

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

**2η εργαστηριακή αναφορά**

στο μάθημα **“Λειτουργικά συστήματα”** 6ου Εξαμήνου

Από τους φοιτητές

Άγγελος Γκίκας , ΑΜ: 03118218

Νικήτας Τσίννας, ΑΜ: 03118187

Ομάδα: oslaba14

Προθεσμία παράδοσης: 20/4/2021

Άσκηση 1: Δημιουργία δεδομένου δέντρου διεργασιών

Ακολουθεί το printout του κώδικα:

#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.*

*\*/*

change\_pname("A");

printf("A: Started**\n**");

pid\_t p;

int status;

p = fork();

**if** (p < 0) {

perror("fork");

exit(1);

}

**if** (p == 0) {

*/\**

*\* in B proccess*

*\*/*

change\_pname("B");

printf("B: Started**\n**");

p = fork();

**if** (p < 0) {

perror("fork");

exit(1);

}

**if** (p == 0) {

*/\**

*\* in D proccess*

*\*/*

change\_pname("D");

printf("D: Started**\n**");

sleep(SLEEP\_PROC\_SEC);

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

exit(13);

}

*/\**

*\* back in B proccess*

*\*/*

change\_pname("B");

printf("B: Created D and waiting for its termination...**\n**");

p = wait(&status);

explain\_wait\_status(p, status);

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

exit(19);

}

*/\**

*\* back in A proccess*

*\*/*

change\_pname("A");

printf("A: Created B and waiting for its termination...**\n**");

p = fork();

**if** (p < 0) {

perror("fork");

exit(1);

}

**if** (p == 0) {

*/\**

*\* in C proccess*

*\*/*

change\_pname("C");

printf("C: Started**\n**");

sleep(SLEEP\_PROC\_SEC);

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

exit(17);

}

*/\**

*\* back in A proccess*

*\*/*

change\_pname("A");

printf("A: Created C and waiting for its termination...**\n**");

p = wait(&status);

explain\_wait\_status(p, status);

p = wait(&status);

explain\_wait\_status(p, status);

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

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(void)

{

pid\_t pid;

int status;

*/\* Fork root of process tree \*/*

pid = fork();

**if** (pid < 0) {

perror("main: fork");

exit(1);

}

**if** (pid == 0) {

*/\* Child \*/*

fork\_procs();

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;

Ενδεικτικά, η έξοδος του προγράμματος είναι η ακόλουθη:

~/labs/os/ex2\_1 ./ask2-fork

A: Started

A: Created B and waiting for its termination...

A: Created C and waiting for its termination...

B: Started

C: Started

B: Created D and waiting for its termination...

D: Started

A(31128)─┬─B(31129)───D(31131)

└─C(31130)

C: Exiting...

D: Exiting...

My PID = 31128: Child PID = 31130 terminated normally, exit status = 17

My PID = 31129: Child PID = 31131 terminated normally, exit status = 13

B: Exiting...

My PID = 31128: Child PID = 31129 terminated normally, exit status = 19

A: Exiting...

My PID = 31127: Child PID = 31128 terminated normally, exit status = 16

Ερωτήσεις:

1. Αν τερματιστεί πρόωρα η διεργασία Α τότε ο kernel γνωρίζει πως δεν θα λάβει κάποιο wait call για την διεργασία παιδί. Επομένως, τα παιδιά της Α, στην προκειμένη περίπτωση η διεργασίες B και C, θα γίνουν “orphan processes” και θα υιοθετηθούν από την κύρια διεργασία του λειτουργικού συστήματος init (pid=1). Η init θα εκτελέσει την wait system call ώστε τα παιδιά (B και C) να μπορούν να τερματιστούν κανονικά.
2. Αν καλέσουμε την συνάρτηση show\_pstree(getpid()) τότε αλλάζει το όρισμα pid που λαμβάνει. Πριν εκτύπωνε το δέντρο διεργασιών με ρίζα την διεργασία Α εφόσον το pid που είχε αποθηκευτεί στην μεταβλητή ήταν αυτό του Α. Τώρα, λαμβάνει το pid της διεργασίας στην οποία το πρόγραμμα βρίσκεται εκείνη την στιγμή μέσω της συνάρτησης getpid(). Δηλαδή, λαμβάνει το process id του ίδιου του προγράμματος που εκτελείται. Οπότε εμφανίζεται το ακόλουθο δέντρο διεργασιών:  
     
     
   ask2-fork(37190)─┬─A(37191)─┬─B(37192)───D(37194)

│ └─C(37193)

└─sh(37195)───pstree(37196)

Παρατηρούμε πως η νέα ρίζα του δέντρου αποτελεί πλέον το εκτελέσιμο πρόγραμμα. Επίσης εμφανίζεται ένα νέο παιδί με όνομα sh. Αυτή η διεργασία ανήκει στο shell του Unix που αποτελεί τον διαμεσολαβητή του χρήστη με τον kernel. Είναι υπεύθυνο για την εκτύπωση της εξόδου του προγράμματος που εκτελούμε. Το παιδί της sh, το pstree, είναι η διεργασία η οποία εκτυπώνει το δέντρο διεργασιών που της ζητείται στο shell.

1. Ένα υπολογιστικό σύστημα έχει συγκεκριμένη χωρητικότητα ενεργών διεργασιών και επομένως αν είναι ανεξέλεγκτη η δημιουργία νέων διεργασιών το λειτουργικό σύστημα μπορεί να εισέλθει σε κορεσμό. Έτσι, δεν θα μπορούν να δημιουργηθούν νέες διεργασίες οι οποίες μπορεί να είναι απαραίτητες για την ορθή λειτουργία του συστήματος. Επομένως, η οριοθέτηση των συνολικών διεργασιών που μπορεί να εκτελεί κάθε χρήστης σε ένα τέτοιο σύστημα είναι επιτακτική.

Άσκηση 2: Δημιουργία αυθαίρετου δέντρου διεργασιών  
  
Ακολουθεί ο κώδικας του ζητούμενου προγράμματος:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.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 3

void

procs\_tree(**struct** tree\_node \*root)

{

printf("%s started...**\n**", root->name);

change\_pname(root->name);

**if** (root->nr\_children >0) {

printf("%s waiting %d children...**\n**", root->name, root->nr\_children);

pid\_t p;

int status;

int i;

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

p=fork();

**if** (p<0) {

perror("fork");

exit(1);

}

**if** (p==0) {

procs\_tree(root->children + i);

}

}

int j;

**for**(j=0; j<root->nr\_children; j++){

p=wait(&status);

explain\_wait\_status(p, status);

}

}

**else** {

printf("%s sleeping...**\n**", root->name);

sleep(SLEEP\_PROC\_SEC);

}

printf("%s exiting...**\n**", root->name);

exit(0);

}

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]);

printf("File Tree:**\n**");

print\_tree(root);

pid\_t pid;

int status;

pid=fork();

**if** (pid<0) {

perror("fork");

exit(1);

}

**if** (pid==0) {

printf("**\n**Creating Proccess Tree...**\n**");

procs\_tree(root);

}

sleep(SLEEP\_TREE\_SEC);

printf("**\n**Proccess Tree:**\n**");

show\_pstree(pid);

pid=wait(&status);

explain\_wait\_status(pid, status);

**return** 0;

}

Καθώς και η έξοδος αυτού με όρισμα ένα αρχείο a.c:

~/labs/os/ex2\_2 ./ask2-tree a.c

File Tree:

A

B

D

C

E

F

Creating Proccess Tree...

A started...

A waiting 3 children...

B started...

C started...

B waiting 1 children...

C sleeping...

E started...

E waiting 1 children...

D started...

D sleeping...

F started...

F sleeping...

Proccess Tree:

A(38763)─┬─B(38764)───D(38767)

├─C(38765)

└─E(38766)───F(38768)

C exiting...

D exiting...

F exiting...

My PID = 38764: Child PID = 38767 terminated normally, exit status = 0

My PID = 38763: Child PID = 38765 terminated normally, exit status = 0

B exiting...

My PID = 38766: Child PID = 38768 terminated normally, exit status = 0

E exiting...

My PID = 38763: Child PID = 38764 terminated normally, exit status = 0

My PID = 38763: Child PID = 38766 terminated normally, exit status = 0

A exiting...

My PID = 38762: Child PID = 38763 terminated normally, exit status = 0

Ερωτήσεις:

1. Παρατηρούμε πως τα μηνύματα έναρξης των διεργασιών εμφανίζονται ανά επίπεδο, δηλαδή πρώτα ξεκινάει η διεργασία Α, μετά οι B, C, E και τέλος οι D και F. Οι διεργασίες δημιουργούνται ανεξέλεγκτα εφόσον το πρόγραμμα δεν προβλέπει κάποιον συγκεκριμένο συγχρονισμό κατά την δημιουργία των διεργασιών.

Οι διεργασίες τερματίζουν βάση της αναμονής των παιδιών (sleep()). Δηλαδή, αφού τερματίσουν οι “διεργασίες φύλλα” μετά την διαδικασία sleep() τότε ακολουθεί ο τερματισμός της αντίστοιχης διεργασίας του “πατέρα”. Έτσι, πρώτα τερματίζονται οι διεργασίες C, D και F που αποτελούν τα φύλλα του δέντρου, μετά τερματίζουν οι B και E και τέλος η Α.

Άσκηση 3: Αποστολή και χειρισμός σημάτων

Ακολουθεί ο κώδικας του ζητούμενου προγράμματος:

#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)

{

*/\**

*\* Start*

*\*/*

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

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

change\_pname(root->name);

*/\**

*\*saving pid of children in an array*

*\*/*

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

int i;

*/\**

*fork in DFS order*

*\*/*

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

pid[i] = fork();

**if** (pid[i] < 0) {

perror("fork");

exit(1);

} **else** **if** (pid[i] == 0) {

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

}

}

wait\_for\_ready\_children(root->nr\_children);

*/\**

*\* Suspend Self*

*\*/*

raise(SIGSTOP);

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

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

*/\**

*wake up children to continue*

*\*/*

int status;

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

kill(pid[i], SIGCONT);

wait(&status);

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

}

printf("PID = %ld, name = %s is exiting...**\n**",

(long)getpid(), root->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[])

{

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(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 \*/*

wait(&status);

explain\_wait\_status(pid, status);

**return** 0;

}

Παράδειγμα εκτέλεσης του προγράμματος:

~/labs/os/ex2\_3 ./ask2-signals a.c ✔

PID = 41948, name A, starting...

PID = 41949, name B, starting...

PID = 41950, name C, starting...

PID = 41951, name E, starting...

My PID = 41948: Child PID = 41950 has been stopped by a signal, signo = 19

PID = 41952, name D, starting...

PID = 41953, name F, starting...

My PID = 41949: Child PID = 41952 has been stopped by a signal, signo = 19

My PID = 41948: Child PID = 41949 has been stopped by a signal, signo = 19

My PID = 41951: Child PID = 41953 has been stopped by a signal, signo = 19

My PID = 41948: Child PID = 41951 has been stopped by a signal, signo = 19

My PID = 41947: Child PID = 41948 has been stopped by a signal, signo = 19

A(41948)─┬─B(41949)───D(41952)

├─C(41950)

└─E(41951)───F(41953)

PID = 41948, name = A is awake

PID = 41949, name = B is awake

PID = 41952, name = D is awake

PID = 41952, name = D is exiting...

My PID = 41949: Child PID = 41952 terminated normally, exit status = 0

PID = 41949, name = B is exiting...

My PID = 41948: Child PID = 41949 terminated normally, exit status = 0

PID = 41950, name = C is awake

PID = 41950, name = C is exiting...

My PID = 41948: Child PID = 41950 terminated normally, exit status = 0

PID = 41951, name = E is awake

PID = 41953, name = F is awake

PID = 41953, name = F is exiting...

My PID = 41951: Child PID = 41953 terminated normally, exit status = 0

PID = 41951, name = E is exiting...

My PID = 41948: Child PID = 41951 terminated normally, exit status = 0

PID = 41948, name = A is exiting...

My PID = 41947: Child PID = 41948 terminated normally, exit status = 0

Ερωτήσεις:

1. Το πλεονέκτημα της χρήσης σημάτων είναι πως ο συγχρονισμός της εκτέλεσης και του τερματισμού των διεργασιών είναι πολύ περισσότερο ελέγξιμος σε σχέση με τον συγχρονισμό μέσω της sleep(). Πιο συγκεκριμένα, σε αυτή την άσκηση έχουμε καταφέρει να εκτελούμε την αφύπνιση των διεργασιών με πολύ συγκεκριμένο τρόπο, δηλαδή με διάρχηση κατά βάθος (DFS) του δέντρου διεργασιών, κάτι που δεν είναι εφικτό με την υλοποίηση της άσκησης 2.
2. H διαδικασία ready\_for\_ready\_children() έχει ως σκοπό την αναμονή της διεργασίας στην οποία καλείται έως την χρονική στιγμή που όλα τα παιδιά της συγκεκριμένης διεργασίας έχουν λάβει το σήμα SIGSTOP.  
   Με την χρήση της εξασφαλίζουμε πως θα έχει ολοκληρωθεί η δημιουργία του δέντρου διεργασιών πριν γίνει η εκτύπωσή του στο terminal. Ποιο συγκεκριμένα, αν δεν καλούσαμε την ready\_for\_ready\_children() τότε δεν θα είχαμε σωστό συγχρονισμό και θα εκτυπωνόταν το δέντρο διεργασιών πριν την δημιουργία όλων των παιδιών διεργασιών.

Άσκηση 4: Παράλληλος υπολογισμός αριθμητικής έκφρασης

Ακολουθεί ο ζητούμενος κώδικας:

#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(int fd, **struct** tree\_node \*root)

{

*/\**

*\* Start*

*\*/*

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

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

change\_pname(root->name);

*/\**

*\*saving pid of children in an array*

*\*/*

pid\_t pid;

int status;

int pfd[2];

int i;

*/\**

*\*creating pipe*

*\*/*

**if** (pipe(pfd) < 0) {

perror("pipe");

exit(1);

}

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

pid = fork();

**if**(pid < 0) {

perror("fork");

exit(1);

}

**else** **if** (pid == 0) {

*// child is going to write to father, so fd = pfd[1]*

fork\_procs(pfd[1], root->children + i);

}

}

*/\**

*if not leaf, read from children // if leaf, skip for*

*\*/*

int ans[2], val;

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

{

printf("Node %s with PID: %ld. reading from child #%d.**\n**",

root->name, (long)getpid(), i+1);

*/\* father reads value from pipe \*/*

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

perror("read from pipe");

exit(1);

}

ans[i]= val;

pid = wait(&status);

explain\_wait\_status(pid, status);

}

*/\**

*compute value according to parent*

*\*/*

**if** ((root->name[0]) == '+') {

val = ans[0] + ans[1];

}

**else** **if** ((root->name[0]) == '\*') {

val = ans[0] \* ans[1];

}

**else** sscanf(root->name, "%d", &val);

*/\**

*now write val to Father*

*\*/*

**if** (write(fd, &val, **sizeof**(val)) != **sizeof**(val)) {

perror("write to pipe");

exit(1);

}

*/\* exit \*/*

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

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

exit(0);

}

int main(int argc, char \*argv[])

{

pid\_t pid;

int status;

int pfd[2];

int ans;

**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]);

print\_tree(root);

*//create pipe in int main(int argc, char const \*argv[]) {*

**if** (pipe(pfd) < 0) {

perror("main: pipe");

exit(1);

}

*/\* Fork root of process tree \*/*

pid = fork();

**if** (pid < 0) {

perror("main: fork");

exit(1);

}

**if** (pid == 0) {

*/\* Child \*/*

fork\_procs(pfd[1],root);

exit(1);

}

*/\**

*\* Father*

*\*/*

*//read values from pipe*

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

perror("read from pipe");

exit(1);

}

wait(&status);

explain\_wait\_status(pid, status);

printf("computation: %d**\n**", ans);

**return** 0;

}

Παράδειγμα εκτέλεσης του προγράμματος:

~/labs/os/ex2\_4 ./ask2-pipes a.c

\*

+

4

5

3

PID = 44089, name \*, starting...

Node \* with PID: 44089. reading from child #1.

PID = 44090, name +, starting...

PID = 44091, name 3, starting...

PID = 44091, name 3, exiting...

Node + with PID: 44090. reading from child #1.

PID = 44093, name 5, starting...

PID = 44092, name 4, starting...

PID = 44092, name 4, exiting...

PID = 44093, name 5, exiting...

My PID = 44089: Child PID = 44091 terminated normally, exit status = 0

Node \* with PID: 44089. reading from child #2.

My PID = 44090: Child PID = 44092 terminated normally, exit status = 0

Node + with PID: 44090. reading from child #2.

My PID = 44090: Child PID = 44093 terminated normally, exit status = 0

PID = 44090, name +, exiting...

My PID = 44089: Child PID = 44090 terminated normally, exit status = 0

PID = 44089, name \*, exiting...

My PID = 44088: Child PID = 44089 terminated normally, exit status = 0

computation: 27

Ερωτήσεις:

1. Στην συγκεκριμένη άσκηση χρειάζεται τουλάχιστον μία σωλήνωση ανά διεργασία πατέρα. Εφόσον οι τελεστές που χρησιμοποιούνται είναι μόνο το \* και το + σημαίνει πως ισχύει η αντιμεταθετικη ιδιότητα, γεγονός που καθιστά αδιάφορη την σειρά προέλευσης των δεδομένων από τα παιδιά στον πατέρα μέσω του pipe.   
   Ωστόσο, σε κάποιο άλλο πρόγραμμα, μπορεί να υπάρχει η ανάγκη αναγνώρισης της προέλευσης των δεδομένων μέσω του pipe ή ακόμα και η σύγχρονη μεταφορά. Τότε, θα χρειαζόμασταν τόσα pipes όσα τα παιδιά της κάθε διεργασίας ανά διεργασία ώστε να είχαμε τον έλεγχο της σειράς και ροής των δεδομένων. Για παράδειγμα, στην συγκεκριμένη άσκηση, αν οι ζητούμενοι τελεστές δεν ήταν μόνο το \* και + αλλά ήταν και η διαίρεση (/) ή η αφαίρεση(-) τότε θα υπήρχε η ανάγκη αναγνώρισης της σειράς των τελεστών. Για αυτό, θα ήταν βολικό να είχαμε πολλαπλά pipes.  
   Όμως, αν θέλαμε να χρησιμοποιήσουμε μόνο μία σωλήνωση για κάθε αριθμητικό τελεστή τότε θα χρειαζόταν να συγχρονίσουμε κατάλληλα την ροή στο μοναδικό pipe μέσω της επικοινωνίας μεταξύ των διεργασιών παιδιών.
2. Σε ένα σύστημα πολλαπλών επεξεργαστών υπάρχει η δυνατότητα παράλληλης εκτέλεσης διεργασιών. Επομένως, στην περίπτωση του δέντρου διεργασιών μπορούν να δημιουργούνται διεργασίες παιδιά, οι οποίες αναλαμβάνουν επιμέρους tasks της διεργασίας πατέρα, τις οποίες αναλαμβάνουν πολλαπλοί επεξεργαστές. Με αυτόν τον τρόπο εξοικονομείται χρόνος καθώς εκμεταλλευόμαστε την υπολογιστική ισχύ που παρέχει το σύστημα. Στην αντίθετη περίπτωση της αποτίμησης από μία μόνο διεργασία, οι διεργασίες θα αναγκάζονταν να εκτελούνται μέσω ενός μόνο επεξεργαστή αυξάνοντας τον χρόνο εκτέλεσης του προγράμματος.