ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Λειτουργικά Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:10

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

1. Σκοπός της άσκησης

- Επικοινωνία διεργασιών με **ουρές μηνυμάτων** (msgget(), msgsnd(), msgrcv(), msgctl()).
- Επικοινωνία διεργασιών με **κοινή μνήμη** (shmget(), shmget(), shmat(), shmdt(), shmctl())
- Επικοινωνία διεργασιών με **απεικόνιση αρχείου στη μνήμη** (open(), mmap(), munma()).
- Ο προσομοιωτής διεργασιών SOSSIM.

2. Παραδοτέα

(A + S) 17+3 ερωτήσεις
(C) 4 ασκήσεις

ΠΡΟΣΟΧΗ: **Το 2015 το FreeBSD αποφάσισε να χρησιμοποιεί και να υποστηρίζει τον μεταγλωττιστή `clang` σε αντίθεση με το linux που χρησιμοποιεί το `gcc`. Σε περίπτωση που εργάζεστε σε FreeBSD μηχάνημα (όπως το zafora) θα πρέπει να αντικαταστήστε τις εντολές `gcc` με τις εντολές `clang` (οι παράμετροι παραμένουν ίδιοι). Σε περίπτωση που εργάζεστε σε Linux μηχάνημα (όπως το pleiades), δε χρειάζεται κάποια αλλαγή.**

3. Επικοινωνία διεργασιών με ουρές μηνυμάτων

Τα σύγχρονα λειτουργικά συστήματα παρέχουν ποικίλες τεχνικές για τη διαδιεργασιακή επικοινωνία. Μια από αυτές είναι η επικοινωνία με ουρές μηνυμάτων. Συγκεκριμένα, μια διεργασία δημιουργεί ή συνδέεται σε ένα γραμματοκιβώτιο *(συνάρτηση `msgget()`)*, και στη συνέχεια στέλνει *(συνάρτηση `msgsnd()`)* και λαμβάνει *(συνάρτηση `msgrcv()`)* μηνύματα από αυτό.

Η συνάρτηση δημιουργίας ή πρόσβασης γραμματοκιβωτίου `msgget()` πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;
_____ **(A1)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση `msgget()` θα αποτύχει: _____ **(A2)**

Η συνάρτηση αποστολής μηνυμάτων `msgsnd()` πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;
_____ **(A3)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση **`msgsnd()`** θα αποτύχει: _____ **(A4)**

Η συνάρτηση λήψης μηνυμάτων **`msgrcv()`** πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;

_____ **(A5)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση **`msgrcv()`** θα αποτύχει: _____ **(A6)**

Πριν την κλήση αποστολής μηνύματος, πρέπει να οριστεί ένα **`struct`** στο οποίο ορίζονται δυο στοιχεία: **(α)** ο τύπος του μηνύματος που ακολουθεί ως long, και **(β)** το κυρίως σώμα του μηνύματος ως char. Για παράδειγμα πρέπει να ορίσετε ένα:

```
struct message {
long mtype;
char mtext[MSGSIZE]; };
```

Αρχικά θα πρέπει να αποκτήσετε πρόσβαση σε ένα γραμματοκιβώτιο με τη συνάρτηση **`msgget()`** . Με τη συνάρτηση αυτή επιστρέφεται ο προσδιοριστής *(resource)* του γραμματοκιβωτίου τον οποίο θα χρησιμοποιήσουμε για να διαβάσουμε ή να γράψουμε μηνύματα.

- Η πρώτη παράμετρος είναι ένας ακέραιος αριθμός *(τύπου key_t)* που χρησιμοποιείται για το γραμματοκιβώτιο *(τον επιλέγουμε εμείς)*.
- Η δεύτερη παράμετρος είναι ένας ακέραιος αριθμός, όπου τοποθετούνται με συμβολικό τρόπο τα δικαιώματα πρόσβασης και επιπρόσθετες επιλογές όπως IPC_CREAT ο οποίος δημιουργεί ένα γραμματοκιβώτιο με το συγκεκριμένο κλειδί ή αν υπάρχει συνδέεται σε αυτό. Συνήθως, δημιουργείται το γραμματοκιβώτιο με τα δικαιώματα 0666.

Η αποστολή μηνύματος γίνεται με τη **`msgsnd()`** η οποία στέλνει ένα μήνυμα στο γραμματοκιβώτιο που ορίζεται στην πρώτη παράμετρο, του οποίου η διεύθυνση δίνεται ως η δεύτερη παράμετρος και έχει μήκος όσο η τρίτη παράμετρος. Η παράμετρος flag είναι 0 τις περισσότερες φορές. Η λήψη μηνύματος γίνεται με τη **`msgrcv()`** με παρόμοιο τρόπο.

Ο χειρισμός του γραμματοκιβωτίου *(ή της ουράς μηνυμάτων)* γίνεται με τη συνάρτηση **`msgctl()`** όπου η πιο διαδεδομένη ενέργεια είναι η IPC_RMID, η οποία καταστρέφει τη συγκεκριμένη ουρά.

(C1) Να γραφεί το πρόγραμμα **c1-server.c** και ακολούθως το **c1-client.c** τα οποία ως διεργασία θα στέλνουν και θα λαμβάνουν μηνύματα μέσω των ουρών του λειτουργικού συστήματος.

Συγκεκριμένα ο **server**:

- Θα ορίζεται το κατάλληλο struct για τις ουρές μηνυμάτων.
- Θα ορίζεται η recv_buffer τύπου struct.

- Θα κάνει κλήση της msgget() για να δημιουργεί ένα γραμματοκιβώτιο. Να προσθέσετε την παράμετρο IPC_NOWAIT ώστε να μη γίνεται block η συνάρτηση λήψης μηνύματος αν δεν υπάρχει μήνυμα και την παράμετρο IPC_CREAT για να δημιουργηθεί το γραμματοκιβώτιο. Επίσης τα δικαιώματα να είναι 0666 *(δηλαδή 0666 | IPC_CREAT.....)*. Επιλέξτε ως κλειδί των αριθμό μητρώου σας.
- Θα ελέγχεται αν έχει δημιουργηθεί το γραμματοκιβώτιο και θα εκτυπώνεται το κατάλληλο μήνυμα *(mailbox created / mailbox not created perror())* και θα εκτυπώνεται ο αριθμός identifier που έχει επιστραφεί.
- Θα μπαίνει σε ένα βρόγχο 3 επαναλήψεων που θα εκτελεί:
    - το sleep(1)
    - τη συνάρτηση λήψης μηνύματος **`msgrcv()`** και θα ελέγχει αν έχει τοποθετηθεί κάποιο μήνυμα στην ουρά. Αν δεν υπάρχει κάποιο μήνυμα *(δηλαδή έχει επιστραφεί -1)* τότε θα εμφανίζει το μήνυμα "`No message,sleeping!`", διαφορετικά "`New Message: %s`" και θα εκτυπώνει το μήνυμα που λήφθηκε. Στη συνάρτηση λήψης μηνύματος μπορείτε να χρησιμοποιήσετε στο μέγεθος μηνύματος το 255, δεδομένου ότι δεν πρόκειται να στέλνετε μεγαλύτερα μηνύματα από αυτό.
- Μετά τις 3 επαναλήψεις θα καταστρέφεται το γραμματοκιβώτιο με την εντολή: msgctl(identifier, IPC_RMID, (struct msqid_ds *) 0)

Ο **client** :

- Θα ορίζεται το κατάλληλο struct για τις ουρές μηνυμάτων.
- Θα ορίζεται η send_buffer τύπου struct.
- Θα συνδέεται στο γραμματοκιβώτιο με τη msgget. Στη δεύτερη παράμετρο θα έχετε το νούμερο 0666.
- Θα τοποθετείται στο send_buffer.mtype μια τιμή long που θα δείχνει τον τύπο του μηνύματος.
- Θα ζητάει από το χρήστη να πληκτρολογήσει μια πρόταση.
- Θα τοποθετείται στο send_buffer.mtext η παραπάνω πρόταση.
- Με τη strlen() θα καταμετράται ο αριθμός bytes του struct σας και θα αναγράφεται στην οθόνη "`I will send XXX number of Bytes`".
- Θα καλείται η msgsnd() η οποία θα στέλνει το send_buffer
- Θα ελέγχεται η τιμή επιστροφής της msgsnd() για να διαπιστωθεί αν στάλθηκε με επιτυχία *(οπότε θα αναγράφει message sent succesfully)* ή όχι *(όταν σταματήσει ο server θα αποτυγχάνει η συνάρτηση)*.
- Η διαδικασία αυτή θα επαναλαμβάνεται 3 φορές.

4. Επικοινωνία διεργασιών με κοινή μνήμη

Ένας διαδεδομένος για τη διαδιεργασιακή επικοινωνία είναι και η κοινή μνήμη. Με παρόμοιο τρόπο όπως και στις ουρές μηνυμάτων, καλείται αρχικά η συνάρτηση **`shmget()`**, η οποία επιστρέφει ένα resource για τη κοινή μνήμη για το συγκεκριμένο κλειδί *(key_t)* η οποία είτε δημιουργείται κατά την κλήση είτε υπάρχει από πριν, δηλαδή έχει δημιουργηθεί από προηγούμενη διεργασία. Οι διεργασίες λοιπόν τοποθετούν δεδομένα στην κοινή περιοχή μνήμης και επικοινωνούν μεταξύ τους. Υπάρχει όμως το πρόβλημα ότι αν οι διεργασίες γράψουν ταυτόχρονα στην ίδια περιοχή μνήμης τότε θα χαθεί μια εγγραφή. Για να το αντιμετωπίσουμε αυτό θα πρέπει να χρησιμοποιήσουμε κάποια τεχνική αμοιβαίου αποκλεισμού όμως είδαμε στη θεωρία.

Η συνάρτηση δημιουργίας ή πρόσβασης κοινής μνήμης shmget()πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας; _____ **(A7)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση shmget() θα αποτύχει: _____ **(A8)**

Αφού καλέσουμε τη shmget()το επόμενο βήμα είναι να προσαρτήσουμε τη συγκεκριμένη περιοχή μνήμης στη διεργασία μας. Αυτό επιτυγχάνεται με την κλήση της συνάρτησης shmat().

Η συνάρτηση προσάρτησης κοινής μνήμης shmat() πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;

_____ **(A9)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση shmat() θα αποτύχει: _____ **(A10)**

Μόλις χρησιμοποιήσουμε τη **`shmat()`** τότε αυτή μας επιστρέφει ένα char *, pointer που δείχνει στην αρχική διεύθυνση του τμήματος. Προκειμένου να μη χάσουμε την αρχική αυτή διεύθυνση, συνήθως αντιγράφουμε αυτή τη διεύθυνση σε ένα άλλο char * pointer τον οποίο θα το χρησιμοποιήσουμε παρακάτω. Δηλαδή, έχουμε ορίσει στην αρχή του προγράμματος:

```
char *shared,*sharedcopy
```

και αφού καλέσουμε τη **`shared=shmat(..)`** και τοποθετηθεί η διεύθυνση που ξεκινάει το shared κομμάτι στο *shared, κάνουμε **`sharedcopy=shared;`** για να αρχίσουμε να τροποποιούμε το sharedcopy. Αν θέλουμε να γράψουμε ή να διαβάσουμε κάτι τότε μπορούμε να χρησιμοποιήσουμε δυο τρόπους:

- είτε με pointer γράφοντας τον χαρακτήρα που θέλουμε άμεσα, ως *sharedcopy=”m” και για την επόμενη θέση *sharedcopy++ κ.ο.κ.

- είτε ως μονοδιάστατο πίνακα γράφοντας τον χαρακτήρα στη θέση sharedcopy[i]=”m”

Σε κάθε περίπτωση στο τέλος πρέπει να τοποθετήσουμε το NULL *(δηλαδή το 0)* γιατί τα strings στη C απαιτούν το \0 για τερματισμό.

Αφού χρησιμοποιήσουμε την κοινή μνήμη, το επόμενο βήμα είναι να αποσπάσουμε τη μνήμη από τη διεργασία. Αυτό επιτυγχάνεται με την κλήση της **`shmdt()`** .

Η συνάρτηση από-προσάρτησης κοινής μνήμης **`shmdt()`** πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;
_____ **(A11)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση **`shmdt()`** θα αποτύχει: _____ **(A12)**

Τέλος, αφού απο-προσαρτήσουμε την κοινή μνήμη απελευθερώνουμε τη μνήμη προς το λειτουργικό σύστημα με τη συνάρτηση **`shmctl()`**. Με παρόμοιο τρόπο με την επικοινωνία με τις ουρές μπορούμε με τη συνάρτηση αυτή να εκτελέσουμε την ενέργεια **IPC_RMID** για να καταστρέψουμε τον πόρο. Σε αυτή την περίπτωση η τελευταία παράμετρος πρέπει να είναι η **`(struct shmid_ds *) 0`**.

(C2) Να γραφεί το πρόγραμμα **c2-server.c** και ακολούθως το **c2-client.c** τα οποία ως διεργασία θα χρησιμοποιούν μια κοινή μνήμη για τη διαδιεργασιακή τους επικοινωνία. Συγκεκριμένα ο server:

- Θα ορίζεται ένα κλειδί με τον ΑΜ σας.
- Θα δημιουργείται μια κοινή περιοχή μνήμης με τη **`shmget()`** με παράμετρο IPC_CREAT | 0666, και μέγεθος τουλάχιστον 50 Bytes.
- Θα εμφανίζεται κατάλληλο μήνυμα επιτυχίας ή αποτυχίας.
- Θα προσαρτάται η κοινή μνήμη στη διεργασία μας.
- Θα γράφει ο server μια αριθμητική τιμή στη διεύθυνση 0 της κοινής μνήμης και το ονοματεπώνυμό σας.
- Θα διαβάζει συνεχώς τη διεύθυνση 0 σε ένα βρόχο while με μια εντολή sleep(1) μέχρι να τροποποιηθεί η τιμή στη διεύθυνση 0.
- Όταν τροποποιηθεί θα εκτυπώνει το μήνυμα που έχει τοποθετηθεί στην κοινή μνήμη.
- Θα απο-προσαρτά την κοινή μνήμη *(εμφανίζοντας κατάλληλο μήνυμα).*
- Θα καταστρέφει την κοινή μνήμη *(εμφανίζοντας κατάλληλο μήνυμα).*

Ο client:

- Θα ορίζεται ένα κλειδί με τον ΑΜ σας.
- Θα συνδέεται στην κοινή περιοχή μνήμης με τη `shmget()` και θα εμφανίζει κατάλληλο μήνυμα
- Θα προσαρτά την κοινή μνήμη στη διεργασία με τη `shmat()` και θα εμφανίζει κατάλληλο μήνυμα. Προσοχή στις παραμέτρους της συνάρτησης. Η πρώτη παράμετρος είναι αυτή που επιστρέφει η `shmget()`.
- Θα ζητάει από το χρήστη να πληκτρολογήσει μια πρόταση.
- Θα τοποθετείται η πρόταση αυτή από τη διεύθυνση μνήμης +1 *(για να μην τροποποιηθεί η διεύθυνση 0).*
- Μόλις γραφεί θα τροποποιείται το περιεχόμενο της διεύθυνσης 0 με το να διαβαστεί και να αυξηθεί κατά 1 *(δηλαδή να τροποποιηθεί το περιεχόμενο της μνήμης που δείχνει η διεύθυνση 0).*

5. Διαδιεργασιακή Επικοινωνία με απεικόνιση αρχείου στη μνήμη

Memory Mapped File

Ένας ακόμη δημοφιλής τρόπος διαδιεργασιακής επικοινωνίας είναι η απεικόνιση αρχείου ή γενικότερα μιας οποιασδήποτε περιοχής που μπορεί να επιτρέπει πρόσβαση επιπέδου μπλοκ στη μνήμη, όπως ένας δίσκος ή μια τμηματοποίηση (partition). Για να λειτουργήσει αυτή η τεχνική, απαιτείται πρώτα να ανοιχτεί η συσκευή ή το αρχείο με `open()` και στη συνέχεια να γίνει κλήση της `mmap()` με τις κατάλληλες παραμέτρους.

Η συνάρτηση απεικόνισης αρχείου στη μνήμη `mmap()` πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;

_____ **(A13)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση `mmap()` θα αποτύχει: _____ **(A14)**

Η συνάρτηση ανοίγματος αρχείου `open()` πόσα ορίσματα λαμβάνει και τι επιστρέφει; Τι επιστρέφει σε περίπτωση αποτυχίας;

_____ **(A15)**

Αναφέρετε τουλάχιστον 2 καταστάσεις που η κλήση στη συνάρτηση `open()` θα αποτύχει: _____ **(A16)**

Με την εντολή `dd` δημιουργήστε στο /tmp ένα αρχείο με όνομα το ΑΜ σας με μέγεθος 30 bytes και αρχικές τιμές 0. Για να γίνει αυτό θα χρησιμοποιήσουμε την παράμετρο

`if=/dev/zero` *(δηλαδή, είσοδος από τη συσκευή που δίνει 0)* και έξοδο **`of=/tmp/AEM`**. Βρείτε τις υπόλοιπες παραμέτρους από τη σελίδα βοήθειας.

Ποια είναι η πλήρης εντολή που χρησιμοποιήσατε: _____ **(A17)**

Επιβεβαιώστε με **`ls -l`** ότι αναγράφεται το σωστό μέγεθος.

(C3) Να γραφεί το πρόγραμμα **c3.c** το οποίο ως διεργασία θα χρησιμοποιεί την απεικόνιση μνήμης. Συγκεκριμένα:

- Θα χρησιμοποιεί την κλήση συστήματος **`open()`** για να ανοίξει το αρχείο**`/tmp/"AEM"`** με παράμετρο εγγραφής/ανάγνωσης **O_RDWR** και **`(mode_t)0600`** προκειμένου να υπάρχουν τα δικαιώματα εγγραφής και ανάγνωσης από διεργασίες του ίδιου ιδιοκτήτη *(μόνο)*.
- Θα γίνεται έλεγχος αν έχει ανοιχτεί το αρχείο, διαφορετικά θα εκτυπώνει κατάλληλο μήνυμα και θα σταματάει η εκτέλεση.
- Η κλήση του συστήματος **`mmap()`** που ακολουθεί επιστρέφει ένα δείκτη σε πίνακα void μεταβλητής. Εμείς θα χρησιμοποιήσουμε χαρακτήρες σε αυτό το αρχείο, οπότε θέλουμε ο δείκτης να είναι δείκτης χαρακτήρων. Να τοποθετήσετε τη δήλωση για το δείκτη mapfile στις μεταβλητές.
- Στη συνέχεια θα καλείται η **`mmap()`** η οποία θα κάνει την αντιστοίχηση ξεκινώντας από τη διεύθυνση 0 *(πρώτη παράμετρος)*, όλο το μέγεθος του αρχείου *(δεύτερη παράμετρος)*, με την τρίτη παράμετρο να δείχνει ότι θα διαβάζονται και θα γράφονται οι σελίδες, με την τέταρτη παράμετρο να δείχνει ότι θα είναι SHARED οι σελίδες αυτές, με την πέμπτη παράμετρο να χρησιμοποιηθεί ο περιγραφέας αρχείου από την open() και την τελευταία παράμετρο που δείχνει την μετατόπιση να είναι 0.
- Θα γίνεται έλεγχος στη συνέχεια αν έχει επιτευχθεί η αντιστοίχηση, διαφορετικά θα εκτυπώνει κατάλληλο μήνυμα και θα σταματάει η εκτέλεση.
- Θα δημιουργείται ένας βρόχος από i=1 έως το μέγεθος του αρχείου:
    - θα εκτυπώνεται ο δείκτης i
    - θα εκτυπώνεται η τρέχουσα τιμή mapfile[i]
    - θα γράφεται η τιμή **`mod((mapfile[i] + i),94)+32`** στη θέση mapfile[i] *{αυτή η έκφραση μας επιστρέφει μια τιμή που ανήκει από 32 έως 128, δηλαδή μια τιμή ενός εκτυπώσιμου ASCII χαρακτήρα}* . Προσέξτε ότι αποκτούμε πρόσβαση στο αρχείο ως πίνακα και μπορούμε απλά να γράφουμε ή να διαβάζουμε με την πρόσβαση στον αντίστοιχο δείκτη.
    - Θα εκτυπώνεται η νέα τιμή mapfile[i]

      δηλαδή:

      "Στη θέση **`i=XX`** διάβασα τιμή **`mapfile[i]`** *(παλαιά τιμή)* και θα την τροποποιήσω σε **`mapfile[i]`** *(νέα τιμή)*"

- Μόλις ολοκληρωθεί η διαδικασία θα καλέσετε την `munmap()` για να απομακρύνετε την απεικόνιση.
- Θα γίνεται έλεγχος και θα εκτυπώνεται κατάλληλο μήνυμα.
- Στο τέλος θα κλείνει ο περιγραφέας αρχείου.

Εκτελέστε το αρχείο και ανοίξτε το αρχείο `/tmp/"AEM"` θα διαπιστώσετε ότι έχει γραφεί το αρχείο με χαρακτήρες ASCII. Τροποποιήστε κάποιους χαρακτήρες και εκτελέστε πάλι το πρόγραμμα, και θα διαπιστώσετε ότι το πρόγραμμά σας εκτυπώνει τις σωστές τιμές.

> ➔ Το μεγάλο πλεονέκτημα αυτής της μεθόδου είναι ότι δε χρειάζεται να διαβάσουμε όλο το αρχείο στη μνήμη. Τροποποιείται μόνο το τμήμα του αρχείου *(ή οι σελίδες του αρχείου)* που χρειαζόμαστε. Έτσι μπορούμε π.χ. να κάνουμε αντιστοίχηση με ένα αρχείο αρκετά μεγάλο *(κάποια GB)* σε μηχάνημα που έχουμε πολύ λίγη μνήμη.

(C4) Να γραφούν τα προγράμματα **c4-server.c** και **c4-client.c** τα οποία ως διεργασία επικοινωνούν με ένα κοινό αρχείο που θα έχει στοιχεία int. Συγκεκριμένα και τα δύο αρχεία θα συνδέονται με read-write σε ένα αρχείο δική σας επιλογής μεγέθους 4*500 *(κάθε ακέραιος έχει 4Bytes και θέλουμε 500 θέσεις).*

- Το **c4-server.c** θα διαβάζει κάθε 1 sec *(χρήση της sleep)* την τιμή του αρχείου στη θέση 0 *(mapfile[0])*. Μόλις διαβάσει μια τιμή διαφορετική από το 0, τότε θα πηγαίνει στην αντίστοιχη θέση mapfile θα διαβάζει την τιμή που υπάρχει, θα την εκτυπώνει, θα την αυξάνει κατά 5 και θα τη γράφει πάλι. Στη συνέχεια θα μηδενίζει την τιμή στη θέση 0. Η διαδικασία επαναλαμβάνεται έως να διαβάσει την τιμή 555 οπότε θα σταματάει.
- Το **c4-client.c** θα ζητάει από το χρήση να πληκτρολογήσει μια διεύθυνση. Αφού γίνει έλεγχος έγκυρης τιμής, η διεργασία θα διαβάζει τη συγκεκριμένη τιμή στη διεύθυνση που έχει δοθεί, θα την εμφανίζει και θα ζητάει από το χρήστη να πληκτρολογήσει τη νέα τιμή. Ο χρήστης θα πληκτρολογεί μια νέα τιμή και αυτή θα τοποθετείται στη διεύθυνση. Ταυτόχρονα ο αριθμός της διεύθυνσης αυτής θα τοποθετηθεί στη θέση 0 *(για να το διαβάσει ο server)*. Η διαδικασία επαναλαμβάνεται έως να διαβάσει την τιμή 555 οπότε τη γράφει στη θέση 0 και σταματάει.
- Να υπάρχουν οι κατάλληλοι έλεγχοι.

Ένα πρόβλημα που υπάρχει είναι ότι η πρόσβαση στα κοινά δεδομένα γίνεται χωρίς αμοιβαίο αποκλεισμό. Για να επιτευχθεί ο αμοιβαίος αποκλεισμός θα πρέπει να χρησιμοποιηθούν mutex ή semaphores, κάτι που θα δούμε στο τελευταίο εργαστήριο.

6. ΠΡΟΣΟΜΟΙΩΣΗ ΔΙΕΡΓΑΣΙΩΝ (SOSSIM)

6.1 Ασκήσεις

Έναρξη -- τερματισμός διεργασιών στον προσομοιωτή διεργασιών λειτουργικού Συστήματος Sosim.

Ο προσομοιωτής SOsim υλοποιεί τις διεργασίες ως Process Control Blocks *(PCBs)*, όπου διατηρούνται οι πληροφορίες της κάθε διεργασίας. Ο χρονοπρογραμματισμός είναι η διαδικασία της απόφασης επιλογής της διεργασίας που θα «κερδίσει» τη CPU.

Υλοποιείται με:

- Προεκτοπιστικούς αλγορίθμους *(preemptive algorithms).*
- Καθορισμός προτεραιοτήτων.
- Χρόνος άφιξης *(δημιουργίας)* διεργασίας.
- Μη-προεκτοπιστικούς αλγορίθμους *(non-preemptive algorithms).*

Ο προσομοιωτής SOSIM έχει 6 παράθυρα:

- Παράθυρο SOsim Console,
- Παράθυρο Processor Manager,
- Παράθυρο Log,
- Παράθυρο Statistics
- Παράθυρο Select a Process,
- Παράθυρο Memory Manager.

Η λειτουργία του παραθύρου SOSim Console:

Δημιουργία νέας διεργασίας *(Process / Create).*

Ορισμός κατηγορίας διεργασίας *(CPU-bound, I/O bound κ.τ.λ)*

- I/O-1, I/O-2, I/O-3 : αν έχουμε ορίσει χρονοπρογραμματισμό με δυναμική προτεραιότητα *(dynamic priority scheduling)*, τα -1, -2, -3 δηλώνουν την αύξηση της προτεραιότητας της διεργασίας κατά +1, +2, +3
- MIX-1:
    - η διεργασία καταναλώνει το χρόνο της τόσο σε I/O όσο και στη CPU ακολουθία καταστάσεων διεργασίας:
      ready->running->ready by time->I/O
    - priority boost +1 *(όταν έχει οριστεί χρονοπρογραμματισμός με δυναμική προτεραιότητα).*
- MIX-2:
    - ακολουθία καταστάσεων διεργασίας:
      ready->running->ready by time->I/O ,
    - priority boost +2
- Ορισμός προτεραιότητας διεργασίας.

Πλήκτρο Run/Stop: παγώνει τις οθόνες των παραθύρων, ώστε να παρατηρήσουμε κάποιες παραμέτρους. Επαναφέρουμε την κίνηση με το Run

- Εμφάνιση παραθύρου μηνυμάτων **(Windows/Log)**
- Εμφάνιση παραθύρου στατιστικών **(Windows/Statistics)**

Πηγαίνετε στο παράθυρο SOSIM Processor Manager.

- Καθορίστε χρονοπρογραμματισμό εκ περιτροπής χωρίς προτεραιότητες (**round-robin scheduling**) *(από Options).*
- Δημιουργήστε δύο διεργασίες προτεραιότητας 0: μια CPU-bound και μια I/O-1 *(από το SOSIM Console, Process/create)*
- Εμφανίστε το παράθυρο Log, αν δεν είναι ορατό. *(από SOSIM CONSOLE Windows Log)*
- Παρατηρήστε πως εναλλάσσονται οι διεργασίες *(πολυπρογραμματισμός)*
- Προσθέστε μια διεργασία I/O-2.
- Παρατηρήστε το παράθυρο Log. Υπάρχει κάποια διαφορά στην εκτέλεση των διεργασιών; _____ **(S1)**

6.2 Καταστάσεις στο Παράθυρο Processor Manager

Ready: δείχνει τις ουρές προτεραιοτήτων των διεργασιών.

Waiting: δείχνει ποιες διεργασίες περιμένουν για I/O.

I/O wait time: δείχνει πόσο χρόνο θα περιμένει μια διεργασία I/O-bound μέχρι να εξυπηρετηθεί. Όταν διαλέγουμε μικρό I/O wait time, τότε διακόπτεται πιο γρήγορα ο χρόνος της CPU για να εκτελεστεί I/O.

time slice: το χρονικό διάστημα *(κβάντο χρόνου)* κατά το οποίο μια διεργασία χρησιμοποιεί τη CPU.

clock: αν αυξηθεί αυτή η παράμετρος τότε παρουσιάζονται πιο γρήγορα οι εναλλαγές των διεργασιών στο παράθυρο log.

Priority preemptive scheduling: Αν είναι τσεκαρισμένο θα πραγματοποιείται χρονοπρογραμματισμός εκ περιτροπής με προτεραιότητες. Αν δεν είναι τσεκαρισμένο θα πραγματοποιείται χρονοπρογραμματισμός εκ περιτροπής χωρίς προτεραιότητες.

Dynamic priority scheduling: Αν είναι τσεκαρισμένο θα ισχύουν δυναμικές προτεραιότητες *(π.χ. Μια διεργασία τύπου I/O-1 θα αυξάνει κατά 1 την προτεραιότητά της, I/O-2 κατά 2 κ.ο.κ).*

Παράθυρο Statistics

#processes : αριθμός διεργασιών.

ready : πόσες διεργασίες είναι έτοιμες.

running: πόσες διεργασίες τρέχουν.

wait: πόσες διεργασίες περιμένουν να εξυπηρετηθούν.

Το άθροισμα των blocks που χρησιμοποιούν μπορεί να είναι μέχρι =20 πριν αρχίσει η εναλλαγή blocks *(4 διεργ. x 5 blocks = 20 memory allocation blocks = 20% του συνολικού χώρου μνήμης).* Μετά αρχίζει εναλλαγή *(swapping).* Δηλαδή μέχρι 4 διεργασίες *(των 5 blocks)* μπορούν να βρίσκονται στη μνήμη κάθε χρονική στιγμή.

Hw Page faults: όταν μια διεργασία δημιουργείται ο αριθμός αυξάνεται κατά τον αρ.των blocks της διεργασίας.

Κάθε φορά που μια διεργασία αφαιρείται από τη μνήμη για να μπει μια άλλη στη θέση της ο αριθμός αυτός αυξάνεται κατά τον αρ.των blocks της νεοεισερχόμενης διεργασίας *(αρ. blocks που ζητήθηκαν).*

Παράδειγμα 1 χρονοδρομολόγησης στο SOSIM

1. Να δημιουργήσετε στον Προσομοιωτή SOSIM τρεις διεργασίες CPU-bound, ίδιας προτεραιότητας (0).
    a. Παρακολουθήστε το παράθυρο Free Page List *(FPL)*:
2. Το **Free Page List** *(FPL)* μικραίνει με κάθε διεργασία που προσθέτουμε
3. Παρακολουθήστε το παράθυρο log:
    a. Οι διεργασίες μοιράζονται τη CPU για ίσο χρόνο.
4. Προσθέστε μια 4η διεργασία CPU-bound, προτεραιότητας 1.
    a. Παρακολουθήστε το παράθυρο log:
    b. Τρέχει μόνο η 4η διεργασία CPU.
5. Από το παράθυρο Process/Select/ επιλέξτε Suspend για την 4η διεργασία.
6. Τρέχουν μόνο οι 3 διεργασίες.
7. Αλλάξτε Priority για την 4η διεργασία από 1 σε 0
8. Επιλέξτε Resume για την 4η διεργασία
9. Παρακολουθήστε το παράθυρο log:
    a. Όλες οι διεργασίες μοιράζονται τώρα τη CPU για ίσο χρόνο.

Αν στο προηγούμενο παράδειγμα δημιουργηθεί πρώτα η CPU-bound διεργασία με προτεραιότητα 1 και μετά οι 3 CPU-bound διεργασίες με προτεραιότητα 0 τί θα συμβεί; _____ **(S2)**

(S3)

- Προσομοιώστε τη λειτουργία χρονοπρογραμματισμού **round - robin** με προτεραιότητες και dynamic priority scheduling με 4 διεργασίες:
    - Μία I/O bound με προτεραιότητα 4.
    - Δύο I/O bound με προτεραιότητα 3.
    - Δύο CPU bound με προτεραιότητα 5.
- Ορίστε το time-slice = 2.

- Ορίστε το I/O wait time = 8.
- Παρατηρήστε για δύο (2) λεπτά το χρόνο εκτέλεσης των διεργασιών.
- Εξυπηρετούνται όλες οι I/O-bound διεργασίες;
- Αν η CPU-bound διεργασία γίνει suspended και μετά resumed, τι συμβαίνει με την εκτέλεση των διεργασιών;
- Παραδώστε το αρχείο log στο οποίο να έχετε συμπληρώσει:
    - Στην αρχή του αρχείου το ονοματεπώνυμό σας και τα PID των διεργασιών σας.
    - Στο τέλος του αρχείου να απαντήσετε τα υποερωτήματα 5 και 6.