

/servers/sched/schedule.c

Η `do_noquantum` καλείται κάθε φορά που το `quantum` μιας διεργασίας τελειώνει. Όπως είναι, την σπρώχνει προς την επόμενη χαμηλότερης προτεραιότητας ουρά εκτέλεσης και ζητά από την `schedule_process()` να την χρονοπρογραμματίσει στην νέα ουρά. Αυτή με τη σειρά της καλεί την `sys_schedule()` για να το επιτύχει.

/lib/libsys/sys\_schedule.c

Κάνει το kernel call για να γίνει `scheduled` μια διεργασία, δηλαδή να τοποθετηθεί μέσα σε κάποια σειρά προτεραιότητας.

/kernel/system/do\_schedule.c

Ο handler του call `SYS_SCHEDULE`. Καλεί την `sched_proc()`

/kernel/system.c

Εδώ βρίσκεται η `sched_proc()`. Καλεί την `RTS_UNSET` για να κάνει `enqueue` την διεργασία

/kernel/proc.h

Εδώ βρίσκεται η `RTS_UNSET`. Καλεί την `enqueue`

/kernel/proc.c

Εδώ βρίσκεται η `enqueue`. Κρατάει δείκτες για το `queue head` και `tail`. Επίσης υπάρχει και η `enqueue_head` που βάζει κάποια διεργασία απευθείας στην πρώτη θέση της λίστας εκτέλεσης.

Με λίγα λόγια

1. Ο scheduler (SCHED) είναι έξω από το kernel και μπορεί να αποφασίσει σε ποιά σειρά προτεραιότητας θα μπει κάποια διεργασία μόλις έχει τελειώσει το `quantum` της.
2. Οι σειρές προτεραιότητας διαχειρίζονται από το kernel με τον παραδοσιακό τρόπο (round robin)
3. Εμείς θέλουμε μόλις τελειώσει μια διεργασία έχει καταναλώσει το `quantum` της, να γνωρίζουμε αν θα πρέπει να τοποθετηθεί κανονικά στο πίσω μέρος της ουράς ή να ξανατοποθετηθεί στο εμπρός μέρος, σύμφωνα με τη ζαριά της. Η ζαριά μπορεί να «πέσει» πριν η διεργασία καταναλώσει το `quantum` ή αφού το έχει μόλις καταναλώσει.
4. Αν είναι μεγαλύτερο από το ζάρι που έριξε η προηγούμενη, τότε ο scheduler θα την κρατήσει στην ίδια ουρά (δηλαδή δεν θα γίνει αυτό που γίνεται τώρα, να την σπρώξει χαμηλότερα) και θα την χρονοπρογραμματίσει έτσι ώστε να γίνει `enqueue` στην κεφαλή της σειράς προτεραιότητας.