Εργαστήριο 12

Συγχρονισμός Νημάτων (χρήση pthread_mutex_t, pthread_cond_t)

Να γράψετε πρόγραμμα που να δημιουργεί 1 νήμα. Έτσι στο πρόγραμμα σας θα υπάρχουν 2 νήματα (το ένα νήμα είναι το αρχικό νήμα που εκτελεί τη main). Το ένα νήμα (αρχικό) θα παίζει το ρόλο του «παραγωγού» το οποίο θα δημιουργεί και θα βάζει 10 κόμβους (nodes) σε μια στοίβα (υλοποιημένη σαν συνδεδεμένη λίστα). Στον κάθε κόμβο θα αποθηκεύεται ένας αριθμός από το 1 έως το 10. Το άλλο νήμα σαν «καταναλωτής» θα αφαιρεί τους κόμβους από την στοίβα και θα εμφανίζει τα περιεχόμενά τους στην οθόνη. Στην υλοποίηση αυτή, ο μοιραζόμενος (από τα νήματα) πόρος είναι η λίστα.

Σημειώσεις:

- Κάθε κόμβος θα αποτελείται από μια δομή όπως φαίνεται πιο κάτω:

```
struct node {
     int n_number;
     struct node *n_next
}
```

- Κάθε νήμα θα πρέπει να κλειδώνει με χρήση δυαδικών σηματοφόρων ή με άλλα λόγια μεταβλητών αμοιβαίου αποκλεισμού (**mut**ual **ex**clusion) τα κοινόχρηστα δεδομένα προτού αποκτήσει πρόσβαση πάνω τους.
- Κάθε φορά που το αρχικό νήμα θα τοποθετεί στη στοίβα ένα κόμβο, θα πρέπει να σηματοδοτεί μια συνθήκη (pthread_cond_t) στην οποία το άλλο νήμα περιμένει. Περισσότερες πληροφορίες για τις μεταβλητές συνθήκης δείτε πιο κάτω.

Οι μεταβλητές συνθήκης (condition variables), παρέχουν ένα άλλο τρόπο συγχρονισμού νημάτων. **Σε αντίθεση με τις μεταβλητές αμοιβαίου αποκλεισμού, οι οποίες χρησιμοποιούνται για το συγχρονισμό των νημάτων ελέγχοντας την πρόσβαση στα προστατευμένα δεδομένα, οι μεταβλητές συνθήκης επιτρέπουν στα νήματα να συγχρονισθούν με βάση τη τιμή των δεδομένων αυτών.** Χωρίς την ύπαρξη των μεταβλητών συνθήκης, ο προγραμματιστής θα έπρεπε να ορίζει τα νήματα να ελέγχουν συνέχει τη τιμή μιας κοινής μεταβλητής για μια συγκεκριμένη τιμή ώστε να ελέγξουν αν έχει ικανοποιηθεί μια συγκεκριμένη συνθήκη. Ο τρόπος αυτός είναι γνωστός ως απασχολημένη αναμονή (busy wait) και καταναλώνει πολύ περισσότερους πόρους συστήματος από ότι είναι αναγκαίο αφού όλα τα νήματα είναι απασχολημένα κάνοντας συνεχείς ελέγχους. **Οι μεταβλητές συνθήκης επιτρέπουν να επιτευχθεί ο σκοπός αυτός χωρίς τα νήματα να τίθενται σε συνεχή αναμονή. Η μεταβλητή συνθήκης χρησιμοποιείται πάντα σε συνεργασία με μια μεταβλητή αμοιβαίου αποκλεισμού.**
Για τη δημιουργία και την καταστροφή μιας μεταβλητής συνθήκης χρησιμοποιούνται οι παρακάτω συναρτήσεις:

```
int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t
*attr);
int pthread_cond_destroy(pthread_cond_t *cond);
```

Αφού μια μεταβλητή συνθήκης δηλωθεί ως `pthread_cond_t`, πρέπει να αρχικοποιηθεί με την `pthread_cond_init` πριν την χρήση της. Μετά τη χρήση της μεταβλητής συνθήκης, οι πόροι που έχουν δεσμευθεί μπορούν να ελευθερωθούν με τη συνάρτηση `pthread_cond_destroy`.

Για την αναμονή μέχρι την ικανοποίηση μια συνθήκης και για τη σηματοδότησή της χρησιμοποιούνται οι πιο κάτω συναρτήσεις:

```
int pthread_cond_wait(pthread cond t *cond, pthread mutex t *mutex)
int pthread_cond_signal(pthread cond t *cond)
int pthread_cond_broadcast(pthread cond t *cond)
```

Ένα νήμα-καταναλωτής που θέλει να αποκτήσει πρόσβαση στον μοιραζόμενο πόρο για τον «καταναλώσει» πρέπει να ελέγξει την τιμή του πόρου ή αν υπάρχει πόρος για να καταναλώσει. Για παράδειγμα, στην άσκησή μας, ο καταναλωτής πρέπει να ελέγξει αν υπάρχει κόμβος στην λίστα. Αν όχι πρέπει να μπει σε αναμονή.
Η συνάρτηση `pthread_cond_wait` θέτει το νήμα (έστω Α – καταναλωτής) που την καλεί σε αναμονή μέχρι τη σηματοδότησή της μεταβλητής συνθήκης `cond`. Η συνάρτηση αυτή πρέπει να κληθεί αφού η μεταβλητή αμοιβαίου αποκλεισμού `mutex` έχει κλειδωθεί (`pthread_mutex_lock`). Όταν καλεστεί η συνάρτηση `pthread_cond_wait`, η μεταβλητή `mutex` ξεκλειδώνεται αυτόματα και το νήμα Α που την κάλεσε μπλοκάρει.
Έτσι μπορεί άλλο νήμα (έστω Β – παραγωγός) να κλειδώσει τη mutex (`pthread_mutex_lock`) για να έχει πρόσβαση στον κοινό πόρο. Όταν το νήμα Β ικανοποιήσει τη συνθήκη την οποία αναμένει το μπλοκαρισμένο νήμα Α (π.χ. βάλει κόμβο μέσα στη λίστα) τότε το νήμα Β πρέπει να καλέσει τη συνάρτηση `pthread_cond_signal` ή `pthread_cond_broadcast`. Για να κληθεί οποιαδήποτε από τις 2 αυτές συναρτήσεις πρέπει η μεταβλητή `mutex` (στο νήμα Β) να είναι κλειδωμένη. Η συνάρτηση `pthread_cond_signal` στέλνει ένα σήμα στο μπλοκαρισμένο νήμα Α για να μπορέσει να ξεμπλοκάρει. Για να ξεμπλοκάρει το νήμα Α πρέπει πρώτα το νήμα Β μετά την κλήση της `pthread_cond_signal` να ξεκλειδώσει τη mutex (`pthread_mutex_unlock`). Όταν γίνει αυτό, τότε η `mutex` ξανακλειδώνεται αυτόματα στο νήμα Α και μπορεί να συνεχίσει για να έχει πρόσβαση στον κοινό πόρο. Όταν τελειώσει το νήμα Α τότε πρέπει να ξεκλειδώσει τη μεταβλητή `mutex` (`pthread_mutex_unlock`) με τη σειρά του.
Η κλήση της συνάρτησης `pthread_cond_signal` ξεμπλοκάρει τουλάχιστον ένα από τα νήματα που είναι μπλοκαρισμένα στη μεταβλητή συνθήκης `cond` (αν υπάρχουν νήματα μπλοκαρισμένα στη μεταβλητή `cond`).
Η κλήση της συνάρτησης `pthread_cond_broadcast` ξεμπλοκάρει όλα τα νήματα που είναι στη τρέχουσα φάση μπλοκαρισμένα στη μεταβλητή συνθήκης `cond`.

Δείτε παράδειγμα πιο κάτω.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>

// compile with gcc mutexcond.c -o mutexcond -Wall -lpthread

pthread_cond_t cond;
pthread_mutex_t mutex;

// shared variable
int x = 0;
```

```
void *thread(void *v) {
      printf("Consumer thread created.\n");
      printf("Locking and waiting.\n");
      pthread_mutex_lock(&mutex);
      if (x == 0)
            pthread_cond_wait(&cond, &mutex);

      printf("I've been unlocked and consumed the shared variable
x=%d.\n", x);
      // shared variable has been consumed
      x -= 10;
      pthread_mutex_unlock(&mutex);
      return NULL;
}

int main() {
      char cmd[10];
      pthread_t t;
      // initializes the mutex
      // upon successful initialization,
      // the state of the mutex becomes initialized and unlocked
      pthread_mutex_init(&mutex, NULL);
      // initializes the condition variable
      // upon successful initialization,
      // the state of the condition variable becomes initialized
      pthread_cond_init(&cond, NULL);


      printf("Type produce to produce a value in the shared
variable.\n");
      printf("Type consume to generate a consumer to consume the value
of the shared variable.\n");
      printf("Type quit to/ exit program.\n");
      while(fscanf(stdin, "%s", cmd) != EOF) {
            if(strcmp(cmd, "produce") == 0) {
                  printf("Shared variable loaded.\n");
                  // lock mutex to produce a value in the shared variable
                  pthread_mutex_lock(&mutex);
                  x += 10;
                  // unblock one thread blocked on the condition variable
                  pthread_cond_signal(&cond);
                  // unlock mutex
                  pthread_mutex_unlock(&mutex);
            } else if(strcmp(cmd, "consume") == 0) {
                  // create a thread to consume shared variable value
                  pthread_create(&t, NULL, thread, NULL);
            } else if(strcmp(cmd, "quit") == 0) {
                  break;
            }
      }
      pthread_mutex_destroy(&mutex);
      pthread_cond_destroy(&cond);
      return 0;
}
```

Παράδειγμα Εκτέλεσης (με κόκκινο η είσοδος του χρήστη). Με κίτρινο highlight το νήμα-παραγωγός (πατέρας) και με πράσινο highlight το νήμα-καταναλωτής.

```
Type produce to produce a value in the shared variable.
Type consume to generate a consumer to consume the value of the shared
variable.
Type quit to/ exit program.
produce
Shared variable loaded.
produce
Shared variable loaded.
produce
Shared variable loaded.
consume
Consumer thread created.
Locking and waiting.
I've been unlocked and consumed the shared variable x=30.
consume
Consumer thread created.
Locking and waiting.
I've been unlocked and consumed the shared variable x=20.
consume
Consumer thread created.
Locking and waiting.
I've been unlocked and consumed the shared variable x=10.
consume
Consumer thread created.
Locking and waiting.
produce
Shared variable loaded.
I've been unlocked and consumed the shared variable x=10.
consume
Consumer thread created.
Locking and waiting.
produce
Shared variable loaded.
I've been unlocked and consumed the shared variable x=10.
quit
```

Μέθοδοι διαχείρισης νημάτων σε πολυνηματικές εφαρμογές

Στα πλαίσια του εργαστηρίου αυτού θα θεωρήσουμε ότι η πολυνηματική εφαρμογή που μας ενδιαφέρει είναι ένας εξυπηρετητής (server). Ο εξυπηρετητής θα πρέπει να είναι σε θέση να εξυπηρετεί «ταυτόχρονα» πολλές αιτήσεις από πελάτες. Δεν πρέπει, δηλαδή, να τελειώσει πρώτα με την εξυπηρέτηση μιας αίτησης και μετά να δέχεται νέες. Για να το πετύχουμε αυτό, θα πρέπει να εκμεταλλευτούμε τη δυνατότητα ύπαρξης πολλών νημάτων μέσα στη διεργασία του εξυπηρετητή. Στο κείμενο που ακολουθεί περιγράφονται δύο διαφορετικοί τρόποι με τους οποίους μπορούμε να προσεγγίσουμε τη διαχείριση των νημάτων:

1. **Τεχνική Α:** Δημιουργία ενός καινούργιου νήματος για την επεξεργασία μιας αίτησης, ενώ το αρχικό νήμα περιμένει νέες αιτήσεις
2. **Τεχνική Β:** Δημιουργία ενός thread-pool για τη διαχείριση των νέων αιτήσεων. Για την περίπτωση αυτή θα δούμε δυο διαφορετικές ιδέες υλοποίησης.

Στα πλαίσια της εργαστηριακής άσκησης 4 για τη δημιουργία ενός εξυπηρετητή (server) όλες οι προσεγγίσεις είναι αποδεκτές (δεν είναι δηλαδή απαραίτητο να χρησιμοποιηθεί thread-pool) όσον αφορά τη λειτουργία του προγράμματος. Όμως η μη χρησιμοποίηση thread-pool ενδέχεται να σας στοιχίσει βαθμούς στο κομμάτι της επίδοσης και της αρχιτεκτονικής σχεδίασης του συστήματος.

Τεχνική Α: Δημιουργία ενός καινούργιου νήματος για την επεξεργασία μιας αίτησης, ενώ το αρχικό νήμα περιμένει νέες αιτήσεις

Η προσέγγιση αυτή αποτελεί τον απλούστερο τρόπο πολυνηματικής εφαρμογής. Υπάρχει το αρχικό νήμα το οποίο περιμένει αιτήσεις από πελάτες (εκτός από το νήμα αυτό μπορεί να υπάρχουν και άλλα νήματα τα οποία διαχειρίζονται άλλου είδους εργασίες πέραν της εξυπηρέτησης πελατών, όπως π.χ., της διαχείρισης γραμμής εντολών). Για κάθε εισερχόμενη αίτηση το αρχικό νήμα δημιουργεί ένα καινούργιο νήμα το οποίο εξυπηρετεί τον αιτητή, και με το πέρας της αποστολής του το νήμα αυτό τερματίζει τη λειτουργία του.

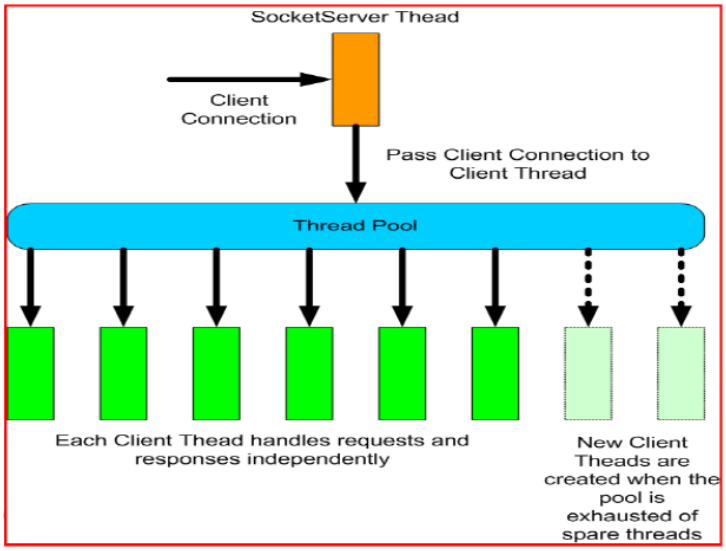
Η προσέγγιση αυτή δεν είναι πολύ καλή για τους ακόλουθους λόγους:

1. Η δημιουργία και καταστροφή νημάτων δεν είναι ιδιαίτερα ελεγχόμενη, γεγονός που μπορεί να αποβεί εξαιρετικά προβληματικό σε κάποιες περιπτώσεις.
2. Η δημιουργία ενός καινούργιου νήματος είναι ακριβή (παρόλο που είναι πιο φθηνή από τη δημιουργία μιας νέας διεργασίας). Στην συνεχεία πρέπει να αποδεσμεύσουμε τα δεδομένα που σχετίζονται με τον νήμα και τέλος να το απελευθερώσουμε, γεγονός που επιφέρει μεγαλύτερο κόστος
3. Ανά πάσα στιγμή δεν μπορούμε να ξέρουμε τον μέγιστο αριθμό νημάτων, επομένως μπορεί η διεργασία του εξυπηρετητή να ξεπεράσει το επιτρεπτό όριο μνήμης με αποτέλεσμα να χάσουμε την υπηρεσία.

Τεχνική Β: Δημιουργία ενός thread-pool για τη διαχείριση των νέων αιτήσεων

Η δεύτερη προσέγγιση αφορά τη δημιουργία ενός thread-pool από το αρχικό νήμα. Με τον όρο thread-pool, αναφερόμαστε στην δημιουργία ενός σταθερού αριθμού νημάτων-εργατών (ο αριθμός επιλέγεται ανάλογα με την εφαρμογή). Τα νήματα που βρίσκονται στο thread-pool συναγωνίζονται για την εξυπηρέτηση κάθε καινούργιας αίτησης. Δηλαδή κάθε καινούργια αίτηση ανατίθεται σε κάποιο από τα νήματα που δεν έχει δουλειά και περιμένει. Τα νήματα του thread-pool, αφού εξυπηρετήσουν ένα πελάτη, δεν τερματίζουν, αλλά μεταβαίνουν σε

κατάσταση αναμονής για να εξυπηρετήσουν άλλο πελάτη. Αν δεν υπάρχει διαθέσιμο νήμα, το αρχικό θα πρέπει να περιμένει μέχρι να υπάρξει, χωρίς να δέχεται νέες αιτήσεις. Πιο συγκεκριμένα εάν υπάρξουν περισσότερες αιτήσεις από το μέγιστο αριθμό νημάτων στο pool τότε το σύστημα απορρίπτει την αίτηση κλείνοντας το socket (χωρίς να επιστρέφει οποιανδήποτε απάντηση στον πελάτη).



Για τη διαχείριση των νημάτων του pool και την εξυπηρέτηση των αιτήσεων μπορούμε να χρησιμοποιήσουμε την πιο κάτω λογική:

1. Τα νήματα εργάτες δημιουργούνται από την αρχή στον εξυπηρετητή από το αρχικό νήμα και τα thread IDs τους αποθηκεύονται σε ένα πίνακα.
2. Δημιουργούμε μια ουρά από εργασίες όπου ο κάθε κόμβος είναι μια νέα εισερχόμενη αίτηση σύνδεσης. Το αρχικό νήμα (παραγωγός) δέχεται αιτήσεις από τους πελάτες και για κάθε νέα σύνδεση εισάγει ένα νέο κόμβο στην ουρά αν υπάρχει διαθέσιμο νήμα (καταναλωτής) για να την εξυπηρετήσει. Αλλιώς (αν δεν υπάρχει διαθέσιμο νήμα καταναλωτής) η αίτηση για σύνδεση απορρίπτεται και το αρχικό νήμα κλείνει το socket που άνοιξε με την accept για εξυπηρέτηση του πελάτη. Η ουρά αυτή είναι ο κοινός (μοιραζόμενος) πόρος.
3. Η πρόσβαση στην ουρά (κοινός πόρος) για εισαγωγή κόμβου (από νήμα-παραγωγός) ή εξαγωγή κόμβου για εξυπηρέτηση (από νήματα-καταναλωτές) πρέπει να ελέγχεται μέσω σηματοφόρου και μεταβλητής συνθήκης. Όταν η ουρά δεν έχει εργασίες για να τύχουν επεξεργασίας, τα νήματα καταναλωτές είναι αδρανή (κολλημένα σε μεταβλητή συνθήκης).