ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα Υπολογιστών
Αναφορά 2ης Εργαστηριακής Άσκησης
Απρίλιος 2021

Λαγός Αναστάσιος - el13531
Κωνσταντίνος Βασιλάκης - el16504

1 Δημιουργία δεδομένου δέντρου διεργασιών

Παρακάτω ο κώδικας του προγράμματος. Η έξοδος φαίνεται στην εικόνα 1.

```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 INIT_PROC_SEC   1
#define SLEEP_TREE_SEC  2


void fork_procs(char *name, int exitno){
        printf("%s: Sleeping...\n", name);
        sleep(SLEEP_PROC_SEC);

        printf("%s: Exiting...\n", name);
        exit(exitno);
}

int main() {
        pid_t pid;
        int status;
    printf("A: Init...\n");
        pid = fork();       //A is created here.
        if (pid < 0) {
                perror("main: fork\n"); //Check errors forking father
                ↪  of A.
                exit(1);
        }
        else if (pid == 0) {        //pid_father == 0  means that this
        ↪  process is a child. In this case A.
        change_pname("A");
        printf("B: Init...\n");
        pid = fork();   //A forks to B.
        if (pid < 0) {
            perror("A: fork -> B\n");
            exit(1);
        }
        else if (pid == 0) {    //B node will enter here.
            change_pname("B");
            printf("D: Init...\n");
            pid = fork();   //D is created here.
```

```
            if (pid < 0) {
                perror("B: fork\n");
                exit(1);
            }
            if (pid == 0) { //D will enter here.
                change_pname("D");
                fork_procs("D", 13);
            }
            printf("B: Waiting for child to terminate...\n");
            change_pname("B");
            pid = wait(&status);
            explain_wait_status(pid, status);
            printf("B: Exiting...\n");
            exit(19);
        }
        printf("C: Init...\n");
        pid = fork();   //C is created here.
        if (pid < 0) {
            perror("A: fork -> C\n");
            exit(1);
        }
        else if (pid == 0) {
            change_pname("C");
            fork_procs("C", 17);
        }
        printf("A: Waiting for child to terminate...\n");
        pid = wait(&status);
        explain_wait_status(pid, status);
        printf("A: Waiting for child to terminate...\n");
        pid = wait(&status);
        explain_wait_status(pid, status);
        printf("A: Exiting...\n");
        exit(16);
        }
    sleep(SLEEP_TREE_SEC);
    show_pstree(getpid());
    pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

1.1 Ερωτήσεις

1 - Τι θα γίνει αν τερματίσετε πρόωρα τη διεργασία A, δίνοντας kill -KILL $< pid >$, όπου $< pid >$ το Process ID της?
- Οποιαδήποτε διεργασία παιδί δεν έχει τερματίσει ακόμα θα κληρονομηθεί από την διεργασία init (PID = 1).

Figure 1: Έξοδος άσκησης 1 με show_pstree(getpid())

2 - Τι γίνεται αν κάνετε show_pstree(getpid()) αντί για show_pstree(pid) στη main()? Ποιές επιπλέον διεργασίες φαίνονται στο δέντρο και γιατί?
- Με την show_pstree(getpid()) θα εμφανιστεί το process tree της main και όχι της A. Η sh(shell) και η pstree δημιουργούνται από την εντολή system(cmd) στην show_pstree για να εμφανίσουν το δέντρο στο terminal.
3 - Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί?
- Ο λόγος είναι πεπερασμένη χωρητικότητα (μνήμη RAM) αλλά και λόγω του ότι μπορούν να δωθούν πεπερασμένος αριθμός PID που εξαρτάται από το datatype pid_t.

2 Δημιουργία αυθαίρετου δέντρου διεργασιών

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

```c
#include "tree.h"

#define SLEEP_PROC_SEC 10
#define OBSV_SLEEP 1

void prong(struct tree_node *node) {
    int status;
    pid_t pid;
    int prongno = node->nr_children;
    if (node->children != NULL) {       //If node != leaf
        while(prongno--) {      //for all children
            pid = fork();       //create them
            if (pid < 0) {      //Error handling
                printf("%s: i have a", node->name);
                perror("pronging error");
                exit(1);
            }
            else if(pid == 0) { //children enter here
                printf("%s: init...\n", (node->children + prongno)->name);
                prong(node->children + prongno); //now im the root of my own
                ↪  tree!
            }
        }
        sleep(OBSV_SLEEP); //give some time to reach end of tree
        printf("%s: waiting for all children to terminate...\n", node->name);
        while((pid = wait(&status)> 0));//wait for all children to finish.
        ↪  wait(&status) returns -1 for no children.
        printf("%s: all children terminated...exiting...\n", node->name);
        exit(1);
    }
    //leafs execute these
    printf("%s: im a leaf! and im sleeping!\n", node->name);
    sleep(SLEEP_PROC_SEC);
    printf("%s: im exiting...\n", node->name);
    exit(1);
}

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\n", argv[0]);
        exit(1);
    }

    root = get_tree_from_file(argv[1]);
```

```c
    pid = fork();   //fork from main to create root process
    if (pid < 0) {
        perror("main fork error");
        exit(1);
    }
    else if (pid == 0){ //root process enters here
        prong(root);
    }
    pid = wait(&status);
    print_tree(root);

    return 0;
}
```

Παρακάτω οι έξοδοι του προγράμματος για διάφορα δέντρα:

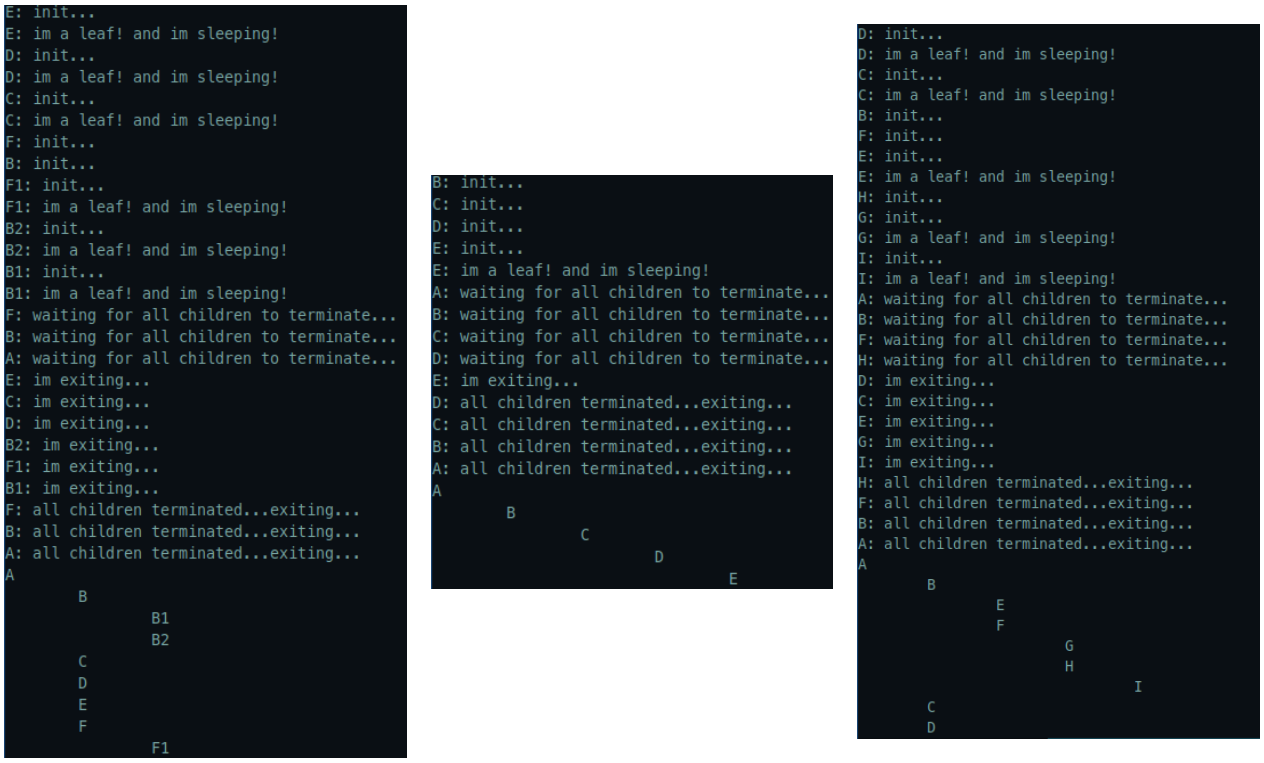


Figure 2: Έξοδος για ενδεικτικά δέντρα εισόδου(αυθαίρετο δέντρο διεργασιών)

2.1 Ερωτήσεις

1 - Με ποιά σειρά εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών?Γιατί?
- Τα μηνύματα έναρξης εμφανίζονται με διαφορετική σειρά απο τα μηνύματα τερματισμού. Αυτό συμβαίνει λόγω του scheduler, ένα στοιχείο του kernel που επιλέγει ποιά διεργασία θα εκτελεστεί.

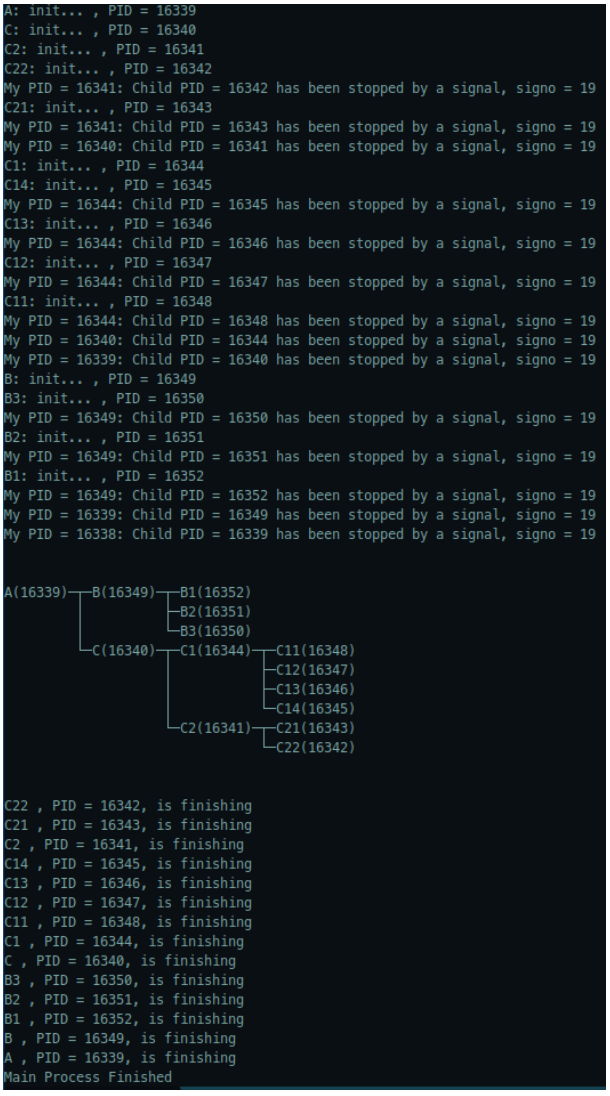
3 Αποστολή και χειρισμός σημάτων

```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 <string.h>

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

void fork_procs(struct tree_node *node)
{
    int status;
    pid_t pid;
    pid_t* child_pids = malloc(node->nr_children * sizeof(pid_t));

    int numberOfChildren = node->nr_children;

    change_pname(node->name);
    printf("%s: init... , PID = %ld\n", node->name, (long) getpid());
    if (node->children != NULL) {       //If node != leaf
        while(numberOfChildren--) {     //for all children
            pid = fork();       //Fork the process

            if (pid < 0) {      //Error handling
                perror("pronging error");
                exit(1);
            }
            else if(pid == 0) { //children enter here
                fork_procs(node->children + numberOfChildren); //now im the
                ↪  root of my own tree!
            } else {
                //If this is the parent of the fork, save the child's pid and
                ↪  wait for it to stop
                child_pids[numberOfChildren-1] = pid;
                wait_for_ready_children(1);
            }
        }
    }

    //Raise the SIGSTOP signal to the process and stop it
    raise(SIGSTOP);

```

```c
        //When the SIGCONT comes from the parent each node signals it's child
        ↪  nodes if it has
        numberOfChildren = node->nr_children;

        if (node->children != NULL) {
            while(numberOfChildren--) {
                kill(child_pids[numberOfChildren-1], SIGCONT);
                    //For each of the children processes that continue, the parent
                    ↪  process waits for them to exit before it continues
                pid = wait(&status)> 0;
                //explain_wait_status(pid, status);
            }
        }

        printf("%s , PID = %ld, is finishing\n", node->name, (long)getpid());

        exit(0);
    }

    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);
        }

        //Waiting the SIGSTOP from the child to continue
        wait_for_ready_children(1);

        show_pstree(pid);
```

```
90
91          kill(pid, SIGCONT);
92
93          /* Wait for the root of the process tree to terminate */
94          wait(&status);
95      //      explain_wait_status(pid, status);
96
97          printf("Main Process Finished\n");
98          return 0;
99      }
```

Παρακάτω οι έξοδοι του προγράμματος για διάφορα δέντρα:



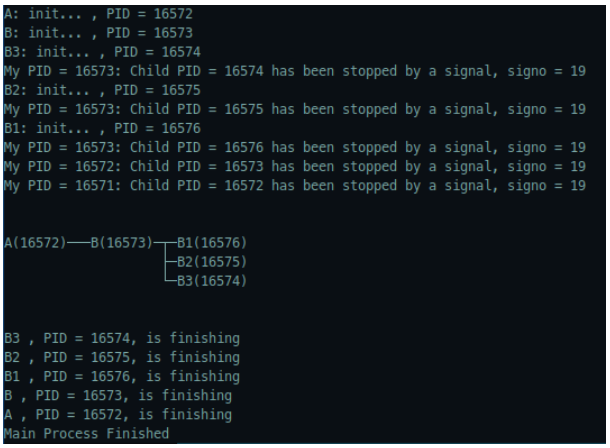


Figure 3: Έξοδος για ενδεικτικά δέντρα εισόδου(αποστολή και χειρισμός σημάτων)

3.1 Ερωτήσεις

1 - Στις προηγούμενες ασκήσεις χρησιμοποιούσαμε την sleep() για συγχρονισμό των διεργασιών. Τι πλεονέκτημα έχει η χρήση σημάτων?
- Χρησιμοποιώντας σήματα μπορούμε να επικοινωνήσουμε με τα διάφορα processes την στιγμή που χρειάζεται, οπότε να διατηρήσουμε έναν έλεγχο στην σειρά εκτέλεσης.

2 - Ποιός ο ρόλος της wait_for_read_children()? Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψη της?
- Η συνάρτηση αυτή χρησιμεύει στο να ελέγξουμε αν κάποιο child process έχει γίνει stop εκτός από exit. Χρησιμοποιείται στον κώδικα για να επιστρέψει τον έλεγχο στον parent που έχει κάνει wait μέχρι να ολοκληρώσει το processing κάποιο παιδί. Με την παράλειψη της δεν θα υπάρχει συγχρονισμός μεταξύ των παιδιών που έκαναν `raise(SIGSTOP)`[γραμμή 42] και περιμένουν, και με την εντολή του parent `kill(child_pid,SIGCONT)`[γραμμή 49,91].

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 <string.h>

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

void fork_procs(struct tree_node *node,int *pfd)
{
    ssize_t rd_error;
    if (node->children != NULL) {       //If node != leaf

        pid_t pid;
        int p0fd[2],p1fd[2];

        //Create pipes
        if (pipe(p0fd) < 0 || pipe(p1fd) < 0 ) {
            perror("Error creating pipe");
            exit(1);
        }

        //If the node has children it has two by the definition
        for (int i = 0; i < 2 ; i++) {
            pid = fork();       //Fork the process

            if (pid < 0) {      //Error handling
                perror("Error creating fork");
```

```
        exit(1);
    } else if(pid == 0) {   //children enter here
        //Each child closes the read endpoint of the pipe
        if (i == 0) {
            close(p0fd[0]);
            fork_procs(node->children + i,p0fd);
        } else if (i == 1) {
            close(p0fd[0]);
            fork_procs(node->children + i,p1fd);
        }
    }
}

//Parent code
//Parent closes the write endpoint of the pipe
close(p0fd[1]);
close(p1fd[1]);

int child0Value, child1Value;

//We read the two values one for each pipe. These pipes block till
↪  bytes are written to the pipe
rd_error = read(p0fd[0], &child0Value, sizeof(int));
if (rd_error < 0) {
    perror("Reading error: ");
    exit(1);
}
close(p0fd[0]);

rd_error = read(p1fd[0], &child1Value,sizeof(int));
if (rd_error < 0) {
    perror("Reading error: ");
    exit(1);
}
close(p1fd[0]);

//The result is calculated according to the operator on the node
int result;

if (strcmp("*",node->name) == 0) {
    result = child0Value * child1Value;
    printf("%d * %d = %d, PID = %ld\n", child0Value, child1Value,
    ↪  result, (long) getpid());
} else if (strcmp("+",node->name) == 0) {
    result = child0Value + child1Value;
    printf("%d + %d = %d, PID = %ld\n", child0Value, child1Value,
    ↪  result, (long) getpid());
} else {
```

```
            perror("Incorrect operator");
            exit(1);
        }

        //The current parent writes the result to its parent
        rd_error = write(pfd[1], &result , sizeof(int));
        if (rd_error < 0) {
            perror("Writing error: ");
            exit(1);
        }
    } else {
        //If we are at a leaf this should be a value so we pass this through
        ↪  the pipe
        int leafValue;
        sscanf(node->name, "%d", &leafValue);

        rd_error = write(pfd[1], &leafValue, sizeof(int));\
        if (rd_error < 0) {
            perror("Writing error");
            exit(1);
        }
        close(pfd[1]);
    }

    exit(0);
}

int main(int argc, char *argv[])
{
    ssize_t rd_error;
    pid_t pid;
    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]);

    int pfd[2];

    if (pipe(pfd) < 0) {
        perror("Error creating pipe");
        exit(1);
    }
```

```c
    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        close(pfd[0]);
        fork_procs(root,pfd);
    }

    close(pfd[1]);

    int result;
    rd_error = read(pfd[0], &result, sizeof(int));
    if (rd_error < 0){
        perror("Reading error main: ");
        exit(1);
    }

    printf("Final Result = %d\n", result);

    return 0;
}
```

Παρακάτω οι έξοδοι:

```
5 + 7 = 12, PID = 20354
12 * 4 = 48, PID = 20353
10 + 48 = 58, PID = 20351
Final Result = 58
```

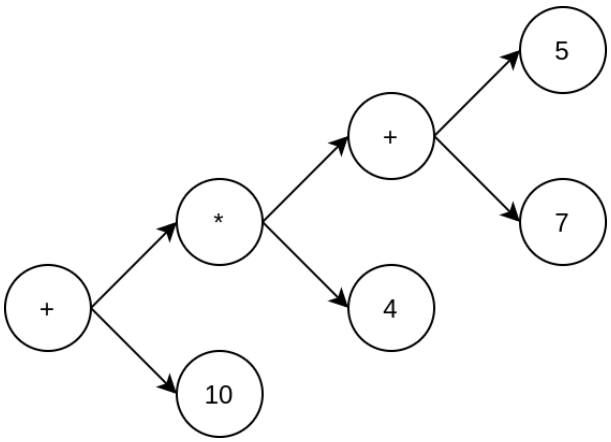


Figure 4: Παράδειγμα εξόδου 1

4.1 Ερωτήσεις

1 - Πόσες σωληνώσεις χρειάζονται στη συγκεκριμένη άσκηση ανά διεργασία? Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μια σωλήνωση για όλες τις διεργασίες παιδιά? Γενικά, μπορεί για κάθε αριθμητικό τελεστή να χρησιμοποιηθεί μόνο μια σωλήνωση?

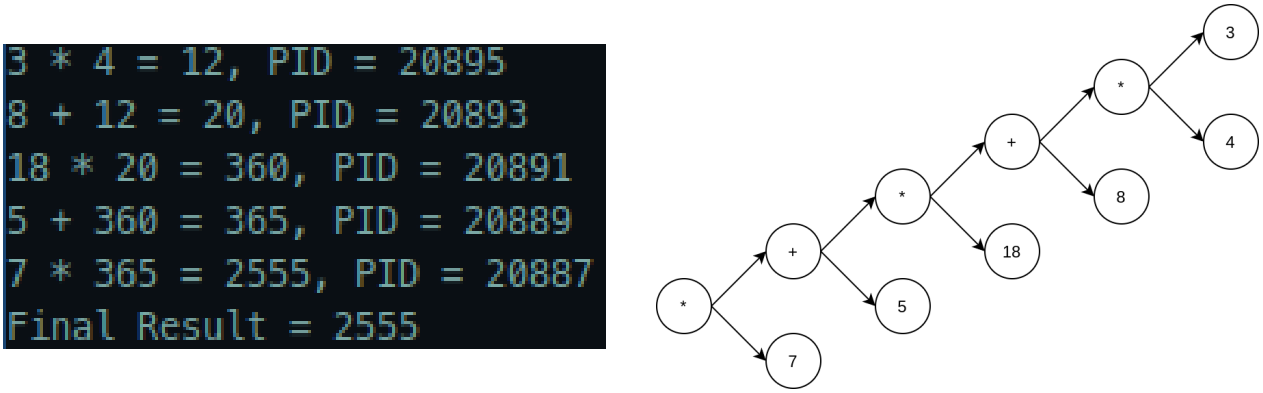


Figure 5: Παράδειγμα εξόδου 2

- Χρειάζεται μια σωλήνωση ανά διεργασία. Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μια σωλήνωση για όλες τις διεργασίες παιδιά αλλά μόνο όταν δεν μας ενδιαφέρει η σειρά με την οποία διαβάζουμε. Συγκεκριμένα σε αριθμητικούς τελεστές με την αντιμεταθετική ιδιοτητα δεν μας ενδιαφέρει η σειρά, αυτό δεν ισχύει για τελεστές που δεν είναι αντιμεταθετικοί. Επίσης, υπάρχουν και οι εξής περιπτώσεις που χρειάζονται προσοχή όταν χρησιμοποιούμε μια σωλήνωση με περισσότερα απο ένα παιδιά:

1. Υπάρχει ένα buffer size πάνω απο το οποίο τα writes δεν είναι ορισμένα. Δηλαδή ένα μήνυμα γράφεται μέχρι την μέση και ολοκληρώνεται μετά από κάποια άλλη σειρά μηνυμάτων.

2. Αν τα μηνύματα έχουν διαφορετικό μήκος ανάλογα με το παιδί τότε ο γονέας δεν ξέρει ακριβώς τι να διαβάσεις.

2 - Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορούν να εκτελούνται παραπάνω απο μια διεργασίες παράλληλα. Σε ένα τέτοιο σύστημα, τι πλεονέκτημα μπορεί να έχει η αποτίμηση της έκφρασης από δέντρο διεργασιών, έναντι της αποτίμησης απο μια μόνο διεργασία?
- Έχοντας πολλές διεργασίες μπορούμε να εκμεταλλευτούμε τις δυνατότητες παράλληλης επεξεργασίας που έχει το σύστημα και να επιταχύνουμε τους υπολογισμούς.