Pthreads

Το **Pthreads** αναφέρεται στο πρότυπο POSIX (IEEE 1003.1c) καθορίζοντας ένα API για τη δημιουργία και το συγχρονισμό των νημάτων. Πρόκειται για μία *προδιαγραφή* για τη συμπεριφορά των νημάτων και όχι για υλοποίηση. Οι σχεδιαστές των λειτουργικών συστημάτων μπορούν να υλοποιήσουν την προδιαγραφή με οποίο τρόπο επιθυμούν. Πολλά συστήματα, συμπεριλαμβανομένων των Solaris, Linux, Mac OS X και Tru64 UNIX, υλοποιούν την προδιαγραφή Pthreads. Υλοποιήσεις *διαμοιραζόμενου λογισμικού* (*shareware*) για τα διάφορα λειτουργικά συστήματα Windows είναι επίσης διαθέσιμες στο ευρύ κοινό.

Το πρόγραμμα C που παρουσιάζεται στο παρακάτω σχήμα επιδεικνύει το βασικό API Pthreads για την κατασκευή ενός πολυνηματικού προγράμματος, το οποίο υπολογίζει το άθροισμα ενός μη αρνητικού ακεραίου σε ένα ξεχωριστό νήμα. Σε ένα πρόγραμμα που χρησιμοποιεί το Pthreads, τα ξεχωριστά νήματα ξεκινούν την εκτέλεση σε μία συγκεκριμένη συνάρτηση. Στο σχήμα αυτό πρόκειται για τη συνάρτηση runner(). Όταν ξεκινήσει αυτό το πρόγραμμα, ένα μόνο νήμα ελέγχου ξεκινάει στη main(). Μετά από κάποιες αρχικοποιήσεις, η main() δημιουργεί ένα δεύτερο νήμα που ξεκινάει τον έλεγχο στη συνάρτηση runner(). Και τα δύο νήματα διαμοιράζονται την καθολική μεταβλητή sum.

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

int sum;
void *runner(void *param);

int main(int argc, char *argv[])
{
    pthread_t tid;
    pthread_attr_t attr;

    if (argc!=2) {
        fprintf(stderr, "usage: a.out <integer value>\n");
        return -1;
    }
    if (atoi(argv[1]) <0) {
        fprintf(stderr, "%d must be >= 0\n", atoi(argv[1]));
        return -1;
    }
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,argv[1]);
    pthread_join(tid,NULL);
    printf("sum=%d\n",sum);
}

void *runner(void *param)
{
    int i, upper=atoi(param);
    sum=0;
```

```
for(i=1; i<=upper; i++)
    sum+=i;
pthread_exit(0);
}
```

Σχήμα: Πολυνηματικό πρόγραμμα C που χρησιμοποιεί το API Pthreads

Ας δούμε με περισσότερη προσοχή αυτό το πρόγραμμα. Όλα τα προγράμματα Pthreads πρέπει να περιλαμβάνουν το αρχείο επικεφαλίδας pthread.h. Η δήλωση pthread_t tid διακηρύσσει τον αναγνωριστή για το νήμα που θα δημιουργήσουμε. Κάθε νήμα έχει ένα σύνολο ιδιοτήτων, συμπεριλαμβανομένων του μεγέθους της στοίβας και πληροφορίας χρονοπρογραμματισμού. Η δήλωση pthread_attr_t attr αντιπροσωπεύει τις ιδιότητες του νήματος. Θέτουμε τις τιμές για τις ιδιότητες στην κλήση της συνάρτησης pthread_attr_init(&attr). Επειδή δεν ορίσαμε ρητά καμία ιδιότητα, χρησιμοποιούμε τις προκαθορισμένες ιδιότητες που παρέχονται. Ένα ξεχωριστό νήμα δημιουργείται με την κλήση συνάρτησης pthread_create(). Πέρα από τον αναγνωριστή και τις ιδιότητες του νήματος, περνάμε επίσης το όνομα της συνάρτησης στην οποία θα ξεκινήσει την εκτέλεση – στην περίπτωσή μας, στη συνάρτηση runner(). Τέλος, περνάμε την ακέραια παράμετρο η οποία δόθηκε στη γραμμή εντολών, argv[1].
Στο σημείο αυτό, το πρόγραμμα έχει δύο νήματα: το αρχικό (ή γονικό) νήμα στη main() και το νήμα του αθροίσματος (ή παιδί) το οποίο πραγματοποιεί την πράξη του αθροίσματος στη συνάρτηση runner(). Μετά τη δημιουργία του νήματος του αθροίσματος, το γονικό νήμα θα περιμένει την ολοκλήρωσή του καλώντας τη συνάρτηση pthread_join(). Το νήμα του αθροίσματος θα ολοκληρώσει όταν θα καλέσει τη συνάρτηση pthread_exit(). Μόλις επιστρέψει το νήμα του αθροίσματος, η διεργασία γονέας θα βγάλει στην έξοδο την τιμή της κοινής μεταβλητής sum.

Άσκηση 3

Παρακάτω δίνεται ένας αλγόριθμος που υπολογίζει το μέγιστο ενός συνόλου ακεραίων:

```
max = x[0];
for (i = 1; i < n; i++)
    if (x[i] > max)
        max = x[i];
```

Αυτό βασίζεται στο γεγονός ότι έχουμε μόνο μία CPU και κάνουμε **n - 1** συγκρίσεις για να βρούμε το μέγιστο. Μπορούμε να κάνουμε κάτι καλύτερο εάν υπάρχουν περισσότερες από μία CPUs ; Η απάντηση είναι φυσικά «ναι». Ας δούμε πως.

Βήμα 1

Ας χρησιμοποιήσουμε ένα thread για να προσομοιώσουμε μια CPU. Έστω ότι έχουμε **n** διαφορετικούς ακεραίους $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, ..., $\boldsymbol{x}_{n-1}$. Αρχικά, αρχικοποιούμε έναν πίνακα

w με **n** στοιχεία με 1. Αυτό μπορεί να γίνει με **n** threads καθένα από τα οποία γράφει ένα 1 σε μία θέση του πίνακα.

Βήμα 2

Για κάθε ζευγάρι ακεραίων x_i και x_j δημιουργούμε ένα thread (η μία CPU) T_{ij}. Αυτό το thread συγκρίνει x_i και x_j και γράφει ένα 0 στην θέση w_i εάν $x_i < x_j$. Διαφορετικά, γράφει ένα 0 στην θέση w_j. Για τον λόγο αυτό σε αυτό το βήμα χρειαζόμαστε $n(n-1)/2$ threads, καθένα από τα οποία εκτελεί ένα βήμα για να συγκρίνει και να γράψει.

Βήμα 3

Αυτό το βήμα απαιτεί **n** threads. Το thread **i** ελέγχει την τιμή w_i. Εάν είναι 0 δεν κάνει τίποτα. Εάν είναι 1 εκτυπώνει την τιμή του x_i επειδή είναι η μέγιστη τιμή που ψάχναμε. Έτσι εάν έχουμε $n(n-1)/2$ threads (ή CPUs) απαιτούνται μόνο 3 συγκρίσεις ανά CPU για να βρούμε το μέγιστο.

Γράψτε ένα πρόγραμμα που να υλοποιεί τον παραπάνω αλγόριθμο. Τα διάφορα θέματα συγχρονισμού που μπορεί να προκύψουν θα πρέπει να λυθούν με κατάλληλο μηχανισμό. Υπάρχει τέτοιος μηχανισμός με χρήση κατάλληλων συναρτήσεων που θα δοθούν παρακάτω.

Η κύρια βιβλιοθήκη που θα χρειαστείτε για να υλοποιήσετε τα παραπάνω είναι η pthread.h, άρα #include <pthread.h> και το compilation θα πρέπει να γίνεται με την παράμετρο -lpthread

Κάποιες από τις συναρτήσεις που θα χρειαστείτε είναι:
pthread_create(...),
pthread_mutex_lock(...),
pthread_mutex_unlock(...),
pthread_mutex_init(...),
pthread_mutex_destroy(...)
pthread_attr_init(...),
pthread_attr_destroy(...)
pthread_exit(...)

Θα πρέπει να δοθεί προσοχή στο γεγονός ότι πριν την προσπέλαση των νημάτων στον πίνακα w πρέπει να γίνεται κλείδωμα ώστε να συγχρονιστούν τα νήματα και να μην υπάρξει ασυνέπεια στις τιμές του πίνακα. Για το λόγο αυτό παρουσιάζεται παρακάτω ένα παράδειγμα νημάτων που περιλαμβάνει συγχρονισμό.

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

int a=0;

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
```

```
void *myThread(void *string)
{
 int i;
 int rc;
 int local;

 for (i=0; i<30; i++)
   {
    rc=pthread_mutex_lock(&mutex);
    local=a;
    sleep(1);
    local=local+1;
    a=local;
    rc=pthread_mutex_unlock(&mutex);
    printf("%d:%s\n",i,string);
   }
}

int main()
{
 char *e_str="Hello!";
 char *f_str="Bonjour!";

 pthread_t e_th1;
 pthread_t f_th1;
 pthread_t e_th2;
 pthread_t f_th2;

 int rc;

 rc=pthread_create(&e_th1, NULL, myThread, (void *)e_str);

 if (rc)
   exit(-1);


 rc=pthread_create(&f_th1, NULL, myThread, (void *)f_str);

 if (rc)
   exit(-1);

 rc=pthread_create(&e_th2, NULL, myThread, (void *)e_str);

if (rc)
  exit(-1);


rc=pthread_create(&f_th2, NULL, myThread, (void *)f_str);

if (rc)
```

```
 exit(-1);

 pthread_join(e_th1, NULL);
 pthread_join(f_th1, NULL);
 pthread_join(e_th2, NULL);
 pthread_join(f_th2, NULL);
 printf("Main finished. a=%d\n",a);
 pthread_exit(NULL);
}
```

Για περισσότερες πληροφορίες για την λειτουργία της κάθε μια από τις παραπάνω συναρτήσεις μπορείτε να χρησιμοποιήσετε την εντολή man και την συνάρτηση για την οποία ενδιαφέρεστε κάθε φορά.

Το πρόγραμμα σας θα πρέπει να δέχεται είσοδο από command line και να εκτυπώνει στην έξοδο τις τιμές μερικών μεταβλητών όπως φαίνεται στο παρακάτω παράδειγμα.

```
./a.out  4  3 1 7 4
Number of input values = 4
Input values        x = 3 1 7 4
After initialization w = 1 1 1 1
    .........
Thread T(1,3) compares 1 and 4, and writes 0 into w[1]
    .........
After Step 2        w = 0 0 1 0
Maximum               = 7
Location              = 2
```