**Ιάσων Μαρινόπουλος – sdi2100097**

Τεκμηρίωση Εργασίας 2

Επεξήγηση setpriority

Η προσθήκη της λειτουργικότητας priority στις διεργασίας υλοποιήθηκε με την προσθήκη ενός πεδίου uint64 priority στη δομή proc. Η κλήση συστήματος setpriority καθαυτή λειτουργεί πολύ απλά: ελέγχει πως η δοσμένη τιμή είναι εντός των ορίων (1<=input<=20) και αναθέτει αυτή την τιμή στο πεδίο priority της διεργασίας.

Επεξήγηση Δομής pstat

Η υλοποίηση της δομής pstat γίνεται με πίνακες μεγέθους NPROC (= μέγιστος αριθμός διεργασιών που μπορούν να υπάρξουν στο kernel). Γίνεται η χρήση πινάκων και όχι μια συνδεδεμένης λίστας, λόγω του μικρού χώρου που απαιτεί η δομή. Ο χώρος είναι τόσο μικρός που χωράει μέσα σε μια σελίδα, με αποτέλεσμα να μπορεί να γίνει εύκολα η ανάθεση, χωρίς την ανάγκη να χρησιμοποιούμε πολύπλοκες δεσμεύσεις μνήμης. Τα δεδομένα που επιλέχθηκαν να αποθηκευτούν στη δομή είναι το όνομα των διεργασιών, το pid τους, η προτεραιότητα, το μέγεθος, η κατάσταση και το pid του γονέα τους.

Επεξήγηση getpinfo

Η κλήση συστήματος getpinfo χρησιμοποιεί για τη λειτουργία της μία δομή pstat στο kernel space. Η μνήμη αυτή πρέπει να ανατεθεί δυναμικά αλλιώς οδηγούμαστε σε kernel panic. Χρησιμοποιείται η συνάρτηση kalloc, επειδή η απαιτήσεις της δομής είναι μικρότερες από το μέγεθος μιας σελίδας (~2172 bytes σε σχέση με το μέγεθος σελίδας 4096 bytes). Γιαυτό τον λόγο, δεσμεύουμε μία σελίδα μόνο. Η κλήση αξιοποιεί έναν βασικό επαναληπτικό βρόχο για κάθε διεργασία και αποθηκεύει στη δομή τις αναγκαίες πληροφορίες. Στις περισσότερες πληροφορίες γίνεται με απλή ανάθεση, στην περίπτωση του πεδίου “ppid” γίνεται με έλεγχο αν υπάρχει γονέας ή όχι (αν δεν υπάρχει δίνεται η τιμή -1) και στο πεδίο “name” χρησιμοποιείται ένας βρόχος που μεταφέρεται η συμβολίσει ένα χαρακτήρα τη φορά. Αυτό συμβαίνει, επειδή οι συναρτήσεις memmove/strncpy/safestrncpy παρουσίαζαν όλες προβλήματα και “έτρωγαν” χώρο από τη δομή που δεν τους αναλογούσε, με αποτέλεσμα να εμφανίζονται λάθος πληροφορίες. Αυτή η διαδικασία γεμίσματος της δομής γίνεται αφού έχουμε κλειδώσει το lock της κάθε διεργασίας και του γονέα της, ώστε να αποφύγουμε τυχόν προβλήματα. Τέλος, μεταφέρουμε τα δεδομένα της δομής από το kernel space στο userspace με τη χρήση της συνάρτησης copyout και αποδεσμεύουμε τη σελίδα που είχαμε προηγουμένως χρησιμοποιήσει για την αποθήκευση των δεδομένων της δομής.

Επεξήγηση ps

Το πρόγραμμα χρήστη ps λειτουργεί πολύ απλά. Αρχικά, δημιουργεί μια δομή pstat και δεσμεύει την αντίστοιχη μνήμη με τη χρήση malloc. Η δομή αυτή βρίσκεται στο user space και σε αυτήν θα μεταφερθεί η δομή που θα δημιουργηθεί από την κλήση συστήματος getpinfo. Αφού ολοκληρωθεί η getpinfo, ελέγχουμε πως εκτελέστηκε ορθά και δεν αντιμετώπισε κάποιο σφάλμα. Στη συνέχεια, δημιουργούμε έναν πίνακα states ώστε να αντιστοιχήσουμε τις αριθμητικές τιμές του pstat→state σε αλφαριθμητικές. Τέλος, εκτυπώνουμε τα αποτελέσματα και αποδεσμεύουμε τη μνήμη που δεσμεύτηκε για την δομή pstat στο user space.

Επεξήγηση scheduler

Για να μετατραπεί ο scheduler σε μορφή round robin με προτεραιότητες γίνονται μερικές προσθήκες στον αρχικό κώδικα. Αρχικά, προσθέτουμε την μεταβλητή cand\_priority και την αρχικοποιούμε σε τιμή 21 στην αρχή κάθε επανάληψης του βρόχου. Αυτό γίνεται ώστε να αποθηκεύεται η υψηλότερη προτεραιότητα που βρέθηκε σε κάποια διεργασία αφού σκανάρουμε όλες τις διαθέσιμες και ώστε σε κάθε επόμενο βρόχο να ξεκινάμε να βρίσκουμε από την αρχή την υψηλότερη προτεραιότητα αντί να συνεχίζουμε από την προηγούμενη τιμή. Παράδειγμα: Ο scheduler εκτελεί μια διεργασία με προτεραιότητα 7. Αφότου ολοκληρωθεί προστίθεται μια διεργασία με προτεραιότητα 8. Αν δεν τεθεί η τιμή του cand\_priority σε 21 ξανά, τότε η διεργασία με προτεραιότητα 8 θα είναι αόρατη και δε θα εκτελεστεί ποτέ.

Επόμενη προσθήκη είναι ένας δεύτερος βρόχος που εκτελείται αφότου ολοκληρωθεί ο πρώτος. Αυτός διατρέχει επίσης όλες τις διαθέσιμες διεργασίες, με διαφορά πως αντί να αποθηκεύει την υψηλότερη προτεραιότητα που βρεθεί, εκτελεί όλες τις διεργασίες με την προτεραιότητα cand\_priority. Κατά τη διάρκεια της διαδικασίας γίνεται acquire και release το p→lock, ώστε να αποφευχθούν προβλήματα.

Αποτελέσματα Usertests

Εκτέλεση πριν οποιαδήποτε αλλαγή στον κώδικα:

|  |  |
| --- | --- |
|  |  |
|  |  |

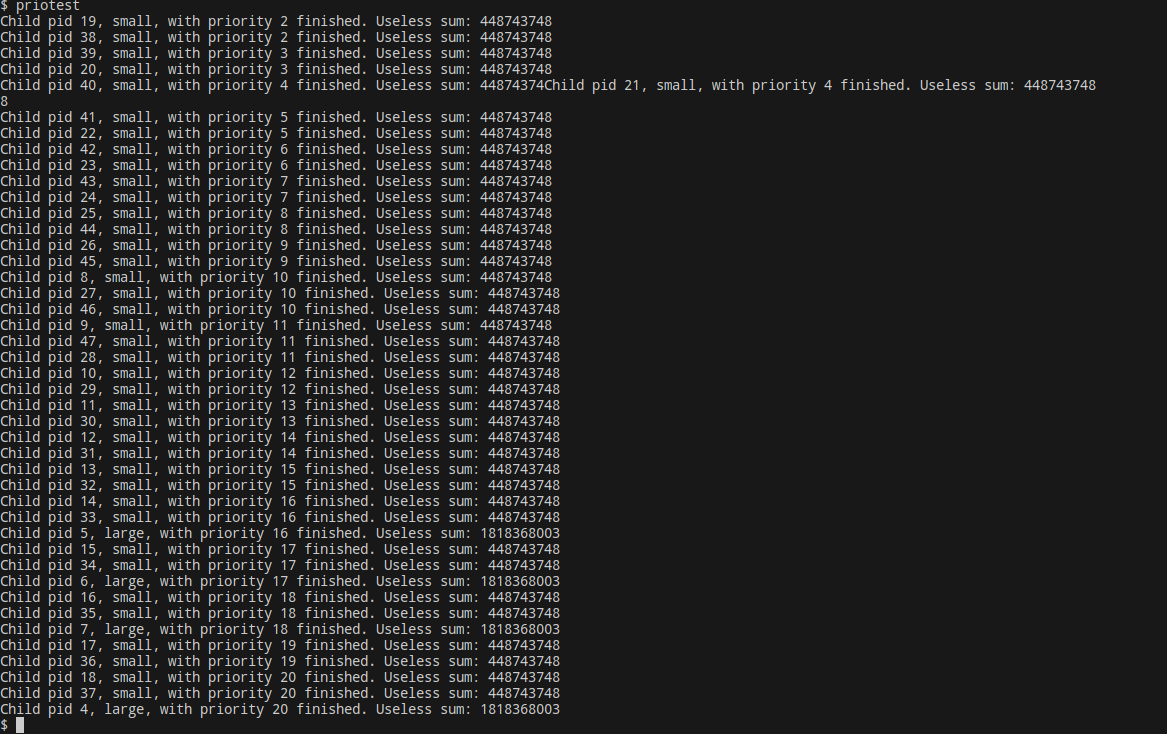
Εκτέλεση μετά την ολοκλήρωση της διεργασίας:

|  |  |
| --- | --- |
|  |  |
|  |  |

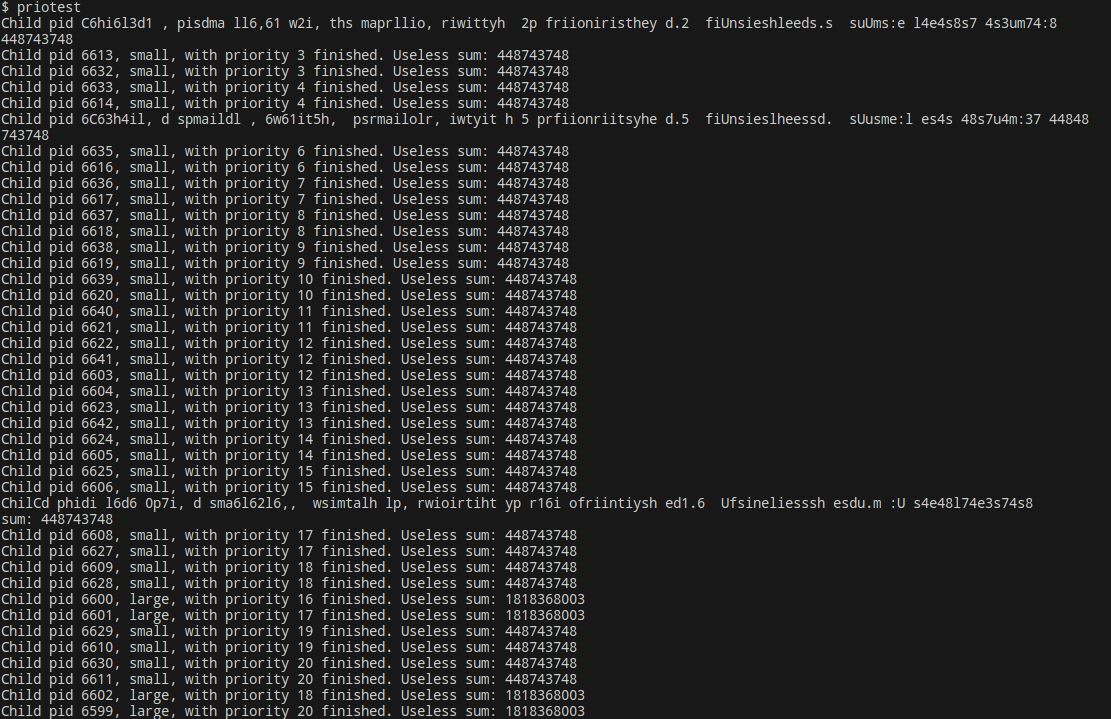
Αποτελέσματα Priotest

Τα αποτελέσματα από την priotest φαίνεται να “διακόπτουν” το ένα το άλλο κατά την εκτύπωση των μηνυμάτων όταν χρησιμοποιούμε 2 CPUS, στην αρχή και στο τέλος της συγγραφής της προσθήκης κώδικα μου. Όταν χρησιμοποιείται 1 CPU αυτό το πρόβλημα δεν εμφανίζεται. Παρατηρούμε πως με την υλοποίηση του scheduler round robin με προτεραιότητες, οι διεργασίες με υψηλότερες προτεραιότητες αρχίζουν την εκτέλεση τους πιο πριν σε σχέση με διεργασίες με χαμηλότερη προτεραιότητα (ανεξαρτήτως του πότε ολοκληρώνουν τη λειτουργία τους και εκτυπώνουν το αντίστοιχο μήνυμα).

Εκτέλεση με 1 CPU:

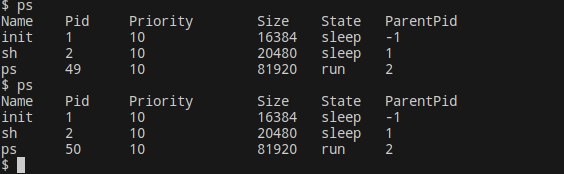


Εκτέλεση με 2 CPUS:



Ενδεικτικές εκτελέσεις ps

Μετά την εκτέλεση priotest:



Εκτέλεση σε σύστημα που μόλις ξεκίνησε:

