Τεκμηρίωση Εργασίας 1 — Λειτουργικά Συστήματα

Εγκλαντίνο Μπρέγκασι AM:1115202000147

1. Εισαγωγή

Η παρούσα εργασία υλοποιεί ένα πολυδιεργασιακό σύστημα ανταλλαγής μηνυμάτων χρησιμοποιώντας τους μηχανισμούς System V Shared Memory και System V Semaphores. Πολλές ανεξάρτητες διεργασίες (`test`) μπορούν να δημιουργήσουν ή να συμμετάσχουν σε διαλόγους και να ανταλλάξουν μηνύματα σε πραγματικό χρόνο. Η επικοινωνία βασίζεται σε κοινόχρηστη μνήμη, ενώ ο συγχρονισμός εξασφαλίζεται με σημαφόρους, ώστε η πρόσβαση στα κοινά δεδομένα να γίνεται με ασφάλεια και χωρίς φαινόμενα race conditions.

Κεντρικός στόχος ήταν η κατασκευή μηχανισμού τύπου *multi-producer / multi-consumer*, με blocking λήψη μηνυμάτων, ασφαλή διαχείριση ουράς (ring buffer) και ομαλή τερματισμό των διαλόγων.

2. Αρχιτεκτονική Συστήματος

2.1 Shared Memory

Η κοινόχρηστη μνήμη χρησιμοποιείται ως ο κεντρικός χώρος δεδομένων. Μέσω των κλήσεων `shmget` και `shmat`, όλες οι διεργασίες αποκτούν πρόσβαση σε κοινό segment, το οποίο περιλαμβάνει:

- τον πίνακα διαλόγων (`dialogs[]`),
- τον κυκλικό buffer μηνυμάτων (`msg_queue[]`),
- τους δείκτες `head`, `tail` και `msg_count`.

Η επιλογή της shared memory προσφέρει εξαιρετική ταχύτητα, καθώς δεν απαιτείται αντιγραφή δεδομένων μεταξύ διεργασιών. Σε αντίθεση με pipes ή sockets, η πρόσβαση είναι άμεση. Ωστόσο, επειδή η κοινόχρηστη μνήμη δεν διαθέτει ενσωματωμένη προστασία, απαιτείται μηχανισμός συγχρονισμού.

2.2 Semaphores

Χρησιμοποιούνται τρεις σημαφόροι System V:

- **spaces**: μετρά τον διαθέσιμο χώρο στην ουρά μηνυμάτων.
- **items**: μετρά τον συνολικο αριθμό εκκρεμών αναγνώσεων.

- **mutex**: binary semaphore για προστασία της shared memory.

Η ροή συγχρονισμού βασίζεται στο κλαΑσικό μοντέλο producer–consumer:

- **Αποστολή (producer)**:
  1. `P(spaces)`
  2. `P(mutex)`
  3. εγγραφή μηνύματος
  4. `V(mutex)`
  5. `V(items)`
- **Λήψη (consumer)**:
  1. `P(items)`
  2. `P(mutex)`
  3. ανάγνωση / επισήμανση μηνύματος
  4. `V(mutex)`
  5. `V(spaces)`

Στο send (enqueue), αφού γραφτεί το μήνυμα, γίνεται V(items, participant count) ώστε να δημιουργηθεί μία "μονάδα item" ανά παραλήπτη. Στο recv, κάθε επιτυχής ανάγνωση καταναλώνει P(items) (μία εκκρεμή παράδοση). Με αυτόν τον τρόπο αποφεύγονται overflows, underflows και race conditions.

3. Διαχείριση Διαλόγων

Κάθε διάλογος αναπαρίσταται από μία δομή `DialogEntry`. Αποθηκεύει το `dialog_id`, τον αριθμό συμμετεχόντων και τα PID των διεργασιών. Κάθε διεργασία αναγνωρίζεται από το λειτουργικό μέσω PID, το οποίο το σύστημα χρησιμοποιεί για να ελέγχει την συμμετοχή σε έναν διάλογο.

3.1 Create

Με την εντολή `create X`, δημιουργείται μία νέα εγγραφή στον πίνακα `dialogs[]`.

3.2 Join

Η εντολή `join X` προσθέτει το PID της διεργασίας στον διάλογο. Η διεργασία αποκτά ένα μοναδικό "reader slot", το οποίο χρησιμοποιείται αργότερα για τον μηχανισμό ανάγνωσης μηνυμάτων.

3.3 Leave

Η εντολή `leave` αφαιρεί το PID από το `participants[]`. Αν ένας διάλογος αδειάσει, το slot μηδενίζεται.

4. Αποστολή Μηνυμάτων

Κάθε μήνυμα αποθηκεύεται στον κυκλικό buffer και περιλαμβάνει:

- κείμενο,
- αποστολέα,
- το `dialog_id`,
- snapshot του αριθμού συμμετεχόντων κατά την αποστολή,
- bitmask (`read_mask`) που υποδεικνύει ποιοι έχουν διαβάσει,
- μετρητή `readers_left`.

Το snapshot εξασφαλίζει ότι όσοι κάνουν join μετά την αποστολή δεν είναι υποχρεωμένοι να λάβουν παλαιότερα μηνύματα.

5. Λήψη Μηνυμάτων

Η λήψη (`recv X`) υλοποιείται ως blocking operation. Η διεργασία περιμένει μέχρι να γίνει διαθέσιμο μήνυμα που δεν έχει διαβάσει ακόμη. Το recv μπλοκάρει αρχικά στο P(items). Επειδή το items είναι global (μετράει deliveries συνολικά) και όχι ανά διάλογο/reader, μπορεί να ξυπνήσει μια διεργασία αλλά να μην υπάρχει μήνυμα που να την αφορά. Σε αυτή την περίπτωση ο κώδικας κάνει V(items) (επιστρέφει το token) και μικρό usleep(10ms) πριν επαναλάβει, ώστε να αποφεύγεται συνεχές spinning.

Η διαδικασία περιλαμβάνει:

- αναμονή μέσω `P(items)`,
- αποκλειστική πρόσβαση μέσω `P(mutex)`,
- σάρωση της ουράς για κατάλληλο μήνυμα,
- ενημέρωση `read_mask` και `readers_left`,
- διαγραφή μηνύματος όταν όλοι οι συμμετέχοντες το έχουν διαβάσει.

Όταν ένα μήνυμα διαβαστεί από όλους (readers left==0), γίνεται garbage collection από το head (gc head). Η gchead() μπορεί να ελευθερώσει πάνω από ένα μηνύματα συνεχόμενα. Για

κάθε μήνυμα που ελευθερώνεται αυξάνεται το spaces κατά 1 (V(spaces, freed)), ώστε να διατηρείται σωστός έλεγχος χωρητικότητας του buffer. Με τον τρόπο αυτό επιτυγχάνεται ακρίβεια: κάθε διεργασία διαβάζει κάθε μήνυμα ακριβώς μία φορά. Η χρήση `blocking` λήψης αποτρέπει το `busy waiting` και μειώνει την άσκοπη κατανάλωση `CPU`.

6. Μήνυμα TERMINATE

Το μήνυμα "`TERMINATE`" λειτουργεί ως συμφωνημένο σήμα τερματισμού. Όταν μία διεργασία το λάβει:

- αποχωρεί από τον διάλογο,
- κάνει `shmdt()` από το shared segment,
- τερματίζει την εκτέλεσή της.

Ο διάλογος διαλύεται αυτόματα όταν όλοι οι συμμετέχοντες έχουν αποχωρήσει.

7. Οδηγίες Εκτέλεσης

Η μεταγλώττιση της εφαρμογής πραγματοποιείται μέσω Makefile. Η εκτέλεση γίνεται σε περιβάλλον Linux,ανοίγοντας πολλαπλά τερματικά, όπου κάθε τερματικό αντιστοιχεί σε μια ανεξάρτητη διεργασία.

Ενδεικτικές εντολές χρήσης:

- create X: δημιουργεί διαλόγου
- join X: είσοδος σε διάλογο
- send X <μήνυμα>: αποστολή μηνύματος
- recv X: blocking λήψη μηνύματος
- terminate X: αποστολή σήματος τερματισμού
- quit: έξοδος διεργασίας

8. Έλεγχος και Δοκιμές

Για την επιβεβαίωση της ορθής λειτουργίας χρησιμοποιήθηκαν τα test files:

- `config_3_100.txt`
- `config_3_1000.txt`
- `config_10_10000.txt`

Τα σενάρια αυτά περιγράφουν ακολουθίες join,send και terminate εντολών για πολλαπλές διεργασίες.

Η εκτέλεση τους πραγματοποιήθηκε μεσω της διεπαφής εντολών `CLI`,σε πολλαπλά τερματικά αναπαράγοντας τα αντίστοιχα χρονικά γεγονότα.

Οι δοκιμές ολοκληρώθηκαν χωρίς deadlocks, race conditions ή απώλεια μηνυμάτων.

9. Συμπεράσματα

Η εργασία υλοποίησε ένα ολοκληρωμένο σύστημα IPC με shared memory και semaphores, χρησιμοποιώντας βασικές αρχές των Λειτουργικών Συστημάτων. Ο συνδυασμός κυκλικής ουράς, σημαφόρων, bitmask και blocking receive παράγει μία αξιόπιστη και ταχεία υλοποίηση, κατάλληλη για πολυδιεργασιακά περιβάλλοντα.

Το σύστημα απέδειξε σταθερότητα και σωστή συμπεριφορά κάτω από υψηλό φόρτο, ολοκληρώνοντας όλα τα διαθέσιμα tests επιτυχώς.