ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ακαδημαϊκό Έτος 2023-2024

2η Εργαστηριακή Άσκηση

Στοιχεία Φοιτητών:
Ονοματεπώνυμο: **ΚΟΥΤΡΟΥΜΠΕΛΑΣ ΒΑΣΙΛΕΙΟΣ**
ΑΜ: **1093397**
email: **up1093397@ac.upatras.gr**

Ονοματεπώνυμο: **ΜΙΝΩΠΕΤΡΟΣ ΦΙΛΙΠΠΟΣ**
ΑΜ: **1093431**
email: **up1093431@ac.upatras.gr**

Περιεχόμενα

Εισαγωγή

Για τις παρακάτω φάσεις της υλοποίησης του χρονοπρογραμματιστή διεργασιών υλοποιήθηκαν, από την πρώτη φάση ο χρονοδρομολογητής και για τις δύο πολιτικές δρομολόγησης, ενώ για την δεύτερη φάση ο χρονοδρομολογητής και μόνο η πολιτική FCFS. Όλα τα τμήματα της άσκησης που υλοποιήθηκαν λειτουργούν σωστά, με βάση και τα αρχεία δοκιμών που δόθηκαν.

Πρώτη Φάση

Για την πρώτη φάση μας ζητήθηκε να υλοποιήσουμε έναν χρονοδρομολογητή διεργασιών με τις πολιτικές χρονοδρομολόγησης First Come First Serve και Round Robin. Αρχικά δημιουργήσαμε μια δομή δεδομένων (struct) Process η οποία διατηρεί στοιχεία για την κάθε διεργασία (όνομα εκτελέσιμου αρ χείου, αναγνωριστικό (pid), κατάσταση εκτέλεσης, χρόνος εισόδου στην ουρά εκτέλεσης) αλλά και πληροφορίες για το που βρίσκεται μεταξύ άλλων διεργασιών στην ουρά εκτέλεσης. Έπειτα υλοποιούμε τις συναρτήσεις οι οποίες εισάγουν στο τέλος της ουράς μια διεργασία (insert_end) και εξάγουν από την αρχή (pop_first).

Στην main συνάρτηση, αρχικά ελέγχουμε τις επιλογές που έχει δώσει ο χρήστης στην εκτέλεση του προγράμματος και περνάμε τα δεδομένα αυτά σε μεταβλητές. Σε περίπτωση που έχει δώσει λανθασμένες επιλογές εμφανίζεται μήνυμα για το πώς πρέπει να γίνει η εκτέλεση και το πρόγραμμα σταματάει. Έπειτα, ανοίγουμε το αρχείο που περιέχει τις διαδρομές των εκτελέσιμων που θα τοποθετήσουμε στην ουρά των διεργασιών. Τοποθετούμε τις διεργασίες στην ουρά (insert_end) και ο δρομολογητής ξεκινάει την λειτουργία του για όσο υπάρχουν διεργασίες στην ουρά. Ο δρομολογητής εξάγει μια διεργασία από την ουρά (pop_first) και ελέγχει αν είναι καινούργια, δηλαδή δεν έχει ξεκινήσει η εκτέλεση της. Αν είναι καινούργια καλεί την συνάρτηση start_process η οποία δημιουργεί ένα αντίγραφο του προγράμματος (fork) και μέσα σε αυτό καλούμε την συνάρτηση execl για να εκτελεστεί η καινούργια διεργασία. Παράλληλα ο δρομολογητής αποθηκεύει το PID της διεργασίας που μόλις ξεκίνησε και συνεχίζει κανονικά η ροή του προγράμματος. Αν πάλι η διεργασία που μόλις εξάγαμε δεν είναι καινούργια, στέλνουμε σε αυτή το σήμα SIGCONT ώστε να συνεχίσει η εκτέλεση της μιας και είναι η σειρά της να απασχολήσει την CPU.

Ενόσω η διεργασία εκτελείται στην CPU ο δρομολογητής “κοιμάται”, με την συνάρτηση sleep_milliseconds, μέχρι να τελειώσει η εκτέλεση της, στην περίπτωση της πολιτικής FCFS, ή μέχρι να τελειώσει το προεπιλεγμένο από τον χρήστη κβάντο χρόνου στην περίπτωση της πολιτικής RR. Για την FCFS έχουμε θέσει ένα πολύ υψηλό κβάντο χρόνου (μέγιστος ακέραιος) στην sleep_milliseconds μιας και δεν θέλουμε να σταματήσει η εκτέλεση μιας διεργασίας μέχρι αυτή να τελειώσει.

Μόλις μια διεργασία (παιδί) τελειώσει την εκτέλεση της στέλνει σήμα SIGCHLD και ο “ύπνος” του δρομολογητή διακόπτεται. Νωρίτερα, έχουμε ορίσει έναν χειριστή του σήματος αυτού ο οποίος ενημερώνει ένα flag (global μεταβλητή) και καταλαβαίνουμε ότι η διεργασία έχει τελειώσει την εκτέλεση της. Αξίζει να σημειωθεί πως, έχουμε ορίσει με σχετικό flag στο sigaction struct να μην ενεργοποιείται ο χειριστής του σήματος σε περίπτωση που η διεργασία σταματήσει (SIGSTOP), αφού δημιουργούσε πρόβλημα στην RR. Δηλαδή, όταν μια διεργασία σταματούσε λόγω λήξης χρόνου ενεργοποιούταν ο χειριστής και ο δρομολογητής θεωρούσε πως η διεργασία που ερχόταν επόμενη τελείωσε.

Έτσι, με το που τελειώσει η εκτέλεση, ενημερώνουμε την κατάσταση της διεργασίας που ολοκληρώθηκε κρατάμε τον χρόνο που ξόδεψε συνολικά στον δρομολογητή, τυπώνουμε τις σχετικές πληροφορίες, το flag που δείχνει πως η διεργασία τελείωσε επαναφέρεται σε 0 για την επόμενη διεργασία και ο δρομολογητής διαλέγει την επόμενη προς εκτέλεση διεργασία.

Στην περίπτωση της πολιτικής RR και εφόσον δεν έχει τελειώσει η διεργασία στον χρόνο που έχει ορίσει ο χρήστης, στέλνεται στην εκτελούμενη διεργασία το σήμα SIGSTOP για να σταματήσει η εκτέλεση της, ενημερώνεται η κατάσταση της και τοποθετείται ξανά στο τέλος της ουράς.

Δεύτερη Φάση

Για την δεύτερη φάση μας ζητήθηκε να επεκτείνουμε την λειτουργιά του χρονοδρομολογητή διεργασιών ώστε να λαμβάνει υπόψη του εφαρμογές που πρόκειται να εκτελέσουν εντολές I/O. Το βασικό πρόβλημα που αντιμετωπίσαμε ήταν να "χτίσουμε" πάνω στον προηγούμενο δρομολογητή ώστε να μπορούν να χρησιμοποιηθούν και οι δύο πολιτικές δρομολόγησης. Το πρόβλημα δεν λύθηκε και έτσι υλοποιήσαμε το ζητούμενο αλλάζοντας το προηγούμενο πρόγραμμα ώστε να υλοποιεί μόνο την πολιτική FCFS και συνεχίσαμε από εκεί.

Για αρχή, προσθέσαμε δυο χειριστές σημάτων. Ο handle_io_start ενημερώνει το global flag io_start σε 1 και ενεργοποιείται από το σήμα SIGUSR1 και ο handle_io_finish ενημερώνει το global flag io_finish σε 1 και ενεργοποιείται από το σήμα SIGUSR2. Έπειτα τροποποιήσαμε κατάλληλα τον δρομολογητή. Ο δρομολογητής παίρνει κάθε φορά την επόμενη προς εκτέλεση διεργασία, αν μετά την εξαγωγή από την ουρά η λίστα μείνει κενή και υπάρχει διεργασία που περιμένει I/O ο δρομολογητής περιμένει μέχρι να ολοκληρωθεί αυτό. Αν οι I/O εντολές δεν έχουν τελειώσει ή η τρέχουσα διεργασία δεν περιμένει αυτές, τότε η διεργασία εκτελείται κανονικά από την αρχή. Αν κατά την διάρκεια της εκτέλεσης η διεργασία ζητήσει να εκτελέσει I/O εντολές ενεργοποιείται το flag io_start από τον χειριστή του σήματος SIGUSR1, ο δείκτης process_io δείχνει πλέον στην τρέχουσα διεργασία, ενημερώνεται η κατάσταση της διεργασίας πως περιμένει I/O εντολές, τοποθετείται πίσω στην ουρά, το io_start επαναφέρεται σε 0 και ο δρομολογητής συνεχίζει με την επόμενη διεργασία. Αλλιώς, η εκτέλεση της διεργασίας συνεχίζει κανονικά μέχρι να τερματίσει. Όταν τελειώσουν οι εντολές I/O (ενεργοποιείται το flag io_finish από τον χειριστή του σήματος SIGUSR2) και η διεργασία που τις περιμένει επιλεγεί από τον δρομολογητή, η διεργασία συνεχίζει στην CPU από εκεί που τελείωσε στέλνοντας της το σήμα SIGCONT, επαναφέρεται το io_finish σε 0 και εκτελείται μέχρι να τελειώσει.

Παρατηρήσεις

- Η βοηθητική συνάρτηση get_wtime() για την χρονομέτρηση του δρομολογητή και των διεργασιών είναι η ίδια με αυτή που χρησιμοποιείται από τον δοθέν κώδικα στο αρχείο integral_mc_seq.c στο Ερώτημα 2 τις πρώτης εργαστηριακής άσκησης. Χρησιμοποιήθηκε χάριν ευκολίας.
- Το αρχείο run.sh τρέχει κανονικά. Στην εκτέλεση της εντολής *./scheduler_io RR 1000* mixed.txt και οποιασδήποτε εκτέλεσης του δρομολογητή με I/O και πολιτική RR εμφανίζεται μήνυμα το οποίο λέει πως η πολιτική αυτή δεν είναι διαθέσιμη.

Παραδείγματα Εκτέλεσης

Παράδειγμα εκτέλεσης FCFS:
./scheduler FCFS reverse.txt
Policy: FCFS

Name: ../work/work7, PID: 41877, State: RUNNING
process 41877 begins
process 41877 ends

Name: ../work/work7, PID: 41877, State: EXITED
Time since entry: 4.58 sec

Name: ../work/work6, PID: 41905, State: RUNNING
process 41905 begins
process 41905 ends

Name: ../work/work6, PID: 41905, State: EXITED
Time since entry: 8.55 sec

Name: ../work/work5, PID: 41922, State: RUNNING
process 41922 begins
process 41922 ends

Name: ../work/work5, PID: 41922, State: EXITED
Time since entry: 11.85 sec

Name: ../work/work4, PID: 41925, State: RUNNING
process 41925 begins
process 41925 ends

Name: ../work/work4, PID: 41925, State: EXITED
Time since entry: 14.50 sec

Name: ../work/work3, PID: 41950, State: RUNNING
process 41950 begins
process 41950 ends

Name: ../work/work3, PID: 41950, State: EXITED
Time since entry: 16.46 sec

Name: ../work/work2, PID: 41953, State: RUNNING
process 41953 begins
process 41953 ends

Name: ../work/work2, PID: 41953, State: EXITED
Time since entry: 17.80 sec

Name: ../work/work1, PID: 41970, State: RUNNING
process 41970 begins
process 41970 ends

Name: ../work/work1, PID: 41970, State: EXITED
Time since entry: 18.47 sec

Total time was 18.47 sec

Παράδειγμα εκτέλεσης RR:

./scheduler RR 1000 reverse.txt
Policy: RR
Quantum: 1000 msec

Name: ../work/work7, PID: 42048, State: RUNNING
process 42048 begins

Name: ../work/work7, PID: 42048, State: STOPPED

Name: ../work/work6, PID: 42054, State: RUNNING
process 42054 begins

Name: ../work/work6, PID: 42054, State: STOPPED

Name: ../work/work5, PID: 42062, State: RUNNING
process 42062 begins

Name: ../work/work5, PID: 42062, State: STOPPED

Name: ../work/work4, PID: 42082, State: RUNNING
process 42082 begins

Name: ../work/work4, PID: 42082, State: STOPPED

Name: ../work/work3, PID: 42092, State: RUNNING
process 42092 begins

Name: ../work/work3, PID: 42092, State: STOPPED

Name: ../work/work2, PID: 42093, State: RUNNING
process 42093 begins

Name: ../work/work2, PID: 42093, State: STOPPED

Name: ../work/work1, PID: 42094, State: RUNNING
process 42094 begins
process 42094 ends

Name: ../work/work1, PID: 42094, State: EXITED
Time since entry: 6.66 sec

Name: ../work/work7, PID: 42048, State: RUNNING

Name: ../work/work7, PID: 42048, State: STOPPED

Name: ../work/work6, PID: 42054, State: RUNNING

Name: ../work/work6, PID: 42054, State: STOPPED

Name: ../work/work5, PID: 42062, State: RUNNING

Name: ../work/work5, PID: 42062, State: STOPPED

Name: ../work/work4, PID: 42082, State: RUNNING

Name: ../work/work4, PID: 42082, State: STOPPED

Name: ../work/work3, PID: 42092, State: RUNNING
process 42092 ends

Name: ../work/work3, PID: 42092, State: EXITED
Time since entry: 11.62 sec

Name: ../work/work2, PID: 42093, State: RUNNING
process 42093 ends

Name: ../work/work2, PID: 42093, State: EXITED
Time since entry: 11.94 sec

Name: ../work/work7, PID: 42048, State: RUNNING

Name: ../work/work7, PID: 42048, State: STOPPED

Name: ../work/work6, PID: 42054, State: RUNNING

Name: ../work/work6, PID: 42054, State: STOPPED

Name: ../work/work5, PID: 42062, State: RUNNING

Name: ../work/work5, PID: 42062, State: STOPPED

Name: ../work/work4, PID: 42082, State: RUNNING
process 42082 ends

Name: ../work/work4, PID: 42082, State: EXITED
Time since entry: 15.55 sec

Name: ../work/work7, PID: 42048, State: RUNNING

Name: ../work/work7, PID: 42048, State: STOPPED

Name: ../work/work6, PID: 42054, State: RUNNING
process 42054 ends

Name: ../work/work6, PID: 42054, State: EXITED
Time since entry: 17.51 sec

Name: ../work/work5, PID: 42062, State: RUNNING
process 42062 ends

Name: ../work/work5, PID: 42062, State: EXITED
Time since entry: 17.83 sec

Name: ../work/work7, PID: 42048, State: RUNNING
process 42048 ends

Name: ../work/work7, PID: 42048, State: EXITED
Time since entry: 18.41 sec

Total time was 18.41 sec

Παράδειγμα εκτέλεσης FCFS με εντολές I/O:

./scheduler_io FCFS mixed.txt
Policy: FCFS

Name: ../work/work5x2_io, PID: 49545, State: RUNNING
process 49545 begins
process 49545 starts io

Name: ../work/work5x2_io, PID: 49545, State: WAITING IO

Name: ../work/work7, PID: 49586, State: RUNNING
process 49586 begins
process 49586 ends

Name: ../work/work7, PID: 49586, State: EXITED
Time since entry: 7.76 sec

Name: ../work/work6, PID: 49602, State: RUNNING
process 49602 begins
process 49602 ends

Name: ../work/work6, PID: 49602, State: EXITED
Time since entry: 11.62 sec

Name: ../work/work5x2_io, PID: 49545, State: RUNNING
process 49545 completed io
process 49545 ends

Name: ../work/work5x2_io, PID: 49545, State: EXITED
Time since entry: 14.89 sec

Total time was 14.89 sec