**Λειτουργικά Συστήματα**

**2022 - 2023**

**2η Εργαστηριακή Άσκηση**

Μέλη Ομάδας:

1. Όνομα : Αγγουρά Ρουμπίνη Μαρία

ΑΜ: 1084634

Email: [up1084634@upnet.gr](mailto:up1084634@upnet.gr)

1. Όνομα : Παυλόπουλος Ιάσονας

ΑΜ: 1084565

Email: [up1084565@upnet.gr](mailto:up1084565@upnet.gr)

**Εισαγωγή**

Πριν ξεκινήσουμε την παρουσίαση του Project, θα θέλαμε να σας ενημερώσουμε πως όλα τα resource files, όπως το αρχείο με τον κώδικα για το project 2 (scheduler.c), συμπεριλαμβανομένης και αυτής της αναφοράς μπορείτε να τα βρείτε στο παρακάτω Github Repository. Επίσης μπορείτε να δείτε τις αλλαγές που έχουν γίνει από την αρχή δημιουργίας του Project (commits).

**Github Repository**: <https://github.com/Roumpini21/Project-2--Operating-Systems>

**Περιγραφή Κώδικα**

Αρχικά πρωτού αρχίσουμε να υλοποιούμε κάποιο αλγόριθμο χρονοπρογραμματισμού, αρχικοποιήσαμε την δομή δεδομένων των διεργασιών μας (proc) με πεδία name, pid, priority, at (Arrival Time), bt (Burst Time), wt (Workload Time), state, status και δύο pointers next και prev οι οποίοι δείχνουν στην επόμενη και προηγούμενη διεργασία από αυτήν αντίστοιχα.

Έπειτα, αρχικοποιήσαμε την δομή της λίστας (queue) μας. Περιέχει δύο Pointers τύπου proc\* με όνομα head και end που σηματοδοτούν την αρχή και το τέλος της ουράς μας αντίστοιχα.

Τέλος αρχικοποιήσαμε έναν proc\* ώστε να αποθηκεύουμε τυχόν διεργασίες που θέλουμε να έχουμε πρόσβαση σε όλες τις συναρτήσεις μας.

Στην συνέχεια, στην main δημιουργήσαμε την βασική ουρά μας queue1 με την χρήση της συνάρτησης CreateQueue (στην οποία δεσμεύουμε χώρο για το struct μας), έναν File Pointer και έναν Integer option. Με βάση τα ορίσματα που δίνουμε στο terminal για την εκτέλεση του προγράμματος, είτε παίρνει το όνομα του αρχείου από το 3ο ή το 4ο όρισμα. Έπειτα συγκρίνοντας, με την χρήση της strcmp, το 2ο όρισμα με το όνομα του αλγορίθμου που θέλουμε να χρησιμοποιήσουμε, επιλέγουμε τις επόμενες κινήσεις μας. Για παράδειγμα, αν επιλεχθεί ο αλγόριθμος Batch, εκτυπώνεται κατάλληλο μήνυμα και εκτελείται η fill\_queue με ορίσματα την ουρά queue1, τον file pointer και το option (που θέτουμε ανάλογα την επιλογή του αλγορίθμου).

Η fill\_queue είναι μία συνάρτηση η οποία ανάλογα την περίπτωση, γεμίζει την ουρά με processes, δίνει τιμές στις μεταβλητές τους και ταξινομεί την ουρά. Συγκεκριμένα, με ένα switch-case statement ελέγχουμε το option και στην περίπτωση, για παράδειγμα, της επιλογής του batch αλγορίθμου, προσκομίζουμε με την χρήση fscanf το όνομα του αρχείου (μεταβλητή str) και το νούμερο που αναγράφεται δίπλα σε κάθε Process (μεταβλητή num). Στην συνέχεια, τρέχουμε την newProc μία συνάρτηση που είναι υπεύθυνη για την εισαγωγή νέων processes στην ουρά. Εμβαθύνοντας, παρατηρούμε πως δεσμεύουμε δυναμικά χώρο για κάθε process και το βάζουμε στο τέλος της ουράς θέτοντας τους κατάλληλους pointers εκεί που πρέπει να δείχνουν κάθε φορά. Αυτή η διαδικασία τρέχει μέχρι να βρούμε χαρακτήρα EOF. Στην περίπτωση του Batch λοιπόν, χρησιμοποιώντας την strcpy, θέτουμε το όνομα ίσο με την μεταβλητή str, το state της διεργασίας ίσο με Ready, καθώς και το arrival time ίσο με num. Αξίζει να αναφερθεί πως σε κάθε περίπτωση, το num χειρίζεται διαφορετικά, όπως για παράδειγμα στην περίπτωση της επιλογής του αλγορίθμου SJF, όπου θέτουμε το burst time της κάθε διεργασίας ίσο με το num. Τέλος, για τους BATCH και SJF εκτελούμε bubble\_sort αλγορίθμους (με τις bubble\_sjf και bubble\_batch) ώστε να ταξινομήσουμε σε κάθε περίπτωση τις διεργασίες με βάση το arrival time και burst time αντίστοιχα.

Γυρίζοντας πίσω στην main, ανάλογα με τον αλγόριθμο που έχουμε επιλέξει τρέχει και η κατάλληλη συνάρτηση. Για τους αλγορίθμους Batch και SJF χρησιμοποιούμε την ίδια συνάρτηση (batch\_sjf) η οποία παίρνει σαν όρισμα την ουρά queue1.

Μέσα στην συνάρτηση αυτή, αρχικοποιούμε μεταβλητές τύπου timespec για την χρονομέτρηση των διεργασιών και μία μεταβλητή τύπου proc\* ώστε να κρατάμε την διεργασία η οποία πρόκειται να εκτελεστεί.

Μπαίνοντας στο while βρόγχο, μέχρι να αδειάσει η λίστα, εκτελούμε τα εξής βήματα:

* Βγάζουμε από την λίστα την διεργασία που πρόκειται να εκτελέσουμε με την χρήση της deQueue η οποία μας επιστρέφει έναν pointer στην διεργασία αυτή.
* Αρχίζουμε την χρονομέτρηση της διεργασίας.
* Κάνουμε fork.
* Ο γονέας θέτει το pid της συγκεκριμένης διεργασίας στο current\_process.
* Ο γονέας θέτει την κατάσταση της διεργασίας σε RUNNING.
* Ταυτόχρονα με execl αρχίζει η εκτέλεση της διεργασίας-παιδί.
* Ο γονέας περιμένει να εκτελεστεί το παιδί.
* Ο γονέας θέτει την κατάσταση της διεργασίας σε EXITED.
* Η χρονομέτρηση τελειώνει και γίνονται οι κατάλληλοι υπολογισμοί.
* Εκτυπώνεται μήνυμα με το ποια διεργασία πήρε πόσο χρόνο να εκτελεστεί και πόση ώρα έχει τρέξει όλο το πρόγραμμα μετά την εκτέλεση της διεργασίας.
* Τέλος εκτυπώνεται ο συνολικός χρόνος εκτέλεσης του προγράμματος.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΈνα παράδειγμα εκτέλεσης της Batch και SJF μπορείτε να δείτε παρακάτω:

Για τον αλγόριθμο Round Robin έπρεπε να υλοποιηθεί διαφορετικά η συνάρτηση του. Συγκεκριμένα, χρησιμοποιούμε τις εντολές sigemptyset και sigaction για να αρχικοποιήσουμε τον handler του SIGCHLD.

Έπειτα αρχικοποιούμε μεταβλητές τύπου timespec που θα τις χρησιμοποιήσουμε στην κλήση της συνάρτησης nanosleep και για την χρονομέτρηση των διεργασιών μας και μία μεταβλητή τύπου proc\* ώστε να κρατάμε την διεργασία η οποία πρόκειται να εκτελεστεί.

Μπαίνοντας στον πρώτο while βρόγχο, μέχρι να αδειάσει η αρχική μας λίστα, εκτελούμε τα εξής βήματα:

* Βγάζουμε από την λίστα την διεργασία που πρόκειται να εκτελέσουμε με την χρήση της deQueue η οποία μας επιστρέφει έναν pointer στην διεργασία αυτή και τον θέτουμε σε έναν Process pointer (p).
* Έπειτα, βάζουμε στο τέλος μιας καινούριας λίστας το process p.

Μπαίνοντας στον δεύτερο while βρόγχο, μέχρι να αδειάσει η καινούρια μας λίστα, εκτελούμε τα εξής βήματα:

* Βγάζουμε από την λίστα την διεργασία που πρόκειται να εκτελέσουμε με την χρήση της deQueue η οποία μας επιστρέφει έναν pointer στην διεργασία αυτή.
* Αρχίζουμε τα Clocks που χρειάζονται για την χρονομέτρηση
* Θέτουμε τον global g\_pointer process ίσο με το p.
* Ελέγχουμε αν η κατάσταση της είναι READY. Αν είναι:
  + Κάνουμε Fork
  + Ταυτόχρονα με execl αρχίζει η εκτέλεση της διεργασίας-παιδί.
* Εάν δεν είναι READY
  + Με την χρήση της kill στέλνεται το σήμα SIGCONT στην διεργασία και κάνει Continue.
* Θέτουμε την κατάσταση της διεργασίας σε RUNNING.
* Καλείται η nanosleep για τον χρόνο που έχει ζητήσει ο χρήστης.
* Ελέγχουμε αν η κατάσταση της είναι EXITED. Αν είναι:
  + Βάζουμε την διεργασία στην αρχική λίστα και κάνουμε continue.
* Αν δεν είναι:
  + Σταματάμε το Clock και υπολογίζουμε το χρόνο που πέρασε.
  + Τον προσθέτουμε στο burst time της διεργασίας
  + Αυξάνουμε τον temp\_time και τον θέτουμε στο workload time της διεργασίας.
* Με την χρήση της kill στέλνεται το σήμα SIGSTOP στην διεργασία και κάνει Stop.
* Αλλάζουμε την κατάσταση της διεργασίας σε STOPPED
* Την βάζουμε στο τέλος της καινούριας λίστας.

Σε περίπτωση που μία διεργασία τελειώσει, στέλνει το σήμα SIGCHLD. Αυτό το σήμα το χειριζόμαστε στην συνάρτηση childHandler. Τα περιεχόμενα του handler φαίνονται παρακάτω: Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, οθόνη

Περιγραφή που δημιουργήθηκε αυτόματα

* Όταν καλείται ο childHandler κάνουμε waitpid για την συγκεκριμένη εργασία που στέλνει το σήμα ώστε να τερματίσει. Σταματάμε το clock και υπολογίζουμε τον χρόνο που πέρασε από το προηγούμενο κβάντο μέχρι να σταματήσει η εκτέλεση της διεργασίας καθώς μπορεί να τελείωσε πριν περάσει όλο το κβάντο χρόνου. Τέλος, Εκτυπώνεται μήνυμα με το ποια διεργασία πήρε πόσο χρόνο να εκτελεστεί και πόση ώρα έχει τρέξει όλο το πρόγραμμα μετά την εκτέλεση της διεργασίας.
* Ένα παράδειγμα εκτέλεσης της Round Robin μπορείτε να δείτε παρακάτω:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα