Ορλάνδος Αλέξιος
sdi2300149

Η εργασία χωρίζεται σε **τρία** κύρια κομμάτια:

1. **chat.c** :
- structs
- Συναρτήσεις Δημιουργίας/Καταστροφής/Διαχείρισης Shared Memory / Semaphore
2. **conversation.c :**
- Συναρτήσεις Αλληλεπιδρασης Process/User και Shared Memory
3. **main_conv.c** :
- Υλοποίηση thread ανάγνωσης μηνυμάτων
- Ανάγνωση εισόδου
- Χρήση συναρτήσεων που προηγουμένως ορίστηκαν για υλοποίηση συνολικου συστήματος

chat.c : Διαχείρισης Shared Memory / Semaphore

1. **structs** :
   Το βασικό struct είναι το **struct SharedMemory** (το οποίο θα ορίσει την μορφή της πραγματικής shared memory). Περιέχει αποθηκευμένες τις συζητησεις (struct Conversation) σε πίνακα conversations[MAX_CONVERSATIONS] και το numConversation.

   Το κάθε **struct Conversation** περιέχει conversation_id, αποθηκευμένα σε πίνακα μηνύματα (struct Message) σε πίνακα messages[MAX_MESSAGES], συμμετέχοντες (struct Participant) σε πίνακα participants[MAX_PARTICIPANTS], καθώς και τα numMessages,
   numParticipants.

   Το κάθε **struct Message** περιέχει message_id , το participant_id(pid) του process που το έστειλε, ένα string, και έναν πίνακα με το αν ο κάθε participant έχει διαβάσει το μήνυμα (hasBeenRead(MAX_PARTICIPANTS)).

   Ο κάθε **struct Participant** περιέχει ένα participant_id(pid).

2. **Δημιουργία/Καταστροφή Shared Memory/Semaphore**
   1) int SetUpSemaphore(semaphore_id) : Δημιουργεί (ή ανοίγει ένα υπάρχον) semaphore για mutual exclusivity. Αν το semaphore δεν υπήρχε του δίνει αρχική τιμή 1. Επιστρέφει 0 σε επιτυχία και 1 σε αποτυχία.

2) int SetUpSharedMemory(sharedMemory_id, sharedMemory_ptr) : Δημιουργεί ή ανοίγει το τμήμα shared memory για αποθήκευση όλων των συνομιλιών. Αν το semaphore δεν υπήρχε του δίνει αρχική τιμή numMessages = 0. Επιστρέφει 0 σε επιτυχία και 1 σε αποτυχία.
3) int CleanUp(semaphore_id, sharedMemory_id, sharedMemory_ptr) : Αν το process που το εκτελεί είναι το τελευταίο ενεργό, κάνει detach από και ελευθερώνει την κοινή μνήμη και το semaphore και επιστρέφει 1. Αλλιώς, απλός κάνει detach από την κοινή μνήμη και επιστρέφει 0. Αν του δοθεί αρνητικό semaphore id, δεν ασχολειται με το semaphore.

3. **Διαχείριση Semaphore**
   - volatile global fatal_error : Σηματοδοτεί ότι υπήρξε λάθος στο lock() ή unlock() και άρα το semaphore θεωρείται χαμένο.
   4) int lock(semaphore_id) : Αυξάνει το semaphore, χρησιμοποιείται στην αρχή ενός critical section. Επιστρέφει 0 σε επιτυχία και 1 σε αποτυχία. Σε αποτυχία, θέτει την volatile global fatal_error σε 1.
   5) int unlock(semaphore_id) : Μειώνει το semaphore (εφόσων είναι >=1), χρησιμοποιείται στο τέλος ενός critical section. Επιστρέφει 0 σε επιτυχία και 1 σε αποτυχία. Σε αποτυχία, θέτει την volatile global fatal_error σε 1.

conversation.c : Συναρτήσεις Αλληλεπιδρασης Process/User και Shared Memory

1. **Συναρτήσεις Αναζήτησης**
   1) int findParticipantIndex(conversation_pointer) : Επιστρέφει το index του παρόν struct Participant στο conversation_pointer->participants[MAX_PARTICIPANTS] συγκρίνοντας το getpid() με το participant_id. Σε αποτυχία επιστρέφει -1.
   2) int findConversationIndex(conversation_id, sharedMemory_pointer) : Επιστρέφει το index του struct conversation με το δοσμένο conversation_id, αναζητώντας στο δοσμένο struct SharedMemory. Σε αποτυχία επιστρέφει -1.

2. **Συναρτήσεις Συμμετοχής στην Συζήτηση**
   1) int joinConversation(conversation_id, semaphore_id, sharedMemory_pointer) : Εισάγει τη διεργασία σε συζήτηση με το δοσμένο conversation_id (ως struct Particpant με participant_id = getpid). Αν η συζήτηση δεν υπάρχει, δημιουργείται νέα (εφόσον υπάρχει διαθέσιμος χώρος). Η διεργασία προστίθεται στον πίνακα συμμετεχόντων της συζήτησης. Χρησιμοποιεί lock() και unlock() για mutual

exclusion κατα την διαρκεια της. Επιστρέφει 0 σε επιτυχία, 1 σε αποτυχία, -1 σε αποτυχία λόγω unresponsive semaphore.
2) int leaveConversation(conversation_id, semaphore_id, sharedMemory_pointer) :
Αφαιρεί τη διεργασία από τη συζήτηση με το δοσμένο conversation_id. Ενημερώνει τον πίνακα συμμετεχόντων και μειώνει το πλήθος τους. Αν το struct conversation μείνει χωρίς ενεργούς struct participants, διαγράφεται από τη shared memory και όλα τα επόμενα struct conversation γίνονται shift left στον πίνακα μηνυμάτων. Χρησιμοποιεί lock() και unlock() για mutual exclusion κατα την διαρκεια της. Επιστρέφει 0 σε επιτυχία, 1 σε αποτυχία, -1 σε αποτυχία λόγω unresponsive semaphore.
3) int sendMessage(conversation_id, semaphore_id, sharedMemory_pointer, string): Στέλνει νέο μήνυμα στη συζήτηση με το δοσμένο id. Το μήνυμα αποθηκεύεται στο struct conversation με το δοσμένο conversation_id και σημειώνεται ως μη αναγνωσμένο (hasBeenRead() == 0) για όλους τους struct participants. Το pid της διεργασίας αποθηκεύεται ως senderId. Χρησιμοποιεί lock() και unlock() για mutual exclusion κατα την διαρκεια της. Επιστρέφει 0 σε επιτυχία, 1 σε αποτυχία, -1 σε αποτυχία λόγω unresponsive semaphore.

3. **Συναρτήσεις Διαγραφής Μηνυμάτων**
1) int messageRemovalCheck(conversation_pointer, msg_index) :
Εάν το μήνυμα στο δοσμένο msg_index (conversation_pointer->messages[msg_index]) έχει διαβαστεί από όλους τους ενεργούς participants (hasBeenRead()) , δηλαδή πρέπει να αφαιρείται, επιστρέφει 1, αλλιώς επιστρέφει 0.
2) void messageRemoval(conversation_pointer, msg_index) :
Αφαιρεί το μήνυμα στο δοσμένο index από τη συζήτηση. Όλα τα επόμενα μηνύματα γίνονται shift left στον πίνακα μηνυμάτων. Μειώνει το numMessages.

main_conv.c : Υλοποίηση Thread Ανάγνωσης Μηνυμάτων / Ανάγνωση εισόδου

1. **Global Μεταβλητές**

- semaphore_id_global, sharedMemory_id_global, sharedMemory_pointer_global, conversation_id_global : Αυτές οι τιμές, αφού οριστούν, παραμένουν σταθερές και σωστές κατα την διαρκεια της εκτέλεσης. Για αυτό τον λόγο και για καλύτερη πρόσβαση τους από το thread, δηλώνονται global.

- volatile int termination_order : Με αρχική τιμή 0, γίνεται 1 όταν διαβαστεί το “TERMINATE”, ή έχει εμφανιστεί κάποιο σφάλμα και πρέπει να τερματιστεί το πρόγραμμα.
-

2. Συνάρτηση για Thread

- void sendMessages(void* arg) :
  Η συνάρτηση που θα δοθεί στο δεύτερο thread (που θα εκτελείται παράλληλα με την main). Ελέγχει περιοδικά τη shared memory για νέα μηνύματα. Για κάθε εκτέλεση καλεί τις lock() και unlock() για mutual exclusion. Χρησιμοποιεί τις findParticipantIndex και findConversationIndex για να αποκτήσει τα σωστά index. Προσπελάζει τα struct messages του παρών conversation, και για κάθε νέο μήνυμα (hasBeenRead[participant_index] = 0) εκτελεί τα ακόλουθα:
  1) Εκτυπώνει το μήνυμα στο terminal.
  2) Σημειώνει το μήνυμα ως διαβασμένο.
  3) Ελέγχει αν το μήνυμα έχει διαβαστεί από όλους και αν ναι το αφαιρεί.

  Εάν το μήνυμα είναι “TERMINATE”:
  1) Το volatile int termination_order γίνεται ίσο με 1 (η main πρόκειται να τερματισει).
  2) Η εκτέλεση του loop τερματίζεται.
  3) Η εκτέλεση της sendMessages() τερματίζεται.

  - int main(argc, argv) :
  Το σημείο εκκίνησης όλου του project. Εκτελεί τα ακόλουθα:
  1) Έλεγχος ορθότητας ορισμάτων.
  2) Δημιουργία ή σύνδεση σε shared memory (setUpSharedMemory).
  3) Δημιουργία ή σύνδεση σε semaphore (setUpSemaphore).
  4) Είσοδος ή δημιουργία συζήτησης με συγκεκριμένο id (joinConversation).
  5) Εκκίνηση του thread ανάγνωσης μηνυμάτων (pthread_create).
  6) Είσοδος σε loop. Το termination_type ορίζει την διαδικασία termination που θα εκτελεστεί μετά το τέλος του loop.
  7) Ανάγνωση εισόδου από τον χρήστη και αποστολή μηνυμάτων (fgets). Η είσοδος του χρήστη διαγράφεται κάθε φορά από το terminal γιατί το μήνυμα του εκτυπώνεται από το thread αφού σταλθεί στην κοινή μνήμη.

8) Έλεγχος για break και ορισμό της termination_type μέσω της volatile int termination_order ή volatile int fatal_error.

9) Αποστολή μηνύματος(sendMessage). Έλεγχος για break και ορισμό της termination_type λόγο error της sendMessage.

10) Τέλος loop.

11) Τερματισμος αναλογα με το termination_type. Σε περίπτωση ομαλού τερματισμού, καλούμε:
    - pthread_join()
    - leaveConversation()
    - cleanUp()
    - return 0

    Σε περίπτωση τερματισμού λόγω fatal_error:
    - termination_order = 1 (Για να τερματίσει το thread)
    - pthread_join()
    - cleanUp()
    - return 1