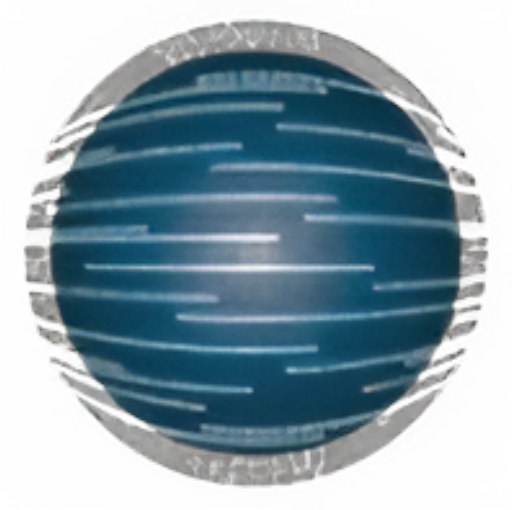
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΉΜΑ ΜΗΧΑΝΙΚΏΝ ΗΛΕΚΤΡΟΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΉΣ



Λειτουργικά Συστήματα

---

Χειμερινό Εξάμηνο

# 2η Εργαστηριακή Άσκηση

---

Καθηγητές: Σ. Σιούτας, Χ. Μακρής, Π. Χατζηδούκας, Α. Ηλίας

| **ΑΜ** | **Επώνυμο** | **Όνομα** | **e-mail** | **Έτος** |
|---|---|---|---|---|
| 1084567 | Βιλλιώτης | Αχιλλέας | up1084567@upnet.gr | $3^o$ |
| 1088098 | Μπαρδάκης | Βασίλειος | up1088098@upnet.gr | $3^o$ |
| 1084589 | Χάλλας | Χαράλαμπος-Μάριος | up1084589@upnet.gr | $3^o$ |

ΠΆΤΡΑ, ΙΑΝΟΥΆΡΙΟΣ, 2023

# Περιεχόμενα

# Μέρος Α: Εισαγωγή

Για την εκπόνηση της δεύτερης εργαστηριακής άσκησης, ακολουθήσαμε το πρότυπο του κ. Χατζηδούκα για την σχεδίαση και οργάνωση του πηγαίου κώδικα. Το παραδοτέο αποτελείται από το αρχείο scheduler.c το οποίο εκτελεί όλες τις προβλεπόμενες λειτουργίες όπως επίσης τροποποιημένα work.c και run.sh αρχεία τα οποία: το πρώτο περιέχει επιπλέον printf και το δεύτερο περιέχει επιπλέον περιπτώσεις.

Το αρχείο scheduler.c περιέχει συμπληρωματικό σχολιασμό σε αυτό το έγγραφο, χωρίς όμως υπερανάλυση βασικών προγραμματιστικών λειτουργιών και συμπεριφορών.

Τέλος, δίνεται και ένα modified.c αρχείο το οποίο περιέχει τις τροποποιήσεις που κάναμε στο τέταρτο μέρος.

Δόθηκε βάση στην δυναμικότητα του χρονοπρογραμματιστή, και έπειτα από πολλούς ελέγχους πιστεύουμε πως είναι πλήρως δυναμικό, κατέχοντας την ικανότητα να αλλάξει σε κάθε χρονική στιγμή τον αλγόριθμο χωρίς να επηρεαστεί η λειτουργικότητα.

Επίσης η δομή του κώδικα επιτρέπει την τυχών προσθήκη και τροποποίηση αλγορίθμων να γίνεται με εύκολο τρόπο. Όσον αφορά τυχών μη-αναμενόμενες συμπεριφορές, μπορέσαμε να εντοπίσουμε παρά μία, η οποία όμως παρατηρείται σπάνια και θα αναφερθούμε σε αυτή αργότερα.

# Μέρος Β: Χρήση του προγράμματος

## Β.1 End User

Ακολουθούν παραδείγματα χρήσης του προγράμματος[1].

```
axillv@axillv-VirtualBox:~/Documents/os_project2/scheduler$ ./scheduler
Wrong amount of arguments given!
Correct usage is: ./scheduler algorithm [quant] sourceFile
Policy and quant combination does not exist! Availabe policies are:
---------------
Static Policies
---------------
fcfs
sjf
----------------
Dynamic Policies
----------------
rr
prio
```

1. Κλήση χωρίς παράμετρους.

```
axillv@axillv-VirtualBox:~/Documents/os_project2/scheduler$ ./scheduler fcfs
Wrong amount of arguments given!
Correct usage is: ./scheduler algorithm [quant] sourceFile
Policy and quant combination does not exist! Availabe policies are:
---------------
Static Policies
---------------
fcfs
sjf
----------------
Dynamic Policies
----------------
rr
prio
```

2. Κλήση με λάθος αριθμό παραμέτρων.

```
axillv@axillv-VirtualBox:~/Documents/os_project2/scheduler$ ./scheduler fcfs reverse.txt
PID 20700 - CMD work7
                        Elapsed Time: 0.248 secs
                        Workload Time: 0.249 secs
PID 20701 - CMD work6
                        Elapsed Time: 0.215 secs
                        Workload Time: 0.464 secs
PID 20702 - CMD work5
                        Elapsed Time: 0.249 secs
                        Workload Time: 0.717 secs
PID 20704 - CMD work4
                        Elapsed Time: 0.142 secs
                        Workload Time: 0.859 secs
PID 20705 - CMD work3
                        Elapsed Time: 0.117 secs
                        Workload Time: 0.976 secs
PID 20706 - CMD work2
                        Elapsed Time: 0.062 secs
                        Workload Time: 1.038 secs
PID 20707 - CMD work1
                        Elapsed Time: 0.079 secs
                        Workload Time: 1.118 secs
```

3. Ορθή κλήση του fcfs.

```
axillv@axillv-VirtualBox:~/Documents/os_project2/scheduler$ ./scheduler prio reverse.txt
Policy and quant combination does not exist! Availabe policies are:
---------------
Static Policies
---------------
fcfs
sjf
----------------
Dynamic Policies
----------------
rr
prio
```

4. Λάθος κλήση δυναμικού αλγόριθμου.

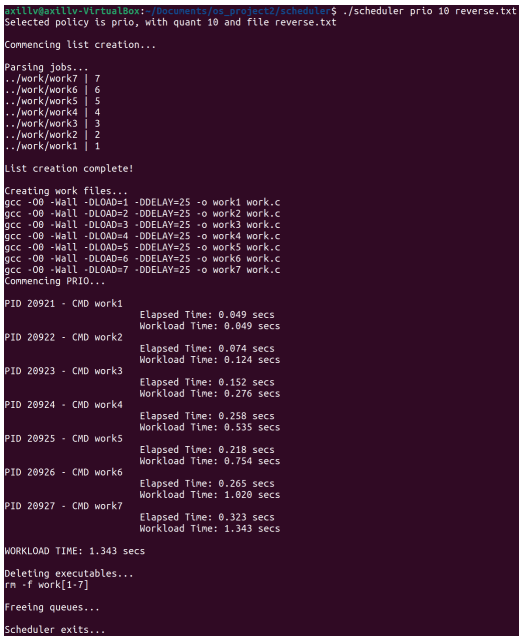
```
axillv@axillv-VirtualBox:~/Documents/os_project2/scheduler$ ./scheduler prio 10 reverse.txt
PID 20775 - CMD work1
                        Elapsed Time: 0.038 secs
                        Workload Time: 0.038 secs
PID 20776 - CMD work2
                        Elapsed Time: 0.074 secs
                        Workload Time: 0.113 secs
PID 20777 - CMD work3
                        Elapsed Time: 0.142 secs
                        Workload Time: 0.255 secs
PID 20778 - CMD work4
                        Elapsed Time: 0.186 secs
                        Workload Time: 0.443 secs
PID 20779 - CMD work5
                        Elapsed Time: 0.256 secs
                        Workload Time: 0.699 secs
PID 20780 - CMD work6
                        Elapsed Time: 0.220 secs
                        Workload Time: 0.920 secs
PID 20781 - CMD work7
                        Elapsed Time: 0.242 secs
                        Workload Time: 1.162 secs

WORKLOAD TIME: 1.162 secs
```

5. Ορθή κλήση του prio.

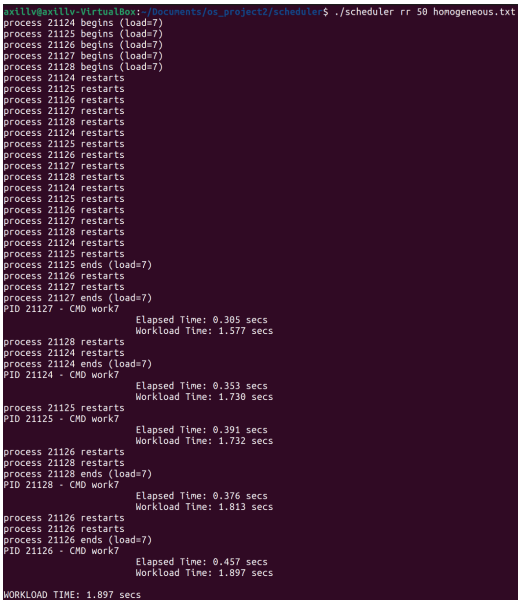
[1] Όλα τα παραδείγματα έτρεξαν σε Intel i5-4460 ενώ χρησιμοποιήθηκε DELAY=25.

## B.2 Debug Flags

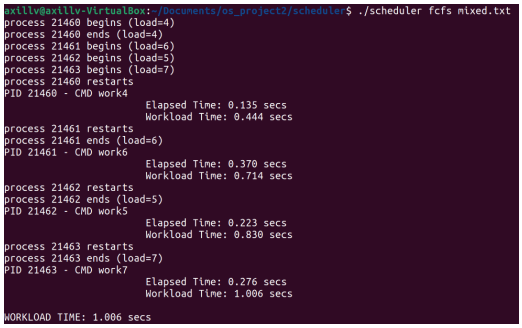
Στην αρχή του πηγαίου κώδικα, υπάρχουν 4 flags τα οποία ενεργοποιούν αντίστοιχα επιπλέον λειτουργίες ή/και cases. Ακολουθούν παραδείγματα με εικόνες[2].



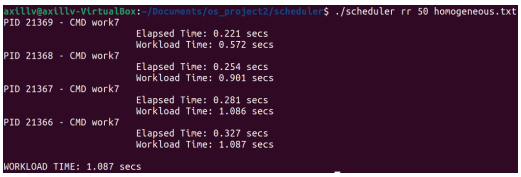

1. INFO_PRINTS - Περισσότερες πληροφορίες για την εκτέλεση.




2. START_STOP_PRINTS - Εμφανίζεται ακριβώς πότε εκτελείται ποιά διεργασία.




3. STATIC_CONTINUE - Προσομοίωση ανάγκης continue σε static αλγόριθμο.




4. SIMULATE..._LOOP - Προσομοίωση να αντιμετωπίζεται μια διεργασία που έχει ήδη τελειώσει.

[2] Όλα τα παραδείγματα έτρεξαν σε Intel i5-4460 ενώ χρησιμοποιήθηκε DELAY=25.

# Μέρος C: Εξήγηση Σχεδίασης

## C.1 main

Στην main γίνεται η ανάθεση της συμπεριφοράς του προγράμματος σε `SIGCHILD` με flag `SA_NOCLDSTOP`, το parsing των δεδομένων input, η δημιουργία των αρχείων `work`, ενώ μετά το πέρας της δρομολόγησης γίνεται διαχείριση της μνήμης.

## C.2 Data Structures

Το πρόγραμμα διαθέτει 2 structs, τα `struct process` και `struct queueNode` τα οποία αντίστοιχα κρατάνε πληροφορία για τη κάθε διεργασία (1 αντίγραφο μόνο) και για καθεμία από τις 3 ουρές.

## C.3 Global Variables

Υπάρχουν οι εξής global μεταβλητές: `struct queueNode *xHead/Tail` όπου x serial,sorted ή exited, οι οποίες είναι στην ουσία οι 3 ουρές στις οποίες θα αναφερθούμε αργότερα. Επίσης το `struct process *activeProcess` το οποίο χρησιμοποιείται στον signal handler, `double executionStartTime` για χρονικούς υπολογισμούς και `char *const xPolicies[]` για την αποθήκευση των διαθέσιμων αλγορίθμων.

## C.4 Functions

`void push`: Παίρνει στοιχεία μιας input διεργασίας και δημιουργεί queueNode το οποίο περνάει στην αντίστοιχη ουρά.

`void pushProcess`: Όπως πάνω, αλλά με όρισμα process που ήδη έχει αρχικοποιηθεί (για μεταφορά στην exitedQueue)

`void removeNode`: Διαγράφει ένα `struct queueNode` από ουρά και εκτελεί διαχείριση μνήμης.

`void freeQueue`: Διαχείριση μνήμης σε ένα ολόκληρο queue και εάν το flag `int freeProcesses` είναι 1 τότε καθαρίζει και τα process μέσω της `void freeProcess` (χρησιμοποιείται μόνο πάνω στην exited queue, υποθέτοντας πως ο δρομολογητής εκτέλεσε όλες τις διεργασίες επιτυχώς).

`void sortedInsert`: Εισαγωγή σε DLL (queue) ενός node.

`void sortSerialIntoSortedDLLAscending`: Ταξινομεί και μεταφέρει ένα αντίτυπο των queueNode(τα οποία όμως δείχνουν στην ίδια διεργασία) στην sorted queue από την serial queue, για χρήση σε δυναμικούς αλγόριθμους.

`void childTerminatedHandler`: Ο handler για το `SIGCHLD` σήμα, διαχειρίζεται το zombie child process και θέτει το status του activeProcess σε EXITED (-1). Επίσης μπλοκάρει τα υπολοίπα σήματα μέχρι να έρθει εις πέρας η διαδικασία.

Οποιαδήποτε άλλη συνάρτηση παραλήφθηκε διότι θεωρήθηκε απλή ή/και δεν αποτελεί βασική συνάρτηση του προγράμματος, παρά βοηθητική για άλλες συναρτήσεις.

## C.5 Algorithms

Ο κώδικας για όλους τους αλγόριθμους ακολουθεί τον εξής σκελετό. Στον κώδικα κάθε αλγορίθμου υπάρχει ένας δείκτης current, ο οποίος ξεκινώντας από το κατάλληλο `head`, εκτελεί προσπέλαση της κατάλληλης queue, μέχρι να είναι άδεια. Σε κάθε επανάληψη, το current ελέγχεται αν είναι `READY` ή `STOPPED`, (η δεύτερη περίπτωση δεν είναι δυνατόν να συμβεί κανονικά στον fcfs και sjf, αλλά υπάρχει για λόγους πληρότητας της δυναμικότητας του κώδικα). Επίσης εάν είναι `EXITED` καθαρίζεται το node και μεταφέρεται στο exited queue, το οποίο υπάρχει επίσης για λόγους πληρότητας και τυχόν επέκτασης του κώδικα.

Σε κάθε περίπτωση, όταν βρεθεί κατάλληλο node, εκτελείται το αντίστοιχο `run` πάνω στην διεργασία του, και έπειτα ελέγχεται έαν όντως τερμάτισε η διεργασία, ώστε να αφαιρεθεί από το queue και να προχωρήσει στο επόμενο node.

Παρουσιάζεται η δομή του κώδικα για τους 4 αλγόριθμους. Για τα είδη των run function, βλ. παρακάτω.

### C.5.i First-Come First-Served

Ο αλγόριθμος καλείται μέσω της `void fcfs()`. Έχει τα εξής χαρακτηριστικά:

- Χρήση serial queue.
- Κλήση `staticRun`.

### C.5.ii Shortest Job First

Ο αλγόριθμος καλείται μέσω της `void sjf()`. Η λογική είναι πιστό αντίγραφο της FCFS, όμως, η static queue ταξινομείται επί της sorted queue, η οποία και τελικά χρησιμοποείται αντί της serial queue.

### C.5.iii Roundrobin

Ο αλγόριθμος καλείται μέσω της `void roundrobinAutonomous(int quant)`. Το κύριο σώμα είναι παρόμοιο των προηγούμενων 2, με τα εξής σημεία:

- Χρήση serial queue.
- Κλήση `dynamicRun` συναρτήσεων αντί των static (βλ. παρακάτω).

### C.5.iv Priority

Ο αλγόριθμος καλείται μέσω της `void priority(int quant)`. Κύρια χαρακτηριστικά της συνάρτησης είναι:

- Χρήση sorted queue.
- Έλεγχος για το πλήθος διεργασιών με ίδιο priority με αυτή που πρόκειται να τρέξει. (`void returnPriorityRRRange`
- Κλήση staticRun αλλά και dynamicRun (μέσω της `void roundrobinPriority`), ανάλογα με το αποτέλεσμα του προηγούμενου σημείου.

Όσον αφορά την `void roundrobinPriority`, είναι μια συνάρτηση η οποία εκτελεί roundrobin όμως σε συγκεκριμένο αριθμό node και ξεκινώντας από αυθαίρετο node.

Γι'αυτό, επιπρόσθετα γίνεται χρήση των `index` και `range`, ώστε ο δείκτης `current` να επιστρέφει στον κατάλληλο κόμβο μετά από κάθε εκτέλεση μιας διεργασίας.

Η φύση των roundrobin και priority επιτρέπει να υπάρχουν `STOPPED` διεργασίες στην ουρά, κάτι για το οποίο έχουμε φροντίσει σε όλους τους αλγόριθμους έτσι κι αλλιώς.

## C.6 Run Functions

Κύριο σημείο αποτελούν οι συναρτήσεις οι οποίες εκτελούν η συνεχίζουν την εκτέλεση των διεργασιών-παιδιών του scheduler, στο σύνολο 4. Αυτές χωρίζονται κατά είδος δρομολόγησης: (static ή run until end και dynamic ή run with nanosleep). Κοινά μέρη αποτελούν ο ορισμός του global pointer `struct process *processToRun`, ο έλεγχος στο τέλος κάθε συνάρτησης με σκοπό την εκτύπωση του κατάλληλου μηνύματος σε περίπτωση τερματισμού της διεργασίας αλλά και η σωστή διαχείρισης των ουρών και του head.

### C.6.i Static Execute Run

Καλείται μέσω της `void staticExecRun(struct *process)`. Αμέσως δημιουργείται το παιδί το οποίο θα τρέξει την διεργασία. Ο πατέρας ξεκινά την χρονομέτρηση, ορίζεται ο global pointer ο οποίος δείχνει στην διεργασία που τρέχει (ώστε να γνωρίζει ο `SIGCHLD` handler) και κοιμάται με nanosleep σε διαστήματα των 100ns μέχρι η διεργασία να τερματίσει και ο handler να αλλάξει το status σε `EXITED`.

### C.6.ii Static Continue Run

Καλείται μέσω της `void staticContinueRun(struct *process)`.Η μόνη ουσιαστική διαφορά που υπάρχει με την πως αντί να δημιουργείται το παιδί, στέλνεται ώστε να συνεχίσει την εκτέλεση.

### C.6.iii Dynamic Execute Run

Καλείται μέσω τής `void dynamicExecRun(struct *process, int)`. Αντίστοιχα, η μόνη διαφορά με την static εκδοχή είναι πως καλείται ένα nanosleep στην scheduler διεργασία, με διαρκεία ίση με το input quant, έπειτα στέλνεται σήμα SIGSTOP και εάν ο handler δεν έτρεξε (άρα η κατάσταση δεν είναι EXITED αλλα RUNNING), θέτεται η κατάσταση σε STOPPED.

### C.6.iv Dynamic Continue Run

Καλείται μέσω της `void dynamicContinueRun(struct *process, int)`. Συνδιασμός των static continue και dynamic execute.

# Μέρος D: Αποτελέσματα και αστοχίες

## D.1 Αποτελέσματα

Πιστεύουμε πως καταφέραμε να σχεδιάσουμε έναν ολοκληρωμένο δρομολογητή, πλήρως δυναμικό και επεκτάσιμο. Γίνεται χρήση POSIX standard συναρτήσεων όπως του sigaction έναντι του signal, ενώ κάθε αλγόριθμος είναι ικανός να συνεχίσει την εκτέλεση σε κάθε (εύλογη) χρονική στιγμή, λαμβάνοντας την σκυτάλη από κάποιον άλλο.

Σημαντικό μέρος των συναρτήσεων αλγορίθμων και εκτέλεσης διεργασιών επαναχρησιμοποιεί κομμάτια κώδικα, το οποίο όμως συνεισφέρει σε πιο καθαρό κώδικα (κατά τη γνώμη μας) και στην εύκολη προσθήκη νέων λειτουργιών.

## D.2 Αστοχίες

Όπως αναφέρθηκε και στην αρχή της αναφοράς, δεν αντιμετωπίσαμε παρά μία αστοχία στην εκτέλεση του προγράμματος.

Συγκεκριμένα, όταν γίνεται χρήση δυναμικής εκτέλεσης διεργασίας, υπάρχει πιθανότητα να σημειωθεί ως EXITED η λάθος (επόμενη κατά σειρά) διεργασία και όταν τελικά ο δρομολογητής επιστρέψει στην αρχικά σωστή, δεν θα τερματίσει ποτέ.

Υπήρξε επικοινωνία με τον κύριο Χατζηδούκα όμως δεν καταφέραμε να λύσουμε το πρόβλημα (όμως μας οδήγησε στην τροποίηση του signal handler).

Έπειτα από δεκάδες ώρες εργασίας δεν καταφέραμε να βρούμε την πηγή του προβλήματος, όμως παρακάτω ακολουθούν τα κύρια σημεία-παρατηρήσεις για την εμφάνιση του προβλήματος καθώς και screenshots.

- Πρέπει να γίνεται χρήση δυναμικής εκτέλεσης είτε execute είτε continue.
- Το testing έγινε κυρίως με `DELAY=25` και `quant=200`, σε όλα τα .txt input αρχεία.
- Όσο πιο υψηλό είναι το `DELAY`, τόσο λιγότερες είναι οι πιθανότητες εμφάνισης του bug (σε `DELAY=1000` δεν αντιμετωπίστηκε ποτέ έπειτα από 100 εκτελέσεις του κώδικα.).
- Η πιθανότητα εμφάνισης του bug υπολογιστήκε (έπειτα απο batch εκτελέσεις του κώδικα) γύρω στο $6,67\%$ ή $\frac{1}{15}$.
- Η θέση στο queue δεν φαίνεται να επηρεάζει την πιθανότητα εμφάνισης.
- Εάν η work δεν περιέχει printf, το bug εξαφανίζεται εντελώς!!!

Ακολουθούν παραδείγματα, με αυξανόμενο insight παράγοντα (περισσότερα debug printf στις δύο τελευταίες φωτογραφίες). Αξιοσημείωτη παρατήρηση το μειωμένο runtime της διεργασίας που λανθασμένα σταματά η εκτέλεση της (όλα τα test έγιναν πάνω στο reverse.txt διότι η prio εκτελείται και στα 5 execs).

```
process 12086 begins (load=7)
process 12086 ends (load=7)
PID 12086 - CMD work7
                        Elapsed Time: 0.222 secs
                        Workload Time: 0.223 secs
process 12087 begins (load=7)
process 12088 begins (load=7)
process 12088 ends (load=7)
PID 12089 - CMD work7
                        Elapsed Time: 0.005 secs
                        Workload Time: 0.594 secs
process 12090 begins (load=7)
process 12087 restarts
process 12088 restarts
process 12090 restarts
process 12087 restarts
process 12088 restarts
process 12090 restarts
process 12090 ends (load=7)
PID 12090 - CMD work7
                        Elapsed Time: 0.275 secs
                        Workload Time: 1.278 secs
process 12087 restarts
process 12087 ends (load=7)
PID 12087 - CMD work7
                        Elapsed Time: 0.313 secs
                        Workload Time: 1.289 secs
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
process 12088 restarts
```

PRIO exec - Η διεργασία 12088 τερματίζει, όμως το πρόγραμμα τερματίζει (ακαριαία) την 12089 που μόλις ξεκινάει.

```
process 12848 begins (load=7)
process 12849 begins (load=7)
process 12850 begins (load=7)
process 12851 begins (load=7)
process 12852 begins (load=7)
process 12848 restarts
process 12849 restarts
process 12850 restarts
process 12850 ends (load=7)
process 12851 restarts
PID 12851 - CMD work7
                        Elapsed Time: 0.104 secs
                        Workload Time: 0.964 secs
process 12852 restarts
process 12852 ends (load=7)
process 12848 restarts
process 12848 ends (load=7)
PID 12848 - CMD work7
                        Elapsed Time: 0.319 secs
                        Workload Time: 1.154 secs
process 12849 restarts
process 12849 ends (load=7)
PID 12849 - CMD work7
                        Elapsed Time: 0.299 secs
                        Workload Time: 1.250 secs
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
process 12850 restarts
```

PRIO cont - Αντίστοιχα η 12850 τερματίζει αλλά το πρόγραμμα σημειώνει την 12851.

```
process 18778 begins (load=7)
process 18779 begins (load=7)
process 18779 ends (load=7)
PID 18780 - CMD work7
                        Elapsed Time: 0.001 secs
                        Workload Time: 0.352 secs
process 18781 begins (load=7)
process 18782 begins (load=7)
process 18778 restarts
process 18779 restarts
process 18781 restarts
process 18781 ends (load=7)
PID 18781 - CMD work7
                        Elapsed Time: 0.258 secs
                        Workload Time: 0.926 secs
process 18782 restarts
process 18778 restarts
process 18778 ends (load=7)
PID 18778 - CMD work7
                        Elapsed Time: 0.287 secs
                        Workload Time: 1.124 secs
process 18779 restarts
process 18782 restarts
process 18779 restarts
process 18782 restarts
process 18782 ends (load=7)
PID 18782 - CMD work7
                        Elapsed Time: 0.372 secs
                        Workload Time: 1.489 secs
process 18779 restarts
process 18779 restarts
process 18779 restarts
process 18779 restarts
process 18779 restarts
process 18779 restarts
process 18779 restarts
process 18779 restarts
```

RR exec- Η 18779 τερματίζει, ο handler δεν τρέχει παρά όταν ξεκινήσει η 18781.

```
process 21920 begins (load=7)
process 21921 begins (load=7)
process 21922 begins (load=7)
process 21923 begins (load=7)
process 21924 begins (load=7)
process 21920 restarts
process 21921 restarts
process 21921 ends (load=7)
process 21922 restarts
PID 21922 - CMD work7
                        Elapsed Time: 0.106 secs
                        Workload Time: 0.810 secs
process 21923 restarts
process 21923 ends (load=7)
PID 21923 - CMD work7
                        Elapsed Time: 0.264 secs
                        Workload Time: 0.969 secs
process 21924 restarts
process 21920 restarts
process 21921 restarts
process 21924 restarts
process 21920 restarts
process 21920 ends (load=7)
PID 21920 - CMD work7
                        Elapsed Time: 0.341 secs
                        Workload Time: 1.394 secs
process 21921 restarts
process 21924 restarts
process 21924 ends (load=7)
PID 21924 - CMD work7
                        Elapsed Time: 0.352 secs
                        Workload Time: 1.529 secs
process 21921 restarts
process 21921 restarts
process 21921 restarts
process 21921 restarts
process 21921 restarts
process 21921 restarts
```

RR cont- Αντίστοιχα σε restarted εκτέλεση για τις 21921 και 21922.

Παραπάνω debug options αυτή τη φορά! SLEEP RESUILT/KILL RESULT: 0 για επιτυχή, -1 για ανεπιτυχή (σύμφωνα με POSIX), STATUS 2=RUNNING, 0=STOPPED, -1=EXITED.

Στην αριστερή φωτογραφία, η 4854 τερματίζει, όμως ο handler δεν τρέχει (πως γίνεται εφόσον μένει μόνο μία εντολή μετά την printf του work, η return), το nanosleep εκτελέστηκε εξ'ολοκλήρου άρα μήπως υπάρχει φαινόμενο πολύ κακού timing;

Στη δεξιά φωτογραφία, η 8295 τελειώνει, το σήμα stop λαμβάνεται επιτυχώς (kill result=0 άρα η διεργασία υπάρχει), το nanosleep τελείωσε κανονικά. Ο handler τρέχει ακριβώς αφού σταλθεί το σήμα στην επόμενη κατα σείρα διεργασία (8296), εκεί λαμβάνεται κανονικά το SIGSTOP, το nanosleep διακόπτεται κανονικά και το status θέτεται σε -1 (EXITED)... αλλά είναι η λάθος διεργασία!

```
HANDLER RAN
process 4851 begins
SLEEP RESUILT : 0 STATUS : 2
process 4852 begins
process 4852 ends

HANDLER RAN
SLEEP RESUILT : 0 STATUS : -1
PID 4852 - CMD work7
                        Elapsed Time: 0.230 secs
                        Workload Time: 0.345 secs
process 4853 begins
SLEEP RESUILT : 0 STATUS : 2
process 4854 begins
process 4854 ends
SLEEP RESUILT : 0 STATUS : 2

HANDLER RAN

REMINING SECONDS AND NANOSECONDS: 0 0
SLEEP RESUILT : -1 STATUS : -1

SETTING TO EXITED
PID 4855 - CMD work7
                        Elapsed Time: 0.000 secs
                        Workload Time: 0.674 secs
process 4851 restarts
SLEEP RESUILT : 0 STATUS : 2
process 4853 restarts
SLEEP RESUILT : 0 STATUS : 2
process 4854 restarts
SLEEP RESUILT : 0 STATUS : 2
process 4851 restarts
process 4851 ends

HANDLER RAN
SLEEP RESUILT : -1 STATUS : -1

SETTING TO EXITED
PID 4851 - CMD work7
                        Elapsed Time: 0.263 secs
                        Workload Time: 1.027 secs
process 4853 restarts
process 4853 ends

HANDLER RAN
SLEEP RESUILT : -1 STATUS : -1

SETTING TO EXITED
PID 4853 - CMD work7
                        Elapsed Time: 0.268 secs
                        Workload Time: 1.093 secs
process 4854 restarts
SLEEP RESUILT : 0 STATUS : 2
process 4854 restarts
SLEEP RESUILT : 0 STATUS : 2
process 4854 restarts
SLEEP RESUILT : 0 STATUS : 2
```

Αυξημένο debugging insight

```
PROCESS WAS RUNNING
EXEC ENDS
EXEC STARTS
process 8295 restarts
sending continue
continue sent
process 8295 ends
sending stop
stop sent
KILL RESULT: 0 SLEEP RESUILT : 0 STATUS : 2

PROCESS WAS RUNNING
EXEC ENDS
EXEC STARTS
process 8296 restarts
sending continue
continue sent

HANDLER RAN
sending stop
stop sent
KILL RESULT: 0 SLEEP RESUILT : -1 STATUS : -1

SETTING TO EXITED
PID 8296 - CMD work7
                        Elapsed Time: 0.104 secs
                        Workload Time: 0.744 secs
EXEC ENDS
EXEC STARTS
process 8297 restarts
sending continue
continue sent
process 8297 ends

HANDLER RAN
sending stop
stop sent
KILL RESULT: -1 SLEEP RESUILT : -1 STATUS : -1

SETTING TO EXITED
PID 8297 - CMD work7
                        Elapsed Time: 0.226 secs
                        Workload Time: 0.789 secs
EXEC ENDS
EXEC STARTS
process 8298 restarts
sending continue
continue sent
sending stop
stop sent
KILL RESULT: 0 SLEEP RESUILT : 0 STATUS : 2
```

**then 8295 restarts**

Ακόμα πιο πολλές printf για debugging