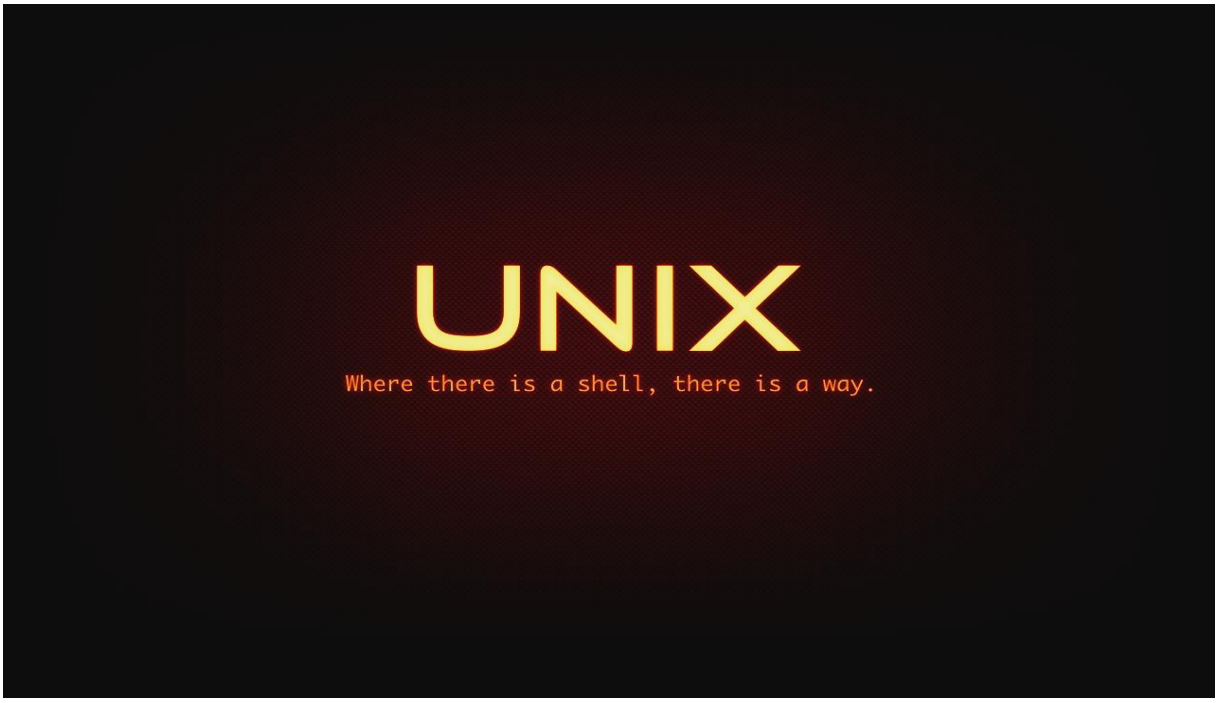
12 Ιουνίου 2020



ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ : 2019-2020

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Άσκηση 4η - Χρονοδρομολόγηση

| Καπαρού Αλεξάνδρα (03117100)
Χλαπάνης Οδυσσέας (03117023)
Ομάδα : oslaba21

Άσκηση 4.1

Υλοποιήσαμε έναν χρονοδρομολογητή κυκλικής επαναφοράς(round-robin) ο οποίος χρησιμοποιεί τα σήματα SIGSTOP και SIGCONT για τον έλεγχο των διεργασιών. Αφού δημιουργηθούν οι διεργασίες με χρήση της fork, τότε ο χρονοδρομολογητής σταματά μια διεργασία μετά την έκπνευση του αντίστοιχου κβάντου χρόνου. Μόλις γίνει αυτό, ο χρονοδρομολογητής στέλνει σήμα να συνεχίσει η επόμενη διεργασία και η όλη διαδικασία συνεχίζεται κυκλικά έως ότου τερματιστούν όλες οι διεργασίες

➢ Έξοδος εκτέλεσης προγραμμάτων της 4.1

(Για πιο καθαρά αποτελέσματα, στις παρακάτω εικόνες ορίσαμε στο αρχείο prog NMSG = 10 αντί για 200)

→ Με δύο διεργασίες:

```
oslaba21@os-node1:~/4/4.alex/4.1$ ./scheduler prog prog
We will fork 2 processes:
Now we fork process number 0
Now we fork process number 1
My PID = 18820: Child PID = 18821 has been stopped by a signal, signo = 19
My PID = 18820: Child PID = 18822 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 10, delay = 101
prog[18821]: This is message 0
prog[18821]: This is message 1
prog[18821]: This is message 2
prog[18821]: This is message 3
prog[18821]: This is message 4
prog[18821]: This is message 5
prog[18821]: This is message 6
--------!!!The time quantum has ended!!!--------
My PID = 18820: Child PID = 18821 has been stopped by a signal, signo = 19
Now process number 0 with id 18821 has been stopped.We now either go to the next process or begin from the start.
------>Moving on to process number 1<------
prog: Starting, NMSG = 10, delay = 147
prog[18822]: This is message 0
prog[18822]: This is message 1
prog[18822]: This is message 2
prog[18822]: This is message 3
prog[18822]: This is message 4
--------!!!The time quantum has ended!!!--------
My PID = 18820: Child PID = 18822 has been stopped by a signal, signo = 19
Now process number 1 with id 18822 has been stopped.We now either go to the next process or begin from the start.
------>Moving on to process number 0<------
prog[18821]: This is message 7
prog[18821]: This is message 8
prog[18821]: This is message 9
My PID = 18820: Child PID = 18821 terminated normally, exit status = 0
Now process number 0 with id 18821 is dead.
------>Moving on to process number 1<------
prog[18822]: This is message 5
prog[18822]: This is message 6
prog[18822]: This is message 7
prog[18822]: This is message 8
prog[18822]: This is message 9
--------!!!The time quantum has ended!!!--------
My PID = 18820: Child PID = 18822 has been stopped by a signal, signo = 19
Now process number 1 with id 18822 has been stopped.We now either go to the next process or begin from the start.
------>Moving on to process number 1<------
My PID = 18820: Child PID = 18822 terminated normally, exit status = 0
Now process number 1 with id 18822 is dead.
Now all processes are dead.
```

→ Με τρεις διεργασίες:

```
oslaba21@os-node1:~/4/4.alex/4.1$ ./scheduler prog prog prog
We will fork 3 processes:
Now we fork process number 0
Now we fork process number 1
Now we fork process number 2
My PID = 18833: Child PID = 18834 has been stopped by a signal, signo = 19
My PID = 18833: Child PID = 18835 has been stopped by a signal, signo = 19
My PID = 18833: Child PID = 18836 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 10, delay = 56
prog[18834]: This is message 0
prog[18834]: This is message 1
prog[18834]: This is message 2
prog[18834]: This is message 3
prog[18834]: This is message 4
prog[18834]: This is message 5
prog[18834]: This is message 6
prog[18834]: This is message 7
prog[18834]: This is message 8
prog[18834]: This is message 9
My PID = 18833: Child PID = 18834 terminated normally, exit status = 0
Now process number 0 with id 18834 is dead.
------>Moving on to process number 1<------
prog: Starting, NMSG = 10, delay = 102
prog[18835]: This is message 0
prog[18835]: This is message 1
prog[18835]: This is message 2
prog[18835]: This is message 3
prog[18835]: This is message 4
prog[18835]: This is message 5
prog[18835]: This is message 6
--------!!!The time quantum has ended!!!--------
My PID = 18833: Child PID = 18835 has been stopped by a signal, signo = 19
Now process number 1 with id 18835 has been stopped.We now either go to the next process or begin from the start.
------>Moving on to process number 2<------
prog: Starting, NMSG = 10, delay = 85
prog[18836]: This is message 0
prog[18836]: This is message 1
prog[18836]: This is message 2
prog[18836]: This is message 3
prog[18836]: This is message 4
prog[18836]: This is message 5
prog[18836]: This is message 6
prog[18836]: This is message 7
--------!!!The time quantum has ended!!!--------
My PID = 18833: Child PID = 18836 has been stopped by a signal, signo = 19
Now process number 2 with id 18836 has been stopped.We now either go to the next process or begin from the start.
------>Moving on to process number 1<------
prog[18835]: This is message 7
prog[18835]: This is message 8
prog[18835]: This is message 9
My PID = 18833: Child PID = 18835 terminated normally, exit status = 0
```

```
------>Moving on to process number 2<------
prog[18836]: This is message 8
prog[18836]: This is message 9
My PID = 18833: Child PID = 18836 terminated normally, exit status = 0
Now process number 2 with id 18836 is dead.
Now all processes are dead.
oslaba21@os-node1:~/4/4.alex/4.1$
```

➢ Ερωτήσεις της 4.1

1. Παρατηρώντας την συνάρτηση install_signal_handlers() και συγκεκριμένα το παρακάτω κομμάτι:

```
    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;
```

βλέπουμε ότι αρχικοποιεί με την sigemptyset ένα σύνολο σημάτων, με την sigaddset προσθέτει τα σήματα SIGCHLD και SIGALRM και

τέλος με την sa_mask φτιάχνει μία μάσκα για τα δύο αυτά σήματα, όπως αναφέρει και το manual page της sigaction:

sa_mask specifies a mask of signals which should be blocked (i.e., added to the signal mask of the thread in which the signal handler is invoked) during execution of the signal handler. In addition, the signal which triggered the handler will be blocked, unless the **SA_NODEFER** flag is used.

με την οποία τα μπλοκάρει όταν εκτελείται ο signal handler. Με αυτόν τον τρόπο, όταν εκτελείται παραδείγματος χάρη η συνάρτηση χειρισμού του σήματος SIGCHLD, τότε μπλοκάρεται μέσω αυτής της μάσκας το σήμα SIGALRM και δεν επηρεάζει την εκτέλεση του άλλου σήματος. Αυτό στη βιβλιογραφία ονομάζεται Preemptive Scheduling. Σε έναν πραγματικό χρονοδρομολογητή χώρου πυρήνα, προτού μια διεργασία επιστρέψει σε user mode, ο πυρήνας κοιτάζει εάν υπάρχουν pending σήματα(δηλαδή σήματα τα οποία περιμένουν την ευκαιρία να ληφθούν από την διεργασία, χωρίς όμως να έχει κρατηθεί ο αριθμός αυτών των σημάτων). Όπως και στη δική μας υλοποίηση, όσο εκτελεί «σημαντικές λειτουργίες» (δηλαδή λειτουργίες που είναι πολύ σημαντικές για να τις διακόψει) μπλοκάρονται κάθε είδους διακοπές, όπως φαίνεται και στο παρακάτω απόσπασμα του βιβλίου του Silberschatz:

Because interrupts can, by definition, occur at any time, and because they cannot always be ignored by the kernel, the sections of code affected by interrupts must be guarded from simultaneous use. The operating system needs to accept interrupts at almost all times; otherwise, input might be lost or output overwritten. So that these sections of code are not accessed concurrently by several processes, they disable interrupts at entry and reenable interrupts at exit. It is important to note that sections of code that disable interrupts do not occur very often and typically contain few instructions.

Εάν όταν επιστρέψει από κάποια «σημαντική λειτουργία» υπάρχουν σήματα προς λήψη τότε ο επεξεργαστής που τρέχει σε kernel mode θα πρέπει προσωρινά να κάνει switch σε user mode, για να εκτελέσει τον signal handler, και τέλος να επιστρέψει σε kernel mode για να ολοκληρωθεί η εκτέλεση του signal handler.

2. Ένα σήμα SIGCHLD στέλνεται στην γονική διεργασία όταν ένα παιδί του τερματιστεί. Αναμένουμε ότι όταν στέλνεται ένα τέτοιο σήμα και το λαμβάνει ο χρονοδρομολογητής, αυτό αναφέρεται στην τρέχουσα διεργασία και έχει σαν αποτέλεσμα τον τερματισμό της. Εάν τερματιστεί αναπάντεχα μια οποιαδήποτε διεργασία-παιδί, τότε ο χρονοδρομολογητής θα λειτουργήσει εσφαλμένα και θα αφαιρέσει την τρέχουσα διεργασία(θέτοντας array[current]=0),

θεωρώντας παράλληλα ότι η διεργασία που έχει τερματιστεί (και για την οποία στάλθηκε το SIGCHLD) είναι ακόμα ζωντανή. Σε μία πιο εύρωστη υλοποίηση θα ελέγχαμε ποια διεργασία έστειλε το σήμα (μέσω του id ή του pid) και θα πράτταμε αναλόγως, αλλά κάτι τέτοιο ξεφεύγει από το σκεπτικό της συγκεκριμένης υλοποίησης.

3. Στην υλοποίηση του χρονοδρομολογητή μας χρησιμοποιήσαμε δύο σήματα, το SIGALRM που σταματά την τρέχουσα διεργασία όταν εκπνεύσει το κβάντο χρόνου, και το SIGCHLD που έχει ως ρόλο, όταν σταματήσει η τρέχουσα διεργασία, να βρει την επόμενη και να την ενεργοποιήσει. Σε περίπτωση που χρησιμοποιούσαμε μόνο το σήμα SIGALRM τότε η υλοποίηση θα ήταν επισφαλής καθώς τότε δεν θα είχαμε γνώση της κατάστασης του παιδιού, αφού θα τερματίζαμε την διεργασία μόνο αφού εξέπνεε το κβάντο χρόνου. Με αυτόν τον τρόπο, εάν τερματιζόταν η διεργασία με ένα σήμα SIGSTOP τότε το σύστημα θα έμενε στον αέρα μέχρι να εκπνεύσει το κβάντο χρόνου για να την τερματίσει. Ακόμα, εάν στελνόταν ένα σήμα SIGSTOP αφού εξέπνεε το κβάντο χρόνου, τότε δεν θα το λάμβανε ποτέ η διεργασία και όταν θα ερχόταν η σειρά της θα εκτελούνταν κανονικά.

Άσκηση 4.2

Σε αυτό το ερώτημα κληθήκαμε να επεκτείνουμε την προηγούμενη άσκηση, ώστε να υποστηρίζεται ο έλεγχος μέσω προγράμματος-φλοιού. Τροποποιώντας τον προηγούμενο κώδικα, οι διεργασίες αποθηκεύονται σε μορφή κυκλικής λίστας για μεγαλύτερη ευκολία στην υλοποίηση του προγράμματος. Αντίστοιχα με πριν, μόλις εκπνεύσει το κβάντο χρόνου, ο χρονοδρομολογητής το αντιλαμβάνεται μέσω της SIGALRM, στέλνει SIGSTOP και μετά στέλνει SIGCONT στην επόμενη διεργασία.

➢ Έξοδος εκτέλεσης προγραμμάτων της 1.2

(Για πιο καθαρά αποτελέσματα απενεργοποιήσαμε τα μηνύματα εκτύπωσης στα αρχεία prog, prog2, prog3)

```
oslaba21@os-node1:~/4/4.alex/4.2$ ./scheduler-shell prog prog2 prog3
My PID = 18971: Child PID = 18972 has been stopped by a signal, signo = 19
My PID = 18971: Child PID = 18973 has been stopped by a signal, signo = 19
My PID = 18971: Child PID = 18974 has been stopped by a signal, signo = 19
My PID = 18971: Child PID = 18975 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 18972 and name = shell is the current process
Process with id = 1, PID = 18973 and name = prog is about to run
Process with id = 2, PID = 18974 and name = prog2 is about to run
Process with id = 3, PID = 18975 and name = prog3 is about to run
Shell> k 2
Shell: issuing request...
Shell: receiving request return value...
We now have killed process with id = 2 and pid = 18974
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 3, PID = 18975 and name = prog3 is the current process
Process with id = 0, PID = 18972 and name = shell is about to run
Process with id = 1, PID = 18973 and name = prog is about to run
Shell> e prog2
Shell: issuing request...
Shell: receiving request return value...
We are going to create process with name = prog2 and id = 4
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 3, PID = 18975 and name = prog3 is the current process
Process with id = 0, PID = 18972 and name = shell is about to run
Process with id = 4, PID = 18983 and name = prog2 is about to run
Process with id = 1, PID = 18973 and name = prog is about to run
Shell> q
Shell: Exiting. Goodbye.
```

Μειώνοντας τον αριθμό των μηνυμάτων σε 10 (NMSG=10 όπως και πριν) βλέπουμε ότι:

```
oslaba21@os-node1:~/4/4.alex/4.2$ ./scheduler-shell prog prog prog
My PID = 19028: Child PID = 19029 has been stopped by a signal, signo = 19
My PID = 19028: Child PID = 19030 has been stopped by a signal, signo = 19
My PID = 19028: Child PID = 19031 has been stopped by a signal, signo = 19
My PID = 19028: Child PID = 19032 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 19029 and name = shell is the current process
Process with id = 1, PID = 19030 and name = prog is about to run
Process with id = 2, PID = 19031 and name = prog is about to run
Process with id = 3, PID = 19032 and name = prog is about to run
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 19029 and name = shell is the current process
Process with id = 1, PID = 19030 and name = prog is about to run
Process with id = 2, PID = 19031 and name = prog is about to run
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 19029 and name = shell is the current process
Process with id = 2, PID = 19031 and name = prog is about to run
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 19029 and name = shell is the current process
Shell> q
Shell: Exiting. Goodbye.
Now all 4 processes are dead
```

,δηλαδή όχι μόνο εξαφανίζονται οι διεργασίες χωρίς να πατήσουμε κάτι αλλά και εμφανίζεται το μήνυμα “Now all 4 processes are dead”. Αυτό συμβαίνει διότι στην δεύτερη περίπτωση, λόγω του μικρού μεγέθους εκτύπωσης, οι διεργασίες τερματίζουν πριν εκπνέυσει το κβάντο χρόνου. Έτσι, χωρίς να πατήσουμε τίποτα οι διεργασίες αφαιρούνται από τον κατάλογο των διεργασιών και εμφανίζεται και το τελικό μήνυμα ενώ όταν πατήσαμε το q στην πρώτη εικόνα δεν είχαν προλάβει να τερματιστούν όλες οι διεργασίες.

➢ Ερωτήσεις της 4.2

1. Κατά την εκτέλεση του προγράμματος βλέπουμε ότι πάντα εμφανίζεται σαν τρέχουσα διεργασία ο φλοιός στην λίστα διεργασιών. Κάτι τέτοιο είναι και το αναμενόμενο καθώς αυτός είναι που εκτελεί την εντολή προκειμένου να τυπώσει τα αποτελέσματα οπότε αν δεν ήταν τρέχουσα διεργασία τότε δεν θα μπορούσαμε να εμφανίσουμε και την λίστα.

```
This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 24798 and name = shell is the current process
Process with id = 1, PID = 24799 and name = prog is about to run
Process with id = 2, PID = 24800 and name = prog2 is about to run
Process with id = 3, PID = 24801 and name = prog3 is about to run
```

   Για να μην συμβαίνει αυτό, θα έπρεπε μόλις γραφτεί η εκάστοτε εντολή στο pipe (μέσω της issue_request του shell)
   και πριν να απενεργοποιηθούν οι διακοπές από την signals_disable(), να εκπνεύσει το κβάντο χρόνου και να προλάβει να αλλάξει την τρέχουσα διεργασία. Έτσι ο sceduler θα έχει πάρει την εντολή και θα έχει πάει στην sched_print_tasks για να τυπώσει άλλη τρέχουσα διεργασία. Επίσης, θα μπορούσαμε να επιλέξουμε ως τρέχουσα διεργασία να εμφανίζεται η διεργασία που είχε το προηγούμενο κβάντο χρόνου από το shell αλλά μία τέτοια σχεδιαστική επιλογή θα αύξανε την πολυπλοκότητα και δεν είναι το ζητούμενο της άσκησης αυτό.

2. Όπως γνωρίζουμε, η συνάρτηση υλοποίησης αιτήσεων του φλοιού επηρεάζει δομές όπως η ουρά εκτέλεσης των διεργασιών. Γι’αυτόν τον λόγο, είναι απαραίτητο να απενεργοποιήσουμε όλα τα σήματα κατά την διάρκεια εκτέλεσης εντολών του φλοιού καθώς σε αντίθετη περίπτωση, αν παραδείγματος χάρη διαγράφαμε μια διεργασία την οποία εκείνη την ώρα ήλεγχε ο φλοιός, τότε θα

έσκαγε το πρόγραμμα μας. Μία άλλη περίπτωση ενδεικτική της λάθους υλοποίησης μη χρήσης signals_disable(), signals_enable() είναι όταν θα εισάγαμε μια νέα διεργασία στον φλοιό (με χρήση της εντολής e), εκείνη την ώρα να ερχόταν ένα σήμα SIGALRM με αποτέλεσμα να άλλαζε το current και να είχαμε λάθος αποτελέσματα στην λίστα διεργασιών.

Άσκηση 4.3

Εδώ κληθήκαμε να επεκτείνουμε το προηγούμενο πρόγραμμα ώστε να υποστηρίζονται οι κλάσεις HIGH και LOW. Οι κλάσεις αυτές χρησιμοποιούνται ώστε να υπάρχει μια ιεραρχία σημαντικών/λιγότερο σημαντικών διεργασιών, δηλαδή όταν υπάρχουν high διεργασίες να εκτελούνται μόνο αυτές. By default θα πρέπει οι διεργασίες να υλοποιούνται με low προτεραιότητα και να αλλάζουν κατά βούληση.

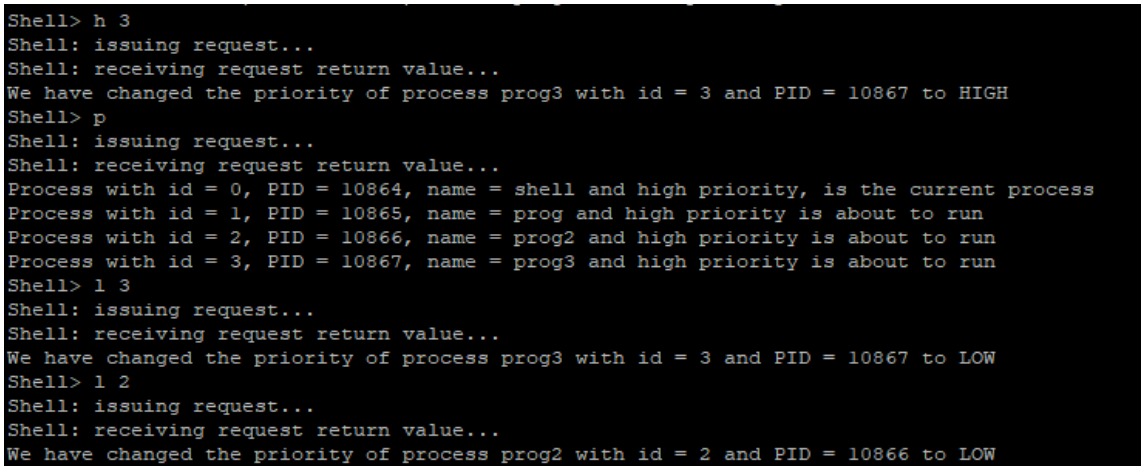
➢ Έξοδος εκτέλεσης προγραμμάτων της 4.3

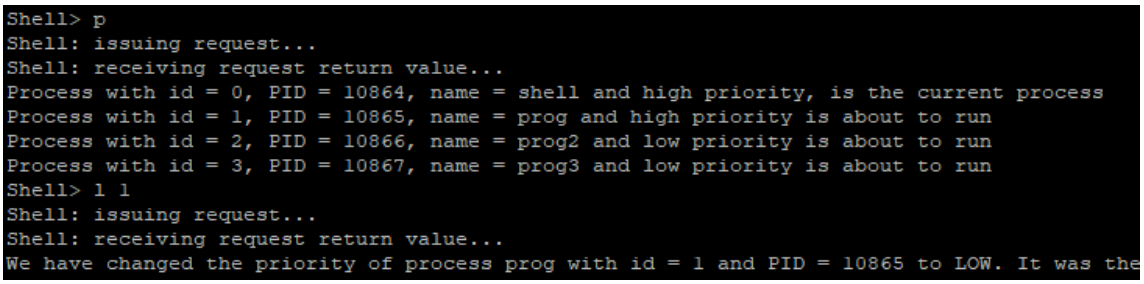
(Για πιο καθαρά αποτελέσματα, τώρα απενεργοποιήσαμε και την εμφάνιση των σχολίων όσον αφορά το ποια διεργασία τρέχει, το οποίο μας επιβεβαίωνε την χρονοδρομολόγηση round-robin.)

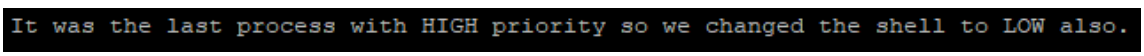
```
oslaba21@os-node1:~/4/4.ody$ ./scheduler-shell prog prog2 prog3
My PID = 10863: Child PID = 10864 has been stopped by a signal, signo = 19
My PID = 10863: Child PID = 10865 has been stopped by a signal, signo = 19
My PID = 10863: Child PID = 10866 has been stopped by a signal, signo = 19
My PID = 10863: Child PID = 10867 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 10864, name = shell and low priority, is the current process
Process with id = 1, PID = 10865, name = prog and low priority is about to run
Process with id = 2, PID = 10866, name = prog2 and low priority is about to run
Process with id = 3, PID = 10867, name = prog3 and low priority is about to run
Shell> h 1
Shell: issuing request...
Shell: receiving request return value...
We have changed the priority of process prog with id = 1 and PID = 10865 to HIGH. It is t
```

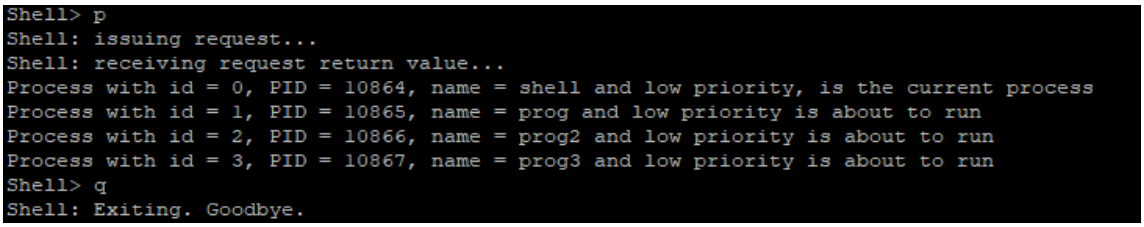
```
It is the first process with HIGH priority so we changed the shell to HIGH also.
```

```
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 10864, name = shell and high priority, is the current process
Process with id = 1, PID = 10865, name = prog and high priority is about to run
Process with id = 2, PID = 10866, name = prog2 and low priority is about to run
Process with id = 3, PID = 10867, name = prog3 and low priority is about to run
Shell> h 2
Shell: issuing request...
Shell: receiving request return value...
We have changed the priority of process prog2 with id = 2 and PID = 10866 to HIGH
Shell> p
Shell: issuing request...
Shell: receiving request return value...
Process with id = 0, PID = 10864, name = shell and high priority, is the current process
Process with id = 1, PID = 10865, name = prog and high priority is about to run
Process with id = 2, PID = 10866, name = prog2 and high priority is about to run
Process with id = 3, PID = 10867, name = prog3 and low priority is about to run
```









Όπως είπαμε και στην αρχή, βλέπουμε αρχικά να υλοποιείται ο φλοιός ως low priority process και να εκτελούνται όλες οι διεργασίες κυκλικά. Στην συνέχεια αλλάζουμε την προτεραιότητα στις διεργασίες και κάθε φορά δείχνουμε πώς εμφανίζονται στον κατάλογο των διεργασιών. Τέλος, αν ενεργοποιήσουμε την εμφάνιση των αντίστοιχων σχολίων, θα παρατηρήσουμε πως πράγματι υπάρχει κυκλική χρονοδρομολόγηση όταν τουλάχιστον 2 διεργασίες είναι high (ή όλες οι διεργασίες είναι low) καθώς εκτελούνται με την σειρά 0-1-2-3-0-1-2-3…

➢ Ερωτήσεις της 4.3

1. Πρόβλημα λιμοκτονίας έχουμε όταν διεργασίες με χαμηλή προτεραιότητα μπορεί να μην εκτελεστούν ποτέ. Κάτι τέτοιο μπορεί να γίνει όταν:

→ Έχουμε τον φλοιό high και όλες τις άλλες διεργασίες low.

→ Έχουμε τον φλοιό low και τουλάχιστον μια διεργασία high που δεν τερματίζει αν δεν την τερματίσουμε εμείς.

→ Έχουμε και τον φλοιό high και τουλάχιστον μία διεργασία high.
Σε αυτές τις περιπτώσεις, επειδή εξαρτάται μόνο από τον χρήστη εάν θα τεθούν ως low οι high ή ως high οι low, υπάρχει κίνδυνος λιμοκτονίας εάν ο χρήστης δεν παρέμβει.

Προαιρετικές ερωτήσεις

1. Ο μηχανισμός της επικοινωνίας μεταξύ του φλοιού και του χρονοδρομολογητή στην άσκηση 1.2 υλοποιείται με τη χρήση pipes. Για την ακρίβεια, κατά τη δημιουργία του shell, δημιουργούμε και δύο pipes το ένα (request_fd) για νέες εντολές του χρήστη τις οποίες θα διαβάσει ο φλοιός και το άλλο (return_fd) για την αποστολή των επεξεργασμένων μέσω της process_request() από τον φλοιό προς τον χρονοδρομολογητή. Αυτά τα pipes δημιουργούνται στο κομμάτι κώδικα που ακολουθεί.

```
/* 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)
...


int pfds_rq[2], pfds_ret[2];
   if (pipe(pfds_rq) < 0 || pipe(pfds_ret) < 0) {
      perror("pipe");
      exit(1);
   }
...
   close(pfds_rq[1]);
   close(pfds_ret[0]);
   *request_fd = pfds_rq[0];
   *return_fd = pfds_ret[1];
```

...

Τα οποία έπειτα χρησιμοποιούνται κάθε φορά που εκτελείται το παρακάτω:

```
static void
shell_request_loop(int request_fd, int return_fd)
{
...
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;
    }
...
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;
    }
```

2. Ας υποθέσουμε ότι έχουμε δύο διεργασίες HIGH προτεραιότητας, όπου η μία εξ αυτών θα είναι ο φλοιός, και κάποιες LOW. Τότε οι LOW προτεραιότητας δεν θα εκτελεστούν ποτέ. Για να αποφύγουμε αυτή τη λιμοκτονία θα μπορούσαμε να υλοποιήσουμε έναν αλγόριθμο γήρανσης. Για παράδειγμα, θα μπορούσαμε να προσθέσουμε στην υλοποίηση του struct proc και μία μεταβλητή int aging η οποία θα αυξάνεται αυτόματα σε κάθε κβάντο χρόνου και θα μηδενίζεται κάθε φορά που δίνουμε κβάντο χρόνου σε κάποια διεργασία. Μετά αν ορίσουμε κάποιο κατώφλι για τον παράγοντα aging, τότε η διεργασία που ξεπερνά το κατώφλι θα μπαίνει προσωρινά σε σειρά προτεραιότητας HIGH (θα μπορούσαμε να ορίσουμε μία νέα κατηγορία TEMP_HIGH ή με οποιονδήποτε άλλο αποδεκτό τρόπο υλοποίησης) και έπειτα να επανέρχεται στην κανονική της προτεραιότητα και με μηδενικό παράγοντα aging.

3. Η κατάσταση που περιγράφεται στην ερώτηση 4.3 είναι προβληματική με την παρούσα υλοποίηση επειδή η διαδικασία M

θα εκτελείται αενάως χωρίς να μπορούν ποτέ ούτε η H ούτε η L να ξεκινήσουν. Επειδή ακόμα και αν ελευθερωθεί ο σημαφόρος της L, επειδή η L έκανε wait πριν την H, η H δεν έχει ποτέ την ευκαιρία να ξεκινήσει παρόλο που σαφώς έχει προτεραιότητα. Η κατάσταση αυτή δεν είναι καθόλου ασυνήθιστη, αφού θα μπορούσε αυτός ο σημαφόρος να παριστάνει αναμονή για Είσοδο/Έξοδο (I/O). Μία απλή λύση θα ήταν να υλοποιήσουμε κάποιο μηχανισμό γήρανσης όπως περιγράψαμε παραπάνω. Εναλλακτικά, αν θέλουμε να αποφύγουμε συγκεκριμένα τέτοιες καταστάσεις, μπορούμε αντί να αναστέλλουμε τη λειτουργία συναρτήσεων που περιμένουν σε σημαφόρους, να τις τοποθετούμε σε κάποια νέα κλάση S η οποία περιοδικά να επιβάλλεται ως HIGH (σύμφωνα με κάποια περίοδο T, ας πούμε 20 κβάντα χρόνου), να τσεκάρουμε αν υπάρχει πρόοδος, και στη συνέχεια να περιμένει πάλι. Μόλις ολοκληρωθεί η αναμονή σε κάποιο σημαφόρο, η κάθε διεργασία επανατοποθετείται στην κλάση της.