2η Εργαστηριακή Άσκηση

Στα πλαίσια της παρούσας εργαστηριακής άσκησης καλείστε να προσθέσετε υποστήριξη σημαφόρων (semaphores) στο MINIX. Το υπάρχον σύστημα δεν προσφέρει τη δυνατότητα χρήσης σημαφόρων για την επίτευξη συγχρονισμού ή αμοιβαίου αποκλεισμού μεταξύ διεργασιών επιπέδου χρήστη. Παρόλα αυτά η σχεδίαση του MINIX καθιστά την υλοποίηση σημαφόρων ιδιαίτερα απλή. Συγκεκριμένα, η υποστήριξη διαδιεργασιακής επικοινωνίας με πέρασμα μηνυμάτων καθώς και το γεγονός ότι οι *servers* του συστήματος εξυπηρετούν τις διεργασίες επιπέδου χρήστη δίχως να αναστέλλεται η λειτουργία τους (*non-preemptive execution*), εξασφαλίζουν την ατομικότητα (*atomicity*) της πράξης DOWN (P) με απλό και άμεσο τρόπο[1]. Οι κλήσεις συστήματος που απαιτούνται για την υποστήριξη των σημαφόρων θα εξυπηρετούνται από τον Memory Manager (MM).

Για την ολοκλήρωση της παρούσας εργαστηριακής άσκησης θα πρέπει να υλοποιήσετε:

1. τις κλήσεις συστήματος που απαιτούνται για την υποστήριξη σημαφόρων στο επίπεδο χρήστη του MINIX.
2. μία εφαρμογή η οποία θα επιλύει μία εκδοχή του προβλήματος *παραγωγού-καταναλωτή* με χρήση σημαφόρων.

Σκοποί της άσκησης αυτής είναι:

- η εκμάθηση της διαδικασίας υλοποίησης και εισαγωγής μίας νέας κλήσης συστήματος στο λειτουργικό σύστημα MINIX.
- η εξοικείωση με τον τρόπο διαδιεργασιακής επικοινωνίας (IPC) που χρησιμοποιεί το MINIX, δηλαδή το blocking πέρασμα μηνυμάτων.
- η κατανόηση των βασικών αρχών συγχρονισμού μέσω της εφαρμογής τους στα πλαίσια ενός πραγματικού λειτουργικού συστήματος.

1. Κλήσεις συστήματος

Στη συνέχεια παρατίθεται η ακριβής περιγραφή του interface που θα πρέπει να υλοποιήσετε:

```
#include <unistd.h>

#include <sys/const.h>

int seminit(int semid, int val)

int semup(int semid)

int semdown(int semid)

int semfin(int semid)

int semstat(int semid, int *nwait)
```

Περιγραφή

Η `seminit()` αρχικοποιεί/δεσμεύει το σημαφόρο με α/α `semid` με την τιμή `val`. Οι κλήσεις `semup()` και `semdown()` εφαρμόζουν την πράξη UP (V) και DOWN (P) αντίστοιχα στο σημαφόρο με α/α `semid`. Η κλήση `semfin()` απελευθερώνει όσες διεργασίες είναι μπλοκαρισμένες στο σημαφόρο με α/α `semid` και στη συνέχεια

[1] Η υλοποίηση σημαφόρων μέσω ενός μηχανισμού περάσματος μηνυμάτων στηρίζεται στην ισοδυναμία των αρχών συγχρονισμού όπως αυτή περιγράφεται στο [1].

τον μαρκάρει ως διαθέσιμο. Η `semstat()` επιστρέφει την τρέχουσα τιμή του σημαφόρου `semid` και αποθηκεύει στη θέση μνήμης που δείχνει ο `nwait`, το πλήθος των διεργασιών που είναι μπλοκαρισμένες στο συγκεκριμένο σημαφόρο.

`semid` ο α/α του σημαφόρου. Η τιμή της παραμέτρου πρέπει να είναι ≥0 και μικρότερη της σταθεράς `MAX_SEM_NR` που ορίζεται στο αρχείο `<unistd.h>`.

`val` η τιμή αρχικοποίησης του σημαφόρου.

`nwait` δείκτης στη θέση μνήμης όπου αποθηκεύεται το πλήθος των διεργασιών που έχουν μπλοκάρει στο σημαφόρο.

Επιστρεφόμενη τιμή

Οι συναρτήσεις `semup()` και `semdown()` στην περίπτωση επιτυχούς ολοκλήρωσης επιστέφουν την τιμή του σημαφόρου μετά την εκτέλεση της πράξης UP και DOWN αντίστοιχα. Η `semstat()` επιστρέφει την τιμή του σημαφόρου την στιγμή που εξυπηρετείται η κλήση. Οι συναρτήσεις `seminit()` και `semfin()` απλά επιστρέφουν 0 αν ολοκληρωθούν επιτυχώς. Σε περίπτωση αποτυχίας ή λήψης κάποιου *singal* όλες οι συναρτήσεις επιστρέφουν `-1` και αποθηκεύουν τον κωδικό λάθους στην *global* μεταβλητή `errno`.

Κωδικοί λαθους

Αν η κλήση αποτύχει η *global* μεταβλητή `errno` περιέχει μία από τις παρακάτω τιμές:

`EINVAL` η τιμής της παραμέτρου `semid` δεν είναι έγκυρη (`0 ≤ semid < MAX_SEM_NR`).

`EINVAL` η τιμή της παραμέτρου `val` δεν είναι έγκυρη (`val ≥ 0`).

`EFAULT` ο δείκτης *nwait* δείχνει σε διεύθυνση εκτός του δεσμευμένου χώρου μνήμης της διεργασίας.

`EPERM` η σημαφόρος είναι ήδη αρχικοποιημένη.

`EACCES` η σημαφόρος χρησιμοποιείται από κάποιον άλλο χρήστη.

`EINTR` η κλήση συστήματος διακόπηκε από κάποιο *signal*.

Σχόλια

1. Κάθε σημαφόρος υπάρχει και διατηρεί το περιεχόμενο του από την στιγμή που αρχικοποιείται μέχρι τη στιγμή που τερματίζεται, ανεξάρτητα από τον κύκλο ζωής των διεργασιών που τον διαχειρίζονται. Έτσι εάν ένας σημαφόρος δεν έχει προηγουμένως τερματιστεί ρητά με κλήση της `semfin()` από τον χρήστη που τον δέσμευσε, υπάρχει ως αντικείμενο ακόμα και μετά τον τερματισμό της διεργασίας.
2. Όταν μία διεργασία που είναι σε αναστολή λόγω της `semdown()` λάβει ένα *signal,* συνεχίζει την εκτέλεση της επιστρέφοντας από την `semdown()` με τιμή -1 και κωδικό λάθους `EINTR` (δηλ. *interrupted system call*).
3. Κάθε διαθέσιμος σημαφόρος[2] μπορεί να αρχικοποιηθεί από οποιονδήποτε χρήστη.
4. Κάθε χρήστης ανά πάσα στιγμή έχει δικαίωμα χρήσης και τερματισμού μόνο στους σημαφόρους που ο ίδιος έχει αρχικοποιήσει και δεν έχει τερματίσει ακόμα.
5. Ο υπερχρήστης *root* έχει δικαίωμα αρχικοποίησης, χρήσης και τερματισμού οποιουδήποτε σημαφόρου.
6. Ένας χρήστης δεν μπορεί να αρχικοποιήσει εκ νέου έναν σημαφόρο αν προηγουμένως δεν τον έχει

[2] Ένας σημαφόρος είναι διαθέσιμος όταν δεν είναι αρχικοποιημένος από κάποια διεργασία.

τερματίσει με την `semfin()`[3].

2. Εφαρμογή

Στα πλαίσια της άσκησης πέρα από την υλοποίηση των κλήσεων συστήματος για την υποστήριξη σημαφόρων, καλείστε να αναπτύξετε μία απλή εφαρμογή που θα χρησιμοποιεί αυτή την υποδομή για την ορθή επίλυση του προβλήματος παραγωγού-καταναλωτή (*bounded buffer problem* ή *single-producer single-consumer problem*).

Η εφαρμογή σας θα πρέπει να ακολουθεί τους επόμενους κανόνες/υποδείξεις:

- το πρόγραμμα δέχεται δύο υποχρεωτικές παραμέτρους γραμμής εντολών:
  i. το συνολικό πλήθος των παραγόμενων αντικειμένων
  ii. το μέγεθος σε αντικείμενα του διαμοιραζόμενου *buffer*
- η αρχική διεργασία θα πρέπει να δημιουργεί (`fork()`) μία διεργασία η οποία θα πρέπει να εκτελεί τον κώδικα του καταναλωτή μέχρι να ειδοποιηθεί από τον παραγωγό ότι δεν θα παραχθούν άλλα αντικείμενα. Η αρχική διεργασία θα εκτελεί τον κώδικα του μοναδικού παραγωγού και θα τερματίζεται μόνο όταν έχουν καταναλωθεί όλα τα αντικείμενα που πρέπει να παράγει - το πλήθος τους έχει δοθεί ως όρισμα στην γραμμή εντολών. Πριν τερματιστεί, η διεργασία του παραγωγού θα πρέπει να ειδοποιήσει και να περιμένει (`wait(),waitpid()`) την διεργασία του καταναλωτή να τερματίσει.
- λόγω της έλλειψης υποστήριξης κοινής μνήμης μεταξύ διεργασιών από το MINIX, ο διαμοιραζόμενος *buffer* θα πρέπει να υλοποιηθεί με τη βοήθεια ενός αρχείου. Το αρχείο αυτό θα προσπελαύνετε από παραγωγό και καταναλωτή μόνο μέσα στο *critical section*.
- τα αντικείμενα που θα παράγει ο παραγωγός θα είναι τυχαίοι ακέραιοι. Η τυχαία ακολουθία που θα προκύπτει θα πρέπει να καταγράφεται σε ένα αρχείο με όνομα *prod*. Ο καταναλωτής θα υπολογίζει το τετράγωνο του ακεραίου/αντικειμένου και θα παράγει ένα αρχείο κειμένου με όνομα *cons* του οποίου κάθε γραμμή θα έχει τη μορφή "<ακέραιος> <τετράγωνο ακεραίου>". Τα αντικείμενα θα γράφονται στα αρχεία *prod/cons* με την ακριβή σειρά με την οποία γράφονται/διαβάζονται από το διαμοιραζόμενο αρχείο.

3. Παραδοτέα

Στα πλαίσια της άσκησης θα πρέπει να παραδώσετε τα εξής:

1. Ένα αρχείο σε .zip *format* που θα περιέχει τα ακόλουθα:

i. Όλα τα αρχεία του πηγαίου κώδικα του συστήματος που αλλάξατε/προσθέσατε. Ο κώδικας θα πρέπει να είναι επαρκώς σχολιασμένος. Στα αρχεία κώδικα που προϋπαρχαν, οι προσθήκες κώδικα θα πρέπει να εσωκλείονται στις εξής γραμμές σχολίων:

```
/* OSLAB START */
κώδικας που προσθέσατε
/* OSLAB END */
```

Αντίστοιχα τα τμήματα του αρχικού κώδικα που δεν θέλετε να περιληφθούν πρέπει να σχολιάζονται με την ένδειξη `CROP START` και `CROP END` στην αρχή και στο τέλος τους αντίστοιχα.

ii. Το *binary* αρχείο του *kernel* με όνομα "`2.0.r<α/α ομάδας>`" (π.χ. για την ομάδα με α/α 17 το όνομα του αρχείου θα πρέπει να είναι `2.0.r17`).

iii. Τον πηγαίο κώδικα του προγράμματος που θα υλοποιεί το πρόβλημα παραγωγού-καταναλωτή που

[3] Είναι προφανές ότι το *schema* χρήσης και δικαιωμάτων που περιγράφεται στα 3-6 είναι πολύ περιοριστικό και ουσιαστικά στερείται των χαρακτηριστικών που πρέπει να παρέχει μία υπηρεσία ενός λειτουργικού συστήματος. Η υλοποίηση χαρακτηριστικών όπως μηχανισμός προστασίας/δικαιωμάτων, όρια πλήθους δεσμευμένων σημαφόρων ανά χρήστη και χρήση αναγνωριστικών τύπου SysV IPC αν και είναι απαραίτητη είναι εκτός του σκοπού της συγκεκριμένης εργαστηριακής άσκησης.

περιγράφεται στην Ενότητα 2.

2. Τεχνική αναφορά (σε μορφή .pdf κατά προτίμηση) όπου θα εξηγείτε με παράθεση σύντομων σχολίων την υλοποίηση της εφαρμογής καθώς και τις αλλαγές που πραγματοποιήσατε στον πηγαίο κώδικα του MINIX.

Τα παραπάνω θα πρέπει να σταλούν με e-mail στο *oslab@ceid.upatras.gr* πριν την ημερομηνία παράδοσης που θα ανακοινωθεί σύντομα στο *forum* του μαθήματος. Το e-mail θα πρέπει να αποσταλεί με *Subject* της μορφής "`OSLAB_2 <α/α ομάδας>`" (π.χ. για την ομάδα με α/α 17, Subject: `OSLAB_2 17`).

4. Υποδείξεις

1. Ακολουθείστε τη γενικότερη υπόδειξη που δίνεται για την υλοποίηση ασκήσεων τέτοιου τύπου και η οποία είναι η σταδιακή προσέγγιση του προβλήματος. Μην προσπαθείτε να υλοποιήσετε το σύνολο των ζητουμένων απευθείας. Κατασκευάστε ενδιάμεσες *λειτουργικές* εκδόσεις που υλοποιούν ένα υποσύνολο των ζητουμένων και των οποίων την ορθότητα μπορείτε εύκολα να επαληθεύσετε. Για παράδειγμα η συγκεκριμένη άσκηση μπορεί να χωριστεί στα εξής τμήματα:

i. υλοποίηση ενός "dummy" system call που εκτυπώνει απλά ένα μήνυμα
ii. επέκταση του (i) έτσι ώστε να εκτυπώνονται τα ορίσματα που περιέχονται στο μήνυμα του system call
iii. επέκταση του (ii) έτσι ώστε υποστηρίζεται η αρχικοποίηση και ο τερματισμός σημαφόρων
iv. επέκταση του (iii) έτσι ώστε υποστηρίζονται οι πράξεις UP και DOWN
v. επέκταση του (iv) έτσι ώστε να λαμβάνεται πρόνοια για την περίπτωση που μία διεργασία λάβει κάποιο *signal* ενώ είναι μπλοκαρισμένη σε κάποιον σημαφόρο
vi. επέκταση του (v) έτσι ώστε να λαμβάνεται πρόνοια για την περίπτωση που μία διεργασία τερματιστεί ενώ είναι μπλοκαρισμένη σε κάποιον σημαφόρο

Κάποια από τα παραπάνω βήματα μπορούν να ενοποιηθούν. Αυτό καθορίζεται ελεύθερα και δυναμικά από εσάς βάσει της εμπειρίας σας.

2. Μελετείστε προσεκτικά τις δομές και τις συναρτήσεις που περιέχονται στα εξής αρχεία:

i. *mproc.h*: περιλαμβάνει το *process table* και τις σταθέρες που περιγράφουν την κατάσταση μίας διεργασίας
ii. *main*.c: περιλαμβάνει τον κώδικα αρχικοποίησης του MM Server
iii. *forkexit.c*: περιέχει τις συναρτήσεις τερματισμού και δημιουργίας μίας διεργασίας
iv. *signal*.c: περιέχει τις συναρτήσεις που υλοποιούν το μηχανισμό των *signals*

3. Η περιγραφή των σημαφόρων ως αρχής συγχρονισμού και αμοιβαίου αποκλεισμού στο [1] θα σας βοηθήσει να κατανοήσετε τα ζητούμενα της άσκησης. Επιπλέον στο [2] περιγράφεται με πληρότητα και λεπτομέρεια η σχεδίαση και λειτουργία όλων των τμημάτων του MM Server του MINIX όσο και ο κώδικας υλοποίησης τους. Ιδιαίτερης σημασίας είναι οι υποενότητες 2.2.5, 2.2.7, 2.5, 4.7, 4.8.1-3, 4.8.6 από το [2].

4. Οδηγίες για την προσθήκη μίας νέας κλήσης συστήματος μπορείτε να βρείτε στο Παράρτημα της 1ης Εργαστηριακής Άσκησης.

5. Σαν *debugging tool* χρησιμοποιείστε την `printf()` (macro name της `printk()`). Η σύνταξη και τα ορίσματα της είναι ίδια με αυτά της συνάρτησης βιβλιοθήκης `printf()` με τη διαφορά ότι τα ορίσματα της εκτυπώνονται πάντοτε στο `tty0`.

6. Θα πρέπει να έχετε δεσμεύσει χώρο στο *address space* του MM εκ των προτέρων για κάθε σημαφόρο καθώς το MINIX δεν προσφέρει μηχανισμό δυναμικής δέσμευσης μνήμης για τους *servers*.

7. Φροντίστε για την αρχικοποίηση κάθε δομής και μεταβλητής που προσθέτετε στο process table του MM.

8. Το διαμοιραζόμενο αρχείο που θα χρησιμοποιήσετε στην εφαρμογή παραγωγού-καταναλωτή θα πρέπει να είναι σε *binary* και όχι σε *text format.*

9. Λάβετε υπόψιν ότι θα πρέπει να αφαιρείτε από την λίστα αναμονής ενός σημαφόρου τις διεργασίες που λαμβάνουν κάποιο *signal* ή τερματίζονται.

10. Μην σβήσετε τον αρχικό kernel */minix/2.0.0* για να μπορέσετε, στη περίπτωση που κάτι απρόβλεπτο συμβεί λόγω του καινούργιου image, να ξεκινήσετε το σύστημα και να διορθώσετε το πρόβλημα που προέκυψε. Λόγω της φύσης της συγκεκριμένης άσκησης υπάρχει ο κίνδυνος ανεπανόρθωτης καταστροφής του συστήματος αρχείων. Διατηρείται τουλάχιστον ένα αντίγραφο ασφαλείας ολόκληρου του *virtual* δίσκου του *Bochs* στον οποίο εργάζεστε.

Βιβλιογραφία

[1] *Modern Operating Systems*, Andrew S. Tanenbaum, 1st ed., Prentice Hall 1992
[2] *Operating Systems, Design and Implementation,* Andrew S. Tanenbaum and Albert S. Woodhull, 2nd ed., Prentice Hall 1996