Ε3

Εγχειρίδιο για τα POSIX threads

1. Εισαγωγή

Τα νήματα καλούνται συχνά «ελαφριές διεργασίες» (*lightweight processes*) και ενώ αυτός ο όρος είναι κάπως είναι υπέρ-απλουστευμένος, ωστόσο έχει κάποιο νόημα. Τα νήματα , που χρησιμοποιούνται ευρέως, αφού είναι διαθέσιμα σε όλες σχεδόν τις πλατφόρμες, μοιάζουν με τις Unix διεργασίες αν και δεν είναι ακριβώς διεργασίες. Μια παραδοσιακή διεργασία στο Unix, αποτελείται από ένα address space και ένα thread of control (νήμα ελέγχου) που εκτελεί τις εντολές του προγράμματος. Έχοντας περισσότερα από ένα νήματα ανά διεργασία δεν υπάρχουν αυτόνομα αντίγραφα της διεργασίας, αλλά όλα τα νήματα εργάζονται στο ίδιο τμήμα δεδομένων με την διεργασία που τα δημιουργεί, εκτελούν τον ίδιο κώδικα και δεν καταχωρούνται στον πίνακα διεργασιών του συστήματος.

Έτσι, σε πολυνηματικές διεργασίες τα νήματα προσφέρουν ίδιες υπηρεσίες και μοιράζονται πληροφορία (δεδομένα). Ωστόσο, έχουν δική

τους στοίβα για προσωρινή αποθήκευση δεδομένων και δικό τους μετρητή προγράμματος για να εκτελούνται ανεξάρτητα. Αυτός είναι και ο λόγος που ονομάζονται «ελαφριές διεργασίες» αφού δημιουργούνται ταχύτατα και δεν καταναλώνουν πολλούς πόρους του συστήματος.

Η πιο διαδεδομένη έκδοση των νημάτων είναι τα POSIX threads ή pthreads, η οποία έχει καθιερωθεί ως στάνταρ που ακολουθούν τα περισσότερα συστήματα UNIX.

Οι καθολικές μεταβλητές μιας διεργασίας είναι κοινές για όλα τα νήματα που αυτή δημιουργεί και γι' αυτό τα POSIX threads προσφέρουν δομές για αμοιβαίο αποκλεισμό (mutexes) και δομές συγχρονισμού (condition variables).

2. Βασικές Λειτουργίες Νημάτων

2.1 *Δημιουργία Νήματος*

Η pthread_create() παίρνει τέσσερα ορίσματα: μία μεταβλητή η οποία είναι το id του νήματος, τα attributes του νέου νήματος, την συνάρτηση την οποία θα καλέσει το νήμα όταν ξεκινήσει η εκτέλεσή του και τα ορίσματα αυτής της συνάρτησης αν υπάρχουν.

Ορισμός

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- Δημιουργεί ένα νήμα που εκτελεί τη διαδικασία start_routine.
- Επιστρέφει ένα id νήματος στο **thread** .
- Όταν το **attr** είναι **NULL**, χρησιμοποιούνται τα προεπιλεγμένα χαρακτηριστικά του νήματος.

Παράδειγμα

```
pthread_t a_thread;
```

```
pthread_attr_t a_thread_attribute;
void thread_function(void *argument);
char *some_argument;
int pthread_create ( &a_thread,
                     a_thread_attribute,
                     (void *)&thread_function,
                     (void *) &some_argument);
```

2.2 *Τερματισμός*

Τα νήματα μπορούν να τερματίσουν την λειτουργία τους καλώντας την συνάρτηση `pthread_exit()`. Καλώντας την συνάρτηση `exit()` τερματίζεται τόσο το νήμα που την κάλεσε όσο και τα υπόλοιπα νήματα καθώς και η διεργασία.

Ορισμός

```
void pthread_exit(void * return_value);
```

- Η διαδικασία `pthread_exit()` τερματίζει το τρέχον νήμα.

2.3 *Detach και Join*

Η κλήση της συνάρτησης `pthread_join()` επιτρέπει σε ένα νήμα να περιμένει τον τερματισμό ενός άλλου νήματος.

Ορισμός

int pthread_join(pthread_t thread, void ** status);

- Το τρέχον νήμα κάνει block μέχρι το νήμα με id thread να τερματίσει.
- Η τιμή που επιστρέφεται από το νήμα τοποθετείται στο **status.**

Παράδειγμα

```
pthread_t some_thread;
void *exit_status;
pthread_join( some_thread, &exit_status );
```

Με την κλήση της συνάρτησης `pthread_detach()` ενημερώνονται τα υπόλοιπα νήματα ότι η κατάσταση τερματισμού του τρέχοντος νήματος δεν θα χρησιμοποιηθεί από ενδεχόμενες κλήσεις της `pthread_join()`. Με άλλα λόγια το νήμα που βρίσκεται σε `detach` κατάσταση δρα ανεξάρτητα από τα άλλα νήματα της διεργασίας.

Ορισμός

int pthread_detach(pthread_t thread);

2.4 *Self και Equal*

Οι δύο συναρτήσεις `pthread_self()` και `pthread_equal()` επιστρέφουν το id του νήματος που την κάλεσε και ελέγχουν αν δύο νήματα έχουν το ίδιο id.

Ορισμός

```
pthread_t pthread_self(void);
int pthread_equal(pthread_1 t1, pthread_t t2);
```

Παράδειγμα Πολυνηματικής Διεργασίας

Στο παράδειγμα που ακολουθεί δημιουργούνται τέσσερα νήματα. Το καθένα από αυτά τυπώνει ένα μήνυμα στην οθόνη, πριν τερματίσει την λειτουργία του.

```
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 4

void *thread_function( void *arg )
{
        int id;
        id = *((int *)arg);
        printf( "Hello from thread %d!\n", id );
        pthread_exit( NULL );
```

```
}


int main( void )
{
      int i, tmp;
      int arg[NUM_THREADS] = {0,1,2,3};

      pthread_t thread[NUM_THREADS];
      pthread_attr_t attr;

      // initialize and set the thread attributes
      pthread_attr_init( &attr );
      pthread_attr_setdetachstate( &attr,
                                   PTHREAD_CREATE_JOINABLE );

      // creating threads
      for ( i=0; i<NUM_THREADS; i++ )
      {
            tmp = pthread_create( &thread[i], &attr,
                                  thread_function,
                                 (void *)&arg[i] );

            if ( tmp != 0 )
            {

                  fprintf(stderr,
                  "Creating thread %d failed!",i);
                   return 1;
            }
      }

      // joining threads
      for ( i=0; i<NUM_THREADS; i++ )
      {
            tmp = pthread_join( thread[i], NULL );
            if ( tmp != 0 )
```

```
            {
                  fprintf(stderr,"Joing thread %d failed!",i);
                  return 1;
            }
    }

    return 0;
}
```

3. Συγχρονισμός Νημάτων

Σε μία πολυνηματική διεργασία όλο το address space μοιράζεται ανάμεσα στα νήματα οπότε οτιδήποτε μπορεί να θεωρηθεί σαν κοινός πόρος. Είναι λοιπόν αναγκαίος ο συγχρονισμός των νημάτων κατά την πρόσβασή τους σε κοινούς πόρους. Τα POSIX Threads παρέχουν δύο τεχνικές συγχρονισμού. Τα mutexes και τα condition variables.

3.1 *Mutexes*

Τα mutexes είναι απλά ένας μηχανισμός κλειδώματος που χρησιμοποιείται για τον έλεγχο πρόσβασης σε κοινούς πόρους. Τα Mutexes χρησιμοποιούνται για τον αμοιβαίο αποκλεισμό των νημάτων. Σειριοποιούν την πρόσβαση σε κρίσιμα τμήματα του κώδικα έτσι ώστε κάθε φορά ένα καθολικό δεδομένο να προσπελαύνεται από ένα μόνο νήμα. Κάθε κοινός πόρος θα πρέπει να αντιστοιχίζεται με ένα mutex.

3.1.1 *Δημιουργία και Καταστροφή των Mutexes*

Τα mutexes αρχικοποιούνται με την κλήση της συνάρτησης `pthread_mutex_init()` και καταστρέφονται καλώντας την συνάρτηση `pthread_mutex_destroy()`.

Ορισμός

```
int pthread_mutex_init(     pthread_mutex_t * mutex,
                            pthread_mutexattr_t *attr);
```

```
int pthread_mutex_destroy(pthread_mutex_t * mutex);
```

3.1.2 *Χρήση των Mutexes*

Ένα mutex μπορεί να κλειδώσει με την κλήση της συνάρτησης `pthread_mutex_lock()`. Αν το mutex είναι κλειδωμένο, το νήμα θα μπλοκάρει περιμένοντας κάποιο άλλο νήμα να το ξεκλειδώσει. Αυτό μπορεί να γίνει με την κλήση της συνάρτησης `pthread_mutex_unlock()`. Τέλος με την συνάρτηση `pthread_mutex_trylock()` ένα νήμα ελέγχει αν ένα mutex είναι κλειδωμένο. Αν δεν είναι τον κλειδώνει, διαφορετικά συνεχίζει τη λειτουργία χωρίς να μπλοκάρει.

Ορισμός

```
int pthread_mutex_lock(pthread_mutex_t * mutex)
int pthread_mutex_unlock(pthread_mutex_t * mutex)
int pthread_mutex_trylock(pthread_mutex_t * mutex)
```

Παράδειγμα Χρήσης Mutexes

Το παράδειγμα που ακολουθεί αποτελείται από ένα reader και ένα writer οι οποίοι επικοινωνούν χρησιμοποιώντας ένα διαμοιραζόμενο buffer στον οποίο η πρόσβαση ελέγχεται με την χρήση ενός mutex.

Σε αυτό το απλό πρόγραμμα υποθέτουμε ότι ο buffer μπορεί να κρατήσει ένα μόνο αντικείμενο. Έτσι μπορεί να βρίσκεται μόνο σε δύο καταστάσεις: να έχει αντικείμενο ή όχι.

Ο writer πρώτος κλειδώνει το mutex, μπλοκάροντας έτσι κάποιο άλλο νήμα που ενδεχομένως θα επιχειρήσει να χρησιμοποιήσει τον buffer, ελέγχει έπειτα για να δει εάν ο buffer είναι κενός. Εάν ο buffer είναι κενός, δημιουργεί ένα νέο αντικείμενο και θέτει το flag, buffer_has_item, έτσι ώστε ο reader να ξέρει ότι ο buffer έχει τώρα ένα αντικείμενο. Ξεκλειδώνει έπειτα το mutex και καθυστερεί για δύο δευτερόλεπτα για να δώσουν στον reader την ευκαιρία να διαβάσει τον buffer.

Ο reader ενεργεί παρόμοια. Κλειδώνει το mutex, ελέγχει για να δει εάν υπάρχει αντικείμενο στον buffer, και σε αυτή την περίπτωση τον διαβάζει. Ξεκλειδώνει το mutex και καθυστερεί δίνοντας στο writer την

ευκαιρία να δημιουργήσει ένα νέο αντικείμενο. Σε αυτό το παράδειγμα ο reader και ο writer τρέχουν για πάντα, παράγοντας και καταναλώνοντας αντικείμενα.

```
void reader_function(void);
void writer_function(void);
char buffer;
int buffer_has_item = 0;
pthread_mutex_t mutex;
struct timespec delay;
main()
{
      pthread_t reader;
      delay.tv_sec = 2;
      delay.tv_nsec = 0;
      pthread_mutex_init(&mutex, pthread_mutexattr_default);
      pthread_create(  &reader, pthread_attr_default,
                       (void*)&reader_function,NULL);
      writer_function();
}

void writer_function(void)
{
      while(1)
      {
            pthread_mutex_lock( &mutex );
            if ( buffer_has_item == 0 )
            {
                  buffer = make_new_item();
                  buffer_has_item = 1;
            }
            pthread_mutex_unlock( &mutex );
            pthread_delay_np( &delay );
      }
}
```

```
void reader_function(void)
{
      while(1)
      {
            pthread_mutex_lock( &mutex );
            if ( buffer_has_item == 1)
            {
                  consume_item( buffer );
                  buffer_has_item = 0;
            }
            pthread_mutex_unlock( &mutex );
            pthread_delay_np( &delay );
      }
}
```

3.2 *Condition Variables*

Τα Condition Variables είναι κοινές μεταβλητές που χρησιμοποιούνται για την προσωρινή απενεργοποίηση νημάτων μέχρι να συμβεί κάποιο γεγονός. Πολλά νήματα μπορούν να περιμένουν για την ίδια μεταβλητή συνθήκης.

3.2.1 *Δημιουργία και καταστροφή μεταβλητών συνθήκης*

Η αρχικοποίηση του condition variable γίνεται με την κλήση της συνάρτησης `pthread_cond_init()`. Μπορεί να καταστραφεί καλώντας την συνάρτηση `pthread_cond_destroy()`.

Ορισμός

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

3.2.2 *Αναμονή για μια μεταβλητή συνθήκης*

Η κλήση της συνάρτησης pthread_cond_wait() θέτει ένα νήμα σε κατάσταση αναμονής με χρήση ενός condition variable. Πριν την κλήση της συνάρτησης απαιτείται το κλείδωμα ενός mutex ενώ μετά την κλήση της ο mutex θα έχει ξεκλειδωθεί.

Ορισμός

```
int pthread_cond_wait(pthread_cond_t * cond, pthread_mutex_t *
mutex)
```

3.2.3 *Αφύπνιση από την κατάσταση αναμονής*

Τα νήματα τα οποία βρίσκονται σε κατάσταση αναμονής μπορούν είτε να ξυπνήσουν όλα μαζί καλώντας την συνάρτηση `pthread_cond_broadcast()` είτε να επιλεγεί τυχαία ένα από αυτά και να ξυπνήσει, καλώντας την συνάρτηση `pthread_cond_signal()`.

Ορισμός

```
int pthread_cond_signal(pthread_mutex_t * cond)
int pthread_cond_broadcast(pthread_mutex_t * cond)
```