ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Λειτουργικά Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:11

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

- Αμοιβαίος αποκλεισμός με σημαφόρους SystemV *semget(), semop(), semctl()) και POSIX *(sem_init(), sem_wait(), sem_post(), sem_destroy()).*
- Διαχείριση αρχείων *(open(), close(), read(), write(), stat(), lseek(), unlink(), basename(), mkstemp(), strcat()).*
- Προσομοιωτής μνήμης MOSS-memory.

2. Παραδοτέα

(A + S) 20 + 6 ερωτήσεις
(C) 7 ασκήσεις

3. Επίτευξη αμοιβαίου αποκλεισμού με τη χρήση σημαφόρων

Οι σημαφόροι είναι ένας μηχανισμός συγχρονισμού διεργασιών για την αποκλειστική χρήση κοινών πόρων όπως μνήμης ή αρχείων. Οι σημαφόροι είναι ειδικού τύπου ακέραιες μεταβλητές με τις οποίες συνδέονται κάποιες ατομικές λειτουργίες *(δηλαδή όλες οι εντολές assembly που αντιστοιχούν εκτελούνται αδιάσπαστα από τον πυρήνα)*. Όταν θέλουμε να εισέλθουμε στο κρίσιμο τμήμα μιας διεργασίας *(δηλαδή την πρόσβαση στο κοινό πόρο)*, τότε εκτελείται μια ενέργεια DOWN στον αντίστοιχο σημαφόρο. Αν ο σημαφόρος έχει τιμή 0 τότε η εντολή μπαίνει σε κατάσταση block-wait. Αν ο σημαφόρος έχει τιμή μεγαλύτερη από 0 τότε εισέρχεται στο κρίσιμο τμήμα αφού μειώσει την τιμή κατά 1. Μόλις βγει από το κρίσιμο τμήμα, τότε εκτελεί μια ενέργεια UP και στην περίπτωση που περιμένει μια διεργασία σε block-wait ενεργοποιείται και ο σημαφόρος παραμένει στο 0, ενώ αν δεν περιμένει μια διεργασία τότε η τρέχουσα τιμή αυξάνεται κατά 1.

Υπάρχουν δυο είδη σημαφόρων:

(α) **`SystemV Semaphores`** και

(b) **`Posix Semaphores`** .

3.1 Οι σημαφόροι SystemV

Οι 3 συναρτήσεις που χρησιμοποιούνται για τη διαχείριση των σημαφόρων είναι οι **`semget()`**, **`semop()`**, **`semctl()`**. Με τη **`semget()`** δημιουργείται ένας περιγραφέας σημαφόρος για να χρησιμοποιηθεί στις υπόλοιπες συναρτήσεις. Με τη **`semop()`** εκτελείται μια πράξη UP ή DOWN στον αντίστοιχο σημαφόρο. Με τη συνάρτηση **`semctl()`** ελέγχεται ο σημαφόρος και μπορεί να καταστραφεί.

Η συνάρτηση δημιουργίας ενός περιγραφέα σημαφόρου **`semget()`** πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A1)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A2)**

Η συνάρτηση εκτέλεσης μιας πράξης σε σημαφόρο **`semop()`** πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A3)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A4)**

Η συνάρτηση ελέγχου λειτουργίας σημαφόρου **`semctl()`** πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει;_____ **(A5)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A6)**

***** Προσοχή:** Κατά τη συγγραφή ενός προγράμματος πρέπει πάντα να ελέγχουμε την τιμή επιστροφής της κλήσης συστήματος και να σταματάμε αμέσως την εκτέλεση, αν επιστρέψει τιμή σφάλματος *(που συνήθως είναι -1, αλλά θα πρέπει να το επιβεβαιώνετε από το man page)*. Επίσης, αν ανιχνευτεί πρόβλημα να χρησιμοποιείτε τη συνάρτηση perror() που εμφανίζει εκτός από το μήνυμα σας και επιπρόσθετες πληροφορίες, αν η κλήση συστήματος ενεργοποιεί την ειδική μεταβλητή errno, το οποίο αναγράφεται στο man page *("errno is set to indicate the error").* Τέλος, ποτέ να μην τοποθετείτε μια κλήση συστήματος χωρίς να αποθηκεύετε την τιμή επιστροφής, ακόμη και αν δεν πρόκειται να χρησιμοποιήσετε αυτή την τιμή. Αυτό συνιστάται να γίνεται για να διευκολύνεστε κατά τη διαδικασία αποσφαλμάτωσης με το gdb. *******

Παράδειγμα:

Καλό παράδειγμα:

```
retvalue = semctl(.....);

if (retvalue == -1 ) { perror ("Parousiasthke problhma"); exit(-1);}
```

Κακό Παράδειγμα:

```
if(semctl(...)==-1){printf("Parousiasthke problhma");exit(-1);}
```

3.1.1 Το παραδοτέο C1

(C1) Να δημιουργήσετε δυο προγράμματα **c1-nomutex.c** και **c1-nomutex2.c** τα οποία ως διεργασία εμφανίζουν και επιδεικνύουν το πρόβλημα της πρόσβασης σε κρίσιμο τμήμα κοινής μνήμης.

Συγκεκριμένα το **c1-nomutex.c**:

- Θα δημιουργεί ένα id κοινής μνήμης με την κλήση της
  `shmget((key_t)4444, 10,0600|IPC_CREAT)`

  *** **Προσοχή:** Να αντικαταστήσετε το 4444 με ένα άλλο κλειδί δικό σας, ιδιαίτερα αν εργάζεστε σε ένα λειτουργικό σύστημα που βρίσκονται πολλοί χρήστες και όλοι θα δημιουργούν περιοχές κοινής μνήμης. ***
- Θα προσαρτάται η κοινή μνήμη στη διεργασία με την
  `shmat(shmid, (char *) 0, 0)`
- Θα τοποθετεί 0 στη θέση 0 της κοινής μνήμης.
- Θα διαβάζει την τιμή στη θέση 0 της κοινής μνήμης και θα αναθέτει την τιμή σε μια μεταβλητή x1.
- Θα κάνει `sleep(15).`
- Θα τοποθετεί στη θέση 0 της κοινής μνήμης την τιμή x1 αφού αυξηθεί κατά 5.
- Θα εκτυπώνει την τιμή που βρίσκεται στη διεύθυνση 0.

Ενώ το **c1-nomutex2.c**:

- Θα δημιουργεί ένα id κοινής μνήμης με βάση το ίδιο κλειδί με την κλήση της
  `shmget((key_t)4444, 10,0600|IPC_CREAT)`
- Θα προσαρτάται η κοινή μνήμη στη διεργασία με την

  `shmat(shmid, (char *) 0, 0)`
- Θα κάνει `sleep(5)`.
- Θα αυξάνει κατά 10 τη θέση 0 της κοινής μνήμης και θα το αποθηκεύει στην ίδια θέση.

  ➔Ύστερα από κάθε κλήση συνάρτησης θα γίνεται έλεγχος αν έχει εκτελεστεί σωστά και θα εκτυπώνεται είτε το μήνυμα επιτυχίας, είτε το μήνυμα αποτυχίας καθώς και το PID της διεργασίας που αναφέρει το μήνυμα.

  ➔Η τιμή που θα πρέπει να τυπώνεται στο τέλος θα πρέπει να είναι 5.

Εκτελέστε ταυτόχρονα και τις δυο διεργασίες *(τη nomutex2 δυο φορές)* αφού τις κάνετε compile με

```
./c1-nomutex & ; ./c1-nomutex2.c & ; ./c1-nomutex2.c &
```

και απαντήστε στο ερώτημα:

Στις διεργασίες του **C1** υπάρχει το πρόβλημα των χαμένων συναλλαγών; Αν ναι που οφείλεται αυτό; _____ **(A7)**

3.1.2 Το παραδοτέο C2

(C2) Αντιγράψτε τα προγράμματα **C1** και τροποποιήστε τα *(**c2-nomutex.c** και **c2-nomutex2.c**)*, ώστε να αποφευχθεί το πρόβλημα των χαμένων συναλλαγών. Για να γίνει αυτό θα χρησιμοποιήσουμε αμοιβαίο αποκλεισμό μέσω σημαφόρων. Συγκεκριμένα και στα δύο προγράμματα θα προσθέσετε κώδικα που:

- θα δημιουργείται με το ίδιο κλειδί σημαφόρου ένα ID σημαφόρου με τη `semget()`.
- Το ένα από τα δύο προγράμματα θα αρχικοποιεί το σημαφόρο σε τιμή 1 με την κλήση

  ```
  union semun arguments;
  arguments.val=1;
  semctl(semid, 0, SETVAL, arguments);
  ```

- Δηλαδή, ότι δηλώνουμε μια μεταβλητή με όνομα `arguments` που είναι της δομής semun. Η δομή semun ορίζει κάποια πεδία. Ένα από αυτά είναι το πεδίο val. Τοποθετούμε σε αυτό (`arguments.val=1`) την τιμή 1, δηλαδή ορίζουμε ότι η σημαφόρα θα έχει αρχική τιμή 1 . Επίσης, η δεύτερη παράμετρος είναι 0, γιατί έχουμε μόνο μια σημαφόρα, οπότε αυτή παίρνει τον αριθμό 0.
- Αμέσως μετά την προσάρτηση της κοινής μνήμης και πριν γίνει η ανάγνωση ή η εγγραφή στην κοινή περιοχή, θα καλείται η `semop(semid, &operationDOWN[0], 1)` ενώ θα έχετε δηλώσει ότι `struct sembuf operationDOWN[1] = {0, -1, 0};` δηλαδή ότι θα επιτελέσετε στη σημαφόρα νούμερο 0 *(πρώτη παράμετρος)*, μια πράξη μείωσης (**DOWN**) κατά -1 *(δεύτερη παράμετρος)*, χωρίς ειδικές σημαίες *(τρίτη παράμετρος)*. Δείτε `man semop` για περισσότερες πληροφορίες.

  *** **Προσοχή:** Οι πίνακες **operationUP** και **operationDOWN** πρέπει να έχουν την τιμή 0 στο πρώτο και τρίτο στοιχείο, όπως ακριβώς αναγράφεται στο εργαστηριακό φυλλάδιο. Διαφορετικά, το πρόγραμμα θα δυσλειτουργήσει. ***

- Αμέσως μετά θα ακολουθούν οι υπόλοιπες ενέργειες ανάγνωσης/εγγραφής ενώ αμέσως μόλις γράφεται η θέση 0 της κοινής μνήμης θα καλείται η `semop(semid,&operationUP[0],1)` και `struct sembuf operationUP[1] = {0, 1, 0};` ώστε να αυξάνεται η σημαφόρος κατά 1.
- Σε μια διεργασία στο τέλος να τοποθετήσετε `sleep(20);` `semctl(semid, 0, IPC_RMID, 0);` προκειμένου να καταστραφεί ο σημαφόρος και να απελευθερωθούν οι πόροι.
- Όταν καταστρέψετε το σημαφόρο εκτυπώστε την τρέχουσα τιμή στη θέση 0.

  ➔ Ύστερα από κάθε κλήση συνάρτησης θα γίνεται έλεγχος αν έχει εκτελεστεί σωστά και θα εκτυπώνεται είτε το μήνυμα επιτυχίας είτε το μήνυμα αποτυχίας καθώς και το PID της διεργασίας που αναφέρει το μήνυμα.

  ➔ Η τιμή που θα πρέπει να τυπώνεται στο τέλος θα πρέπει να είναι 25.

Εκτελέστε ταυτόχρονα και τις δυο διεργασίες *(τη mutex2 δύο φορές)* αφού τις κάνετε compile με

`./c2-mutex & ; ./c2-mutex2.c & ; ./c2-mutex2.c &`

και απαντήστε στο ερώτημα:
Στις διεργασίες του C2 υπάρχει το πρόβλημα των χαμένων συναλλαγών; Αν ναι που οφείλεται αυτό; _____ **(A8)**

Όταν εργαζόμαστε με σημαφόρους SystemV, μπορούμε να χρησιμοποιήσουμε την εντολή του ΛΣ `ipcs` η οποία αναφέρει όλους τους σημαφόρους και τις περιοχές κοινής μνήμης που έχουν δημιουργηθεί, μαζί με τα αναγνωριστικά, τα δικαιώματα πρόσβασης και τον ιδιοκτήτη. Τέλος, μπορούμε να χρησιμοποιήσουμε την εντολή `ipcrm` για να διαγράψουμε *(~καταστρέψουμε)* σημαφόρους ή κοινές περιοχές μνήμης που δε χρειαζόμαστε.

3.2 Οι σημαφόροι POSIX

Εκτός από τους σημαφόρους SystemV παρόμοια λειτουργία μπορεί να επιτευχθεί και με τους σημαφόρους POSIX. Συνήθως τα σύγχρονα λειτουργικά συστήματα υποστηρίζουν και τους δυο τρόπους σημαφόρων. Για να γίνει εύκολος ο διαχωρισμός οι συναρτήσεις POSIX semaphores έχουν ένα underscore στο όνομα. Οι πιο σημαντικές συναρτήσεις POSIX semaphores είναι η `sem_init()`, η `sem_wait()`, η `sem_post()` και η `sem_destroy()`.

Η `sem_init()` αρχικοποιεί και δημιουργεί μια σημαφόρο. Η συνάρτηση `sem_wait()` εκτελεί μια λειτουργία **DOWN** στη συγκεκριμένη σημαφόρο. Η συνάρτηση s`em_post()` εκτελεί μια λειτουργία **UP** στη συγκεκριμένη σημαφόρο. Τέλος, η συνάρτηση `sem_destroy()` καταστρέφει τη συγκεκριμένη σημαφόρο.

Η συνάρτηση δημιουργίας ενός περιγραφέα σημαφόρου `sem_init()` πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A9)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A10)**

Η συνάρτηση εκτέλεσης μιας λειτουργίας DOWN `sem_wait()` πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A11)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A12)**

Η συνάρτηση εκτέλεσης μιας λειτουργίας UP `sem_post()` πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A13)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A14)**

Η συνάρτηση καταστροφής σημαφόρου `sem_destroy()` πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A15)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει στο ΛΣ Freebsd: _____ **(A16)**

3.2.1 Το παραδοτέο C3

(C3) Να δημιουργήσετε ένα πρόγραμμα (**c3.c**) το οποίο ως διεργασία θα δημιουργεί πολλαπλές θυγατρικές διεργασίες που όλες θα χρησιμοποιούν την ίδια περιοχή μνήμης *(πρόβλημα του κρίσιμου τμήματος)* και θα επιτυγχάνουν τον αμοιβαίο αποκλεισμό μέσω των σημαφόρων POSIX.

Συγκεκριμένα:

- Θα ορίσετε μια μεταβλητή τύπου sem_t η οποία θα είναι η σημαφόρος.
- Θα κάνετε κλήση της `sem_init()` με την παράμετρο να είναι κοινόχρηστη και να έχει αρχική τιμή 1 γιατί μόνο 1 διεργασία θέλουμε να έχει πρόσβαση κάθε φορά σε αυτή.
- Θα αρχικοποιεί μια κοινή περιοχή μνήμης μεγέθους 20 bytes και θα την προσαρτά στη διεργασία.
- Θα δημιουργεί με τη fork() 3 θυγατρικές διεργασίες οι οποίες όλες μαζί με τη γονική θα εκτελούν διαρκώς (`while(1)`) τα παρακάτω:
    - Θα εκτελούν τη συνάρτηση `sleep(3)`.
    - Θα εκτελούν τη συνάρτηση **POSIX Semaphore DOWN** για να αποκτήσουν πρόσβαση στο κρίσιμο τμήμα.

- Μόλις εκτελεστεί η συνάρτηση θα αναφέρεται το μήνυμα "`(PID) Είσοδος στο κρίσιμο τμήμα της διεργασίας με (count)`", με PID το pid της διεργασίας που εκτύπωσε το μήνυμα, και count ένας μετρητής που θα διατηρεί η κάθε διεργασία και θα τον αυξάνει κάθε φορά που μπαίνει στο κρίσιμο τμήμα.
- Θα διαβάζουν την τιμή στη κοινή θέση 0 και θα την αυξάνουν κατά 5 αναφέροντας το μήνυμα "`(PID) read ORIGINAL and changed value to FINAL`", όπου PID, ORIGINAL και FINAL το process ID η αρχική και η τελική τιμή που γράφτηκε αντίστοιχα.
- Μόλις γραφεί η νέα τιμή στην ίδια θέση *(θέση 0)* τότε θα απελευθερώνεται το κρίσιμο τμήμα με τη συνάρτηση POSIX Semaphore UP.
- Μόλις εκτελεστεί η συνάρτηση θα αναφέρεται το μήνυμα "`(PID) Έξοδος από το κρίσιμο τμήμα της διεργασίας`", με PID το pid της διεργασίας που εκτύπωσε το μήνυμα.

➔ Ύστερα από κάθε **`printf()`** να εκτελείται το **`fflush(NULL)`** προκειμένου να εκτυπώνεται το μήνυμα άμεσα, διαφορετικά θα πρέπει να γεμίσει το buffer της γραμμής και ενδέχεται να καθυστερεί η εκτύπωση.

➔ Σημειώστε ότι επειδή δημιουργούνται θυγατρικές διεργασίες με τη **`fork()`** αντιγράφονται οι μεταβλητές, οπότε η διεύθυνση της κοινής περιοχής μνήμης είναι κοινή και στις θυγατρικές διεργασίες και δεν απαιτείται κάποια άλλη συνάρτηση για να συνδεθούν οι θυγατρικές στην κοινή περιοχή μνήμης.

➔ Να εκτελέσετε το πρόγραμμα και να απαντήσετε:

Όλες οι διεργασίες *(γονική+θυγατρικές)* καταφέρνουν να μπουν στο κρίσιμο τμήμα; Υπάρχει δικαιοσύνη ως προς τον αριθμό που έχει εισέλθει η κάθε διεργασία στο κρίσιμο τμήμα ή κάποια διεργασία έχει εισέλθει πολλές περισσότερες φορές; Γιατί;
_____ **(A17)**

4. Διαχείριση αρχείων

Στο POSIX υπάρχει ένα πλήθος συναρτήσεων χειρισμού αρχείων. Οι βασικές συναρτήσεις περιγράφονται στη συνέχεια. Με τη συνάρτηση **`open()`** ανοίγουμε ένα αρχείο ή ένα άλλο αντικείμενο *(όπως socket)* και μας επιστρέφεται ένας περιγραφέας αρχείου. Με τη συνάρτηση **`close()`** κλείνουμε το συγκεκριμένο περιγραφέα. Με τη συνάρτηση **`read()`** διαβάζουμε από ένα συγκεκριμένο περιγραφέα. Με τη συνάρτηση **`write()`** γράφουμε σε ένα συγκεκριμένο περιγραφέα. Με τη συνάρτηση **`stat()`** επιστρέφονται τα μεταδεδομένα που συνδέονται με ένα συγκεκριμένο αρχείο. Τέλος, με τη συνάρτηση **`lseek()`** τοποθετούμε το pointer εγγραφής ανάγνωσης σε ένα συγκεκριμένο σημείο.

Η συνάρτηση επιστροφής μεταδεδομένων αρχείου `stat()` *(προσοχή τμήμα 2 manpage)* πόσα ορίσματα δέχεται στην είσοδο και τι επιστρέφει; _____ **(A18)**

Αναφέρετε δυο περιπτώσεις στις οποίες η ανωτέρω συνάρτηση θα αποτύχει: _____ **(A19)**

Αν θέλουμε να χρησιμοποιήσουμε παραμέτρους της γραμμής εντολών στη διεργασία μας θα χρησιμοποιήσετε τα ορίσματα `argc,argv` στη main. Δηλαδή θα δηλώσετε `int main (int argc, char *argv[])`, ενώ το `argc` θα έχει τον αριθμό των ορισμάτων, ενώ το `argv[i]` θα έχει την τιμή του `i` ορίσματος *(δηλαδή της `i` παραμέτρου)*.

4.1 Το παραδοτέο C4

(C4) Να δημιουργήσετε ένα πρόγραμμα **c4.c** το οποίο ως διεργασία θα ανοίγει το αρχείο που δίνει ο χρήστης θα διαβάζει το περιεχόμενο και θα το αντικαθιστά με αυτό που δίνει ο χρήστης. Συγκεκριμένα:

- δέχεται μια και μόνο μια παράμετρο στη γραμμή εντολών. Θα αναφέρει είτε την παράμετρο που έδωσε ο χρήστης είτε ότι δεν έδωσε παράμετρο και θα σταματάει η εκτέλεση.
- Θα χρησιμοποιεί τη `stat()` για να ελέγχει αν υπάρχει το συγκεκριμένο αρχείο που έχει δοθεί ως πρώτη παράμετρο.
- Αν υπάρχει το αρχείο:
    - θα ανοίγεται με την `open()` για ανάγνωση και εγγραφή
    - Θα καθορίζει και θα εκτυπώνει το μέγεθος του αρχείου είτε με τα στοιχεία που επιστρέφει η `stat()` είτε με το να εκτελείται συνεχώς η `while(read())` για να διαβάζει 1 byte, αυξάνοντας ένα μετρητή.
    - Επίσης θα εκτυπώνονται τα περιεχόμενα του αρχείου.
    - θα τοποθετεί το pointer εγγραφής στη θέση 0
- Αν δεν υπάρχει το αρχείο
    - θα ανοίγει με την `open()` για δημιουργία και εγγραφή
    - θα τοποθετεί το pointer εγγραφής στη θέση 0
- Στη συνέχεια, θα ρωτάει το χρήστη να πληκτρολογήσει μια πρόταση.
- Η πρόταση αυτή θα τοποθετηθεί με τη `write()` *{προσοχή: όχι με άλλη συνάρτηση}* στο αρχείο.

- Θα κλείνει το αρχείο, αφού αναφερθεί το μέγεθος του νέου αρχείου.

Μέσα στο πλήθος των συναρτήσεων χειρισμού αρχείων και καταλόγου είναι η συνάρτηση `unlink()` η οποία διαγράφει ένα αρχείο. Η `basename()` επιστρέφει το όνομα ενός αρχείου μόνο από την πλήρη διαδρομή *(δηλαδή αν δοθεί το **/usr/local/etc/passwd** θα επιστρέψει μόνο το **passwd**)*. Η συνάρτηση `mkstemp()` ανοίγει ένα προσωρινό αρχείο για εγγραφή το οποίο είναι μοναδικό με συνέπεια να μη δημιουργείται πρόβλημα με ένα αρχείο που υπάρχει.

Αναφέρετε τι παραμέτρους δέχεται και τι επιστρέφει η συνάρτηση `unlink()` **(A20)**

4.2 Τα παραδοτέα C5 έως C7

(C5) Να δημιουργήσετε ένα πρόγραμμα **c5.c** το οποίο ως διεργασία θα δημιουργεί ένα αρχείο με μέγεθος όσο έχει δηλώσει ο χρήστης στη γραμμή εντολών. Συγκεκριμένα:

- δέχεται δυο και μόνο δυο παραμέτρους στη γραμμή εντολών. Θα αναφέρει είτε τις παραμέτρους που έδωσε ο χρήστης είτε ότι δεν έδωσε σωστές παραμέτρους και θα σταματάει η εκτέλεση. Η πρώτη παράμετρος αντιστοιχεί σε ένα όνομα αρχείου, ενώ η δεύτερη στο επιθυμητό μέγεθος σε Bytes.
- Θα χρησιμοποιεί τη `stat()` για να ελέγχει αν υπάρχει το συγκεκριμένο αρχείο που έχει δοθεί ως πρώτη παράμετρο.
- Αν υπάρχει το αρχείο, το πρόγραμμα θα ρωτάει αν επιθυμεί ο χρήστης να διαγράψει το συγκεκριμένο αρχείο. Αν πατήσει y τότε θα διαγράφεται το αρχείο με τη χρήση της `unlink().` Αν πατήσει n τότε δε θα διαγράφεται το αρχείο και η διεργασία θα σταματάει την εκτέλεση.
- Θα ανοίγει με την `open()` για δημιουργία και εγγραφή το αρχείο που έχει δοθεί ως πρώτη παράμετρος.
- Θα εκτελεί τη συνάρτηση `lseek()` με πρώτη παράμετρο το περιγραφέα αρχείου, με δεύτερη παράμετρο το μέγεθος αρχείου-1 και τρίτη παράμετρο το SEEK_SET.
- Θα γράφει με τη write 1 byte (το '\0') για το τέλος του αρχείου.
- Θα κλείνει το αρχείο.

Εκτελέστε το πρόγραμμα και επιβεβαιώστε την ορθή λειτουργία. Όταν σταματάει η εκτέλεση να δίνεται ls -l για να διαπιστώνετε αν το μέγεθος του αρχείου που έχετε δημιουργήσει συμφωνεί με τα δεδομένα που δώσατε.

(C6) Να δημιουργήσετε ένα πρόγραμμα **c6.c** το οποίο ως διεργασία θα δημιουργεί 10 νήματα τα οποία όλα θα προσπαθούν να αυξήσουν μια φορά μόνο μια global μεταβλητή κατά 1. Θα υπάρχει αμοιβαίος αποκλεισμός για να μη χάνεται καμία συναλλαγή. Συγκεκριμένα:

- θα υπάρχει μια μεταβλητή με καθολική τοπικότητα και αρχική τιμή 0 *(έξω από το* `main()` *)*
- θα δημιουργείται ένας σημαφόρος POSIX με αρχική τιμή 1.
- θα δημιουργούνται 10 νήματα τα οποία όλα θα προσπαθούν να εκτελέσουν τη συνάρτηση increment_shared, η οποία θα κάνει το εξής:
    - στην αρχή θα κάνει `sleep(randomval)` όπου η τιμή randomval υπολογίζεται με κλήση της συνάρτησης `random()` ή `rand()`, αφού γίνει το σωστό seeding της συνάρτησης *(όπως αναγράφεται στο manpage)*, και στη συνέχεια χρησιμοποιήσουμε την τιμή που θα βγει με `modulo 10` *(ώστε να προκύψουν τιμές από 0 έως 9)*.
    - θα κάνει down το σημαφόρο για να εισέλθει στην κρίσιμη περιοχή.
    - Θα διαβάζει και θα αυξάνει την κοινή μεταβλητή κατά 1
    - θα εκτυπώνει το μήνυμα "`TID entered critical region and changed value to XXX`", όπου TID το threadID ενώ το XXX είναι η νέα τιμή που γράφτηκε.
    - θα κάνει up το σημαφόρο για να απελευθερώσει την κρίσιμη περιοχή.
    - Θα σταματάει η εκτέλεση του συγκεκριμένου νήματος.
- Το κυρίως πρόγραμμα θα περιμένει την ολοκλήρωση και των 10 νημάτων και στη συνέχεια θα εκτυπώνει την τελική τιμή της κοινής μεταβλητής `(Final Shared Value is: XX)`. Η τιμή αυτή θα πρέπει να είναι 10.

(C7) Να δημιουργήσετε ένα πρόγραμμα με όνομα **c7.c** το οποίο ως διεργασία θα δημιουργεί ένα μοναδικό προσωρινό αρχείο *(κλήση της* `mkstemp()` *)* στο οποίο θα τοποθετεί το όνομά σας *(κλήση της* `write()` *)* θα περιμένει 10 sec για να δείτε τα περιεχόμενα του αρχείου και στη συνέχεια θα καλεί close για να κλείσει το προσωρινό περιγραφέα αρχείου και στη συνέχεια να διαγραφεί αυτόματα. Συγκεκριμένα:

- το `main()` θα έχει τις δηλώσεις των `argc, argv` προκειμένου να βρούμε το όνομα της διεργασίας.
- Η μεταβλητή `argv[0]` διατηρεί το πλήρες όνομα της διεργασίας όπως την εκτελέσαμε. Δηλαδή, αν έχουμε δώσει **/usr/local/c7-mkstemp** τότε αυτή την τιμή θα έχει η `argv[0]`.
- Θα καλείται η `basename()` με όρισμα το `argv[0]` για να βρούμε μόνο το όνομα της διεργασίας.

- Θα εκτυπώνεται το όνομα της διεργασίας *(δηλαδή, χωρίς την πλήρη διαδρομή).*
- Θα καλείται η **`strcat()`**, ώστε να προστεθεί το string `".XXXXX"` στο τέλος του ονόματος. Δηλαδή, αν το όνομα είναι c7-mkstemp τότε το νέο όνομα που θα δημιουργείται θα είναι το `c7-mkstemp.XXXXX` . Χρειαζόμαστε το **XXXXX** γιατί θα δοθεί ως template στη συνάρτηση δημιουργίας προσωρινού αρχείου. Μπορείτε να χρησιμοποιήσετε και μεγαλύτερο αριθμό από X ώστε να δημιουργείται ακόμη πιο random όνομα αρχείου.
- Θα καλείται η **`mkstemp()`** για να δημιουργηθεί το προσωρινό αρχείο με το template που έχει δημιουργηθεί πριν.
- Θα τοποθετείται με τη **`write()`** το ονοματεπώνυμο σας στο αρχείο.
- Θα κάνει sleep(15)
- Θα κλείνει με τη **`close()`** το αρχείο. Ως εκ τούτου θα διαγράφεται αυτόματα το αρχείο.
- Θα εκτυπώνει το μήνυμα “Closed Descriptor”.

➔ Να φροντίστε να υπάρχουν οι κατάλληλοι έλεγχοι στη **`mkstemp()`**, **`write()`**, **`close()`** με τα κατάλληλα μηνύματα.

➔ Όπως πάντα, να κάνετε man την κάθε συνάρτηση που χρησιμοποιείτε, ώστε να βρείτε τα include αρχεία που απαιτεί. Αν δεν τοποθετήσετε τα κατάλληλα **`#include`** τότε δε θα μπορεί να γίνει compile το πρόγραμμα, ή θα αναφέρει warnings και θα δυσλειτουργεί.

Κάντε compile και εκτελέστε το ως **./c7 &** για να τοποθετηθεί στο παρασκήνιο. Δώστε μετά **ls -la** και θα δείτε ότι θα έχει δημιουργηθεί το προσωρινό αρχείο **c7.??** όπου **??** είναι δυο τυχαίοι χαρακτήρες. Δώστε **cat c7.??** για να δείτε το περιεχόμενο. Μετά από λίγα δευτερόλεπτα αφού εμφανιστεί το μήνυμα `"Closed Descriptor"` δώστε πάλι **ls -la** και θα δείτε ότι θα έχει διαγραφεί το αρχείο.

5. Το πρόγραμμα moss-memory

Προσομοίωση του συστήματος διαχείρισης μνήμης του λειτουργικού συστήματος με το πρόγραμμα moss-memory. Ερμηνεία Page-faults.

Ο προσομοιωτής **moss-memory** παρουσιάζει τη συμπεριφορά ενός συστήματος σελιδοποίησης εικονικής μνήμης όταν συμβαίνουν σφάλματα σελίδας ***(page faults)***. Περιλαμβάνει ένα γραφικό περιβάλλον, το οποίο επιτρέπει στο χρήστη την παρακολούθηση της λειτουργίας των αλγορίθμων αντικατάστασης σελίδας *(page replacement algorithms).*

Ο προσομοιωτής διαβάζει:

- την αρχική κατάσταση ενός πίνακα σελίδων.
- μια ακολουθία εντολών εικονικής μνήμης *(virtual memory).*

Δημιουργεί *(προαιρετικά)* ένα αρχείο παρακολούθησης *(log file)*, το οποίο παρουσιάζει το αποτέλεσμα της εκτέλεσης της κάθε εντολής.

Ο χρήστης μπορεί:

- να δει βηματικά την προσομοίωση, ώστε να παρακολουθήσει τη δράση σε κάθε εκτέλεση εντολής
- ή να «τρέξει» την προσομοίωση μέχρι αυτή να ολοκληρωθεί.
- Όταν όλες οι εντολές εκτελεστούν η προσομοίωση σταματάει.

Ο προσομοιωτής είναι γραμμένος σε JAVA οπότε απαιτείται η εγκατάσταση στον υπολογιστή του JAVA RUNTIME.

Για την εκτέλεση του προγράμματος ανοίξτε το `cmd.exe`, μεταβείτε στον κατάλογο που έχετε εγκαταστήσει το moss-memory και εισάγετε την επόμενη εντολή:

```
java MemoryManagement commands memory.conf
```

Το πρόγραμμα θα εμφανίσει ένα παράθυρο, επιτρέποντας σας να τρέξετε τον προσομοιωτή. Στην αριστερή πλευρά του παραθύρου εμφανίζονται 2 στήλες με “πλήκτρα” σελίδων. Στη δεξιά πλευρά υπάρχει μια οθόνη πληροφοριών. Στο πάνω μέρος υπάρχουν πλήκτρα, που καθορίζουν την εκτέλεση του προσομοιωτή *(run, step, reset, exit).*

5.1 Τα αρχεία commands και memory.conf

Το **αρχείο commands** είναι το αρχείο όπου καθορίζεται η ακολουθία των εντολών μνήμης που πρόκειται να εκτελεστούν. Κάθε εντολή είναι είτε μια **READ**, είτε μια **WRITE** εντολή και περιλαμβάνει μια διεύθυνση εικονικής μνήμης που πρέπει να γραφεί ή να διαβαστεί. Αν η διεύθυνση εικονικής μνήμης είναι παρούσα στη φυσική μνήμη, η λειτουργία *(εντολή)* θα εκτελεστεί επιτυχώς, διαφορετικά θα προκληθεί ένα σφάλμα σελίδας *(page fault).*

Το **αρχείο memory.conf** είναι το αρχείο που χρησιμοποιείται για να καθορίσει τα αρχικά περιεχόμενα του χάρτη εικονικής μνήμης *(virtual memory map)*, δηλαδή ποιες σελίδες της εικονικής μνήμης έχουν αντιστοιχισθεί σε σελίδες της φυσικής μνήμης. Επίσης, καθορίζει άλλου είδους πληροφορίες καθορισμού της εκτέλεσης, π.χ. αν η λειτουργία αποθηκευτεί σε αρχείο.

Στην οθόνη του προσομοιωτή φαίνονται τα παρακάτω πεδία:

- **time:** Αριθμός σε “ns” από την αρχή της προσομοίωσης.
- **instruction:** READ ή WRITE. Η τελευταία εντολή που εκτελέστηκε.
- **address:** Η διεύθυνση εικονικής μνήμης της εντολής που εκτελέστηκε τελευταία.
- **page fault:** εάν η τελευταία εντολή προκάλεσε σφάλμα σελίδας (YES / NO).
- **virtual page:** Ο αριθμός της σελίδας εικονικής μνήμης που παρουσιάζεται στα παρακάτω πεδία. Αυτός μπορεί να είναι είτε η πιο πρόσφατη σελίδα εικονικής μνήμης που χρησιμοποίησε ο προσομοιωτής, είτε η τελευταία σελίδα της οποίας πατήθηκε το πλήκτρο page n.
- **physical page:** Η σελίδα φυσικής μνήμης γι αυτή τη σελίδα εικονικής μνήμης *(αν υπάρχει)*. Το -1 δείχνει ότι καμία σελίδα φυσικής μνήμης δεν είναι συνδεδεμένη με αυτή τη σελίδα εικονικής μνήμης.
- **R:** Αν έχει διαβαστεί η σελίδα είναι 1, αν όχι είναι 0.
- **M:** Αν έχει τροποποιηθεί η σελίδα είναι 1, αν όχι είναι 0.
- **inMemTime:** Αριθμός σε “ns” που δηλώνει πριν από πόσο χρόνο δεσμεύτηκε η φυσική σελίδα σε αυτή την εικονική σελίδα.
- **lastTouchTime:** Αριθμόςσε “ns” που δηλώνει πριν από πόσο χρόνο τροποποιήθηκε η φυσική σελίδα σε αυτή την εικονική σελίδα.
- **page fault:** Εάνη τελευταία εντολή προκάλεσε σφάλμα σελίδας.
- **low:** Η“χαμηλή” διεύθυνση εικονικής μνήμης για τη συγκεκριμένη σελίδα εικονικής μνήμης.
- **high:** Η“υψηλή” διεύθυνση εικονικής μνήμης για τη συγκεκριμένη σελίδα εικονικής μνήμης.

Καθορίζονται 2 λειτουργίες: READ, WRITE

```
READ: READ address    READ random
WRITE: WRITE address  WRITE random
```

address: ο αριθμός της διεύθυνσης εικονικής μνήμης. Προαιρετικά προηγείται αυτού μια από τις λέξεις κλειδιά bin, oct, hex. Αν δεν προηγείται κάτι από τα παραπάνω, ο αριθμός θεωρείται δεκαδικός.

random: θα δημιουργήσει μια εικονική διεύθυνση μνήμης.

Το **αρχείο memory.conf** χρησιμοποιείται για να καθορίσει τα αρχικά περιεχόμενα του χάρτη εικονικής μνήμης *(virtual memory map)*, δηλαδή ποιες σελίδες της εικονικής μνήμης έχουν αντιστοιχιστεί σε σελίδες της φυσικής μνήμης. Επίσης καθορίζει άλλου είδους πληροφορίες προσδιορισμού της εκτέλεσης, π.χ. αν η λειτουργία θα αποθηκευτεί σε αρχείο.

Η **εντολή memset** χρησιμοποιείται για να αρχικοποιήσει κάθε είσοδο του χάρτη εικονικής μνήμης, και ακολουθείται από 6 ακέραιες τιμές:

- Τον αριθμό σελίδας εικονικής μνήμης που θα αρχικοποιήσει.
- Τη φυσική σελίδα που σχετίζεται με αυτή τη σελίδα εικονικής μνήμης *(-1 αν δεν αντιστοιχίζεται καμία σελίδα).*
- Αν έχει πραγματοποιηθεί ανάγνωση της σελίδας (**R**) *(0=όχι, 1=ναι).*
- Αν η σελίδα έχει τροποποιηθεί (**M**) *(0=όχι, 1=ναι).*
- Πόσο χρόνο βρίσκεται η σελίδα στη μνήμη *(σε ns).*
- Την τελευταία χρονική στιγμή που τροποποιήθηκε η σελίδα *(σε ns).*

Οι **2** πρώτες παράμετροι καθορίζουν την αντιστοίχιση μεταξύ μιας σελίδας εικονικής μνήμης και μιας σελίδας φυσικής μνήμης, αν υπάρχει.

Οι **4** τελευταίες παράμετροι είναι τιμές που μπορούν να χρησιμοποιηθούν από έναν αλγόριθμο αντικατάστασης σελίδας.

Παράδειγμα: `memset 34 23 0 0 0 0`

Καθορίζει ότι η σελίδα εικονικής μνήμης 34 αντιστοιχίζεται στη σελίδα φυσικής μνήμης 23 και ότι η σελίδα δεν έχει αναγνωσθεί ούτε τροποποιηθεί.

Προσοχή:

- Κάθε φυσική σελίδα πρέπει να αντιστοιχεί σε μια μόνο εικονική.
- Το πλήθος των σελίδων εικονικής μνήμης είναι 64: 0..63 *(αναφερόμαστε στον προσομοιωτή).*
- Το πλήθος των σελίδων φυσικής μνήμης δεν μπορεί να ξεπερνά τις 64 σελίδες: 0..63.
- Αν μια σελίδα εικονικής μνήμης δεν καθορίζεται σε κάποια εντολή memset, αυτό σημαίνει ότι δεν έχει αντιστοιχιστεί σε σελίδα φυσικής μνήμης.

Άλλοι παράμετροι του αρχείου είναι:

enable_logging: Καθορίζει ν θα εμφανίζεται ή όχι μήνυμα για την εκτέλεση της κάθε εντολής READ ή WRITE. προκαθορισμένη τιμή FALSE.

log_file: Το όνομα του αρχείου όπου θα γράφονται τα μηνύματα. Αν δε δίνεται όνομα αρχείου, τότε τα μηνύματα γράφονται στην οθόνη.

pagesize: Το μέγεθος της σελίδας σε bytes, δοσμένο ως δύναμη του 2.

addressradix: Η βάση στην οποία παρουσιάζονται οι αριθμοί. Η προκαθορισμένη βάση είναι το **2**, μπορεί όμως, να είναι και το **8** *(octal)*, **10** *(decimal)* ή **16** *(hexadecimal)*

5.2 Ερωτήσεις S1 έως S6

Εξαγάγετε τα αρχεία της άσκησης σε ένα κατάλογο της επιλογής σας.

(S1) Δώστε **notepad memory.conf** για να δείτε το όνομα του αρχείου καταγραφής.

Εκτελέστε την προσομοίωση όπως φαίνεται παραπάνω. Πατήστε run και όταν ολοκληρωθεί exit. Σημειώστε το όνομα του αρχείου καταγραφής.

(S2) Ανοίξτε το αρχείο καταγραφής με την εντολή **notepad αρχείο_καταγραφής**

Έχει δημιουργηθεί πουθενά Page fault;

Στο τέλος του αρχείου commands προσθέστε την εντολή **WRITE random**

(S3) Τρέξτε την προσομοίωση και σημειώστε αν δημιουργηθεί page fault & τη διεύθυνση.

Στο τέλος του αρχείου Commands προσθέστε την εντολή **READ 10**.

(S4) Τρέξτε την προσομοίωση πάλι και σημειώστε αν δημιουργηθεί page fault & τη διεύθυνση του:

Αντιγράψτε το αρχείο commands στο commands1: `copy commands commands1`

Τροποποιήστε το αρχείο commands1ώστε να υπάρχει η εξής σειρά εντολών.

```
READ 4,  READ 19, READ 600000 , WRITE 100 , WRITE 200, WRITE
800000 , READ 20000, READ 400
```

(S5) Εκτελέστε τον προσομοιωτή και ελέγξτε τις σελίδες στις οποίες βρίσκονται οι διευθύνσεις εικονικής μνήμης 3 και 6 του παραπάνω αρχείου. Σημειώστε που έχει γίνει Page fault και γιατί *(ανοίξτε το tracefile)*

(S6) Ομοίως για το παράδειγμα με τις εξής εντολές *(τροποποιήστε κατάλληλα το αρχείο commands1)*:

```
READ 140000, READ 190000, READ 700000, READ 900000, WRITE 100,
READ 17000, READ 20000
```
