Κωνσταντίνος Σκορδούλης

ΑΜ : 1115 2016 00155

Η εργασία έχει υλοποιηθεί σε C++ (and compiled with g++), βέβαια γίνεται περισσότερη χρήση συναρτήσεων/βιβλιοθηκών της C . Για την shared memory, χρησιμοποιήθηκε η αντίστοιχη **System V Shared Memory** (αν δεν κάνω λάθος) και όσον αφορά semaphores => **POSIX semaphores**.

Από δομές δεδομένων, χρησιμοποίησα μια **απλή συνδεδεμένη λίστα** (single linked list **τοπική** για τον **port-master**) για να διατηρώ τη σειρά (more on this later). Όσον αφορά τη κοινή μνήμη, χρησιμοποίησα διάφορα δικά μου structs όπως **SHM, Application_form, Position_Port** . Πέρα από αυτά χρησιμοποίησα static sized πίνακες, εντός των structs, για αποθήκευση πληροφοριών (strings κλπ) .

Source Files: **myport.cpp(root), vessel.cpp, port-master.cpp, monitor.cpp, PriorityList.**

Όσον αφορά τους **semaphores**, έχω δώσει εκτενής περιγραφή στο **Mystructs.hpp** αρχείο.

Για τις **προτιμήσεις (upgrades**), των πλοίων. 1) Τσεκάρονται τελευταίες, ο καταναλωτής δεν θέλει να πληρώσει παραπάνω. 2) Ακριβής προτίμηση θέσης, πχ αν upgrade = Large, ψάχνω θέση μόνο για Large και όχι για Medium.

Γενικά τα βασικά συστατικά της υλοποίησης μου είναι τα εξής :

1. ***Queue*(semaphore):** Η Queue είναι μια **ουρά αιτημάτων**(τόσο για είσοδο, όσο και για έξοδο). Δηλαδή μας ενδιαφέρουν τα αιτήματα και όχι τα vessels (αποκλειστικά). Κατά αυτό τον τρόπο διατηρούμε (εν μέρει) το **FCFS**. Αυτός ο σημαφόρος αρχικοποιείται με 0 , έτσι ώστε κάθε αίτημα να μπλοκάρεται , μέχρι ο port-master να είναι διαθέσιμος να το εξυπηρετήσει.
2. ***Application_Form*(Struct in Shared Memory):** Αφού ξυπνήσει το **vessel**, πρέπει κάπως να ενημερώσει τον **port-master** για το αίτημα του. Για αυτό έχουμε το application form, κατά το οποίο το vessel γράφει τα στοιχεία του στη μνήμη, και μετά τα διαβάζει ο port-master. Αφού γράψει το αίτημα του , μπλοκάρει στο semaphore **in_evaluation**.
3. ***In_evaluation(*Semaphore):** Εδώ μπλοκάρει ο **vessel** αφού γράψει το αίτημα του, και περιμένει να αποφασίσει ο port-master, αλλάζοντας τη μεταβλητή **bool approved** μέσα στο **Application_Form** .
4. ***Full(Semaphore):*** Εδώ μπλοκάρει ο **port-master**, αφού ξυπνήσει κάποιο αίτημα/vessel και περιμένει μέχρι να γράψει, το **vessel**, το αίτημα του στο **application_form** , αλλάζοντας τη μεταβλητή **bool approved** μέσα στο **Application_Form.**
5. ***Maneuver_pm(*Semaphore):** Εδώ μπλοκάρει ο **port-master**, αφού **εγκρίνει** κάποιο αίτημα/vessel και περιμένει μέχρι να τελειώσει την κίνηση του vessel στο λιμάνι. Κατά αυτό τον τρόπο, εγγυούμαστε ότι δεν κινείται κανένα άλλο vessel(ένα vessel γράφει στο **application_form** , και γενικά μόνο ένα εξυπηρετείται τη φορά). Ξυπνάει ο port-master , μόνο όταν του δώσει σήμα το αντίστοιχο vessel.

6. ***PriorityQueues(*** **6 Semaphores):** Ένα **αίτημα εισόδου** , μπορεί να **αποτύχει** μόνο αν δεν υπάρχει διαθέσιμη θέση για να τον καλύψει (ούτε για το size του, ούτε για μεγαλύτερο(upgraded)) (όσον αφορά τη μοναδικότητα στη κίνηση, ο παραπάνω semaphore την καλύπτει. Άρα όταν **αποτύχει** η αίτηση μπαίνει σε ουρά προτεραιότητα/**semaphores**, έτσι ώστε με το που αδειάσει κάποια θέση, να εξυπηρετήσουμε πρώτα αυτούς και να διατηρήσουμε **FCFS & Available.**Κάθε ουρά είναι αρχικοποιημένη στο **0** , για να **μπλοκάρει**. Για αυτό το λόγο έχουμε ***6 PriorityQueues:*** (3 normal + 3 upgrade)
    a. **Pr_Small:** Για τα **small** καράβια, που **δεν έχουν upgrade**.
    b. **Pr_Medium:** Για τα **medium** καράβια, που **δεν έχουν upgrade**.
    c. **Pr_Large:** Για τα **large** καράβια, που **δεν έχουν upgrade**.
    d. **Pr_uSM:** Για τα **small** καράβια, που έχουν προτίμηση **upgrade** για **Medium** θέση.
    e. **Pr_uSL:** Για τα **small** καράβια, που έχουν προτίμηση **upgrade** για **Large** θέση.
    f. **Pr_uML:** Για τα **medium** καράβια, που έχουν προτίμηση **upgrade** για **Large** θέση.

    *** Σημείωση, τα πλοία που θέλουν να βγουν, πάντα εγκρίνεται η αίτηση τους , όταν έρθει η σειρά τους(γιατί μόνη προϋπόθεση είναι να μην κινείται κανένας στο λιμάνι ➔ no priority queue).

7. ***PriorityList***(Single Linked List): Είναι **τοπικά αποθηκευμένη** στον **port-master**. Αν διασκορπίσουμε τις αποτυχημένες αιτήσεις στα priority queues, πως κρατάμε την σειρά εξυπηρέτησης των **priority**? Με το ***PriorityList***. Κάθε φορά που δεν εγκρίνεται το αίτημα, δημιουργείται ένας κόμβος της λίστας , ο οποίος μπαίνει πάντα στο τέλος (ουρά FIFO με λίστα). Αυτός ο κόμβος περιέχει ένα **string** ,το οποίο δείχνει ποιον **PriorityQueue** να ξυπνήσουμε (εφόσον υπάρχει διαθέσιμος χώρος για να τον ικανοποιήσουμε, αλλιώς μεταβαίνουμε στον επόμενο κόμβο: **from head ➔tail**). Επομένως, πετυχαίνουμε πραγματικά **FCFS & Available** .

    ** Σημείωση: Κάθε φορά ο **port-master**, τσεκάρει πρώτα αν μπορεί να εξυπηρετήσει τα **priority αιτήματα** και μετά κοιτάζει το **semaphore queue**.

Διάφορες Σημειώσεις:

1. Θεωρώ **χρόνο αναμονής** , τόσο ,την "ανταπόκριση" του port-master, για το αίτημα **εισόδου**, όσο και της **εξόδου**.
2. **Κοστολογώ** την ώρα, από τη στιγμή που θα πάρει έγκριση και θα μπει, μέχρι να πάρει έγκριση και να βγει από το λιμάνι. Δηλαδή κοστολογώ και τα 2 maneuvers του vessel, καθώς και τη διάρκεια ελλιμενισμού/sleep().
3. Το πλοίο ξύπναει και βλέπει το κόστος του , στα **k/2** του χρόνου sleep().
4. ***Shared Memory:*** Αρο **πάνω προς τα κάτω**:

a. ***Struct SHM:*** εδώ βρίσκονται οι περισσότερες **μεταβλητές** και τα απαραίτητα ***offset***.
b. ***Struct Position:*** (Capacity1+Capacity2+Capacity3) *sizeof(struct Position). Εδώ είναι οι συνεχόμενες **θέσεις του λιμανιού**, και τις χειριζόμαστε με τα κατάλληλα offsets. (Public ledger)
c. ***PL_Records(char*):*** **20 * sizeof(char**). Εδώ υπάρχει το **όνομα** του ιστορικού του Public Ledger(άρα και πρόσβαση σε αυτό).
d. ***Log_Name(char*):*** **10 * sizeof(char)**. Εδώ υπάρχει το **όνομα** του Logger(άρα και πρόσβαση σε αυτό).
e. ***Struct Application_Form***: sizeof(Application_Form). Πληροφορίες για αυτό, υπάρχει παραπάνω

5. ***Σημαντικό***, για πρόσβαση στη μνήμη , ακολουθούμε τη ***λύση του προβλήματος readers/writers***. Δηλαδή , 2 semaphores(**mutex**, rw_**mutex**) και 1 counter ➔ **readers_count**. Με αυτό τον τρόπο εξασφαλίζουμε ότι κανένας δεν γράφει, αν κάποιος διαβάζει και αντίστοιχα κανένας δεν διαβάζει αν κάποιος γράφει .