ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ακαδημαϊκό Έτος 2022-2023

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ

2η Εργαστηριακή Άσκηση

ΤΟΠΑΛΗΣ ΛΑΖΑΡΟΣ

upnet email ceid email

Α.Μ.: 1088101

ΣΑΜΑΡΑ ΧΡΙΣΤΙΝΑ – ΕΛΕΑΝΝΑ

upnet email ceid email

Α.Μ.: 1084622

Πάτρα 2022-23

Περιεχόμενα

Πρόλογος

Όλα τα μέρη του project υλοποιήθηκαν πλήρως. Οι αλγόριθμος και τυχόν εικόνες που χρησιμοποιήθηκαν, βρίσκονται και στο github.

Εισαγωγή

Η υλοποίηση του προγράμματος πραγματοποιείται σε δύο μέρη· την εισαγωγή των διεργασιών σε λίστα σύμφωνα με την σειρά που θα εκτελεστούν και τον έλεγχο εκτέλεσης της κάθε μίας.

Η βιβλιοθήκη structures.h εμπεριέχει τις βασικές δομές και συναρτήσεις που είναι χρήσιμες για την υλοποίηση του προγράμματος. Η σημαντικότερη δομή είναι η **process_list**, σκοπός της οποίας είναι να διατηρεί τις πληροφορίες κάθε διεργασίας και αποτελεί και τον κόμβο της διπλά συνδεδεμένης λίστας. Το πεδίο με τις πληροφορίες της διεργασίας αποτελεί μία άλλη δομή **process_info**, η οποία διατηρεί πληροφορίες, όπως το pid, το όνομα-διαδρομή της διεργασίας, χρόνοι εκτέλεσης, την προτεραιότητα καθώς και μία λίστα τύπου **history_data**, η οποία περιέχει τις εναλλαγές στην κατάσταση της διεργασίας (**READY, BLOCKED, RUNNING**) και την ώρα που συνέβη.

Επιπλέον, υπάρχουν συναρτήσεις που υλοποιούν ορισμένες χρήσιμες λειτουργίες. Η **copyInfoStructure** παίρνει δύο ορίσματα τύπου **process_info** και αντιγράφει τις τιμές του src στο dest. Επίσης, η **toString** τυπώνει μορφοποιημένα τις πληροφορίες ενός **process_info** στο αρχείο file. Η συνάρτηση **is_first** ελέγχει αν το node τύπου **process_list** είναι το πρώτο της λίστας. Μία αρκετά χρήσιμη συνάρτηση είναι, επιπλέον, η **print_to_file**, η οποία με όρισμα την ρίζα της λίστας με τις ολοκληρωμένες διεργασίες, τυπώνει τις πληροφορίες εκτέλεσης σε ένα αρχείο ονόματι **output.txt**, ενώ τις πληροφορίες του ιστορικού εκτέλεσης τις τυπώνει σε ένα αρχείο **history.txt**. Τα αρχεία αυτά είναι αποθηκευμένα σε έναν φάκελο **output** που δημιουργεί η ίδια η συνάρτηση στον τρέχοντα φάκελο. Τέλος, η συνάρτηση **get_wtime** επιστρέφει το πλήθος των δευτερολέπτων που έχουν περάσει από την ημέρα “γέννησης” του UNIX (01/01/1971).

Λίστα Αναμονής

Εξυπηρέτηση με βάση τη σειρά άφιξης (FCFS)

Το πρώτο βήμα της υλοποίησης του περιβάλλοντος χρονοπρογραμματισμού είναι η δημιουργία της λίστας αναμονής. Η εξυπηρέτηση με βάση την σειρά άφιξης (First In First Served) δίνει προτεραιότητα στις διεργασίες οι οποίες ζητούν πρώτες την χρήση της CPU και εγγυάται ότι θα την πάρουν.

Με την συνάρτηση **FCFSadd()** προσθέτουμε στην λίστα αναμονής τις διεργασίες που διαβάζουμε από το αρχείο. Η συνάρτηση παίρνει ως ορίσματα: το **data** τύπου **process info** και το **root** τύπου **process_list *,** δείκτη στην αρχή της λίστας. Αρχικά, διατρέχουμε την λίστα έτσι ώστε να βρούμε την τελευταία διεργασία που εισήχθη. Μόλις βρεθεί, δεσμεύουμε χώρο χρησιμοποιώντας την **malloc** για να προστεθεί η νέα διεργασία ως επόμενή της. Ορίζουμε επίσης ως προηγούμενη της νέας διεργασίας την τρέχουσα τελευταία διεργασία. Τέλος, με την συνάρτηση **copyInfoStructure()** αντιγράφουμε τα δεδομένα του **data** στο πεδίο **info** του νέου κόμβου και ορίζουμε ως επόμενό του το **NULL**.

Εξυπηρέτηση με βάση τη μικρότερη διάρκεια (SJF)

Ο αλγόριθμος της SJF αποτελεί μία βελτίωση του αλγορίθμου FCFS. Αντί, όμως, να εκτελεί τις διεργασίες τυφλά όπως έρχονται στην CPU, ελέγχει τους χρόνους εκτέλεσης που χρειάζεται για την διεκπεραίωση της διεργασίας και δίνει μεγαλύτερη προτεραιότητα σε αυτές με τον λιγότερο χρόνο εκτέλεσης.

Για να πραγματοποιηθεί αυτή η λειτουργικότητα ξεκινώντας από τον κόμβο-ρίζα ψάχνουμε να βρούμε το σημείο που θα πρέπει να τοποθετηθεί ο κόμβος αυτός. Αυτό επιτυγχάνεται αναζητώντας τον κόμβο με μεγαλύτερο priority σε σχέση με το δικό του. Εφόσον βρούμε αυτόν τον κόμβο, τον τοποθετούμε στην λίστα. Οι πιθανές θέσεις είναι στην αρχή, στο τέλος ή εμφωλευμένη ανάμεσα σε δύο υπάρχοντες κόμβους. Ο νέος κόμβος δημιουργείται με δέσμευση μνήμης και έπειτα με χρήση της συνάρτησης **copyInfoStructure**, ώστε να αντιγραφούν τα στοιχεία από το data στον κόμβο της λίστας.

Δρομολόγηση εκ περιτροπής (RR)

Η δημιουργία της λίστας αναμονής στη περίπτωση της δρομολόγησης εκ περιτροπής (Round Robin) γίνεται όμοια με την **FCFS**.

Δρομολόγηση με βάση την προτεραιότητα (PRIO)

Η δημιουργία της λίστας αναμονής στη περίπτωση της δρομολόγησης εκ περιτροπής (Round Robin) γίνεται όμοια με την **SJF**.

Δρομολογητής

Μετά την δημιουργία της λίστας αναμονής των διεργασιών, σειρά έχει η δρομολόγησή τους με την σωστή σειρά.

Εξυπηρέτηση με βάση τη σειρά άφιξης (FCFS)

Για τον αλγόριθμο **FCFS**, οι διεργασίες πρέπει να εξυπηρετηθούν με βάση την σειρά άφιξής τους. Η δρομολόγηση γίνεται με την συνάρτηση **FCFS()**, η οποία παίρνει ως ορίσματα δείκτες στις δύο λίστες: την λίστα αναμονής και την τελική λίστα, στην οποία θα αποθηκεύονται μια μια κάθε διεργασία που τερματίζει.
Κάθε φορά, η επόμενη προς εκτέλεση διεργασία θα είναι η πρώτη της λίστας αναμονής. Η διεργασία-πατέρας, χρησιμοποιώντας την συνάρτηση **fork()** δημιουργεί μια διεργασία-παιδί, η οποία με την συνάρτηση **execvp()** εκτελεί την καθορισμένη διαδικασία. Με το που αρχίσει η εκτέλεση, το ιστορικό του αντίστοιχου κόμβου διεργασίας (**history node**) ενημερώνεται και παίρνει την τιμή **“RUNNING”**. Η διεργασία-πατέρας περιμένει το αντίγραφο να τελειώσει με χρήση της εντολής **waitpid()**. Τότε, ενημερώνεται πάλι το ιστορικό του κόμβου διεργασίας σε “**EXITED**”. Η διεργασία αυτή διαγράφεται από την αρχική λίστα αναμονής και προστίθεται στο τέλος της τελικής λίστας (**finished**). Η διαδικασία αυτή επαναλαμβάνεται για όλους τους κόμβους της λίστας αναμονής, έως ότου φτάσει στο τέλος της.

Εξυπηρέτηση με βάση τη μικρότερη διάρκεια (SJF)

Ο χρονοδρομολογητής του αλγορίθμου SJF είναι ο ίδιος με αυτόν του **FCFS**, όπως παρουσιάστηκε προηγουμένως.

Δρομολόγηση εκ περιτροπής (RR)

Το σκεπτικό του αλγορίθμου RR είναι ότι κάθε διεργασία έχει συγκεκριμένο (τον ίδιο) χρονικό διάστημα εκτέλεσης έτσι ώστε να μην υπάρχουν διεργασίες οι οποίες να βρίσκονται μεγάλο χρονικό διάστημα σε αναμονή.

Ο αλγόριθμος RR, για να πετύχει αυτήν την λειτουργικότητα, δημιουργεί αρχικά για κάθε νέα διεργασία ένα πιστό αντίγραφο του εαυτού του έτσι ώστε να εκτελεστεί το πρόγραμμα που απαιτείται. Παράλληλα, η διεργασία-πατέρας ενημερώνει την λίστα **history_node** του κάθε κόμβου-διεργασίας θέτοντας στο status την τιμή “**RUNNING”**, έτσι ώστε να υπάρχει ιστορικό με όλα τις καταστάσεις από τις οποίες πέρασε η εκάστοτε διεργασία. Έπειτα, η γονική διεργασία περιμένει για ένα χρονικό διάστημα, το οποίο δόθηκε από τον χρήστη με την αρχή του προγράμματος και αποτελεί το κβάντο (δίνεται σε ms), την χρονική διάρκεια που έχει κάθε διεργασία για να εκτελεστεί μέχρι να σταματήσει και να πάρει σειρά η επόμενη. Μετά το πέρας αυτού του χρονικού διαστήματος, η διεργασία-γονέας στέλνει στην διεργασία που εκτελείται σήμα να σταματήσει με χρήση της εντολής **kill**. Το σήμα στέλνεται με χρήση του PID της εκτελούμενης διεργασίας το οποίο το έχει ο γονέας κρατημένο στον κόμβο της λίστας μαζί με επιπλέον πληροφορίες, όπως αναλύθηκε στο κεφάλαιο Λίστα Αναμονής. Αν το σήμα δεν σταλεί επιτυχώς τυπώνεται στην οθόνη αντίστοιχο ενημερωτικό μήνυμα. Η διεργασία-γονέας στην συνέχεια, μέσω της συνάρτησης **waitpid**() περιμένει έως ότου η συγκεκριμένη διεργασία σταματήσει και κρατάει στην μεταβλητή **status** την επιστρεφόμενη τιμή της διεργασίας-παιδί. Με χρήση αυτής, θα μπορέσει να καταλάβει πώς τερμάτισε η διεργασία.

Εάν η διεργασία ολοκληρώθηκε πλήρως και γι’ αυτό τερμάτισε τότε η συνάρτηση **WIFEXITED()** θα επιστρέψει *true*. Σε αυτήν την περίπτωση, η διεργασία-πατέρας ενημερώνει το ιστορικό της

διεργασίας με **status** την τιμή **“EXITED”** και τοποθετεί τον κόμβο με τις πληροφορίες της διεργασίας στο τέλος μίας άλλη λίστα **finished_node**, σκοπός της οποίας είναι να κρατά όσες διεργασίες έχουν ολοκληρωθεί πλήρως. Εφόσον ο κόμβος αυτός αφαιρέθηκε από την λίστα root τότε σαν πρώτος κόμβος βρίσκεται η επόμενη προς εκτέλεση διεργασία. Εάν δεν υπάρχει άλλη διεργασία, τότε όλες έχουν εκτελεστεί και ο αλγόριθμος RR τερματίζει. Εάν, αντίθετα, υπάρχει κι άλλη διεργασία, τότε ελέγχεται αν η διεργασία αυτή έχει ήδη εκτελεστεί, έστω και μία φορά ή όχι. Ο έλεγχος γίνεται με χρήση της τιμής του PID που είναι αποθηκευμένο στον κόμβο **process_info** (0: δεν έχει εκτελεστεί, ≠0: έχει εκτελεστεί έστω και μία φορά με το συγκεκριμένο PID). Εφόσον έχει ήδη εκτελεστεί στέλνουμε ένα σήμα ***SIGCONT*** με χρήση της ***kill*** ώστε να βάλουμε την συγκεκριμένη εργασία να ξεκινήσει πάλι, αλλιώς δημιουργεί μία νέα διεργασία και αντικαθιστά τον κώδικα της με τον κώδικα της προς εκτέλεση διεργασίας.

Αντίθετα, αν η διεργασία σταμάτησε λόγω του μηνύματος που στάλθηκε, τότε η συνάρτηση **WIFEXITED()** επιστρέφει *false,* ενώ η ***WIFSTOPPED()*** *true.* Σε αυτήν την περίπτωση ενημερώνεται και πάλι το ιστορικό της συγκεκριμένης διεργασίας θέτοντας ***status = “STOPPED”*** και έπειτα ο συγκεκριμένος κόμβος με τα στοιχεία της διεργασίας τοποθετείται στο τέλος της λίστας root για να συνεχίσει την εκτέλεση της αργότερα. Έπειτα, επιλέγουμε πάλι την διεργασία που βρίσκεται στον πρώτο κόμβο της λίστας root, καθώς αυτή είναι η επόμενη προς εκτέλεση και γίνονται οι ίδιοι έλεγχοι με πριν εφόσον η διεργασία έχει ήδη εκτελεστεί ή όχι. Τέλος, ενημερώνει το ιστορικό του συγκεκριμένου κόμβου θέτοντας τιμή στο ***status = “RUNNING”***.

Δρομολόγηση με βάση την προτεραιότητα (PRIO)

Ο αλγόριθμος προτεραιότητας PRIO (**Priority Scheduling Algorithm**) εξυπηρετεί τις διεργασίες που φτάνουν στο σύστημα με βάσει την προτεραιότητά τους. Για αυτόν τον λόγο, η λίστα αναμονής που χρειάζεται ένας δρομολογητής που εκτελεί χρονοπρογραμματισμό PRIO περιλαμβάνει τις διεργασίες ταξινομημένες με αύξουσα τιμή προτεραιότητας. Ο αλγόριθμος αποτελεί έναν συνδυασμό των SJF και RR, καθώς όταν οι διεργασίες έχουν διαφορετικές προτεραιότητες εκτελούνται η μια μετά την άλλη, ενώ όταν βρεθούν διεργασίες με την ίδια προτεραιότητα εκτελούνται εκ περιτροπής.

Η υλοποίηση του αλγορίθμου αυτού έγινε με την χρήση βοηθητικών λιστών, μια για κάθε τιμή προτεραιότητας. Διατρέχοντας την λίστα αναμονής, κάθε κόμβος-διεργασία τοποθετήθηκε στην αντίστοιχη λίστα σύμφωνα με την τιμή που είχε στο πεδίο **priority**. Έτσι για παράδειγμα, οι διεργασίες με priority = 7 θα βρεθούν στην λίστα **array_list[7]**, ενώ η διεργασία με priority = 3 πάει στην λίστα **array_list[3]**. Με τον τρόπο αυτό μπορούμε να ξεχωρίσουμε ποιες διεργασίες θα εκτελεστούν με χρήση δρομολόγησης SJF και ποιες θα χρειαστούν δρομολόγηση RR, ανάλογα με το πόσα στοιχεία έχει κάθε λίστα. Συγκεκριμένα, ελέγχουμε αν υπάρχει ή όχι δεύτερο στοιχείο στην λίστα, και βάζουμε την λίστα να εκτελεστεί με FCFS ή RR ανάλογα.