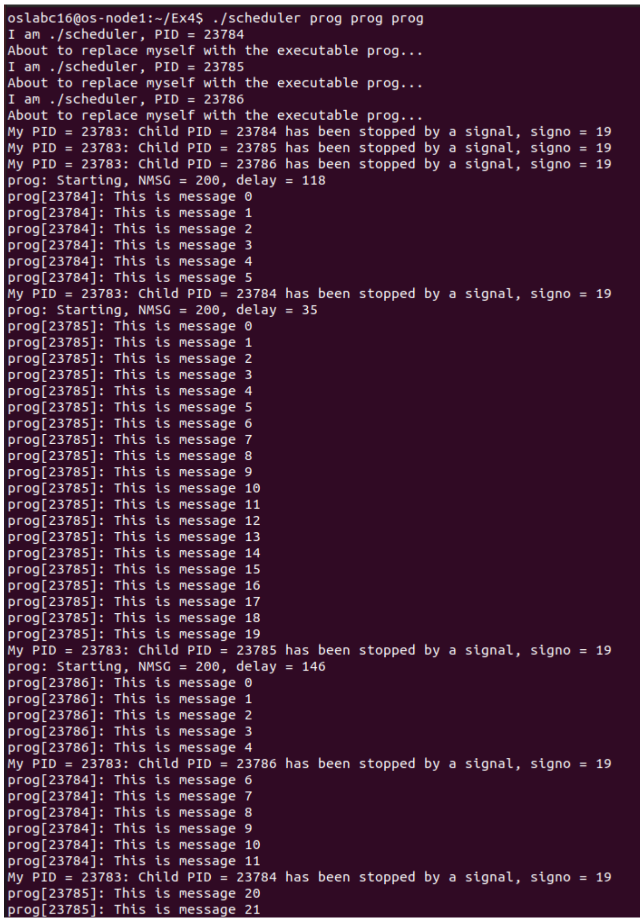
ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

4η Άσκηση: Χρονοδρομολόγηση

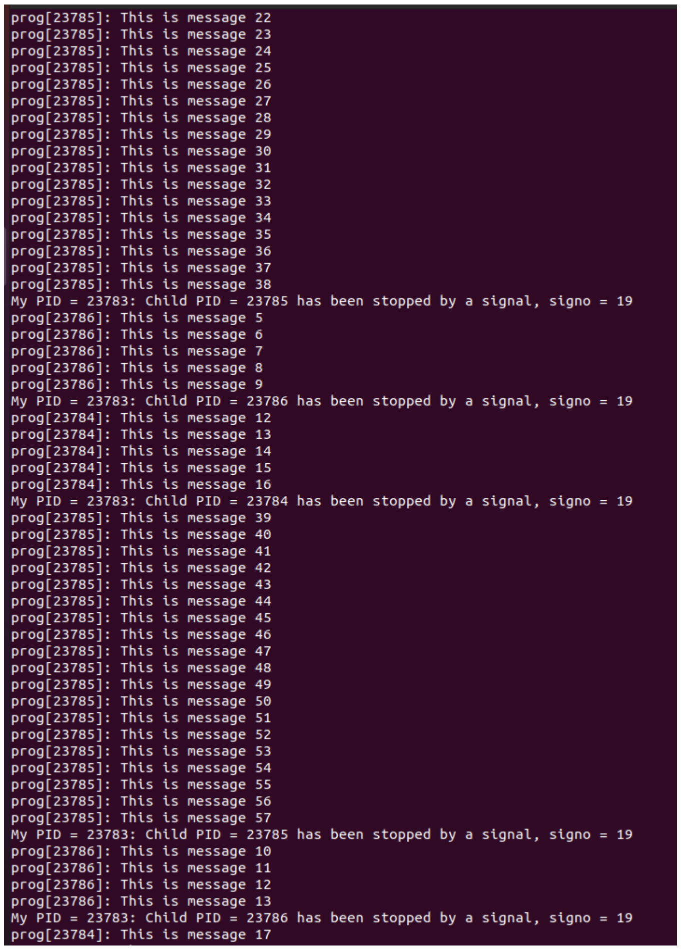
ΦΟΙΤΗΤΕΣ: ΚΥΡΙΑΚΟΥ ΔΗΜΗΤΡΗΣ 03117601
ΧΑΤΖΗΧΡΙΣΤΟΦΗ ΧΡΙΣΤΟΣ 03117711
ΟΜΑΔΑ: OSLABC16
ΕΞΑΜΗΝΟ: 6ο

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

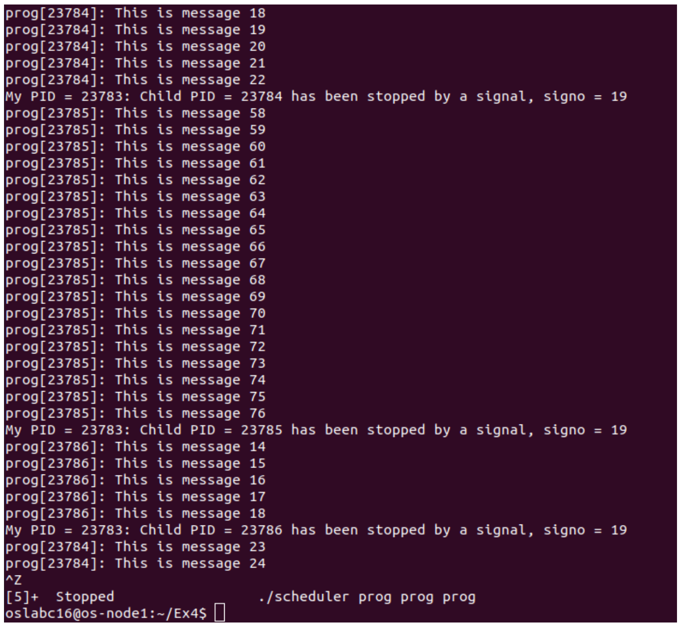
Ενδεικτική έξοδος εκτέλεσης:



```
oslabc16@os-node1:~/Ex4$ ./scheduler prog prog prog
I am ./scheduler, PID = 23784
About to replace myself with the executable prog...
I am ./scheduler, PID = 23785
About to replace myself with the executable prog...
I am ./scheduler, PID = 23786
About to replace myself with the executable prog...
My PID = 23783: Child PID = 23784 has been stopped by a signal, signo = 19
My PID = 23783: Child PID = 23785 has been stopped by a signal, signo = 19
My PID = 23783: Child PID = 23786 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 118
prog[23784]: This is message 0
prog[23784]: This is message 1
prog[23784]: This is message 2
prog[23784]: This is message 3
prog[23784]: This is message 4
prog[23784]: This is message 5
My PID = 23783: Child PID = 23784 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 35
prog[23785]: This is message 0
prog[23785]: This is message 1
prog[23785]: This is message 2
prog[23785]: This is message 3
prog[23785]: This is message 4
prog[23785]: This is message 5
prog[23785]: This is message 6
prog[23785]: This is message 7
prog[23785]: This is message 8
prog[23785]: This is message 9
prog[23785]: This is message 10
prog[23785]: This is message 11
prog[23785]: This is message 12
prog[23785]: This is message 13
prog[23785]: This is message 14
prog[23785]: This is message 15
prog[23785]: This is message 16
prog[23785]: This is message 17
prog[23785]: This is message 18
prog[23785]: This is message 19
My PID = 23783: Child PID = 23785 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 146
prog[23786]: This is message 0
prog[23786]: This is message 1
prog[23786]: This is message 2
prog[23786]: This is message 3
prog[23786]: This is message 4
My PID = 23783: Child PID = 23786 has been stopped by a signal, signo = 19
prog[23784]: This is message 6
prog[23784]: This is message 7
prog[23784]: This is message 8
prog[23784]: This is message 9
prog[23784]: This is message 10
prog[23784]: This is message 11
My PID = 23783: Child PID = 23784 has been stopped by a signal, signo = 19
prog[23785]: This is message 20
prog[23785]: This is message 21
```



```
prog[23785]: This is message 22
prog[23785]: This is message 23
prog[23785]: This is message 24
prog[23785]: This is message 25
prog[23785]: This is message 26
prog[23785]: This is message 27
prog[23785]: This is message 28
prog[23785]: This is message 29
prog[23785]: This is message 30
prog[23785]: This is message 31
prog[23785]: This is message 32
prog[23785]: This is message 33
prog[23785]: This is message 34
prog[23785]: This is message 35
prog[23785]: This is message 36
prog[23785]: This is message 37
prog[23785]: This is message 38
My PID = 23783: Child PID = 23785 has been stopped by a signal, signo = 19
prog[23786]: This is message 5
prog[23786]: This is message 6
prog[23786]: This is message 7
prog[23786]: This is message 8
prog[23786]: This is message 9
My PID = 23783: Child PID = 23786 has been stopped by a signal, signo = 19
prog[23784]: This is message 12
prog[23784]: This is message 13
prog[23784]: This is message 14
prog[23784]: This is message 15
prog[23784]: This is message 16
My PID = 23783: Child PID = 23784 has been stopped by a signal, signo = 19
prog[23785]: This is message 39
prog[23785]: This is message 40
prog[23785]: This is message 41
prog[23785]: This is message 42
prog[23785]: This is message 43
prog[23785]: This is message 44
prog[23785]: This is message 45
prog[23785]: This is message 46
prog[23785]: This is message 47
prog[23785]: This is message 48
prog[23785]: This is message 49
prog[23785]: This is message 50
prog[23785]: This is message 51
prog[23785]: This is message 52
prog[23785]: This is message 53
prog[23785]: This is message 54
prog[23785]: This is message 55
prog[23785]: This is message 56
prog[23785]: This is message 57
My PID = 23783: Child PID = 23785 has been stopped by a signal, signo = 19
prog[23786]: This is message 10
prog[23786]: This is message 11
prog[23786]: This is message 12
prog[23786]: This is message 13
My PID = 23783: Child PID = 23786 has been stopped by a signal, signo = 19
prog[23784]: This is message 17
```

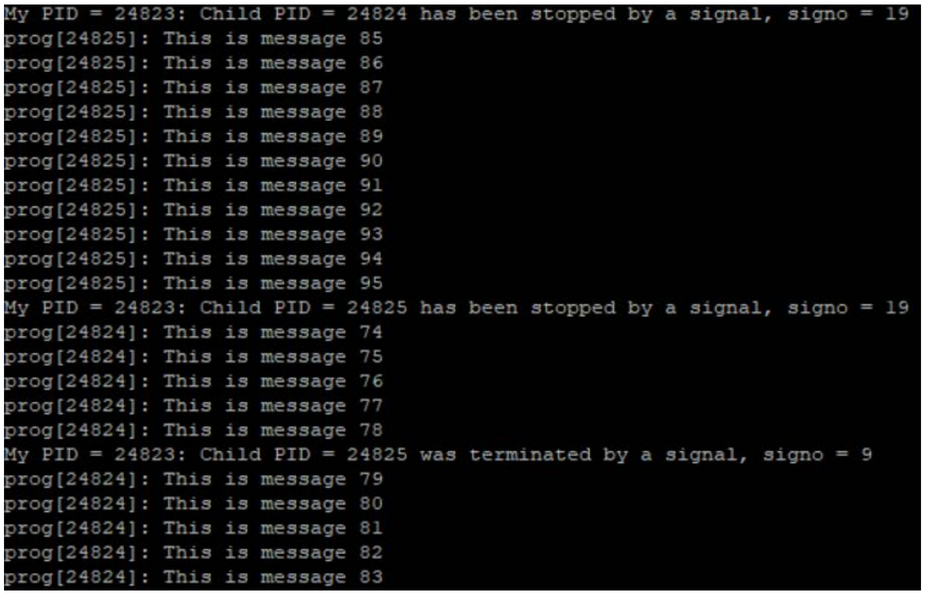


```
prog[23784]: This is message 18
prog[23784]: This is message 19
prog[23784]: This is message 20
prog[23784]: This is message 21
prog[23784]: This is message 22
My PID = 23783: Child PID = 23784 has been stopped by a signal, signo = 19
prog[23785]: This is message 58
prog[23785]: This is message 59
prog[23785]: This is message 60
prog[23785]: This is message 61
prog[23785]: This is message 62
prog[23785]: This is message 63
prog[23785]: This is message 64
prog[23785]: This is message 65
prog[23785]: This is message 66
prog[23785]: This is message 67
prog[23785]: This is message 68
prog[23785]: This is message 69
prog[23785]: This is message 70
prog[23785]: This is message 71
prog[23785]: This is message 72
prog[23785]: This is message 73
prog[23785]: This is message 74
prog[23785]: This is message 75
prog[23785]: This is message 76
My PID = 23783: Child PID = 23785 has been stopped by a signal, signo = 19
prog[23786]: This is message 14
prog[23786]: This is message 15
prog[23786]: This is message 16
prog[23786]: This is message 17
prog[23786]: This is message 18
My PID = 23783: Child PID = 23786 has been stopped by a signal, signo = 19
prog[23784]: This is message 23
prog[23784]: This is message 24
^Z
[5]+  Stopped                 ./scheduler prog prog prog
oslabc16@os-node1:~/Ex4$ 
```

Ερωτήσεις

1. Με τη χρήση μάσκας ο χρονοδρομολογητής μέσω της συνάρτησης install_signal_handlers εξασφαλίζει ότι μόνο ένα εκ των δύο σημάτων εκτελείται κάθε στιγμή και ποτέ και τα δύο. Σε αντίθεση με αυτή την υλοποίηση που χρησιμοποιεί σήματα, ένας πραγματικός χρονοδρομολογητής χρησιμοποιεί διακοπές υλικού (hardware interrupts). Με αυτό τον τρόπο το σύστημα γλιτώνει τυχόν καθυστερήσεις από τον έλεγχο για την κατάσταση των σημάτων.

2. Η εντολή SIGCHLD αναφέρεται στην επόμενη διεργασία από αυτήν που εκτελείται και της δίνει σήμα να ξεκινήσει. Όταν μία οποιαδήποτε διεργασία παιδί τερματίζεται αναπάντεχα τότε αφαιρείται από την ουρά και εκτελούνται κανονικά οι υπόλοιπες. Το πρόγραμμα αντιμετωπίζει μια διεργασία που τερματίστηκε από σήμα όπως ακριβώς θα αντιμετώπιζε μια διεργασία που τερματίστηκε κανονικά. Στο παράδειγμα που ακολουθεί δόθηκε σήμα SIGKILL στη διεργασία με pid=24825

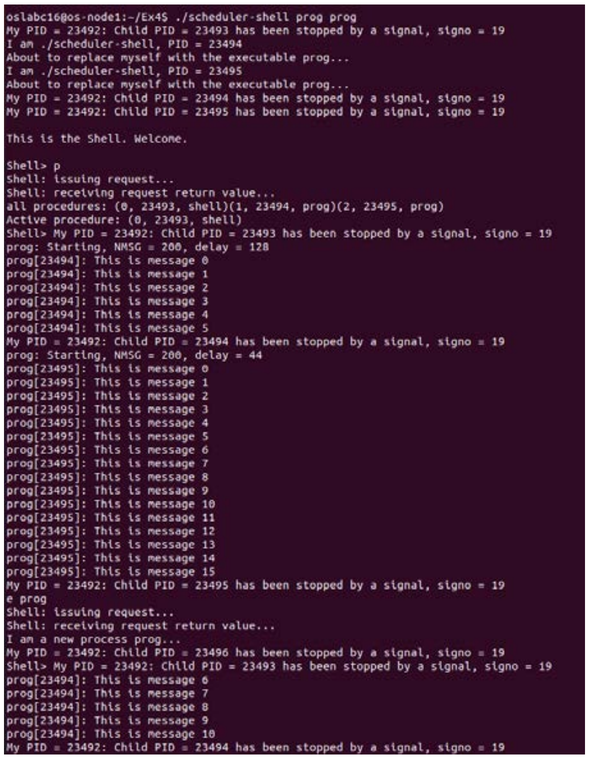

```
My PID = 24823: Child PID = 24824 has been stopped by a signal, signo = 19
prog[24825]: This is message 85
prog[24825]: This is message 86
prog[24825]: This is message 87
prog[24825]: This is message 88
prog[24825]: This is message 89
prog[24825]: This is message 90
prog[24825]: This is message 91
prog[24825]: This is message 92
prog[24825]: This is message 93
prog[24825]: This is message 94
prog[24825]: This is message 95
My PID = 24823: Child PID = 24825 has been stopped by a signal, signo = 19
prog[24824]: This is message 74
prog[24824]: This is message 75
prog[24824]: This is message 76
prog[24824]: This is message 77
prog[24824]: This is message 78
My PID = 24823: Child PID = 24825 was terminated by a signal, signo = 9
prog[24824]: This is message 79
prog[24824]: This is message 80
prog[24824]: This is message 81
prog[24824]: This is message 82
prog[24824]: This is message 83
```

3. Η διαίρεση σε δύο σήματα SIGCHLD και SIGALARM και με τη χρήση του system call sigaction εξασφαλίζεται ότι η τρέχουσα διεργασία έχει σταματήσει πριν ξεκινήσει η επόμενη. Αν είχε χρησιμοποιηθεί μόνο ένα σήμα που θα σταματούσε μια διεργασία και θα ξεκινούσε την επόμενη δεν θα μπορούσε να διασφαλιστεί ότι έχει σταματήσει πλήρως η μία διεργασία πριν να ξεκινήσει η άλλη. Επομένως δεν θα λειτουργούσε σωστά η χρονοδρομολόγηση.

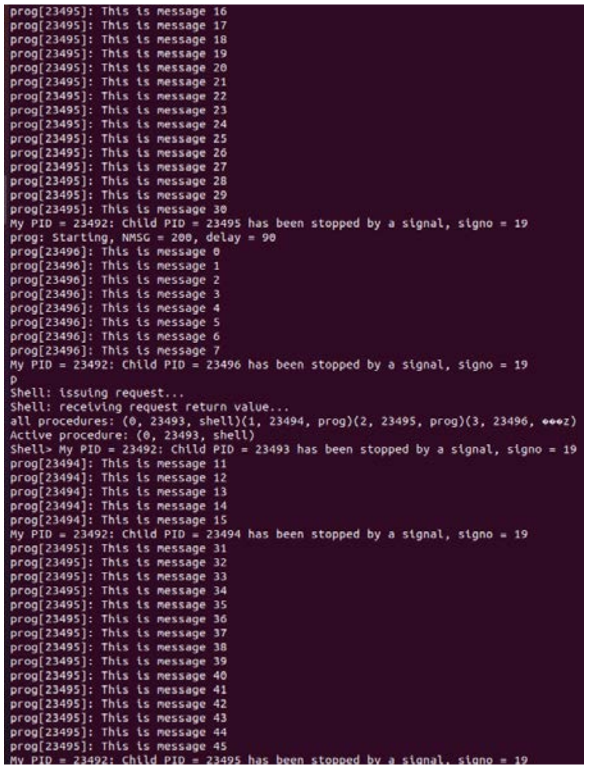
παραπομπή στον κώδικα

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή

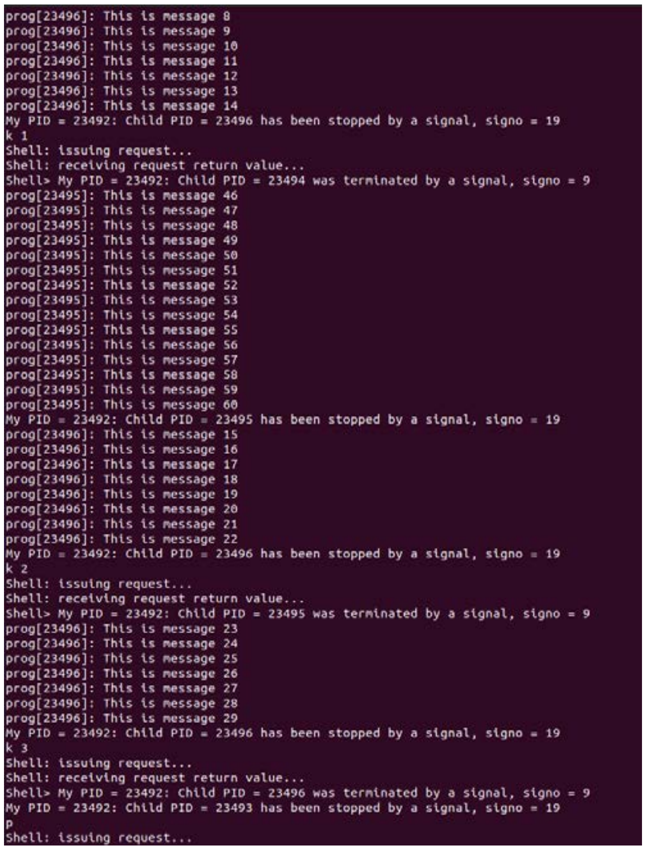
Ενδεικτική έξοδος εκτέλεσης:



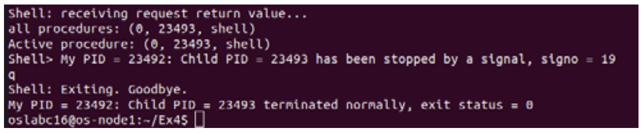
```
oslabc16@os-node1:~/Ex4$ ./scheduler-shell prog prog
My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 23494
About to replace myself with the executable prog...
I am ./scheduler-shell, PID = 23495
About to replace myself with the executable prog...
My PID = 23492: Child PID = 23494 has been stopped by a signal, signo = 19
My PID = 23492: Child PID = 23495 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
all procedures: (0, 23493, shell)(1, 23494, prog)(2, 23495, prog)
Active procedure: (0, 23493, shell)
Shell> My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 128
prog[23494]: This is message 0
prog[23494]: This is message 1
prog[23494]: This is message 2
prog[23494]: This is message 3
prog[23494]: This is message 4
prog[23494]: This is message 5
My PID = 23492: Child PID = 23494 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 44
prog[23495]: This is message 0
prog[23495]: This is message 1
prog[23495]: This is message 2
prog[23495]: This is message 3
prog[23495]: This is message 4
prog[23495]: This is message 5
prog[23495]: This is message 6
prog[23495]: This is message 7
prog[23495]: This is message 8
prog[23495]: This is message 9
prog[23495]: This is message 10
prog[23495]: This is message 11
prog[23495]: This is message 12
prog[23495]: This is message 13
prog[23495]: This is message 14
prog[23495]: This is message 15
My PID = 23492: Child PID = 23495 has been stopped by a signal, signo = 19
e prog
Shell: issuing request...
Shell: receiving request return value...
I am a new process prog...
My PID = 23492: Child PID = 23496 has been stopped by a signal, signo = 19
Shell> My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
prog[23494]: This is message 6
prog[23494]: This is message 7
prog[23494]: This is message 8
prog[23494]: This is message 9
prog[23494]: This is message 10
My PID = 23492: Child PID = 23494 has been stopped by a signal, signo = 19
```



```
prog[23495]: This is message 16
prog[23495]: This is message 17
prog[23495]: This is message 18
prog[23495]: This is message 19
prog[23495]: This is message 20
prog[23495]: This is message 21
prog[23495]: This is message 22
prog[23495]: This is message 23
prog[23495]: This is message 24
prog[23495]: This is message 25
prog[23495]: This is message 26
prog[23495]: This is message 27
prog[23495]: This is message 28
prog[23495]: This is message 29
prog[23495]: This is message 30
My PID = 23492: Child PID = 23495 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 90
prog[23496]: This is message 0
prog[23496]: This is message 1
prog[23496]: This is message 2
prog[23496]: This is message 3
prog[23496]: This is message 4
prog[23496]: This is message 5
prog[23496]: This is message 6
prog[23496]: This is message 7
My PID = 23492: Child PID = 23496 has been stopped by a signal, signo = 19
p
Shell: issuing request...
Shell: receiving request return value...
all procedures: (0, 23493, shell)(1, 23494, prog)(2, 23495, prog)(3, 23496, ◆◆◆z)
Active procedure: (0, 23493, shell)
Shell> My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
prog[23494]: This is message 11
prog[23494]: This is message 12
prog[23494]: This is message 13
prog[23494]: This is message 14
prog[23494]: This is message 15
My PID = 23492: Child PID = 23494 has been stopped by a signal, signo = 19
prog[23495]: This is message 31
prog[23495]: This is message 32
prog[23495]: This is message 33
prog[23495]: This is message 34
prog[23495]: This is message 35
prog[23495]: This is message 36
prog[23495]: This is message 37
prog[23495]: This is message 38
prog[23495]: This is message 39
prog[23495]: This is message 40
prog[23495]: This is message 41
prog[23495]: This is message 42
prog[23495]: This is message 43
prog[23495]: This is message 44
prog[23495]: This is message 45
My PID = 23492: Child PID = 23495 has been stopped by a signal, signo = 19
```



```
prog[23496]: This is message 8
prog[23496]: This is message 9
prog[23496]: This is message 10
prog[23496]: This is message 11
prog[23496]: This is message 12
prog[23496]: This is message 13
prog[23496]: This is message 14
My PID = 23492: Child PID = 23496 has been stopped by a signal, signo = 19
k 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 23492: Child PID = 23494 was terminated by a signal, signo = 9
prog[23495]: This is message 46
prog[23495]: This is message 47
prog[23495]: This is message 48
prog[23495]: This is message 49
prog[23495]: This is message 50
prog[23495]: This is message 51
prog[23495]: This is message 52
prog[23495]: This is message 53
prog[23495]: This is message 54
prog[23495]: This is message 55
prog[23495]: This is message 56
prog[23495]: This is message 57
prog[23495]: This is message 58
prog[23495]: This is message 59
prog[23495]: This is message 60
My PID = 23492: Child PID = 23495 has been stopped by a signal, signo = 19
prog[23496]: This is message 15
prog[23496]: This is message 16
prog[23496]: This is message 17
prog[23496]: This is message 18
prog[23496]: This is message 19
prog[23496]: This is message 20
prog[23496]: This is message 21
prog[23496]: This is message 22
My PID = 23492: Child PID = 23496 has been stopped by a signal, signo = 19
k 2
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 23492: Child PID = 23495 was terminated by a signal, signo = 9
prog[23496]: This is message 23
prog[23496]: This is message 24
prog[23496]: This is message 25
prog[23496]: This is message 26
prog[23496]: This is message 27
prog[23496]: This is message 28
prog[23496]: This is message 29
My PID = 23492: Child PID = 23496 has been stopped by a signal, signo = 19
k 3
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 23492: Child PID = 23496 was terminated by a signal, signo = 9
My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
p
Shell: issuing request...
```



```
Shell: receiving request return value...
all procedures: (0, 23493, shell)
Active procedure: (0, 23493, shell)
Shell> My PID = 23492: Child PID = 23493 has been stopped by a signal, signo = 19
q
Shell: Exiting. Goodbye.
My PID = 23492: Child PID = 23493 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex4$ 
```

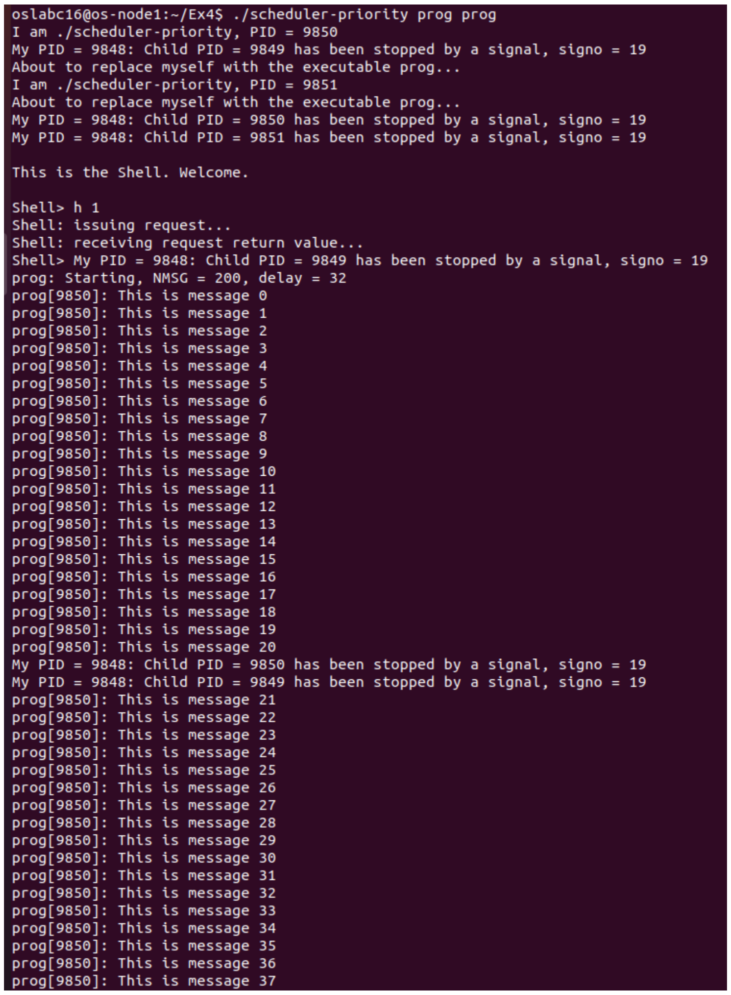
Ερωτήσεις

1. Όταν ο φλοιός υφίσταται χρονοδρομολόγηση τότε εμφανίζεται πάντα αυτός ως η τρέχουσα διεργασία όταν εκτελείται η εντολή p. Αυτό συμβαίνει διότι η εντολή p εκτελείται μόνο όταν τρέχει η διεργασία του φλοιού. Επομένως με την παρούσα υλοποίηση δεν θα μπορούσε να εμφανιστεί άλλη διεργασία σαν τρέχουσα, παρόλο που όλες τρέχουν κανονικά.
2. Οι κλήσεις signals_disable(),signals_enable() απαγορεύουν τη λήψη των σημάτων SIGALRM και SIGCHLD την ώρα που εκτελούνται οι αιτήσεις φλοιού. Αυτό εξασφαλίζει ότι δεν θα λειτουργεί μια διεργασία ταυτόχρονα με τις αιτήσεις και θα ξεκινήσουν κανονικά όταν αυτές ολοκληρωθούν. Το πιο πάνω είναι απαραίτητο αφού οι αιτήσεις φλοιού μπορούν να αλλάξουν την ουρά εκτέλεσης (προσθήκη νέας διεργασίας ή διαγραφή υπάρχουσας) άρα είναι σημαντικό να μην εκτελούνται ταυτόχρονα οι διεργασίες, εξάλλου αυτός είναι και ο λόγος ύπαρξης του φλοιού στη στοίβα διεργασιών.

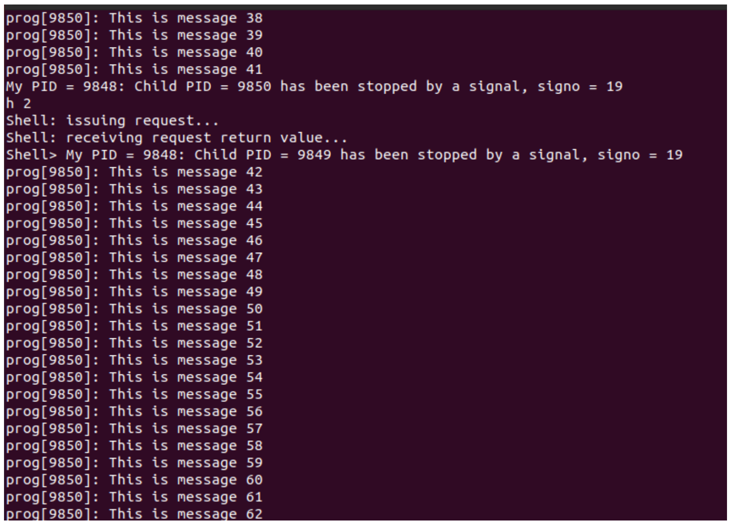
παραπομπή στον κώδικα

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

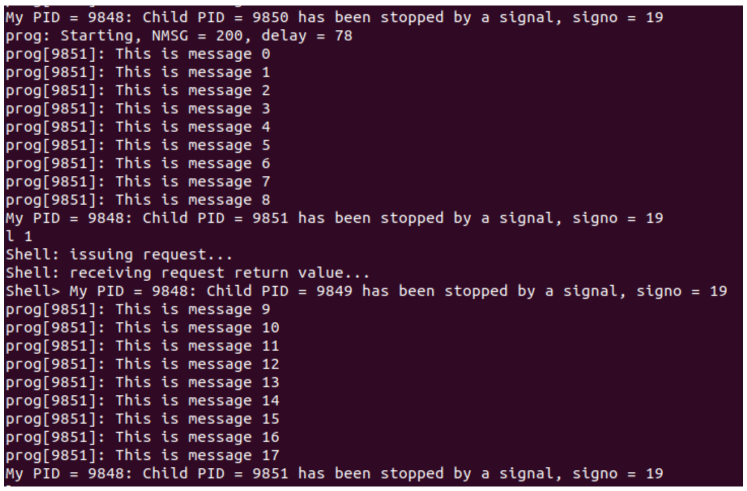
Ενδεικτική έξοδος εκτέλεσης:



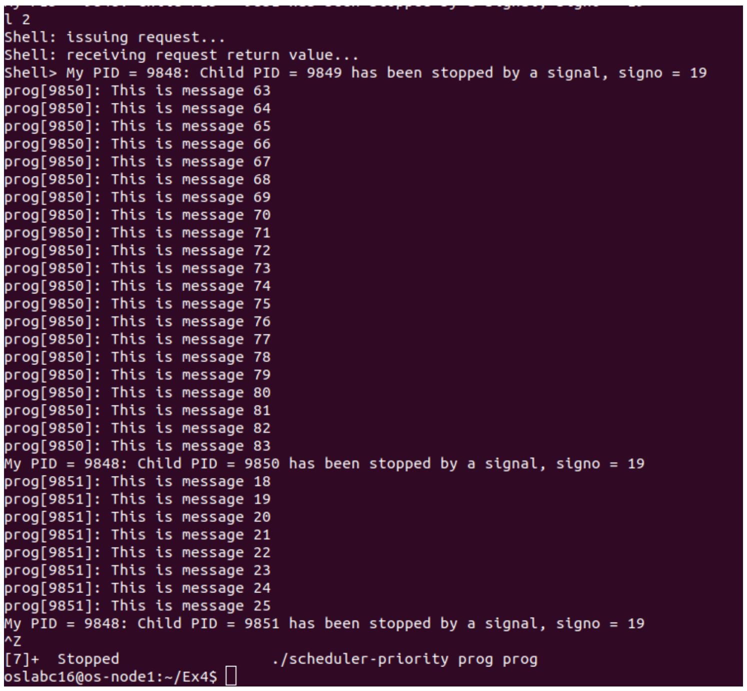
```
oslabc16@os-node1:~/Ex4$ ./scheduler-priority prog prog
I am ./scheduler-priority, PID = 9850
My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
About to replace myself with the executable prog...
I am ./scheduler-priority, PID = 9851
About to replace myself with the executable prog...
My PID = 9848: Child PID = 9850 has been stopped by a signal, signo = 19
My PID = 9848: Child PID = 9851 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> h 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 32
prog[9850]: This is message 0
prog[9850]: This is message 1
prog[9850]: This is message 2
prog[9850]: This is message 3
prog[9850]: This is message 4
prog[9850]: This is message 5
prog[9850]: This is message 6
prog[9850]: This is message 7
prog[9850]: This is message 8
prog[9850]: This is message 9
prog[9850]: This is message 10
prog[9850]: This is message 11
prog[9850]: This is message 12
prog[9850]: This is message 13
prog[9850]: This is message 14
prog[9850]: This is message 15
prog[9850]: This is message 16
prog[9850]: This is message 17
prog[9850]: This is message 18
prog[9850]: This is message 19
prog[9850]: This is message 20
My PID = 9848: Child PID = 9850 has been stopped by a signal, signo = 19
My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
prog[9850]: This is message 21
prog[9850]: This is message 22
prog[9850]: This is message 23
prog[9850]: This is message 24
prog[9850]: This is message 25
prog[9850]: This is message 26
prog[9850]: This is message 27
prog[9850]: This is message 28
prog[9850]: This is message 29
prog[9850]: This is message 30
prog[9850]: This is message 31
prog[9850]: This is message 32
prog[9850]: This is message 33
prog[9850]: This is message 34
prog[9850]: This is message 35
prog[9850]: This is message 36
prog[9850]: This is message 37
```



```
prog[9850]: This is message 38
prog[9850]: This is message 39
prog[9850]: This is message 40
prog[9850]: This is message 41
My PID = 9848: Child PID = 9850 has been stopped by a signal, signo = 19
h 2
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
prog[9850]: This is message 42
prog[9850]: This is message 43
prog[9850]: This is message 44
prog[9850]: This is message 45
prog[9850]: This is message 46
prog[9850]: This is message 47
prog[9850]: This is message 48
prog[9850]: This is message 49
prog[9850]: This is message 50
prog[9850]: This is message 51
prog[9850]: This is message 52
prog[9850]: This is message 53
prog[9850]: This is message 54
prog[9850]: This is message 55
prog[9850]: This is message 56
prog[9850]: This is message 57
prog[9850]: This is message 58
prog[9850]: This is message 59
prog[9850]: This is message 60
prog[9850]: This is message 61
prog[9850]: This is message 62
```



```
My PID = 9848: Child PID = 9850 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 78
prog[9851]: This is message 0
prog[9851]: This is message 1
prog[9851]: This is message 2
prog[9851]: This is message 3
prog[9851]: This is message 4
prog[9851]: This is message 5
prog[9851]: This is message 6
prog[9851]: This is message 7
prog[9851]: This is message 8
My PID = 9848: Child PID = 9851 has been stopped by a signal, signo = 19
l 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
prog[9851]: This is message 9
prog[9851]: This is message 10
prog[9851]: This is message 11
prog[9851]: This is message 12
prog[9851]: This is message 13
prog[9851]: This is message 14
prog[9851]: This is message 15
prog[9851]: This is message 16
prog[9851]: This is message 17
My PID = 9848: Child PID = 9851 has been stopped by a signal, signo = 19
```



```
l 2
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 9848: Child PID = 9849 has been stopped by a signal, signo = 19
prog[9850]: This is message 63
prog[9850]: This is message 64
prog[9850]: This is message 65
prog[9850]: This is message 66
prog[9850]: This is message 67
prog[9850]: This is message 68
prog[9850]: This is message 69
prog[9850]: This is message 70
prog[9850]: This is message 71
prog[9850]: This is message 72
prog[9850]: This is message 73
prog[9850]: This is message 74
prog[9850]: This is message 75
prog[9850]: This is message 76
prog[9850]: This is message 77
prog[9850]: This is message 78
prog[9850]: This is message 79
prog[9850]: This is message 80
prog[9850]: This is message 81
prog[9850]: This is message 82
prog[9850]: This is message 83
My PID = 9848: Child PID = 9850 has been stopped by a signal, signo = 19
prog[9851]: This is message 18
prog[9851]: This is message 19
prog[9851]: This is message 20
prog[9851]: This is message 21
prog[9851]: This is message 22
prog[9851]: This is message 23
prog[9851]: This is message 24
prog[9851]: This is message 25
My PID = 9848: Child PID = 9851 has been stopped by a signal, signo = 19
^Z
[7]+  Stopped                 ./scheduler-priority prog prog
oslabc16@os-node1:~/Ex4$
```

Ερώτηση

1. Λιμοκτονία διεργασίας παρατηρείται όταν μια διεργασία χαμηλής προτεραιότητας παραμελείται επ' άπειρον από το σύστημα διότι εκτελούνται πάντα διεργασίες με υψηλή προτεραιότητα. Για να συμβεί αυτό πρέπει να δίνεται συνεχώς εντολή στο φλοιό για δημιουργία νέων διεργασιών με υψηλή προτεραιότητα ενώ αρχικά υπήρχε στην ουρά κάποια διεργασία με χαμηλή προτεραιότητα. Αυτή η διεργασία δεν θα εκτελεστεί ποτέ με αποτέλεσμα να παρατηρείται το φαινόμενο της λιμοκτονίας. Επίσης αν υπήρχε χρονικός περιορισμός για το χρόνο που μπορεί να παραμείνει μια διεργασία στο σύστημα και μια διεργασία χαμηλής προτεραιότητας είχε παραμελήθηκε από το σύστημα για περισσότερο χρόνο τότε πάλι θα παρατηρηθεί φαινόμενο λιμοκτονίας.

παραπομπή στον κώδικα

Παράρτημα: Κώδικας της άσκησης

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

```c
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <assert.h>
#include <sys/wait.h>
#include <sys/types.h>
#include "proc-common.h"
#include "request.h"
/* Compile-time parameters. */
#define SCHED_TQ_SEC 2                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */

//creating a struct, every procedure is saved in a node
//nodes are linked to create a custom queue structure
struct node{
      pid_t pid;
      int numproc;
      struct node *next;
};

//declare the head, tail and it nodes of the queue
struct node *head = NULL, *tail = NULL, *it = NULL;

//function used to delete an element from the queue based on its pid
void deleteNode(pid_t p){
      struct node *toBeDeleted=head;
      while (toBeDeleted->pid != p)
            toBeDeleted = toBeDeleted->next;
      if (toBeDeleted!=NULL && (toBeDeleted->next != toBeDeleted)){
            struct node *q = head;
            while (q->next!=toBeDeleted)
                  q = q->next;
            if(toBeDeleted==head)
                  head=head->next;
            if(toBeDeleted==tail)
                  tail=q;
            q->next=toBeDeleted->next;
            free(toBeDeleted);
      }
      else {
            head = NULL;
            tail = NULL;
            free(toBeDeleted);
      }
}

//SIGALRM handler to stop a procedure
```

```
static void sigalrm_handler(int signum){
      kill(it->pid,SIGSTOP);
}

//SIGCHLD handler to start the next procedure
static void
sigchld_handler(int signum)
{
      pid_t p;
      int status;
      for(;;){
            p = waitpid(-1, &status, WUNTRACED | WNOHANG);
            if (p < 0){
                  perror("waitpid");
                  exit(1);
            }
            if (p == 0)
                  break;
            explain_wait_status(p,status);
            if (WIFEXITED(status) || WIFSIGNALED(status) ||
WIFSTOPPED(status)){
//if child was terminated normally or by signal, it is removed from the queue
                  if (WIFEXITED(status) || WIFSIGNALED(status)){
                        deleteNode(p);
                        if (head == NULL)
                              exit(0);
                  }
                  //gives signal to the next procedure to start
                  it = it->next;
                  kill(it->pid,SIGCONT);
                  alarm(SCHED_TQ_SEC);
            }
      }
}
/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.*/
static void install_signal_handlers(void){
      sigset_t sigset;
      struct sigaction sa;
      sa.sa_handler = sigchld_handler;
      sa.sa_flags = SA_RESTART;
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGCHLD);
      sigaddset(&sigset, SIGALRM);
      sa.sa_mask = sigset;
      if (sigaction(SIGCHLD, &sa, NULL) < 0) {
            perror("sigaction: sigchld");
            exit(1);
      }
      sa.sa_handler = sigalrm_handler;
      if (sigaction(SIGALRM, &sa, NULL) < 0) {
            perror("sigaction: sigalrm");
            exit(1);
      }
      /*
```

```
         * Ignore SIGPIPE, so that write()s to pipes
         * with no reader do not result in us being killed,
         * and write() returns EPIPE instead.
         */
        if (signal(SIGPIPE, SIG_IGN) < 0) {
                perror("signal: sigpipe");
                exit(1);
        }
}

int main(int argc, char *argv[])
{
        int nproc,i;
        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */
        for(i=1;i<argc;i++){
                struct node *curr = (struct node*) malloc(sizeof(struct node));
                curr->pid = fork();
                if(curr->pid<0){
                        perror("fork");
                        exit(1);
                }
                if(curr->pid==0){
                        char *executable = argv[i];
                        char *newargv[] = { executable, NULL, NULL, NULL };
                        char *newenviron[] = { NULL };
                        printf("I am %s, PID = %ld\n", argv[0], (long)getpid());
                        printf("About to replace myself with the executable
%s...\n", executable);
                        sleep(2);
                        raise(SIGSTOP);
                        execve(executable, newargv, newenviron);
                        /* execve() only returns on error */
                        perror("execve");
                        exit(1);
                }
                curr->numproc = i-1;
                //if queue is empty then curr node is added as both head and tail
                if (head == NULL){
                        head = curr;
                        head->next = head;
                        tail = head;
                }
                //else, curr node is added to the end of the queue
                else{
                        tail->next = curr;
                        curr->next = head;
                        tail=curr;
                }
        }
        nproc = argc-1; /* number of proccesses goes here */
        /* Wait for all children to raise SIGSTOP before exec()ing. */
        wait_for_ready_children(nproc);
        /* Install SIGALRM and SIGCHLD handlers. */
        install_signal_handlers();
        //checks if there are any procedures
        if (nproc == 0) {
              fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
              exit(1);
        }
        //it sends signal to the first procedure to start
        it = head;
        if(kill(head->pid, SIGCONT) < 0){
          perror("First child Cont error");
          exit(1);
      }
        alarm(SCHED_TQ_SEC);
        /* loop forever  until we exit from inside a signal handler. */
        while (pause())
              ;
        /* Unreachable */
        fprintf(stderr, "Internal error: Reached unreachable point\n");
        return 1;
}
```

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή

```
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <assert.h>
#include <sys/wait.h>
#include <sys/types.h>
#include "proc-common.h"
#include "request.h"

/* Compile-time parameters. */
#define SCHED_TQ_SEC 2                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

//creating a struct, every procedure is saved in a node
//nodes are linked to create a custom queue structure
//exec attribute was added to the struct to keep the name of the procedure
//and determine the shell procedure
struct node{
      pid_t pid;
      int numproc;
      char exec[TASK_NAME_SZ];
      struct node *next;
};

//declare the head, tail and it nodes of the queue
struct node *head = NULL, *tail = NULL, *it = NULL;

//function used to delete an element from the queue based on its pid
void deleteNode(pid_t p){
      struct node *toBeDeleted=head;
      while (toBeDeleted->pid != p)
            toBeDeleted = toBeDeleted->next;
      if (toBeDeleted!=NULL && (toBeDeleted->next != toBeDeleted)){
            struct node *q = head;
            while (q->next!=toBeDeleted)
                  q = q->next;
            if(toBeDeleted==head)
                  head=head->next;
            if(toBeDeleted==tail)
                  tail=q;
            q->next=toBeDeleted->next;
            free(toBeDeleted);
      }
      else {
            head = NULL;
            tail = NULL;
            free(toBeDeleted);
      }
}
```

```
/* Print a list of all tasks currently being scheduled.  */
static void sched_print_tasks(void){
      struct node *temp=head;
      printf("all procedures: ");
      do{
            printf("(%d, %d, %s)",temp->numproc,temp->pid,temp->exec);
            temp=temp->next;
      }while(temp!=head);
      printf("\nActive procedure: (%d, %d, %s)\n",it->numproc,it->pid,it-
>exec);
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int sched_kill_task_by_id(int id){
      struct node *temp=head;
      while(temp->numproc!=id){
            temp=temp->next;
/*if there is no process with an id that matches the one given as a parameter
then temp will reach the head of the queue in that case it returns the signal
ESRCH
ESRCH: “No such process.” No process matches the specified process ID
            */
            if(temp==head)
                  return -ESRCH;
      }
      return kill(temp->pid,SIGKILL);
}
// Create a new task using the given executable name and adding it to tail of
the queue
static void sched_create_task(char *executable){
      struct node *curr = (struct node*) malloc(sizeof(struct node));
      curr->pid = fork();
      if(curr->pid<0){
            perror("fork");
            exit(1);
      }
      if(curr->pid==0){
            char *newargv[] = { executable, NULL, NULL, NULL };
            char *newenviron[] = { NULL };
            printf("I am a new process %s...\n", executable);
            raise(SIGSTOP);
            execve(executable, newargv, newenviron);
            /* execve() only returns on error */
            perror("execve");
            exit(1);
      }
      curr->numproc = (tail->numproc)+1;
      strcpy(curr->exec ,executable);
      tail->next = curr;
      curr->next = head;
      tail=curr;
      wait_for_ready_children(1);
}
```

```
/* Process requests by the shell.  */
static int process_request(struct request_struct *rq){
      switch (rq->request_no) {
            case REQ_PRINT_TASKS:
                  sched_print_tasks();
                  return 0;

            case REQ_KILL_TASK:
                  return sched_kill_task_by_id(rq->task_arg);

            case REQ_EXEC_TASK:
                  sched_create_task(rq->exec_task_arg);
                  return 0;

            default:
                  return -ENOSYS;
      }
}

//SIGALRM handler to stop a procedure
static void sigalrm_handler(int signum){
      kill(it->pid,SIGSTOP);
}

//SIGCHLD handler to start the next procedure
static void sigchld_handler(int signum){
      pid_t p;
      int status;
      for(;;){
            p = waitpid(-1, &status, WUNTRACED | WNOHANG);
            if (p < 0){
                  perror("waitpid");
                  exit(1);
            }
            if (p == 0)
                  break;
            explain_wait_status(p,status);
            if (WIFEXITED(status) || WIFSIGNALED(status) ||
WIFSTOPPED(status)){
//if child was terminated normally or by signal then it must be removed from
the queue
                  if (WIFEXITED(status) || WIFSIGNALED(status)){
                        deleteNode(p);
                        if (head == NULL)
                              exit(0);
                  }
//gives signal to the next procedure to start
                  it = it->next;
                  kill(it->pid,SIGCONT);
                  alarm(SCHED_TQ_SEC);
            }
      }
}

/* Disable delivery of SIGALRM and SIGCHLD. */
static void signals_disable(void){
	sigset_t sigset;
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGALRM);
	sigaddset(&sigset, SIGCHLD);
	if (sigprocmask(SIG_BLOCK, &sigset, NULL) < 0) {
		perror("signals_disable: sigprocmask");
		exit(1);
	}
}

/* Enable delivery of SIGALRM and SIGCHLD.  */
static void signals_enable(void){
	sigset_t sigset;
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGALRM);
	sigaddset(&sigset, SIGCHLD);
	if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
		perror("signals_enable: sigprocmask");
		exit(1);
	}
}

/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */
static void install_signal_handlers(void){
	sigset_t sigset;
	struct sigaction sa;
	sa.sa_handler = sigchld_handler;
	sa.sa_flags = SA_RESTART;
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGCHLD);
	sigaddset(&sigset, SIGALRM);
	sa.sa_mask = sigset;
	if (sigaction(SIGCHLD, &sa, NULL) < 0) {
		perror("sigaction: sigchld");
		exit(1);
	}
	sa.sa_handler = sigalrm_handler;
	if (sigaction(SIGALRM, &sa, NULL) < 0) {
		perror("sigaction: sigalrm");
		exit(1);
	}
	/*
	 * Ignore SIGPIPE, so that write()s to pipes
	 * with no reader do not result in us being killed,
	 * and write() returns EPIPE instead.
	 */
	if (signal(SIGPIPE, SIG_IGN) < 0) {
		perror("signal: sigpipe");
		exit(1);
	}
}

static void do_shell(char *executable, int wfd, int rfd){
		char arg1[10], arg2[10];
		char *newargv[] = { executable, NULL, NULL, NULL };
		char *newenviron[] = { NULL };
		sprintf(arg1, "%05d", wfd);
		sprintf(arg2, "%05d", rfd);
		newargv[1] = arg1;
		newargv[2] = arg2;
		raise(SIGSTOP);
		execve(executable, newargv, newenviron);
		/* execve() only returns on error */
		perror("scheduler: child: execve");
		exit(1);
}

/* Create a new shell task.
 * The shell gets special treatment:
 * two pipes are created for communication and passed
 * as command-line arguments to the executable.
 */
static void sched_create_shell(char *executable, int *request_fd, int
*return_fd){
		pid_t p;
		int pfds_rq[2], pfds_ret[2];
		if (pipe(pfds_rq) < 0 || pipe(pfds_ret) < 0) {
			perror("pipe");
			exit(1);
		}
		p = fork();
		if (p < 0) {
			perror("scheduler: fork");
			exit(1);
		}
		if (p == 0) {
			/* Child */
			close(pfds_rq[0]);
			close(pfds_ret[1]);
			do_shell(executable, pfds_rq[1], pfds_ret[0]);
			assert(0);
		}
		/* Parent */
		head->pid = p;    //shell's pid is saved in the structure
		close(pfds_rq[1]);
		close(pfds_ret[0]);
		*request_fd = pfds_rq[0];
		*return_fd = pfds_ret[1];
}

static void shell_request_loop(int request_fd, int return_fd){
		int ret;
		struct request_struct rq;
		/*
		 * Keep receiving requests from the shell.
		 */
		for (;;) {
			if (read(request_fd, &rq, sizeof(rq)) != sizeof(rq)) {
				perror("scheduler: read from shell");
				fprintf(stderr, "Scheduler: giving up on shell request 
processing.\n");
				break;
			}
			signals_disable();
			ret = process_request(&rq);
			signals_enable();
			if (write(return_fd, &ret, sizeof(ret)) != sizeof(ret)) {
				perror("scheduler: write to shell");
				fprintf(stderr, "Scheduler: giving up on shell request 
processing.\n");
				break;
			}
		}
}

int main(int argc, char *argv[]){
		int nproc,i;
		/* Two file descriptors for communication with the shell */
		static int request_fd, return_fd;
		head = (struct node*) malloc(sizeof(struct node));
		/* Create the shell. */
		sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);
		//create a node for the shell and place it as the head of the queue
		strcpy(head->exec , SHELL_EXECUTABLE_NAME);
		head->numproc=0;
		head->next=head;
		tail=head;
		nproc = argc-1; /* number of proccesses goes here */
		/*
		 * For each of argv[1] to argv[argc - 1],
		 * create a new child process, add it to the process list.
		 */
		for(i=1;i<=nproc;i++){
			struct node *curr = (struct node*) malloc(sizeof(struct node));
			curr->pid = fork();
			if(curr->pid<0){
				perror("fork");
				exit(1);
			}
			if(curr->pid==0){
				char *executable = argv[i];
				char *newargv[] = { executable, NULL, NULL, NULL };
				char *newenviron[] = { NULL };
				printf("I am %s, PID = %ld\n", argv[0], (long)getpid());
				printf("About to replace myself with the executable 
%s...\n", executable);
				sleep(2);
				raise(SIGSTOP);
				execve(executable, newargv, newenviron);
				/* execve() only returns on error */
				perror("execve");
				exit(1);
			}
			curr->numproc = i;
			strcpy(curr->exec ,argv[i]);
```

```
            //if queue is empty then curr node is added as both head and tail
            if (head == NULL){
                  head = curr;
                  head->next = head;
                  tail = head;
            }
            //else, curr node is added to the end of the queue
            else{
                  tail->next = curr;
                  curr->next = head;
                  tail=curr;
            }
      }
      /* Wait for all children to raise SIGSTOP before exec()ing. */
      wait_for_ready_children(nproc+1);
      /* Install SIGALRM and SIGCHLD handlers. */
      install_signal_handlers();
      if (nproc == 0) {
            fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
            exit(1);
      }
      //it sends signal to the first procedure to start
      it = head;
      if(kill(head->pid, SIGCONT) < 0){
        perror("First child Cont error");
        exit(1);
    }
      alarm(SCHED_TQ_SEC);
      shell_request_loop(request_fd, return_fd);
      /* Now that the shell is gone, just loop forever
       * until we exit from inside a signal handler.
       */
      while (pause())
            ;
      /* Unreachable */
      fprintf(stderr, "Internal error: Reached unreachable point\n");
      return 1;
}
```

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

```
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <assert.h>
#include <sys/wait.h>
#include <sys/types.h>
#include "proc-common.h"
#include "request.h"

/* Compile-time parameters. */
#define SCHED_TQ_SEC 2                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

//creating a struct, every procedure is saved in a node
//nodes are linked to create a custom queue structure
//exec attribute was added to the struct to keep the name of the procedure
//and determine the shell procedure
struct node{
      pid_t pid;
      int numproc;
      char exec[TASK_NAME_SZ];
      char priority;
      struct node *next;
};

//declare the head, tail and it nodes of the queue
struct node *head = NULL, *tail = NULL, *it = NULL;

//function used to delete an element from the queue based on its pid
void deleteNode(pid_t p){
      struct node *toBeDeleted=head;
      while (toBeDeleted->pid != p)
            toBeDeleted = toBeDeleted->next;
      if (toBeDeleted!=NULL && (toBeDeleted->next != toBeDeleted)){
            struct node *q = head;
            while (q->next!=toBeDeleted)
                  q = q->next;
            if(toBeDeleted==head)
                  head=head->next;
            if(toBeDeleted==tail)
                  tail=q;
            q->next=toBeDeleted->next;
            free(toBeDeleted);
      }
      else {
            head = NULL;
            tail = NULL;
            free(toBeDeleted);
      }
}
```

```
/* Print a list of all tasks currently being scheduled.
   This time it also prints the priority of each process*/
static void sched_print_tasks(void){
	struct node *temp=head;
	printf("all procedures: ");
	do{
		printf("(%d, %d, %s, %c)",temp->numproc,temp->pid,temp-
>exec,temp->priority);
		temp=temp->next;
	}while(temp!=head);
	printf("\nActive procedure: (%d, %d, %s, %c)\n",it->numproc,it->pid,it-
>exec,temp->priority);
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int sched_kill_task_by_id(int id){
	struct node *temp=head;
	while(temp->numproc!=id){
		temp=temp->next;
/*if there is no process with an id that matches the one given as a parameter
then temp will reach the head of the queue in that case it returns the signal
ESRCH
ESRCH: .No such process.. No process matches the specified process ID
			  */
		if(temp==head)
			return -ESRCH;
	}
	return kill(temp->pid,SIGKILL);
}

// Create a new task using the given executable name and adding it to tail of
the queue
// and setting its priority to low
static void sched_create_task(char *executable){
	struct node *curr = (struct node*) malloc(sizeof(struct node));
	curr->pid = fork();
	if(curr->pid<0){
		perror("fork");
		exit(1);
	}
	if(curr->pid==0){
		char *newargv[] = { executable, NULL, NULL, NULL };
		char *newenviron[] = { NULL };
		printf("I am a new process %s...\n", executable);
		raise(SIGSTOP);
		execve(executable, newargv, newenviron);
		/* execve() only returns on error */
		perror("execve");
		exit(1);
	}
	curr->priority='l';
	curr->numproc = (tail->numproc)+1;
	strcpy(curr->exec ,executable);
	tail->next = curr;
	curr->next = head;
	tail=curr;
	wait_for_ready_children(1);
}

//change the priority of a process from low to high and relocate it in the
queue
static int sched_high_task_by_id(int id){
	struct node *temp=head->next;
	struct node *prev_temp=head;
	//find the process and its previous process
	while(temp->numproc!=id){
		prev_temp=temp;
		temp=temp->next;
		if(temp==head)
			return -ESRCH;
	}
	if(temp->priority=='h'){
		printf("process is already high priority\n");
		return 0;
	}
	//set its priority to high
	temp->priority='h';
	//if it was the only process with low priority then it stays at the
tail of the queue
	if((temp==tail)&&(prev_temp->priority='h'))
		return 0;
	prev_temp->next=temp->next;
	//if it was at the tail of the queue but there are more processes with
low priority, tail changes
	if(temp==tail)
		tail=prev_temp;
	//finds the last process with high priority and places the current
process there
	struct node *lastHigh=head;
	while(((lastHigh->next)->priority=='h')&&(lastHigh->next!=head))
		lastHigh=lastHigh->next;
	temp->next=lastHigh->next;
	lastHigh->next=temp;
	return 0;
}

//change the priority of a process from high to low and relocate it in the
queue
static int sched_low_task_by_id(int id){
	struct node *temp=head->next;
	struct node *prev_temp=head;
	//find the process and its previous process
	while(temp->numproc!=id){
		prev_temp=temp;
		temp=temp->next;
		if(temp==head)
			return -ESRCH;
	}
	if(temp->priority=='l'){
		printf("process is already low priority\n");
		return 0;
	}
	//set its priority to low
	temp->priority='l';
	//if it is already at the tail of the queue then it doesn't change 
place
	if(temp==tail)
		return 0;
	//current process is relocated to the tail of the queue
	prev_temp->next=temp->next;
	temp->next=tail->next;
	tail->next=temp;
	tail=temp;
	return 0;
}

/* Process requests by the shell.
   proccesses REQ_HIGH_TASK, REQ_LOW_TASK are added*/
static int process_request(struct request_struct *rq){
	switch (rq->request_no) {
		case REQ_PRINT_TASKS:
			sched_print_tasks();
			return 0;

		case REQ_KILL_TASK:
			return sched_kill_task_by_id(rq->task_arg);

		case REQ_EXEC_TASK:
			sched_create_task(rq->exec_task_arg);
			return 0;

		case REQ_HIGH_TASK:
			return sched_high_task_by_id(rq->task_arg);

		case REQ_LOW_TASK:
			return sched_low_task_by_id(rq->task_arg);

		default:
			return -ENOSYS;
	}
}

//SIGALRM handler to stop a procedure
static void sigalrm_handler(int signum){
	kill(it->pid,SIGSTOP);
}

//SIGCHLD handler to start the next procedure
static void sigchld_handler(int signum){
	pid_t p;
	int status;
	for(;;){
		p = waitpid(-1, &status, WUNTRACED | WNOHANG);
		if (p < 0){
			perror("waitpid");
```

```
				exit(1);
			}
			if (p == 0)
				break;
			explain_wait_status(p,status);
			if (WIFEXITED(status) || WIFSIGNALED(status) ||
WIFSTOPPED(status)){
				//if child was terminated normally or by signal then it
must be removed from the queue
				if (WIFEXITED(status) || WIFSIGNALED(status)){
					deleteNode(p);
					if (head == NULL)
						exit(0);
				}
				//if the current process is the shell then start the next
process no matter what
				if(it==head){
					it = it->next;
					kill(it->pid,SIGCONT);
					alarm(SCHED_TQ_SEC);
				}
				//if the current process has high priority then start the
next process only if it also has high priority
				else if(it->priority=='h'){
					if(it->next->priority=='h'){
						it = it->next;
						kill(it->pid,SIGCONT);
						alarm(SCHED_TQ_SEC);
					}
					//if not then start the shell
					else{
						it = head;
						kill(it->pid,SIGCONT);
						alarm(SCHED_TQ_SEC);
					}
				}
				//if the current process has low priority then start the
next process no matter what
				else{
					it = it->next;
					kill(it->pid,SIGCONT);
					alarm(SCHED_TQ_SEC);
				}
			}
		}
}
/* Disable delivery of SIGALRM and SIGCHLD. */
static void signals_disable(void){
		sigset_t sigset;
		sigemptyset(&sigset);
		sigaddset(&sigset, SIGALRM);
		sigaddset(&sigset, SIGCHLD);
		if (sigprocmask(SIG_BLOCK, &sigset, NULL) < 0) {
			perror("signals_disable: sigprocmask");
			exit(1);
		}
```

```
}

/* Enable delivery of SIGALRM and SIGCHLD.  */
static void signals_enable(void){
      sigset_t sigset;
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGALRM);
      sigaddset(&sigset, SIGCHLD);
      if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
            perror("signals_enable: sigprocmask");
            exit(1);
      }
}

/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */
static void install_signal_handlers(void){
      sigset_t sigset;
      struct sigaction sa;
      sa.sa_handler = sigchld_handler;
      sa.sa_flags = SA_RESTART;
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGCHLD);
      sigaddset(&sigset, SIGALRM);
      sa.sa_mask = sigset;
      if (sigaction(SIGCHLD, &sa, NULL) < 0) {
            perror("sigaction: sigchld");
            exit(1);
      }
      sa.sa_handler = sigalrm_handler;
      if (sigaction(SIGALRM, &sa, NULL) < 0) {
            perror("sigaction: sigalrm");
            exit(1);
      }
      /*
       * Ignore SIGPIPE, so that write()s to pipes
       * with no reader do not result in us being killed,
       * and write() returns EPIPE instead.
       */
      if (signal(SIGPIPE, SIG_IGN) < 0) {
            perror("signal: sigpipe");
            exit(1);
      }
}

static void do_shell(char *executable, int wfd, int rfd){
      char arg1[10], arg2[10];
      char *newargv[] = { executable, NULL, NULL, NULL };
      char *newenviron[] = { NULL };
      sprintf(arg1, "%05d", wfd);
      sprintf(arg2, "%05d", rfd);
      newargv[1] = arg1;
      newargv[2] = arg2;
      raise(SIGSTOP);
```

```
		execve(executable, newargv, newenviron);
		/* execve() only returns on error */
		perror("scheduler: child: execve");
		exit(1);
}

/* Create a new shell task.
 * The shell gets special treatment:
 * two pipes are created for communication and passed
 * as command-line arguments to the executable.
 */
static void sched_create_shell(char *executable, int *request_fd, int
*return_fd){
	pid_t p;
	int pfds_rq[2], pfds_ret[2];
	if (pipe(pfds_rq) < 0 || pipe(pfds_ret) < 0) {
		perror("pipe");
		exit(1);
	}
	p = fork();
	if (p < 0) {
		perror("scheduler: fork");
		exit(1);
	}
	if (p == 0) {
		/* Child */
		close(pfds_rq[0]);
		close(pfds_ret[1]);
		do_shell(executable, pfds_rq[1], pfds_ret[0]);
		assert(0);
	}
	/* Parent */
	head->pid = p;    //shell's pid is saved in the structure
	close(pfds_rq[1]);
	close(pfds_ret[0]);
	*request_fd = pfds_rq[0];
	*return_fd = pfds_ret[1];
}

static void shell_request_loop(int request_fd, int return_fd){
	int ret;
	struct request_struct rq;
	/*
	 * Keep receiving requests from the shell.
	 */
	for (;;) {
		if (read(request_fd, &rq, sizeof(rq)) != sizeof(rq)) {
			perror("scheduler: read from shell");
			fprintf(stderr, "Scheduler: giving up on shell request
processing.\n");
			break;
		}
		signals_disable();
		ret = process_request(&rq);
		signals_enable();
		if (write(return_fd, &ret, sizeof(ret)) != sizeof(ret)) {
```

```
                        perror("scheduler: write to shell");
                        fprintf(stderr, "Scheduler: giving up on shell request
processing.\n");
                        break;
                }
        }
}

int main(int argc, char *argv[]){
        int nproc,i;
        /* Two file descriptors for communication with the shell */
        static int request_fd, return_fd;
        head = (struct node*) malloc(sizeof(struct node));
        /* Create the shell. */
        sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);
        //create a node for the shell and place it as the head of the queue
        strcpy(head->exec ,SHELL_EXECUTABLE_NAME);
        head->numproc=0;
        head->next=head;
        //shell always has high priority
        head->priority='h';
        tail=head;
        nproc = argc-1; /* number of proccesses goes here */
        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */
        for(i=1;i<=nproc;i++){
                struct node *curr = (struct node*) malloc(sizeof(struct node));
                curr->pid = fork();
                if(curr->pid<0){
                        perror("fork");
                        exit(1);
                }
                if(curr->pid==0){
                        char *executable = argv[i];
                        char *newargv[] = { executable, NULL, NULL, NULL };
                        char *newenviron[] = { NULL };
                        printf("I am %s, PID = %ld\n", argv[0], (long)getpid());
                        printf("About to replace myself with the executable
%s...\n", executable);
                        sleep(2);
                        raise(SIGSTOP);
                        execve(executable, newargv, newenviron);
                        /* execve() only returns on error */
                        perror("execve");
                        exit(1);
                }
                curr->numproc = i;
                strcpy(curr->exec ,argv[i]);
                //every new process has low priority
                curr->priority='l';
                //if queue is empty then curr node is added as both head and tail
                if (head == NULL){
                        head = curr;
                        head->next = head;
                        tail = head;
                }
                //else, curr node is added to the end of the queue
                else{
                        tail->next = curr;
                        curr->next = head;
                        tail=curr;
                }
        }
        /* Wait for all children to raise SIGSTOP before exec()ing. */
        wait_for_ready_children(nproc+1);
        /* Install SIGALRM and SIGCHLD handlers. */
        install_signal_handlers();
        if (nproc == 0) {
                fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
                exit(1);
        }
        //it sends signal to the first procedure to start
        it = head;
        if(kill(head->pid, SIGCONT) < 0){
                perror("First child Cont error");
                exit(1);
        }
        alarm(SCHED_TQ_SEC);
        shell_request_loop(request_fd, return_fd);
        /* Now that the shell is gone, just loop forever
         * until we exit from inside a signal handler.
         */
        while (pause())
                ;
        /* Unreachable */
        fprintf(stderr, "Internal error: Reached unreachable point\n");
        return 1;
}
```