ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Λειτουργικά Συστήματα

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:08

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

- Διεργασίες.
- Νήματα *(δημιουργία, καταστροφή, συγχρονισμός).*
- **Συναρτήσεις:** time(), fork(), vfork(), pthread_create(), pthread_exit(), pthread_join(), fflush(), pthread_attr_init(), pthread_self(), pthread_attr_destroy(), pthread_kill(), pthread_cancel().

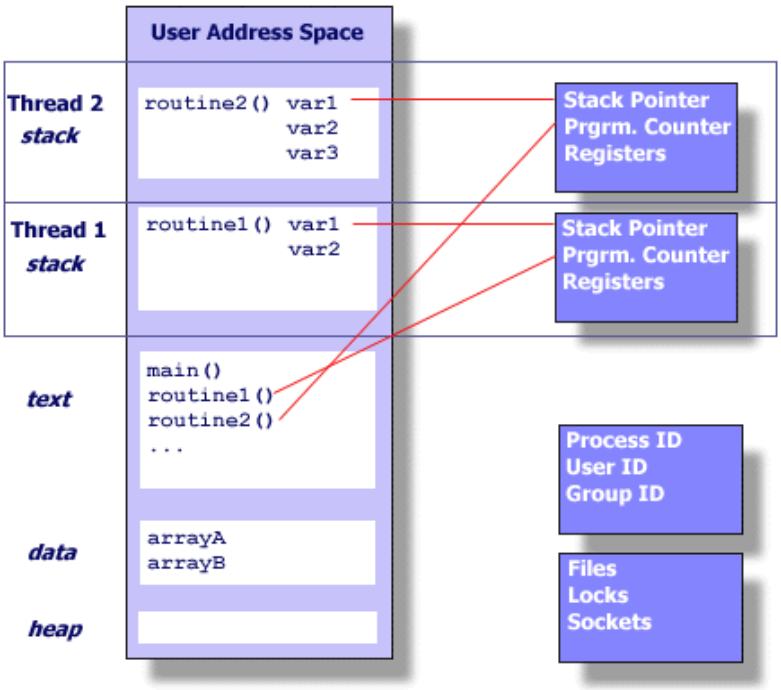
2. Παραδοτέα

(A) 19 ερωτήσεις
(C) 10 ασκήσεις

ΠΡΟΣΟΧΗ: **Το 2015 το FreeBSD αποφάσισε να χρησιμοποιεί και να υποστηρίζει τον μεταγλωττιστή `clang` σε αντίθεση με το linux που χρησιμοποιεί το `gcc`. Σε περίπτωση που εργάζεστε σε FreeBSD μηχάνημα (όπως το zafora) θα πρέπει να αντικαταστήστε τις εντολές `gcc` με τις εντολές `clang` (οι παράμετροι παραμένουν ίδιοι). Σε περίπτωση που εργάζεστε σε Linux μηχάνημα (όπως το pleiades), δε χρειάζεται κάποια αλλαγή.**

3. Περιγραφή εργαστηριακής άσκησης

Μια τεχνική για να αυξηθεί η παραλληλία σε ένα σύστημα είναι η χρήση των νημάτων. Σε αντίθεση με τις διεργασίες, όταν θέλουμε να δημιουργήσουμε ένα νήμα θα πρέπει να προσδιορίσουμε κάθε φορά τη συνάρτηση που θα εκτελεσθεί στη συνάρτηση δημιουργίας νήματος. Ένα νήμα είναι μια ανεξάρτητη ροή εντολών η οποία μπορεί να χρονοπρογραμματιστεί για εκτέλεση από τον πυρήνα. Οπτικά αν θέλουμε να δούμε τα νήματα και τις διεργασίες θα έχουμε την παρακάτω εικόνα:



Δηλαδή, τα νήματα μπορούμε να τα δούμε ως συναρτήσεις τις οποίες μέσα στο κυρίως πρόγραμμα τις ορίζουμε ότι θα εκτελεστούν ως ξεχωριστά νήματα. Οι μεταβλητές που έχουν οριστεί στο main θα είναι προσβάσιμες και κοινές και στα νήματα. Το κάθε νήμα όμως μπορεί να έχει δικές του τοπικές μεταβλητές.

Υπάρχουν διάφοροι τρόποι να υλοποιήσουμε τα νήματα. Εμείς θα χρησιμοποιήσουμε το αρκετά δημοφιλές και προτυποποιημένο τρόπο με τη βιβλιοθήκη Pthreads όπως έχουν οριστεί από το IEEE POSIX 1003.1c standard. Αυτή η υλοποίηση ονομάζεται Posix Threads ή απλά Pthreads και υποστηρίζεται από πλήθος λειτουργικών συστημάτων *(ακόμη και των windows αν εγκατασταθούν οι κατάλληλες βιβλιοθήκες)*.

Τα threads όταν δημιουργούνται επιβαρύνουν το σύστημα πολύ λιγότερο από ότι οι διεργασίες. Είναι λοιπόν αποδοτικότερο να δημιουργούμε threads παρά διεργασίες στην εφαρμογή μας. Αυτό θα το διαπιστώσετε παρακάτω.

Η συνάρτηση `time()` της C *{προσοχή man section 3}* τι παραμέτρους δέχεται και τι επιστρέφει;_____**(A1)**

(C1) Να δημιουργήσετε το πρόγραμμα **c1.c** το οποίο:

- Θα ορίσετε μια συνάρτηση `nothing()` η οποία θα ορίζει μια μεταβλητή `int x=0` και θα κάνει την πράξη `x=x+1`. Αυτή η συνάρτηση δεν κάνει τίποτα χρήσιμο, αλλά απλώς υπάρχει για να μας βοηθήσει στη μέτρηση.
- Μέσα στο `main()` θα εκτελεί μια φορά τη συνάρτηση `time()` με τις κατάλληλες παραμέτρους, θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή start και αμέσως μετά θα εκτυπώνεται το μήνυμα “Αρχική τιμή δευτερολέπτων” ακολουθούμενο από τη start.

- Στη συνέχεια θα υπάρχει ένας βρόχος **`while()`** ο οποίος θα δημιουργεί 100 διεργασίες (δοκιμάστε και για 5000, 10000) με τη **`fork()`** οι οποίες όλες θα εκτελούν τη **`nothing()`**. Προσοχή, ο πατέρας θα δημιουργήσει όλες τις διεργασίες και όχι η κάθε διεργασία θα δημιουργεί άλλη διεργασία.
- Μόλις δημιουργηθούν οι 100 διεργασίες και μόνο τότε θα εκτελεί ο πατέρας 100 φορές τη **`waitpid()`** ώστε να περιμένει την επιτυχή ολοκλήρωση όλων των θυγατρικών.
- Στη συνέχεια θα καλείται η **`time()`**, θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή end και αμέσως μετά θα εκτυπώνεται το μήνυμα "Τελική τιμή δευτερολέπτων" ακολουθούμενη από την end.
- Θα γίνεται η πράξη end-start, θα εκτυπώνεται το αποτέλεσμα ενώ θα εκτυπώνεται και το αποτέλεσμα **"(end-start)/100"** για να εμφανιστεί ο μέσος χρόνος δημιουργίας, εκτέλεσης και τερματισμού των 100 διεργασιών.

Πόσος είναι ο συνολικός χρόνος και ο μέσος χρόνος δημιουργίας εκτέλεσης και τερματισμού των 100 διεργασιών στο σύστημα σας *(μη ξεχάσετε τη μονάδα μέτρησης);* _____ **(A2)**

Το πρόγραμμα συστήματος time που χρησιμοποιείται και τι στοιχεία μας εκτυπώνει στην έξοδο όταν χρησιμοποιηθεί για μια διεργασία;_____ **(A3)**

Απομακρύνετε από τον κώδικα **c1.c** τις εντολές που κάνουν printf() όπως και τις κλήσεις στην time και δημιουργήστε το αρχείο **c2b.c** το οποίο απλώς δημιουργεί 1000 διεργασίες. Κάντε το compile, και εκτελέστε το με τη χρήση του προγράμματος time.

Ποιοι είναι οι χρόνοι που αναφέρονται από το πρόγραμμα time για την εκτέλεση της προηγούμενης διεργασίας; Συγκρίνετε τους χρόνους με το ερώτημα **A2**. Είναι ίδιοι, έχουν μικρή ή μεγάλη απόκλιση κατά τη γνώμη σας;_____ **(A4)**

4. Η συνάρτηση vfork()

Μέσα στο πρότυπο POSIX ορίζονται πολλαπλές συναρτήσεις δημιουργίας διεργασιών. Ως τώρα χρησιμοποιήσαμε τη **`fork()`**. Εκτός από τη **`fork()`** υπάρχει και η **`vfork()`** η οποία όμως έχει μια διαφορετική λειτουργία.

Τοποθετήστε τον παρακάτω κώδικα στο πρόγραμμα vfork1.c και εκτελέστε τον. Το πρόγραμμα σας θα σταματήσει απότομα εμφανίζοντας το μήνυμα **`"Segmentation Fault"`** *(ή αντίστοιχο).*

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
main ()
{pid_t p;
if (p=vfork()==0)
```

```
     { puts("I am the child");}
else { puts("I am the parent");}
puts("End");
}
```

Χρησιμοποιώντας τη σελίδα βοήθειας της **vfork()**, τα σχετικά βοηθήματα και το Internet βρείτε γιατί το συγκεκριμένο πρόγραμμα έχει αυτό το πρόβλημα. _____ **(A5)**

Πως μπορεί να τροποποιηθεί το πρόγραμμα ώστε να μην κάνει **Segmentation Fault**; _____ **(A6)**

(C2) Τροποποιήστε το πρόγραμμα δημιουργώντας το **c2.c**, ώστε να καλεί με επιτυχία τη **vfork()** να εκτυπώνονται από τη θυγατρική διεργασία τα "I am the child" και "End" και ομοίως να εκτυπώνει η γονική τα αντίστοιχα, χωρίς να δημιουργεί το Segmentation Fault.

5. Δημιουργία του πρώτου προγράμματος με threads.

5.1 Δημιουργία & Καταστροφή νημάτων.

Προκειμένου να χρησιμοποιήσουμε τα νήματα Pthreads θα πρέπει να τοποθετήσουμε στα include το **<pthread.h>** το οποίο μας παρέχει τις δηλώσεις των συναρτήσεων που θα χρησιμοποιήσουμε. Επίσης, κατά τη διαδικασία του compilation θα πρέπει αμέσως μετά το gcc να τοποθετήσουμε το **-lpthread** ώστε να γίνει σύνδεση με την αντίστοιχη βιβλιοθήκη *(pthread)* των νημάτων.

Προκειμένου να δημιουργήσουμε ένα νήμα θα πρέπει:

- να **ορίσουμε μια συνάρτηση** το όνομα της οποίας θα είναι pointer στη συνάρτηση, ενώ και οι παράμετροι θα είναι pointers. Δηλαδή η δήλωση θα είναι με * *(αστέρι)*, όπως π.χ. **void *PrintHello (void *)**

- μέσα στο σημείο που θέλουμε να δημιουργήσουμε το νήμα να **εκτελέσουμε τη συνάρτηση** **pthread_create()** στην οποία θα δώσουμε ως δείκτες όλα τα στοιχεία που απαιτούνται: **(α)** το pointer της συνάρτησης που θα εκτελεστεί, **(β)** το pointer των παραμέτρων καθώς και **(γ)** άλλες παραμέτρους που αφορούν τα νήματα.

Αφού διαβάσετε τη σελίδα βοήθειας της **pthread_create()**, τα βοηθητικά κείμενα που συνοδεύουν το εργαστήριο και πληροφορίες από το Internet, καταγράψτε πόσες παραμέτρους απαιτεί η συνάρτηση **pthread_create()** και τι σημαίνει η κάθε παράμετρος_____**(A7)**

Προκειμένου να τερματίσουμε το νήμα θα πρέπει:

- είτε να εκτελέσουμε την εντολή `pthread_exit( )`

- είτε να τοποθετήσουμε μια εντολή `return`, η οποία προκαλεί έμμεση εκτέλεση της `pthread_exit()`

5.2 Δημιουργία νημάτων χωρίς παραμέτρους

(C3) Κατασκευάστε ένα αρχείο με όνομα **c3.c** στο οποίο θα δημιουργήστε μια συνάρτηση `*helloworld (void *arg)` η οποία θα εκτυπώνει το μήνυμα "Hello World from Thread". **Η συνάρτηση των νημάτων δηλώνεται ως pointer, όπως και οι παράμετροι της.** Μέσα στο κυρίως πρόγραμμα θα ερωτάται ο χρήστης πόσα νήματα θέλει να δημιουργήσει και στη συνέχεια θα υπάρχει μια δομή βρόχου `for()` για τον αριθμό των επαναλήψεων που έχει δηλώσει ο χρήστης. Μέσα στο for θα εκτυπώνεται το μήνυμα "Δημιουργείται το thread XXX", όπου **xxx** η τρέχουσα τιμή του μετρητή for και αμέσως μετά θα δημιουργούνται τα νήματα με τη `pthread_create()` και παράμετρο τη συνάρτηση `helloworld`. Σημειώστε ότι δε θα ρυθμίσουμε χαρακτηριστικά νημάτων, ούτε θα δώσουμε παραμέτρους, οπότε αντιστοίχως η 2η και η 4η παράμετροι της `pthread_create()` θα είναι **NULL**.

Προκειμένου να αποθηκεύουμε το κάθε id θα δηλώσουμε έναν πίνακα από id νημάτων ως: `pthread_t threads[number_of_threads];` και αυτό σημαίνει ότι η 1η παράμετρος της `pthread_create()` θα είναι η `&threads[i]`.

Να κάνετε compile το πρόγραμμα και να επιβεβαιώσετε την ορθή λειτουργία.

Που χρησιμοποιείται η εντολή `pthread_join()`; _____ **(A8)**

Ποιοι είναι οι παράμετροι της `pthread_join()`; _____ **(A9)**

(C4) Να δημιουργήσετε το πρόγραμμα **c4.c** το οποίο:

- θα ορίσετε μια συνάρτηση `nothing()` η οποία θα ορίζει μια μεταβλητή `int x=0` και θα κάνει την πράξη `x=x+1`. Αυτή η συνάρτηση δεν κάνει τίποτα χρήσιμο, αλλά απλώς υπάρχει για να μας βοηθήσει στη μέτρηση.
- Μέσα στο `main()` θα εκτελεί μια φορά τη συνάρτηση `time()` με τις κατάλληλες παραμέτρους, θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή start και αμέσως μετά θα εκτυπώνεται το μήνυμα "Αρχική τιμή δευτερολέπτων" ακολουθούμενο από τη start.
- Στη συνέχεια θα υπάρχει ένας βρόχος `while()` ο οποίος θα δημιουργεί 100 **νήματα** με τη `pthread_create()`, τα οποία όλα θα εκτελούν τη `nothing()`. *Προσοχή,* ο πατέρας θα δημιουργήσει 100 νήματα. Το thread id θα αποθηκεύεται σε έναν πίνακα:

```
pthread_t threads [number_of_threads];
```

- Μόλις δημιουργηθούν οι 100 νήματα και μόνο τότε θα εκτελεί ο πατέρας ένα νέο βρόχο for για 100 φορές τη `pthread_join(threads[i],(void *) NULL)` ώστε να περιμένει την επιτυχή ολοκλήρωση όλων των νημάτων. Παρατηρήστε ότι η 2η παράμετρος είναι `NULL` γιατί δε μας ενδιαφέρει η τιμή επιστροφής του κάθε νήματος.
- Στη συνέχεια θα καλείται η `time()`, θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή end και αμέσως μετά θα εκτυπώνεται το μήνυμα "Τελική τιμή δευτερολέπτων" ακολουθούμενη από την end.
- Θα γίνεται η πράξη end-start, θα εκτυπώνεται το αποτέλεσμα ενώ θα εκτυπώνεται και το αποτέλεσμα "`(end-start)/100`" για να εμφανιστεί ο μέσος χρόνος δημιουργίας, εκτέλεσης και τερματισμού των 100 διεργασιών.

Πόσος είναι ο συνολικός χρόνος και ο μέσος χρόνος δημιουργίας εκτέλεσης και τερματισμού των 100 νημάτων στο σύστημα σας *(μη ξεχάσετε τη μονάδα μέτρησης)*; _____ **(A10)**

Απομακρύνετε από τον κώδικα **c4.c** τις εντολές που κάνουν `printf()` όπως και τις κλήσεις στην time και δημιουργήστε το αρχείο **c4.c** το οποίο απλώς δημιουργεί 1000 νήματα. Κάντε το compile, και εκτελέστε το με τη χρήση του προγράμματος time.

Ποιοι είναι οι χρόνοι που αναφέρονται από το πρόγραμμα time για την εκτέλεση της προηγούμενης διεργασίας; Συγκρίνετε τους χρόνους με το ερώτημα Α2. Είναι ίδιοι, έχουν μικρή ή μεγάλη απόκλιση κατά τη γνώμη σας; _____ **(A11)**

Συγκρίνετε τους χρόνους δημιουργίας διεργασιών και τους χρόνους δημιουργίας νημάτων και σχολιάστε ποια τεχνική είναι πιο γρήγορη και πόσες φορές:_____ **(A12)**

Που χρησιμοποιείται και γιατί η συνάρτηση `fflush()`; _____ **(A13)**

Τι θα επιτευχθεί αν γίνει η κλήση `fflush(NULL)` *(δηλαδή κλήση με καμία παράμετρο)*; _____ **(A14)**

5.3 Δημιουργία νημάτων με πέρασμα μιας παραμέτρου

(C5) Να δημιουργήσετε το πρόγραμμα **c5.c** το οποίο:

- Θα ορίζει μια συνάρτηση `void *PrintHello(void *threadid)` η οποία θα εκτυπώνει με `printf()` το string hello world και θα ακολουθεί ο αριθμός του threadid. Θα εκτελείται η συνάρτηση `fflush(NULL)` και θα καλείται η συνάρτηση τερματισμού thread.

- Στο κυρίως πρόγραμμα ορίστε `#define NUM_THREADS 15`
    - Δημιουργήστε ένα βρόχο επαναλήψεων, το οποίο δημιουργεί τόσα threads όσα το προηγούμενο define (`for(i=0;i<NUM_THREADS`). Όλα τα thread θα εκτελούν τη συνάρτηση PrintHello. Η παράμετρος που θα στέλνουμε κάθε φορά θα είναι ο αριθμός του thread, δηλαδή η τιμή της μεταβλητής i του βρόχου for. Αυτό μπορείτε να το περάσετε ως `(void *)i`. Μπορείτε να χρησιμοποιήσετε τη δήλωση:
      `pthread_t threads[NUM_THREADS];`
      ώστε να μην απαιτείται κάθε φορά να δηλώνετε ένα ένα όλα τα pthread_t που χρειάζεστε.
    - Ελέγξτε την τιμή που επιστρέφεται (από τη `pthread_create()`) κάθε φορά που δημιουργείται ένα νέο νήμα. Αν η τιμή δεν είναι σωστή *(δείτε το man page)* τότε θα εμφανίζεται ένα μήνυμα αποτυχίας σχετικά με το ποιο thread δε μπόρεσε να δημιουργηθεί.

Κάντε compile & επιβεβαιώστε την ορθή λειτουργία.

(C6) Να δημιουργήσετε το πρόγραμμα **c6.c** το οποίο θα μας βοηθήσει να βρούμε ποιο είναι το μέγιστο όριο νημάτων που επιτρέπεται να δημιουργήσουμε στο σύστημα μας. Συγκεκριμένα:

- Θα χρησιμοποιεί το **C5** κώδικα.
- Να απομακρύνετε ή να τοποθετήσετε σε σχόλιο όλες τις εντολές εκτύπωσης μηνύματος.
- Το NUM_THREADS θα έχει μια πολύ μεγάλη τιμή (π.χ. 32000 ή και μεγαλύτερη).
- Μόλις εκτελείται η `pthread_create()` θα ελέγχεται η τιμή επιστροφής. Αν δεν έχει δημιουργηθεί με επιτυχία το νήμα τότε θα εκτυπώνεται το μήνυμα "`No more threads. Thread Count: XXX`", όπου **XXX** θα είναι η τρέχουσα τιμή του βρόχου for.

Πόσος είναι ο μέγιστος αριθμός νημάτων ανά διεργασία στο σύστημα σας σύμφωνα με το **C6**;_____ **(A15)**

Ψάξτε στο internet και βρείτε που ορίζεται αυτός ο μέγιστος αριθμός και πως μπορεί να τροποποιηθεί: _____ **(A16)**

5.4 Πέρασμα παραμέτρων σε νήματα

5.4.1 Ένα ακόμη παράδειγμα με κοινά δεδομένα

Στο επόμενο παράδειγμα θα δημιουργήσουμε ένα πρόγραμμα το οποίο δημιουργεί έναν αριθμό από νημάτων. Το κάθε νήμα εκτυπώνει μια διαφορετική γραμμή ενός πίνακα κοινού.

(C7) Να δημιουργήσετε το πρόγραμμα **c7.c** το οποίο:

- Χρησιμοποιεί τον κώδικα του **C5**.
- Μέσα στη συνάρτηση των thread το printf θα τροποποιηθεί στο `printf("Thread %d: %s\n", taskid, messages[taskid]);` όπου taskid είναι ο αριθμός του thread και messages ένας πίνακας που έχει οριστεί στο κυρίως πρόγραμμα.
- Έξω από κάθε συνάρτηση ορίστε τη global μεταβλητή, προκειμένου να είναι προσβάσιμη τόσο από το κυρίως πρόγραμμα όσο και από τα νήματα: `char *messages[NUM_THREADS];`
- Στο κυρίως πρόγραμμα τοποθετήστε στη global μεταβλητή messages τα παρακάτω μηνύματα:

```
messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: NuqneH Hoch!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";
```

- Μέσα στο βρόχο του κυρίως προγράμματος που δημιουργούνται τα threads (από **`t=0 έως t<NUM_THREADS`**) θα έχουμε:

```
taskids[t] = (int *) malloc(sizeof(int));
*taskids[t] = t;
```

******* Γιατί το κάνουμε αυτό; Αν στέλναμε κατευθείαν τη μεταβλητή t σε κάθε thread τότε θα στέλναμε τη διεύθυνση μνήμης της μεταβλητής t, η οποία έχει δεδομένα που τροποποιούνται συνέχεια στο βρόχο *(αρχικά τα δεδομένα είναι 0 μετά είναι 1, κ.ο.κ.)*. Κατασκευάζουμε μια μεταβλητή για κάθε νήμα λοιπόν, η οποία δεν αλλάζει καθώς αλλάζει ο δείκτης του βρόχου. *******

- Η παράμετρος που θα στέλνουμε σε κάθε thread μέσα στη συνάρτηση pthread_create() είναι η `(void *) taskids[t]`
- Ορίστε το **`NUM_THREADS`** σε 8.
- Προσοχή στη συνάρτηση νήματος:
  - η συνάρτηση στον ορισμό της δέχεται τα ορίσματα ως διευθύνσεις, οπότε θα έχει ως παράμετρο ορισμού: **`void * parameter`**

- προκειμένου να διαβάσουμε την ακέραια τιμή που βρίσκεται στη διεύθυνση μνήμης που δείχνει ο pointer **`* parameter`** θα πρέπει να ορίσουμε ότι θα χρησιμοποιήσουμε ένα νέο pointer ακεραίων **`*ptr`** ως εξής: **`int *ptr;`**
- Μετά θα τοποθετήσουμε στον pointer **`*ptr`** που έχουμε δηλώσει πριν, τον αρχικό pointer **`*parameter`** που έχουμε δεχθεί ως παράμετρο της συνάρτησης: **`ptr=(int *) parameter;`**
- Θα δηλώσουμε μια ακέραια τιμή **`int index`** για να τοποθετήσουμε σε αυτό την τιμή που δέχθηκε το νήμα μας.
- Για να διαβάσουμε λοιπόν την ακέραια τιμή που δείχνει ο pointer **`*ptr`** αρκεί να δώσουμε **`index=*ptr;`**
- Την τιμή **`index`** που είναι μια ακέραια τιμή θα χρησιμοποιήσουμε στο printf.

Κάντε compile και επιβεβαιώστε την ορθή λειτουργία. Θα πρέπει να εκτυπώνονται όλα τα μηνύματα του πίνακα.

5.4.2 Πολλαπλές τιμές

Υπάρχουν περιπτώσεις που θέλουμε να στείλουμε πολλαπλά δεδομένα στα νήματα και όχι μόνο μια ακέραια τιμή. Για να γίνει αυτό δημιουργούμε ένα struct στο οποίο τοποθετούμε όλα τα δεδομένα μας και εμείς στέλνουμε τη διεύθυνση του struct στο νήμα. Αυτό θα το δούμε με το παρακάτω παράδειγμα:

(C8) Να δημιουργήσετε το πρόγραμμα **c8.c** το οποίο:

- Χρησιμοποιεί τον κώδικα του **C7.**
- Ορίζει ως global μια δομή *(struct)* στην οποία θα τοποθετήσουμε 3 παραμέτρους: τον αριθμό του νήματος, μια ακέραια τιμή, και ένα μήνυμα. Αυτό θα γίνει με τη δήλωση:

```
struct thread_data{
int thread_id;
int sum;
char *message;
};
```

- Συσχετίστε αυτή τη δομή με μια μεταβλητή *(ή με άλλα λόγια ορίστε μια μεταβλητή τύπου thread_data)*, η οποία θα είναι ξεχωριστή για κάθε νήμα. Ο καλύτερος τρόπος να το κάνουμε αυτό είναι να ορίσουμε έναν πίνακα όπου κάθε καταχώρηση θα είναι τύπου struct thread_data. Αυτό γίνεται με τη δήλωση:
  `struct thread_data thread_data_array[NUM_THREADS];`
- Το επόμενο βήμα είναι να τροποποιηθεί η δήλωση της συνάρτησης printhello, ώστε να δέχεται ως όρισμα το `(void *threadarg)` και ότι θα

χρησιμοποιήσει το struct. Θα πρέπει να δηλωθεί λοιπόν μέσα στη συνάρτηση αυτή μια μεταβλητή pointer `*my_data` *(είναι pointer γιατί οι παράμετροι στα νήματα είναι πάντα pointer)* που θα δείχνει σε μια δομή struct, όπως παραπάνω:

```
struct thread_data *my_data;
```

- Μέσα στη συνάρτηση ορίστε ως ακέραια τιμή μια μεταβλητή **taskid**, η οποία θα φέρει κάθε φορά τον ακέραιο αριθμό αριθμού thread που βρίσκεται μέσα στο struct thread_data.
- Στη συνέχεια μέσα στη συνάρτηση printhello, διαβάστε όλες τις τιμές που έχουν έρθει ως pointer κατά την κλήση και αντιστοιχούν στο struct που έχουμε δηλώσει, με τον παρακάτω τρόπο. Παρατηρήστε ότι αρχικά ο pointer `*threadarg` που μας έρχεται από την κλήση αρχικά τοποθετείται στο τοπικό pointer και στη συνέχεια μπορούμε να διαβάσουμε συγκεκριμένα στοιχεία με τη χρήση του βέλους `- >`

```
my_data = (struct thread_data *) threadarg;
taskid = my_data->thread_id;
```

- Μέσα στη συνάρτηση printhello, εκτυπώστε με printf το κάθε στοιχείο *(δηλαδή, τον αριθμό του thread, τον αριθμό sum και το μήνυμα)*, ώστε στο τέλος να εμφανίζονται τα μηνύματα ως εξής:

```
...
Thread 4: German: Guten Tag, Welt!  Sum=10
Thread 5: Russian: Zdravstvytye, mir!  Sum=15
Thread 6: Japan: Sekai e konnichiwa!  Sum=21
Thread 7: Latin: Orbis, te saluto!  Sum=28
....
```

- Στο κυρίως πρόγραμμα πριν από το κάθε pthread_create() θα πρέπει να τοποθετήσετε τις γραμμές που γράφουν δεδομένα στο struct, όπως παρακάτω και τα οποία τα στέλνουμε με το pointer που τους αντιστοιχούν στο κάθε νήμα.

```
thread_data_array[t].thread_id = t;
thread_data_array[t].sum = sum;
thread_data_array[t].message = messages[t];
```

- Το sum *(που το έχετε ορίσει ως ακέραιο μέσα στο main() για να συμφωνεί με το struct)* να το υπολογίζετε κάθε φορά ως:

```
sum = sum + t + 1;
```

- Η κλήση της δημιουργίας νημάτων θα έχει για 4η παράμετρο το δείκτη που αντιστοιχεί στο struct που θέλουμε να στείλουμε και ο οποίος είναι το thread_data_array[t]. Αυτό θα γίνει ως:

```
(void *) &thread_data_array[t]
```

- Παρατηρήστε ότι επιβάλετε να χρησιμοποιούμε το `(void *)` στη pthread_create() στο πέρασμα των παραμέτρων.

Κάντε compile και επιβεβαιώστε την ορθή λειτουργία.

ΠΡΟΣΟΧΗ:

Η παρακάτω δομή είναι λάθος, επειδή στέλνει τη διεύθυνση της μεταβλητής **t** η οποία βρίσκεται σε κοινή μνήμη και προσβάσιμη από κάθε νήμα. Καθώς το loop προχωράει η τιμή του t αλλάζει *(αρχικά t=0, μετά t=1,..)* οπότε μέχρι να διαβάσει το νήμα την τιμή έχει ήδη αλλάξει αυτή και μας είναι άχρηστη.

Για αυτό το λόγο προσέχουμε αν θα στείλουμε τη διεύθυνση της μεταβλητής με το & μπροστά ή την τιμή της μεταβλητής χωρίς το & από μπροστά.

Λάθος:

```
int rc;long t;
for(t=0; t<NUM_THREADS; t++)
{
printf("Creating thread %ld\n", t);
rc = pthread_create(&threads[t], NULL, PrintHello, (void
*) &t);
...}
```

Σωστό:

```
int rc;long t;
for(t=0; t<NUM_THREADS; t++)
{
  printf("Creating thread %ld\n", t);
  rc   =   pthread_create(&threads[t],   NULL,   PrintHello,
(void *) t);
    ...}
```

5.5 Συγχρονισμός διεργασιών με pthread_join()

Υπάρχουν περιπτώσεις που πρέπει το κυρίως πρόγραμμα να περιμένει να ολοκληρωθούν κάποια νήματα πριν συνεχίσει την εκτέλεση. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τη συνάρτηση **pthread_join()**.

Για παράδειγμα θα κατασκευάσουμε μια διαδικασία που θα εκτελείται από κάθε νήμα προκειμένου να κάνει κάποιους υπολογισμούς. Το κυρίως πρόγραμμα θα περιμένει να ολοκληρωθούν όλοι οι υπολογισμοί *(να τερματίσουν δηλαδή όλα τα νήματα)*. Όταν κάνουν join όλα τα νήματα, τότε θα ολοκληρώνεται και το κυρίως πρόγραμμα.

Η 2η παράμετρος της **pthread_create()** που χρησιμοποιείται; Δώστε ένα παράδειγμα:_____ **(A17)**

Η συνάρτηση **pthread_attr_init** που χρησιμοποιείται, πόσες παραμέτρους και τι σημαίνει η κάθε παράμετρος; _____ **(A18)**

(C9) Να δημιουργηθεί το πρόγραμμα **c9.c**

- Κάντε include το header file των μαθηματικών σχέσεων math.h γιατί θα χρησιμοποιήσουμε τριγωνομετρικές συναρτήσεις.

***** Όταν χρησιμοποιούμε μαθηματικές συναρτήσεις στη C (gcc) θα πρέπει κατά το compile να κάνουμε link με τη μαθηματική βιβλιοθήκη. Αυτό γίνεται με το να προσθέσουμε το διακόπτη -lm στην εντολή gcc …. *****

- Κάντε include όλα τα υπόλοιπα header file που θα χρησιμοποιήσετε.
- Ορίστε στο define τον αριθμό των threads σε 4.
- Ορίστε μια διαδικασία που θα εκτελούν τα threads ως computethread με:

  ```
  void *computethread(void *t)
  ```

  - Η διαδικασία θα εκτυπώνει ένα μήνυμα ότι ξεκινάει το νήμα με αριθμό **xxx** *(ο αριθμός που δίνεται στην παράμετρο* ***t****)*.
  - Στη συνέχεια θα εκτελεί ένα βρόχο από i=0 έως i=107 με την πράξη:

  ```
  result = result + sin(i) * tan(i);
  ```

  (μέσα στη συνάρτηση να έχει οριστεί `double result=`***0.0)***

- Όταν υπολογιστεί ο αριθμός result το νήμα εκτυπώνει ότι "`Το νήμα με αριθμό XXXX, τελείωσε. Το result είναι YYYYY`"
- Στην κυρίως συνάρτηση *(main).*
  - Ορίστε για κάθε thread τον τύπο pthread_t *(σε μορφή πίνακα όπως το είχατε κανει πριν).*
  - Ορίστε μια παράμετρο **attr** τύπου pthread_attr_t :
    ```
    pthread_attr_t attr;
    ```
  - Αρχικοποιήστε το attr με τη χρήση της συνάρτησης **pthread_attr_init**
  - Ορίστε ότι το attr θα είναι `JOINABLE` με τη χρήση της συνάρτησης **pthread_attr_setdetachstate()**, δηλαδή η κατάσταση να είναι `PTHREAD_CREATE_JOINABLE`.
  - Δημιουργήστε ένα βρόχο for-loop από t=0 έως t<NUM_THREADS, ο οποίος θα εκτυπώνει ένα μήνυμα "`MAIN: Creating thread Number...`" και θα δημιουργεί το νήμα με παράμετρο το **(void *)t** και το χαρακτηριστικό **&attr**.
  - Να υπάρχει έλεγχος ότι δημιουργείται το thread. Αν δε μπορεί να δημιουργηθεί το thread τότε να εμφανίζεται σχετικό μήνυμα.

- Όταν ολοκληρωθεί το παραπάνω loop, τότε δε θα χρειάζεται πια το attribute και μπορούμε να το απομακρύνουμε με την κλήση της συνάρτησης **`pthread_attr_destroy()`**
- Δημιουργήστε ένα νέο βρόχο for-loop από t=0 έως t<NUM_THREADS, ο οποίος θα κάνει κλήση της συνάρτησης **`pthread_join`** με πρώτη παράμετρο το pthread_t του κάθε thread, και δεύτερη παράμετρο το &status.
- Στο status θα τοποθετείται η τιμή που επιστρέφει το κάθε νήμα όταν τερματίζει.
- Να ελέγξετε αν εκτελέστηκε κάθε φορά με επιτυχία η pthread_join με μια δομή if. Αν δεν έχει εκτελεστεί σωστά να εκτυπώνεται το αντίστοιχο μήνυμα.
- Το main να εκτυπώνει στο τέλος κάθε join που γίνεται μαζί με την τιμή που επιστρέφει *(Main: completed join with thread number XXXX having a status of YYYYY).*
- Χρησιμοποιήστε σε αυτήν την άσκηση τη συνάρτηση **`pthread_self()`** για να εκτυπώνεται το thread ID.

5.6 Καταστρέφοντας τα νήματα

Η συνάρτηση **`pthread_cancel()`** που χρησιμοποιείται, πόσους παραμέτρους δέχεται και τι σημαίνει η κάθε παράμετρος;_____ **(A19)**

Πολλές φορές είναι επιθυμητό να καταστρέφεται ένα νήμα ιδιαίτερα αν για κάποιο λόγο καθυστερεί να ολοκληρώσει το έργο του, ενώ θα έπρεπε να το έχει ήδη ολοκληρώσει. Η συνάρτηση που χρησιμοποιείται για την ακύρωση είναι η **`pthread_cancel()`**. Η συνάρτηση που χρησιμοποιείται για να ελεγχθεί αν εκτελείται ακόμη ένα νήμα, είναι η συνάρτηση **`pthread_kill()`** με παράμετρο 0 *(δείτε man page).*

(C10) Να δημιουργηθεί το πρόγραμμα **c10.c**, το οποίο δημιουργεί δύο νήματα. Το κάθε νήμα εκτελεί μια διαφορετική συνάρτηση.

- Το πρώτο νήμα με όνομα **`sleeper`** θα εκτυπώνει ένα μήνυμα ότι ενεργοποιήθηκε, θα ζητάει από το χρήστη μια τιμή και θα κάνει ένα sleep για όσα δευτερόλεπτα έχει δώσει πριν ο χρήστης. Στη συνέχεια θα κάνει pthread_exit() .
- Το δεύτερο νήμα με όνομα **`watchdog`** θα εκτελείται με παράμετρο το thread_id του προηγούμενου νήματος θα εκτυπώνει ένα μήνυμα ότι ενεργοποιήθηκε και μετά από sleep(10) θα ελέγχει με τη συνάρτηση pthread_kill() αν το προηγούμενο νήμα έχει ολοκληρωθεί ή όχι. Αν έχει ολοκληρωθεί θα κάνει κλήση της pthread_exit() και θα ολοκληρώνεται και

αυτό, διαφορετικά θα εκτυπώνει ένα κατάλληλο μήνυμα και θα κάνει pthread_cancel() καταστρέφοντας το προηγούμενο νήμα.

- Το κυρίως πρόγραμμα θα εκτελεί αμέσως μετά τη δημιουργία των νημάτων pthread_join και για τα δυο, προκειμένου να μην ολοκληρωθεί η εκτέλεση *(γιατί σε αυτήν την περίπτωση αμέσως όλα τα νήματα θα καταστραφούν).*