Λειτουργικά Συστήματα, 6^{ο} Εξάμηνο ΗΜΜΥ

Ακαδημαϊκή Περίοδος 2019-2020

Άσκηση 2: Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Ομάδα: oslabb33

Γιαννούλης Παναγιώτης (03117812)

Κανελλόπουλος Σωτήρης (03117101)

1.1. Χρησιμοποιώντας την κλήση sleep(), καθώς και τις έτοιμες βοηθητικές συναρτήσεις που δίνονται, γράψαμε τον παρακάτω κώδικα, ο οποίος δημιουργεί το ζητούμενο δέντρο διεργασιών. Συγκεκριμένα με fork δημιουργούμε την διεργασία A στην συνάρτηση main(). Έπειτα από την διεργασία A καλούμε την fork_procs όπου δημιουργεί την B και την C ομοίως με fork. Η B με τη σειρά της δημιουργεί την D. Τα φύλλα κάνουν sleep για μικρό χρονικό διάστημα και έπειτα τερματίζονται ομαλά. Οι γονείς περιμένουν πρώτα να τερματιστούν όλα τους τα παιδιά.

```c
void fork_procs(void)
{
  /*
   * initial process is A.
   */
  pid_t pB, pC, pD, p;
  int status;

  change_pname("A");
  fprintf(stderr, "A process , PID = %ld: Creating child B process...\n", (long)getpid());

  /* A process creating two childs */
  pB = fork();
  if (pB < 0) {
    perror("A: error while forking to B");
    exit(1);
  }
  if (pB == 0) {
    /* In child (B) process */

    /* Creating D process */
    pD = fork();
    if (pD < 0) {
      perror("B: error while forking to D");
      exit(1);
    }
    if (pD == 0) {
      /* In child (D) process */
      change_pname("D");
      printf("D: Sleeping...\n");
      sleep(SLEEP_PROC_SEC);
      printf("D: Exiting...\n");
      exit(13);
    }

    change_pname("B");
    printf("Child process B, PID = %ld: Created child D with PID = %ld, waiting for it to terminate...\n",
        (long)getpid(), (long)pD);
    pD = wait(&status);
    explain_wait_status(pD, status);
    exit(19);
  }
```

```
  fprintf(stderr, "A process , PID = %ld: Creating child C process...\n", (long)getpid());
  pC = fork();
  if (pC < 0) {
    perror("A: error while forking to C");
    exit(1);
  }
  if (pC == 0) {
    change_pname("C");
    printf("C: Sleeping...\n");
    sleep(SLEEP_PROC_SEC);

    printf("C: Exiting...\n");
    exit(17);
  }

  /*show_pstree(pid)*/

  printf("Parent A, PID = %ld: Created children B and C with pB = %ld and pC = %ld, waiting for them to terminate...\n", (long

  p = wait(&status);
  explain_wait_status(p, status);
  p = wait(&status);
  explain_wait_status(p, status);

  /*printf("A: Exiting...\n");*/
  exit(16);
}
```

```
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);
  }
  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 κώδικά μας:

```
A process , PID = 7089: Creating child B process...
A process , PID = 7089: Creating child C process...
Parent A, PID = 7089: Created childs C and B with pB = 7090 and pC = 7091, waiting for them to termina
te...
C: Sleeping...
Child process B, PID = 7090: Created child D with PID = 7092, waiting for it to terminate...
D: Sleeping...


A(7089)─┬─B(7090)──D(7092)
        └─C(7091)


C: Exiting...
D: Exiting...
My PID = 7089: Child PID = 7091 terminated normally, exit status = 17
My PID = 7090: Child PID = 7092 terminated normally, exit status = 13
My PID = 7089: Child PID = 7090 terminated normally, exit status = 19
My PID = 7088: Child PID = 7089 terminated normally, exit status = 16
```

Απαντήσεις στις ερωτήσεις:

1. Γνωρίζουμε από τη θεωρία ότι αν «πεθάνει» ο γονιός μιας διεργασίας πριν την ίδια, τότε αυτή υιοθετείται από την αρχική διαδικασία init με PID = 1.

Παρακάτω φαίνεται η έξοδος του προγράμματός μας αν τερματίσουμε πρόωρα τη διεργασία A με kill:

```
A process , PID = 43755: Creating child B process...
A process , PID = 43755: Creating child C process...
Parent A, PID = 43755: Created children B and C with pB = 43756 and pC = 43757, waiting for them to terminate
...
C: Sleeping...
Child process B, PID = 43756: Created child D with PID = 43758, waiting for it to terminate...
D: Sleeping...


A(43755)─┬─B(43756)──D(43758)
         └─C(43757)


My PID = 43754: Child PID = 43755 was terminated by a signal, signo = 9
pangiann@WhiteRose13:~/6ο Εξάμηνο/opsys/2herg/2_1\∞ C: Exiting...
D: Exiting...
My PID = 43756: Child PID = 43758 terminated normally, exit status = 13
```

Παρατηρούμε ότι τα παιδία της διεργασίας A τερματίζουν κανονικά, παρόλο που η A έχει τερματιστεί πριν από αυτά.

2. Αν χρησιμοποιήσουμε show_pstree(getpid()) στην main, τότε στο δέντρο που τυπώνεται υπάρχει και η διεργασία στην οποία εκτελείται το ίδιο το πρόγραμμα, η οποία είναι γονιός της Α, καθώς και ένα άλλο υποδέντρο με διεργασίες οι οποίες χρησιμοποιούνται για την εμφάνιση του δέντρου διεργασιών στο χρήστη. Η διαφορά αυτή έγκειται στο γεγονός ότι τώρα βάζουμε στην show_pstree το pid της main, και έτσι εμφανίζει το δέντρο διεργασιών που έχει ρίζα αυτή τη διεργασία. Αντιθέτως, πριν εμφανίζαμε το δέντρο διεργασιών με ρίζα την Α.

Παρακάτω φαίνεται η έξοδος του προγράμματος, χρησιμοποιώντας show_pstree(getpid()):

```
A process , PID = 7024: Creating child B process...
A process , PID = 7024: Creating child C process...
Parent A, PID = 7024: Created childs C and B with pB = 7025 and pC = 7026, waiting for them to terminate...
Child process B, PID = 7025: Created child D with PID = 7027, waiting for it to terminate...
C: Sleeping...                                          D: Sleeping...


fork-tree(7023)─┬─A(7024)─┬─B(7025)──D(7027)
                │         └─C(7026)
                └─sh(7028)──pstree(7029)


C: Exiting...
My PID = 7024: Child PID = 7026 terminated normally, exit status = 17
D: Exiting...
My PID = 7025: Child PID = 7027 terminated normally, exit status = 13
My PID = 7024: Child PID = 7025 terminated normally, exit status = 19
My PID = 7023: Child PID = 7024 terminated normally, exit status = 16
```

3. Σε έναν υπολογιστή εκτελούνται πολλές διεργασίες παράλληλα και, αν το πλήθος αυτών αυξηθεί ανεξέλεγκτα, είναι πιθανό ο υπολογιστής να υπερφορτωθεί και να μην μπορεί να λειτουργήσει με την επιθυμητή ταχύτητα ή τον επιθυμητό τρόπο. Για αυτό το λόγο είναι λογικό σε ένα σύστημα με πολλούς χρήστες να υπάρχει ένα όριο στο πλήθος των διεργασιών που μπορεί να δημιουργήσει ο κάθε χρήστης.

1.2. Παρακάτω φαίνεται ο κώδικας που γράψαμε, ο οποίος δημιουργεί ένα δέντρο διεργασιών που καθορίζει ο χρήστης από την είσοδο. Στην αρχή ομοίως με πριν δημιουργείται η διεργασία Α. Εκείνη με τη σειρά της μέσω ενός for δημιουργεί όλα τα παιδιά της. Κάθε παιδί που δεν είναι φύλλο καλεί αναδρομικά την create_fork_tree(). Οι διεργασίες φύλλα εκτελούν την ίδια λειτουργία με πριν. Κοιμούνται για μικρό χρονικό διάστημα και στη συνέχεια τερματίζονται ομαλά. Για να περιμένουν τώρα οι γονείς να τερματιστούν όλα τους τα παιδιά χρησιμοποιούμαι ένα for με τόσες επαναλήψεις όσα και τα παιδιά κάθε διεργασίας όπου καλείται η συνάρτηση wait().

```c
void create_fork_tree(struct tree_node *node)
{
  pid_t pid;
  int status, i;

  printf("Node with name %s: Creating %d children...\n", node->name, node->nr_children);
  for (i = 0; i < node->nr_children; i++) {

    pid = fork();
    if (pid < 0) {
      perror("error while forking");
      exit(1);
    }
    if (pid == 0) {
      change_pname((node->children + i)->name);
      if ((node->children+i)->nr_children != 0) {
        /* MIDDLE NODE */
        create_fork_tree(node->children+i);
        exit(17);

      }
      else {
        /* LEAF NODE */
        printf("Node %s, is sleeping now...\n", (node->children + i)->name);
        sleep(SLEEP_PROC_SEC);

        printf("Node %s: Woke up and exiting...\n",  (node->children + i)->name);
        exit(42);
      }
    }
  }

  for (i = 0; i < node->nr_children; i++) {
    pid = wait(&status);
    explain_wait_status(pid, status);
  }

  exit(17);


}
```

```c
void create_fork_tree(struct tree_node *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\n", argv[0]);
    exit(1);
  }

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

  pid = fork();
  if (pid < 0) {
    perror("main: fork");
    exit(1);
  }
  if (pid == 0) {
    /* child */
    change_pname(root->name);
    create_fork_tree(root);
    exit(0);
  }

  sleep(SLEEP_TREE_SEC);
  show_pstree(pid);

  pid = wait(&status);
  explain_wait_status(pid, status);

  return 0;
}
```

Παρακάτω φαίνονται οι έξοδοι του προγράμματός μας για δύο παραδείγματα:

```
A
3
B
C
D

B
2
E
F

E
0

F
0

C
0

D
0
```

```
A
        B
                E
                F
        C
        D
Node with name A: Creating 3 children...
Node C, is sleeping now...
Node D, is sleeping now...
Node with name B: Creating 2 children...
Node F, is sleeping now...
Node E, is sleeping now...


A(8794)─┬─B(8795)─┬─E(8798)
        │         └─F(8799)
        ├─C(8796)
        └─D(8797)


Node C: Woke up and exiting...
Node D: Woke up and exiting...
My PID = 8794: Child PID = 8796 terminated normally, exit status = 42
My PID = 8794: Child PID = 8797 terminated normally, exit status = 42
Node F: Woke up and exiting...
My PID = 8795: Child PID = 8799 terminated normally, exit status = 42
Node E: Woke up and exiting...
My PID = 8795: Child PID = 8798 terminated normally, exit status = 42
My PID = 8794: Child PID = 8795 terminated normally, exit status = 17
My PID = 8793: Child PID = 8794 terminated normally, exit status = 17
```

```
A
2
B
C

B
3
D
E
F

D
1
G

G
0

E
0

F
0

C
0
```

```
A
        B
                D
                        G
                E
                F
        C
Node with name A: Creating 2 children...
Node with name B: Creating 3 children...
Node C, is sleeping now...
Node with name D: Creating 1 children...
Node E, is sleeping now...
Node F, is sleeping now...
Node G, is sleeping now...


A(9070)─┬─B(9071)─┬─D(9073)──G(9076)
        │         ├─E(9074)
        │         └─F(9075)
        └─C(9072)


Node C: Woke up and exiting...
My PID = 9070: Child PID = 9072 terminated normally, exit status = 42
Node E: Woke up and exiting...
Node F: Woke up and exiting...
My PID = 9071: Child PID = 9074 terminated normally, exit status = 42
My PID = 9071: Child PID = 9075 terminated normally, exit status = 42
Node G: Woke up and exiting...
My PID = 9073: Child PID = 9076 terminated normally, exit status = 42
My PID = 9071: Child PID = 9073 terminated normally, exit status = 17
My PID = 9070: Child PID = 9071 terminated normally, exit status = 17
My PID = 9069: Child PID = 9070 terminated normally, exit status = 17
```

Σημειώνουμε ότι επιλέξαμε οι διεργασίες-φύλλα να έχουν exit status 42 και οι υπόλοιπες 17.

Απαντήσεις στις ερωτήσεις:

1. Τα μηνύματα ενάρξεων φαίνεται να εμφανίζονται σε «επίπεδα», δηλαδή πρώτα εμφανίζεται η διεργασία-ρίζα, μετά τα παιδιά αυτής, μετά τα παιδιά αυτών των παιδιών κ.ο.κ. Αυτό είναι αναμενόμενο από τον τρόπο που έχει γραφεί ο κώδικας, δηλαδή πρώτα μία διεργασία «δημιουργεί» τα παιδιά της και μετά το κάθε παιδί της «δημιουργεί» τα δικά του.

Τα μηνύματα τερματισμού ακολουθούν μια διαφορετική σειρά: πρώτα τερματίζονται οι διεργασίες-φύλλα και μετά οι γονείς αυτών κ.ο.κ. Έτσι, π.χ. στο πρώτο παράδειγμα, βλέπουμε ότι παρόλο που οι διεργασίες B, C, D είναι όλες στο ίδιο «επίπεδο» (και επομένως δημιουργούνται μαζί), δεν τερματίζονται μαζί, αφού οι διεργασίες C και D είναι φύλλα και τερματίζονται αρκετά νωρίτερα από την B (η οποία πρέπει να περιμένει τον τερματισμό των παιδιών της).

Επίσης, παρατηρούμε ότι τα «αριστερότερα» φύλλα τερματίζονται νωρίτερα από τα υπόλοιπα, το οποίο είναι λογικό αφού δημιουργήθηκαν νωρίτερα και επομένως είχαν περισσότερο χρόνο να εκτελέσουν τις αντίστοιχες εντολές.

1.3. Επεκτείναμε το προηγούμενο πρόγραμμα, έτσι ώστε οι ενάρξεις και οι τερματισμοί να ελέγχονται με σήματα και επομένως να μπορούμε να καθορίσουμε με περισσότερη σιγουριά τη σειρά με την οποία εκτελούνται οι διεργασίες. Η διαδικασία δημιουργίας του δέντρου είναι ίδια με πριν. Αυτή τη φορά όμως, κάθε φορά που δημιουργείται μια διεργασία (αφού δημιουργήσει όλα της τα παιδιά και βεβαιωθεί ότι και αυτά έχουν ολοκληρώσει τη διαδικασία δημιουργίας των δικών τους παιδιών) σταματάει (SIGSTOP) και περιμένει σήμα SIGCONT από τον γονέα του ώστε να συνεχίσει την λειτουργία του. Με αυτόν τον τρόπο εξασφαλίζεται ότι θα ξεκινήσει η λειτουργεία του δέντρου διεργασιών που φτιάξαμε αφού δημιουργηθεί ολόκληρο. Ο πρώτος που στέλνει το πρώτο σήμα SIGCONT είναι η ρίζα των διεργασιών και εκκινεί την διεργασία A, η οποία με τη σειρά της εκκινεί της επόμενες. Για να επιτευχθεί το DFS, κάθε διεργασία γονιός στέλνει σήμα SIGCONT σε ένα παιδί και ταυτόχρονα περιμένει να τερματιστεί αυτό το παιδί και μετά συνεχίζει με το επόμενο παιδί.

Παρακάτω φαίνεται ο κώδικάς μας:

```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 */
    change_pname(root->name);
    create_fork_tree(root);
    exit(1);
  }
  /*
   * Father
   */
  wait_for_ready_children(1);

  /* 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;
}
```

```c
void create_fork_tree(struct tree_node *node)
{
  /*
   * Start
   */
  pid_t pid_arr[node->nr_children], pid;
  int status, i;

  printf("PID = %ld, name %s, starting...\n",
      (long)getpid(), node->name);
  for (i = 0; i < node->nr_children; i++) {
    pid_arr[i] = fork();
    if (pid_arr[i] < 0) {
      perror("error while forking");
      exit(1);
    }
    if (pid_arr[i] == 0) {
      change_pname((node->children+i)->name);
      if ((node->children+i)->nr_children != 0) {
          /* MIDDLE NODE */
          create_fork_tree(node->children+i);
          exit(17);
      }
      else {
          /* LEAF NODE */
          printf("Node %s, is sleeping now...\n", (node->children + i)->name);
          raise(SIGSTOP);

          printf("Node %s: Woke up and exiting...\n",  (node->children + i)->name);
          exit(42);
      }
    }
    wait_for_ready_children(1);
  }
  raise(SIGSTOP);
  printf("PID = %ld, name = %s is awake\n",
    (long)getpid(), node->name);


  /*
   * Suspend Self
   */
  for (i = 0; i < node->nr_children; i++) {
    kill(pid_arr[i], SIGCONT);
    pid = wait(&status);
    explain_wait_status(pid, status);
  }

  exit(17);
  /* ... */
}
```

Παραθέτουμε και τις εξόδους του προγράμματος για τις δύο εισόδους που δοκιμάσαμε και στο προηγούμενο ερώτημα:

```
PID = 10066, name A, starting...
PID = 10067, name B, starting...
Node E, is sleeping now...
My PID = 10067: Child PID = 10068 has been stopped by a signal, signo = 19
Node F, is sleeping now...
My PID = 10067: Child PID = 10069 has been stopped by a signal, signo = 19
My PID = 10066: Child PID = 10067 has been stopped by a signal, signo = 19
Node C, is sleeping now...
My PID = 10066: Child PID = 10070 has been stopped by a signal, signo = 19
Node D, is sleeping now...
My PID = 10066: Child PID = 10071 has been stopped by a signal, signo = 19
My PID = 10065: Child PID = 10066 has been stopped by a signal, signo = 19


A(10066)─┬─B(10067)─┬─E(10068)
         │          └─F(10069)
         ├─C(10070)
         └─D(10071)


PID = 10066, name = A is awake
PID = 10067, name = B is awake
Node E: Woke up and exiting...
My PID = 10067: Child PID = 10068 terminated normally, exit status = 42
Node F: Woke up and exiting...
My PID = 10067: Child PID = 10069 terminated normally, exit status = 42
My PID = 10066: Child PID = 10067 terminated normally, exit status = 17
Node C: Woke up and exiting...
My PID = 10066: Child PID = 10070 terminated normally, exit status = 42
Node D: Woke up and exiting...
My PID = 10066: Child PID = 10071 terminated normally, exit status = 42
My PID = 10065: Child PID = 10066 terminated normally, exit status = 17
```

```
PID = 10172, name A, starting...
PID = 10173, name B, starting...
PID = 10174, name D, starting...
Node G, is sleeping now...
My PID = 10174: Child PID = 10175 has been stopped by a signal, signo = 19
My PID = 10173: Child PID = 10174 has been stopped by a signal, signo = 19
Node E, is sleeping now...
My PID = 10173: Child PID = 10176 has been stopped by a signal, signo = 19
Node F, is sleeping now...
My PID = 10173: Child PID = 10177 has been stopped by a signal, signo = 19
My PID = 10172: Child PID = 10173 has been stopped by a signal, signo = 19
Node C, is sleeping now...
My PID = 10172: Child PID = 10178 has been stopped by a signal, signo = 19
My PID = 10171: Child PID = 10172 has been stopped by a signal, signo = 19


A(10172)─┬─B(10173)─┬─D(10174)──G(10175)
         │          ├─E(10176)
         │          └─F(10177)
         └─C(10178)


PID = 10172, name = A is awake
PID = 10173, name = B is awake
PID = 10174, name = D is awake
Node G: Woke up and exiting...
My PID = 10174: Child PID = 10175 terminated normally, exit status = 42
My PID = 10173: Child PID = 10174 terminated normally, exit status = 17
Node E: Woke up and exiting...
My PID = 10173: Child PID = 10176 terminated normally, exit status = 42
Node F: Woke up and exiting...
My PID = 10173: Child PID = 10177 terminated normally, exit status = 42
My PID = 10172: Child PID = 10173 terminated normally, exit status = 17
Node C: Woke up and exiting...
My PID = 10172: Child PID = 10178 terminated normally, exit status = 42
My PID = 10171: Child PID = 10172 terminated normally, exit status = 17
```

Από τα παραπάνω φαίνεται ότι τα μηνύματα των διεργασιών όντως τυπώνονται με σειρά DFS.

Απαντήσεις στις ερωτήσεις:

1. Με τη χρήση σημάτων μπορούμε να συγχρονίσουμε τις διεργασίες ευκολότερα με τον τρόπο που θέλουμε, αφού μπορούμε με σαφήνεια να καθορίσουμε πότε θα σταματήσει ή θα ξεκινήσει η κάθε διεργασία. Έτσι επιτυγχάνεται η διάσχιση κατά βάθος. Αυτό βεβαίως θα μπορούσε να γίνει και χωρίς σήματα, με τη χρήση της sleep(), όμως για να μπορεί αυτό να λειτουργήσει ντετερμινιστικά θα πρέπει να βάλουμε ένα αρκετά μεγάλο χρονικό περιθώριο στη sleep(), με αποτέλεσμα το πρόγραμμα να γίνεται αργό χωρίς λόγο.

2. Η wait_for_ready_children() εξασφαλίζει ότι πρώτα θα δημιουργηθούν όλες οι διεργασίες, πριν αρχίσουν να ενεργοποιούνται. Αν την παραλείπαμε θα ήταν πιθανό να μην δημιουργηθεί εγκαίρως κάποια διεργασία και επομένως θα ήταν αδύνατη η διάσχιση κατά βάθος.

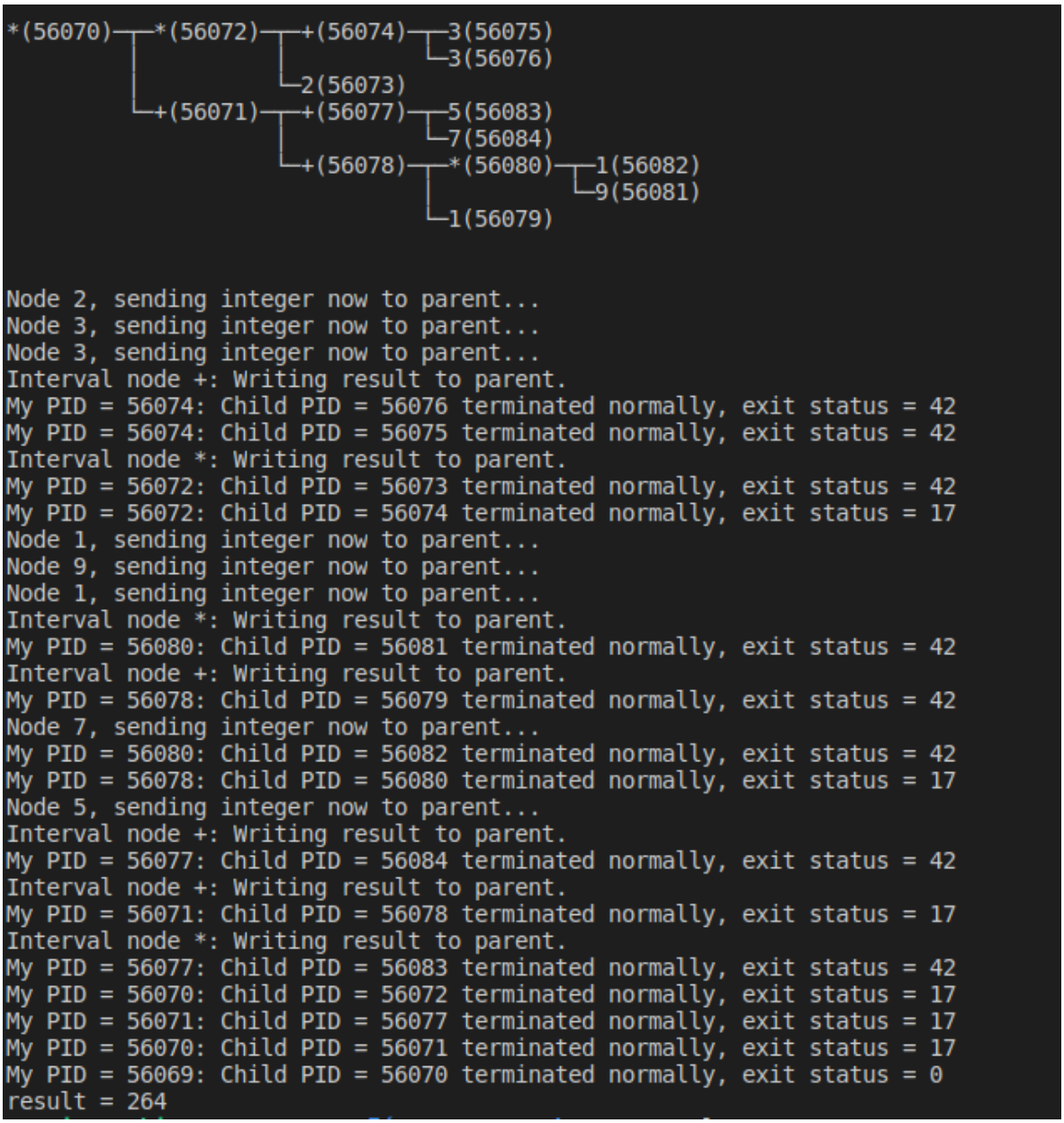
1.4. Ο κώδικας επειδή είναι λίγο μεγαλύτερος και δεν θα ήταν βολικό για ανάγνωση μέσω εικόνας, παρατίθεται όπως και οι άλλοι φυσικά στον φάκελο με όλα τα απαραίτητα αρχεία. Το δέντρο διεργασιών και πάλι δημιουργείται με τον ίδιο τρόπο. Αυτή τη φορά, κάθε γονιός δημιουργεί και ένα pipe με τα δύο του παιδιά. Οι διεργασίες φύλλα στέλνουν τον αριθμό που τους αναλογεί μέσω του pipe στον γονιό τους. Ενώ οι διεργασίες-ενδιάμεσοι κόμβοι περιμένουν δύο αριθμούς από τα δύο τους παιδιά και εκτελούν την πράξη που τους αντιστοιχεί. Με τη σειρά τους στέλνουν το αποτέλεσμα στους γονείς. Η υλοποίηση μπορεί να πραγματοποιηθεί είτε με σήματα είτε με sleep functions.

Παραθέτουμε τις εξόδους του προγράμματός μας για δύο παραδείγματα:

```
*
        +
                5
                6
        *
                2
                5
Node with name *: Creating 2 children...
Node with name *: Creating pipe
Interval node *: Receiving integer from child.
Node with name +: Creating 2 children...
Node with name +: Creating pipe
Node with name *: Creating 2 children...
Node with name *: Creating pipe
Interval node +: Receiving integer from child.
Interval node *: Receiving integer from child.


*(55692)─┬─*(55695)─┬─2(55697)
         │          └─5(55699)
         └─+(55694)─┬─5(55696)
                    └─6(55698)


Node 2, sending integer now to parent...
Node 5, sending integer now to parent...
Node 6, sending integer now to parent...
Interval node +: Writing result to parent.
My PID = 55694: Child PID = 55696 terminated normally, exit status = 42
My PID = 55694: Child PID = 55698 terminated normally, exit status = 42
Node 5, sending integer now to parent...
Interval node *: Writing result to parent.
Interval node *: Writing result to parent.
My PID = 55695: Child PID = 55697 terminated normally, exit status = 42
My PID = 55692: Child PID = 55694 terminated normally, exit status = 17
My PID = 55695: Child PID = 55699 terminated normally, exit status = 42
My PID = 55692: Child PID = 55695 terminated normally, exit status = 17
My PID = 55691: Child PID = 55692 terminated normally, exit status = 0
result = 110
```



Απαντήσεις στις ερωτήσεις:

1. Στη συγκεκριμένη άσκηση χρειαζόμαστε μόνο μία σωλήνωση ανά διεργασία, αφού οι πράξεις που εκτελούνται (πρόσθεση, πολλαπλασιασμός) είναι προσεταιριστικές και επομένως δεν μας αφορά με ποια σειρά θα επιστρέψουν τα δύο παιδιά το αποτέλεσμά τους στον γονιό. Έτσι, μπορούμε να χρησιμοποιήσουμε κοινό pipe για τα δύο παιδιά, από το οποίο ο γονιός θα πραγματοποιεί ανάγνωση δύο φορές. Αυτό δεν θα μπορούσε να γίνει για κάθε αριθμητικό τελεστή, αφού αν είχαμε π.χ. αφαίρεση ή διαίρεση θα έπρεπε να διαβάζει η γονική διεργασία τους αριθμούς από δύο διαφορετικά pipes, ώστε να γνωρίζει π.χ. ποιος είναι ο διαιρετέος και ποιος ο διαιρέτης.

2. Η εκτέλεση των πράξεων με δέντρο διεργασιών θα είχε το πλεονέκτημα ότι θα μπορούσαν οι πράξεις που βρίσκονται στο ίδιο «επίπεδο» του δέντρου να εκτελούνται παράλληλα και επομένως γρηγορότερα. Αντίθετα, αν εκτελούνταν όλες οι πράξεις από μία διεργασία, θα είχαμε σειριακή εκτέλεση, η οποία είναι προφανώς πιο αργή. Αναφέρουμε ότι η χρονική διαφορά είναι σημαντική καθώς στην πρώτη περίπτωση θα είχαμε λογαριθμική πολυπλοκότητα, ενώ στη δεύτερη γραμμική.