**Λειτουργικά Συστήματα**

**Άσκηση 2: Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία**

**1.1 Δημιουργίαδεδομένουδέντρουδιεργασιών**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <assert.h>

#include <sys/types.h>

#include <sys/wait.h>

#include "proc-common.h"

#define SLEEP\_PROC\_SEC 5

#define SLEEP\_TREE\_SEC 2

/\*

\* Create this process tree:

\* A-+-B---D

\* `-C

\*/

void fork\_procsC(void)

{

change\_pname("c");

sleep(SLEEP\_PROC\_SEC);

printf("C: Exiting...\n");

exit(17);

}

void fork\_procsD(void)

{

change\_pname("D");

sleep(SLEEP\_PROC\_SEC);

printf("D: Exiting...\n");

exit(13);

}

void fork\_procsB(void)

{

change\_pname("B");

pid\_tpid;

int status;

pid = fork();

if (pid< 0) {

perror("main: fork");

exit(1);

}

if (pid == 0) {

fork\_procsD();

exit(1);

}

pid = wait(&status);

explain\_wait\_status(pid, status);

printf("B: Exiting...\n");

exit(19);

}

void fork\_procs(void)

{

change\_pname("A");

pid\_t pid1,pid2;

int status1,status2;

pid1 = fork();

if (pid1 < 0) {

perror("main: fork");

exit(1);

}

if (pid1 == 0) {

fork\_procsB();

exit(1);

}

pid2 = fork();

if (pid2 < 0) {

perror("main: fork");

exit(1);

}

if (pid2 == 0) {

fork\_procsC();

exit(1);

}

pid1 = wait(&status1);

explain\_wait\_status(pid1, status1);

pid2 = wait(&status2);

explain\_wait\_status(pid2, status2);

printf("A: Exiting...\n");

exit(16);

}

int main(void)

{

pid\_tpid;

int status;

pid = fork();

if (pid< 0) {

perror("main: fork");

exit(1);

}

if (pid == 0) {

/\* Child \*/

fork\_procs();

exit(1);

}

sleep(SLEEP\_TREE\_SEC);

/\* Print the process tree root at pid \*/

show\_pstree(pid);

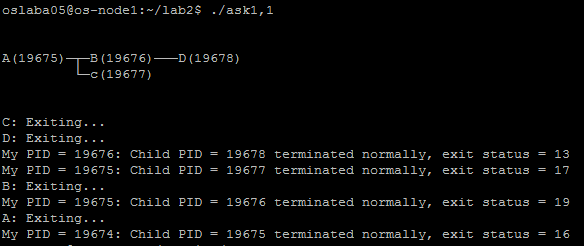
/\* Wait for the root of the process tree to terminate \*/

pid = wait(&status);

explain\_wait\_status(pid, status);

return 0;

}

Παρακάτω φαίνεται η έξοδος του προγράμματος

**Απαντήσεις**:

1. Τερματίζοντας μια διεργασία, έστω την “Α” που έχει παιδιά, με “kill –Kill<pid>” τότε οι θυγατρικές διεργασίες ανατίθενται στην “init”. Η διεργασία που δημιούργησε την “Α” δεν ενημερώνεται για τον τερματισμό της. Αυτό γίνεται διότι το σήμα –Kill (-9) προέρχεται από τον πυρήνα και δεν μπορεί μία διεργασία να το χειριστεί. Η διεργασία που δημιούργησε την “Α” περιμένει για κάποιο χρονικό διάστημα μέχρι το wait() να γίνει time-out, δηλαδή να σταματήσει να περιμένει καθώς η διεργασία “Α” δεν βρέθηκε.
2. Με την show\_pstree(getpid()), θα έχουμε ως κεφαλή του δέντρου μας την final()(όνομα εκτελέσημου αρχείου, στη περίπτωσή μας ask1,1) αντί τη διεργασία “Α”. Οπότε θα εμφανιστούν στο δέντρο διεργασιών εκτός από τις διεργασίες παιδία που δημιουργήσαμε και οι διεργασίες “sh” και “pstree” οι οποίες θα είναι επίσης παιδιά της final(). Ο λόγος είναι γιατί καλώντας με κεφαλή τo PID της main, η show\_pstree θα κάνει κάποια systemcalls, τα οποία θα δημιουργήσουν τις παραπάνω διεργασίες και οπότε θα είναι παιδιά της final().
3. Αυτό γίνεται διότι υπάρχει η πιθανότητα ένας χρήστης να δημιουργήσει μια διεργασία που να δημιουργεί με την σειρά της ανεξέλεγκτα πολλές διεργασίες. Έτσι θέτοντας όριο στις διεργασίες που δημιουργούνται από κάθε χρήστη ο διαχειριστής επιτυγχάνει με αυτό το τρόπο ένα είδος μηχανισμού ελέγχου και περιορισμού της υπολογιστικής ισχύος που χρησιμοποιείται από κάθε χρήστη με αποτέλεσμα μια πιο δίκαιη κατανομή της ισχύος.

**1.2Δημιουργίααυθαίρετουδέντρουδιεργασιών**

O κώδικας φαίνεται πιο κάτω:

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <sys/types.h>

#include <sys/wait.h>

#include "tree.h"

#include "proc-common.h"

#define SLEEP\_PROC\_SEC 10

#define SLEEP\_TREE\_SEC 4

void fork\_procs(struct tree\_node \*root)

{

change\_pname(root->name); //dinoumeonomastindiergasia

if (root->nr\_children> 0) //an exeipediaksekinanakanei fork

{

pid\_tpid[root->nr\_children];

int status[root->nr\_children],i;

for (i=0; i< root->nr\_children; i++)

{

pid[i] = fork();

if (pid[i] < 0) {

perror("main: fork");

exit(1);

}

if (pid[i] == 0) {

fork\_procs(root->children + i); //arxizoumetidimiourgiadiergasiwn

exit(1);

}

}

for (i=0; i< root->nr\_children; i++)

{

pid[i] = wait(&status[i]); //perimeno ton prwtokomvo

explain\_wait\_status(pid[i], status[i]);

}

}

else sleep(SLEEP\_PROC\_SEC); //idergasiakanei sleep k termatizei

exit(10);

}

intmain(intargc, char \*argv[])

{

struct tree\_node \*root;

//elexosotiiparxeiarxio

if (argc != 2) {

fprintf(stderr, "Usage: %s <input\_tree\_file>\n\n", argv[0]);

exit(1);

}

root = get\_tree\_from\_file(argv[1]); //an iparxe, dimiourgoume to structure k pernoumetiriza

pid\_tpid;

int status;

pid = fork(); //dimiourgiaprwtoukomvou

if (pid< 0) {

perror("main: fork");

exit(1);

}

if (pid == 0) {

fork\_procs(root); //arxizoumetidimiourgiadiergasiwn

exit(1);

}

sleep(SLEEP\_TREE\_SEC); //parousiasidentroudiergasiwn

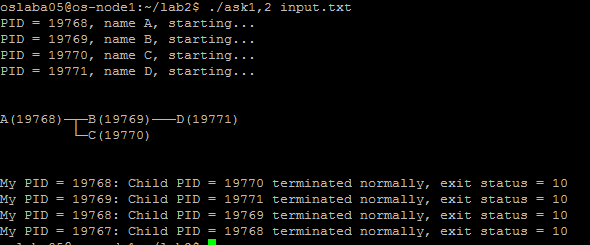
show\_pstree(pid);

pid = wait(&status); //perimeno ton prwtokomvo

explain\_wait\_status(pid, status);

return 0;

}

Παρακάτω φαίνεται η έξοδος του προγράμματος

**Απαντήσεις**:

1. Η σειρά εμφάνισης των μηνυμάτων έναρξης και τερματισμού, όπως φαίνεται και πιο πάνω, γίνεται με τυχαία σειρά. Το μόνο σίγουρο είναι ότι πριν τερματιστεί μια διεργασία έχουν προηγουμένως τερματιστεί όλες οι διεργασίες-παιδια της.

**1.3Αποστολή και χειρισμός σημάτων**

O κώδικας φαίνεται πιο κάτω:

#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 fork\_procs(struct tree\_node \*root)

{

inti;

printf("PID = %ld, name %s, starting...\n",

(long)getpid(), root->name);

change\_pname(root->name);

if (root->nr\_children> 0)

{

pid\_tpid[root->nr\_children];

int status[root->nr\_children];

for (i=0; i< root->nr\_children; i++)

{

pid[i] = fork();

if (pid[i] < 0) {

perror("main: fork");

exit(1);

}

if (pid[i] == 0) {

fork\_procs(root->children + i);

exit(1);

}

printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to terminate...\n",(long)getpid(),(long)pid[i]);

wait\_for\_ready\_children(1);

}

raise(SIGSTOP);

printf("PID = %ld, name = %s is awake\n",

(long)getpid(), root->name);

for (i=0; i< root->nr\_children; i++)

{

kill(pid[i],SIGCONT);

pid[i] = wait(&status[i]);

explain\_wait\_status(pid[i], status[i]);

}

}

else {raise(SIGSTOP); printf("PID = %ld, name = %s is awake\n",

(long)getpid(), root->name); }

exit(0);

}

intmain(intargc, char \*argv[])

{

pid\_tpid;

int status;

struct tree\_node \*root;

//elexosotiiparxeiarxio

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]);

pid = fork(); //dimiourgiaprwtoukomvou

if (pid< 0) {

perror("main: fork");

exit(1);

}

if (pid == 0) {

/\* Child \*/

fork\_procs(root); //arxizoumetidimiourgiadiergasiwn

exit(1);

}

wait\_for\_ready\_children(1); //perimeneina alla3ei //katastasi to paidi tis (diladinakanei stop i root)

show\_pstree(pid); //etsiexoun //dimiourgitheiola ta paidia kai tipwnw to dentro

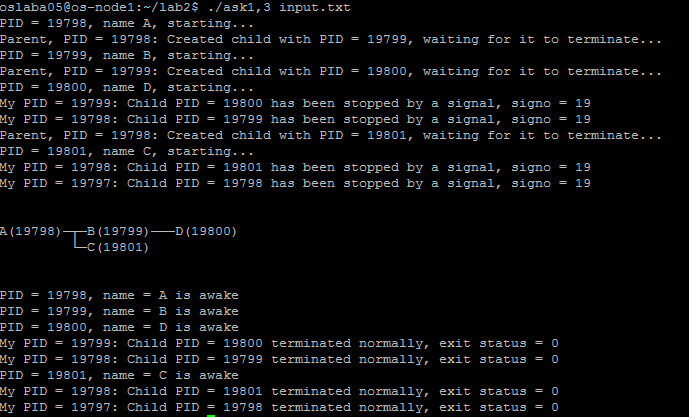
kill(pid, SIGCONT); //stelnwshmastopaidi ///gianasinexistoun oi diergasies

wait(&status); //perimeno ton //prwtokomvonapethanei

explain\_wait\_status(pid, status);

return 0;

}

Παρακάτω φαίνεται η έξοδος του προγράμματος

**Απαντήσεις:**

1. Χρησιμοποιώντας την sleep() μια διεργασία “κοιμάται” για προκαθορισμένο χρονικό διάστημα. Σε κάθε περίπτωση η διεργασία θα γίνει “ready” μόνο όταν περάσει αυτό το χρονικό διάστημα. Η χρήση των σημάτων μας επιτρέπει να καθορίσουμε εμείς σε ποιο σημείο της εκτέλεσης του κώδικα θα γίνει η διεργασία “ready”. Με αυτό τον τρόπο μια διεργασία δεν πρόκειται να γίνει “ready” νωρίτερα από ότι χρειάζεται για την ορθή εκτέλεση του προγράμματος. Επίσης δεν πρόκειται να παραμείνει σε κατάσταση αναμονής για περισσότερο χρόνο από όσο χρειάζεται.
2. Ο ρόλος της wait\_for\_ready\_children(n) είναι να περιμένει μέχρι n-παιδιά της διεργασίας που την καλεί, να στείλουν μήνυμα SIGSTOP. Τελικά, με την χρήση της συνάρτησης αυτής εξασφαλίζουμε ότι μια διεργασία – πατέρας θα περιμένει όλα τα παιδιά του να δημιουργηθούν και να μπουν σε κατάσταση αναμονής. Με τον τρόπο που την χρησιμοποιούμε στον κώδικά μας εξασφαλίζουμε, επίσης, και το Depth-First. Η τυχόν παράλειψη αυτής της εντολής θα είχε την εξής συνέπεια. Κάθε διεργασία-πατέρας θα δημιουργούσε τα παιδιά της και πιθανότατα θα τερμάτιζε πριν προλάβουν τα παιδιά της τα τεραματιστούν. Τελικά, το δέντρο διεργασιών δεν θα ήταν το επιθυμητό. Κάποιες διεργασίες θα ανατίθονταν στην “init”, ενώ, δεν θα μπορούσαμε να εξασφαλίσουμε ούτε διάσχυση κατά βάθος.

**1.4Παράλληλος υπολογισμός αριθμητικής έκφρασης**

O κώδικας φαίνεται πιο κάτω:

#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 fork\_procs(struct tree\_node \*root, int \*pfd)

{

printf("PID = %ld, name %s, starting...\n",

(long)getpid(), root->name);

change\_pname(root->name); //dinwonomasti //diergasia

intnum = atoi (root->name);

if (root->nr\_children> 0) //an exeipediaksekinana //kanei fork

{

pid\_tpid[2];

int status[2],ch\_pfd[2],i,num1,num2,ans;;

if (pipe(ch\_pfd) < 0) {

perror("pipe");

exit(1);

}

for (i=0; i< 2; i++)

{

pid[i] = fork();

if (pid[i] < 0) {

perror("main: fork");

exit(1);

}

if (pid[i] == 0) {

fork\_procs(root->children + i,ch\_pfd);

exit(1);

}

}

for (i=0; i< 2; i++){

pid[i] = wait(&status[i]);

explain\_wait\_status(pid[i], status[i]);

}

if (read(ch\_pfd[0], &num1, sizeof(int) ) != sizeof(int) ) {

perror("read from pipe");

exit(1);

}

if (read(ch\_pfd[0], &num2, sizeof(int) ) != sizeof(int) ) {

perror("read from pipe");

exit(1);

}

int x = root->name[0];

if (x == 42) ans = (num1)\*(num2);

else if (x == 43) ans = num1+num2;

else exit(1);

printf("%d %d %d\n\n",ans,num1,num2);

if (write(pfd[1],&ans, sizeof(int)) != sizeof(int) ) {

perror("write to pipe");

exit(1);

}

}

else {

if (write(pfd[1],&num, sizeof(int)) != sizeof(int) ) { //writes to the pipe

perror("write to pipe");

exit(1);

}

}

exit(0);

}

intmain(intargc, char \*argv[])

{

pid\_tpid;

intstatus,ans,pfd[2];

struct tree\_node \*root;

//elexosotiiparxeiarxio

if (argc<2){

fprintf(stderr, "Usage: %s <tree\_file>\n", argv[0]);

exit(1);

}

root = get\_tree\_from\_file(argv[1]); //an iparxei //dimiourgoume to structure

print\_tree(root);

if (root==NULL) //an einai //adeio exit

exit(1);

if (pipe(pfd) < 0) {

perror("pipe");

exit(1);

}

pid = fork(); //dimiourgiaprwtoukomvou

if (pid< 0) {

perror("main: fork");

exit(1);

}

if (pid == 0) {

fork\_procs(root,pfd); //arxizoumeti //dimiourgiadiergasiwn

exit(1);

}

wait(&status);

explain\_wait\_status(pid, status);

if (read(pfd[0], &ans, sizeof(ans)) != sizeof(ans)) {

perror("read from pipe");

exit(1);

}

printf("The answer is: %d\n\n",ans);

return 0;

}

Παρακάτω φαίνεται η έξοδος του προγράμματος

**Απαντήσεις:**

1. Στην άσκηση αυτή μπορούμε να έχουμε μέχρι και ένα pipe ανά κόμβο-τελεστή. Αυτό επιτρέπεται καθώς η πρόσθεση και ο πολλαπλασιασμός είναι αντιμεταθετικές πράξεις. Αυτό δεν μπορεί να ισχύει στην περίπτωση που είχαμε σαν τελεστές την αφαίρεση και την διαίρεση, διότι παίζει ρόλο η σειρά των πράξεων.
2. Το πλεονέκτημα στην περίπτωση αυτή είναι πως μπορούμε να έχουμε παράλληλη εκτέλεση των διεργασιών με αποτέλεσμα η τελική αποτίμηση να είναι ταχύτερη από ότι αν εκτελείτο από μία διεργασία.Όμως χρειάζεται χρόνος για την δημιουργία κάθε διεργασίες και αυτό είναι το τίμημα που πληρώνουμε.

O κώδικας του **Makefile** φαίνεται πιο κάτω:

.PHONY: all clean

all: ask1,1 ask1,2 ask1,3 ask1,4

CC = gcc

CFLAGS = -g -Wall -O2

SHELL= /bin/bash

ask1,1: ask1,1.o proc-common.o

$(CC) $(CFLAGS) $^ -o $@

ask1,2: ask1,2.otree.o proc-common.o

$(CC) $(CFLAGS) $^ -o $@

ask1,3: ask1,3.otree.o proc-common.o

$(CC) $(CFLAGS) $^ -o $@

ask1,4: ask1,4.otree.o proc-common.o

$(CC) $(CFLAGS) $^ -o $@

%.s: %.c

$(CC) $(CFLAGS) -S -fverbose-asm $<

%.o: %.c

$(CC) $(CFLAGS) -c $<

%.i: %.c

gcc -Wall -E $< | indent -kr> $@

clean:

rm -f \*.o ask1,1 ask1,2 ask1,3 ask1,4 -{fork,tree,signals,pipes}