 has been stopped by a signal, signo = 19


A(12834)─┬─B(12835)─┬─E(12836)
         │          └─F(12837)
         ├─C(12838)
         └─D(12839)


PID = 12834, name = A is awake
PID = 12835, name = B is awake
PID = 12836, name = E is awake
My PID = 12835: Child PID = 12836 terminated normally, exit status = 0
PID = 12837, name = F is awake
My PID = 12835: Child PID = 12837 terminated normally, exit status = 0
My PID = 12835: Child PID = 12835 terminated normally, exit status = 0
My PID = 12834: Child PID = 12835 terminated normally, exit status = 0
PID = 12838, name = C is awake
My PID = 12834: Child PID = 12838 terminated normally, exit status = 0
PID = 12839, name = D is awake
My PID = 12834: Child PID = 12839 terminated normally, exit status = 0
My PID = 12834: Child PID = 12834 terminated normally, exit status = 0
My PID = 12833: Child PID = 12834 terminated normally, exit status = 0
```

Ερωτήσεις:

1. Στις προηγούμενες ασκήσεις καταφέρναμε να εξασφαλίσουμε αρκετό χρόνο αδρανοποιώντας τα παιδία μέσω της εντολής sleep. Ωστόσο, αυτό ήταν κάπως αυθαίρετο διότι διαλέγαμε χρόνο ώστε να είναι αρκετός άλλα όχι απαραίτητα βέλτιστος (περιττή καθυστέρηση). Η σωστή προσέγγιση είναι μέσω μέσω των σημάτων όπου μια διεργασία ‘παγώνει’ έως ώτου να λάβει μήνυμα έναρξης.
2. Η wait_for_ready_children() είναι μια συνάρτηση που αναστέλλει την λειτουργία της διεργασίας γονέα μέχρι τα παιδιά της να αδρανοποιηθούν. Ελέγχει ουσιαστικά τη σημαία WIFSTOPPED, που δείχνει αν το παιδί έχει σταματήσει και σε αντίθετη περίπτωση εμφανίζει μήνυμα για τον «ξαφνικό» θάνατο του παιδιού και κάνει

exit(). Η χρήση της εξασφαλίζει πως όλα τα παιδιά είναι ζωντανά και έχουν σταματήσει, καθιστώντας πλήρες και σωστό το δέντρο διεργασιών που θα απεικονιστεί. Αν δε τη χρησιμοποιήσουμε, τότε ο πατέρας είναι πιθανό να στείλει μήνυμα SIGCONT σε παιδί που δεν έχει κάνει raise SIGSTOP και άρα το παιδί αυτό θα αδρανοποιηθεί για πάντα αφού δεν θα ξαναλάβει τέτοιο μήνυμα.

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

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

void calculate (struct tree_node *root,int* pipefd) {
        change_pname(root->name);
        int status;
        int new_pipe[2];
        pid_t *p;
        int result;
        int number1,number2;
        unsigned i;
        // Case 1
        if (root->nr_children==0) { //termatikos komvos ara arithmos
                result=atoi(root->name); //metatropi tou string arithmou se int
                sleep(2);
                if (write(pipefd[1],&result,sizeof(result))!=sizeof(result)){
                        perror("apotuxia eggrafis\n");
                        exit(1);
                }
                exit(0);
        }
```

```
        // Case 2
        else { // mi termatikos komvos ara telestis
                if (pipe(new_pipe)<0) { // kataskeui pipe
                        perror("error creating pipe");
                        exit(1);
                }
                pid_t next;
                p=calloc(root->nr_children,sizeof(pid_t));
                for (i=0;i<root->nr_children;i++) {
                        next=fork();
                        if (next<0) {
                                perror("provlima");
                                exit(1);
                        }
                        if (next==0) {
                                close(new_pipe[0]); // to paidi borei mono na grafei
                                calculate(root->children+i,new_pipe);
                                exit(0);
                        }
                        else {
                        p[i]=next;
                        }
                }
                close(new_pipe[1]); // o parent mono diabazei

                for(i=0;i<root->nr_children;i++){ //perimenoume na teliwsoume ta paidia
                        wait(&status);
                        explain_wait_status(p[i],status);
                }
                if (read(new_pipe[0],&number1,sizeof(number1))!=sizeof(number1)) {
                        perror("apotuxia diavasmatos");
                        exit(1);
                }

                if (read(new_pipe[0],&number2,sizeof(number2))!=sizeof(number2)) {
                        perror("apotuxia diavasmatos");
                        exit(1);
                }
                if (strcmp(root->name,"*")==0){
                        result=number1*number2;
                }
                else if (strcmp(root->name,"+")==0){
                        result=number1+number2;
                        printf("%s %d\n","to apotelesma einai",number2);
                }
                if (write(pipefd[1],&result,sizeof(result))!=sizeof(result)) {
                        perror("failed to write to parent"); //apotelesma pra3is gia na to lavei o parent
                        exit(1);
                }
                exit(0);
        }
}
```

```c
int main (int argc, char* argv[]) {
        int result;
        int fds[2];
        pid_t pid;
        struct tree_node *root;
        int status;
        if (argc!=2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
                exit(1);
        }
        root=get_tree_from_file(argv[1]);
        if (pipe(fds)<0) { // an to pipe epistrepsei -1 exoume error kanonika prepei na epistrefei 0
                perror("pipe in main");
                exit(1);
        }
        pid=fork();
        if (pid<0) {
                perror("Provlima stin main");
                exit(1);
        }
        if (pid==0) {
                close(fds[0]); // kathe child einai write-only
                calculate(root,fds);
                exit(0);
        }
        close(fds[1]); //kathe goneas einai read-only
        sleep(1);
        show_pstree(pid);
        wait(&status);
        explain_wait_status(pid, status);
        if (read(fds[0],&result,sizeof(result)) != sizeof(result)){ //read from the pipe the final result
                perror("read from pipe final \n");
                exit(1);
        }
        printf("Teliko apotelesma dentrou %d\n",result);
        return 0;
}
```

```
oslabb30@orion:~/giorgos/exercise2$ ./main4 expr.tree


+(12729)─┬─*(12731)─┬─+(12732)─┬─5(12734)
         │          │          └─7(12735)
         │          └─4(12733)
         └─10(12730)


My PID = 12729: Child PID = 12730 terminated normally, exit status = 0
My PID = 12731: Child PID = 12732 terminated normally, exit status = 0
My PID = 12732: Child PID = 12734 terminated normally, exit status = 0
My PID = 12732: Child PID = 12735 terminated normally, exit status = 0
to apotelesma einai 7
My PID = 12731: Child PID = 12733 terminated normally, exit status = 0
My PID = 12729: Child PID = 12731 terminated normally, exit status = 0
to apotelesma einai 48
My PID = 12728: Child PID = 12729 terminated normally, exit status = 0
Teliko apotelesma dentrou 58
```

Ερωτήσεις:

1. Στην παρούσα άσκηση απαιτείται η χρήση ακριβώς μίας σωλήνωσης. Αυτό οφείλεται στο γεγονός ότι το δέντρο διασχίζεται κατά βάθος (DFS) και φτάνοντας στα φύλλα με το μεγαλύτερο βάθος γίνονται οι πράξεις της πρόσθεσης και του πολλαπλασιασμού, που είναι **αντιμεταθετικές πράξεις**. Στην περίπτωση που γινόντουσαν πράξεις όπως αφαίρεση και διαίρεση όπου δεν ισχύει η αντιμεταθετική ιδιότητα θα χρειαζόντουσαν τουλάχιστον δύο σωληνώσεις.
2. Σε ένα υπολογιστικό σύστημα με πολλαπλούς επεξεργαστές οι διεργασίες θα εκτελούνταν παράλληλα οδηγώντας σε σημαντική μείωση χρόνου σε σχέση με την αποτίμηση της έκφρασης από μία μόνο διεργασία.