Λειτουργικά συστήματα K22

Τεκμηρίωση 2ης εργασίας

- ΑΜ: 1115201300202 - Κώστας Χατζόπουλος -

Παραμετροποίηση του **xv6** scheduler

Όταν ξεκινήσει το λειτουργικό xv6, αντί του κανονικού **scheduler** που έχει, θα τρέξει ο **min_heap_scheduler** ο οποίος θα μπει σε μια ατέρμον λούπα και θα προσπαθεί συνεχώς να κάνει εξαγωγή τη πρώτη διαθέσιμη διεργασία από τη δομή **min-heap**. Για να γίνει αυτό, εκτός ότι πρέπει να υπάρχει κάποιο διαθέσιμο **process** να εξαχθεί, θα χρειαστεί εκείνη τη στιγμή να είναι κλειδωμένος ο πίνακας διεργασιών ώστε να μην προσπαθήσει κάποια άλλη **CPU** να κάνει τυχόν αλλαγές σε αυτόν όπου θα μπορούσαν να προκαλέσουν την κατάρρευση του συστήματος.

Έχοντας μια έγκυρη διεργασία στη μεταβλητή **p**, ο **scheduler** θα προσπαθήσει να κάνει **context switch** με την τρέχουσα κρατώντας "φρεναρισμένες" τις υπόλοιπες **CPU** που θέλουν εκείνη τη στιγμή να έχουν πρόσβαση στον πίνακα διεργασιών.

Χρονοπρογραμματισμός

Η λογική που ακολουθώ για τη διαδικασία του χρονοπρογραμματισμού έχει ως εξής:
Αρχικά, όταν γίνεται δέσμευση και δημιουργία μιας νέας διεργασίας (συνάρτηση **allocproc**) στον πίνακα διεργασιών, γίνεται αρχικοποίηση των τιμών: p->cr = ticks; (Creation time), p->sch = 0; (Schedule times),p->rt = 0; (Running Time), p->pr = 0.0; (Priority), p->mhi = -1; (Min-Heap Index) αμέσως μετά αφού δημιουργηθεί ο κατάλληλος χώρος για αυτή και όταν ακόμη βρίσκεται σε κατάσταση **EMBRYO** θα πρέπει να ενημερωθεί η κατάσταση της σε **RUNNABLE** (Ready). Για να γίνει αυτό, θα κληθεί η συνάρτηση **setReady** όπου δέχεται ως όρισμα ένα **process** και έναν **αριθμό προτεραιότητας.** Αν ο αριθμός αυτός ισοδυναμεί με **AUTO_PR** τότε η συνάρτηση **calc_pr** αναλαμβάνει να υπολογίσει μια προτεραιότητα για τη διεργασία η οποία υπολογίζεται από τον τύπο:

$$\text{p->pr} = \frac{p{-}{>}rt}{ticks - p{-}{>}cr}$$

διαφορετικά η μεταβλητή **priority** της διεργασίας θα πάρει την τιμή του ορίσματος **pr**.

Αμέσως μετά καλείται η **recalcpr** όπου υπολογίζει ξανά το priority για όλες τις υπόλοιπες διεργασίες που εκείνη τη στιγμή βρίσκονται σε κατάσταση **RUNNABLE**, (και συνεπώς είναι μέσα στο σωρό) κάνοντας ταυτόχρονα, για κάθε μία από αυτές αναδιάταξη του "δέντρου" της δομής min-heap ώστε ο κάθε κόμβος του να βρίσκεται ξανά στη σωστή θέση. Αφού ολοκληρωθεί ο επαναυπολογισμός των προτεραιοτήτων, τότε με την κλήση της συνάρτησης **min_heap_push** γίνεται εισαγωγή της νέας διεργασίας στο σωρό και τέλος το state της τίθεται σε κατάσταση **RUNNABLE.**

Κλήσεις συνάρτησης void setReady(struct proc *p, double pr);

Η συνάρτηση **setReady** αρχικά καλείται από τη συνάρτηση **userinit** όταν δημιουργείται ένα process όπως περιγράφηκε και παραπάνω, επιπλέον καλείται και στις εξής περιπτώσεις/συναρτήσεις:

- **Fork**: Η fork αναλαμβάνει να δημιουργήσει ένα αντίγραφο - παιδί της τρέχουσας διεργασίας. Για να γίνει αυτό, δημιουργείται ένα νέο process με τη βοήθεια της **allocproc** κάτι που σημαίνει πως οι τιμές των πεδίων **pid**, **state**, **rt**, **sch**, **cr**, **pr**, **mhi** αρχικοποιούνται εκ νέου, δηλαδή δεν αντιγράφονται από τη γονική διεργασία με αποτέλεσμα το παιδί να έχει τις δικές του τιμές για αυτά τα πεδία.
- **Yield - Λήξη timeslice:** Όταν ένα process ολοκληρώσει το **timeslice** του διότι προέκυψε κάποιο **interrupt** λήξης χρόνου (**IRQ_TIMER**), τότε αναλαμβάνει η **setReady** να το κάνει **RUNNABLE** από **RUNNING**. Αμέσως μετά καλείται η **sched** ώστε να γίνει **context switch** και να τρέξει άλλη διεργασία.
- **wakeup1:** Η συνάρτηση **wakeup** καλείται συνεχώς σε κάθε **clock tick** και αναλαμβάνει να "ξυπνήσει" τα process εκείνα τα οποία έχουν διακοπεί και περιμένουν να τελειώσει κάποιο I/O το οποίο ακόμη εκκρεμεί. Αρχικά σαρώνει όλο τον πίνακα διεργασιών (**ptable**) μέχρι να βρεθεί το process που την αφορά, το οποίο είναι σε κατάσταση **SLEEPING.** Όταν και αν βρεθεί, καλείται η **setReady** ώστε να το προετοιμάσει να τρέξει στον επόμενο γύρο (σε σειρά προτεραιότητας).

Κλήση συστήματος printRunningProc

CR = Creation time, **R** = Running time, **SCH** = Scheduled times, **INC** = Incomplete timeslices

Ενδεικτική εκτέλεση των **5 workloads** μαζί με το επαναληπτικό πρόγραμμα **ps** όπου χρησιμοποιεί **sleep()**

CR	NAME	PID	PPID	PR	S	R	LIFE	SCH	INC
0	init	1	0	0.0	S	1	13144	25	24
24	sh	2	1	0.0	S	4	13120	35	31
12737	ps	50	1	0.1	R	5	407	387	382
2863	w1	5	1	0.25	*	2602	10281	2608	6
2875	w2	7	1	0.11	S	1158	10269	7431	6273
2876	w3	9	1	0.11	S	1157	10268	7082	5925
2879	w4	11	1	0.11	S	1157	10265	7465	6308
2881	w5	13	1	0.11	S	1156	10263	7112	5956
8749	w1	29	1	0.25	R	1113	4395	1120	7
4503	w1	17	1	0.25	*	2187	8641	2194	7
4506	w2	19	1	0.11	S	975	8638	6335	5360
4510	w3	21	1	0.11	*	973	8634	6041	5068
4515	w4	23	1	0.11	S	972	8629	6273	5301
4543	w5	25	1	0.11	S	968	8601	6021	5053
8749	w2	31	1	0.11	S	496	4395	3120	2624
8761	w3	33	1	0.11	S	494	4383	2796	2302
8761	w4	35	1	0.11	S	495	4383	3080	2585
8763	w5	37	1	0.11	S	493	4381	2912	2419
9236	w1	40	1	0.25	*	989	3908	997	8
9238	w2	42	1	0.11	S	440	3906	2751	2311
9253	w3	44	1	0.11	S	438	3891	2613	2175
9266	w4	46	1	0.11	S	437	3878	2722	2285
9275	w5	48	1	0.11	S	436	3869	2531	2095