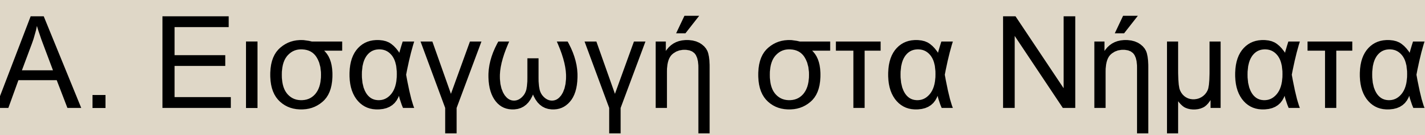
Διάλεξη 19: Νήματα: Εισαγωγή και Έλεγχος (Threads: Introduction & Control)

(Κεφάλαιο 11 - Stevens & Rago)

Δημήτρης Ζεϊναλιπούρ

Περιεχόμενο Διάλεξης

A. Εισαγωγή και Αναπαράσταση Νημάτων
B. Πλεονεκτήματα Νημάτων
C. Είδη Νημάτων
D. Πολυνηματικά Μοντέλα
E. Η βιβλιοθήκη <pthread.h>
   – Δημιουργία/Τερματισμός Νημάτων
   – Αναγνώριση Νημάτων
   – “Ορφανά” και Zombie Νήματα
   – Αναμονή Νημάτων
   – Απόσπαση (detachment) Νημάτων

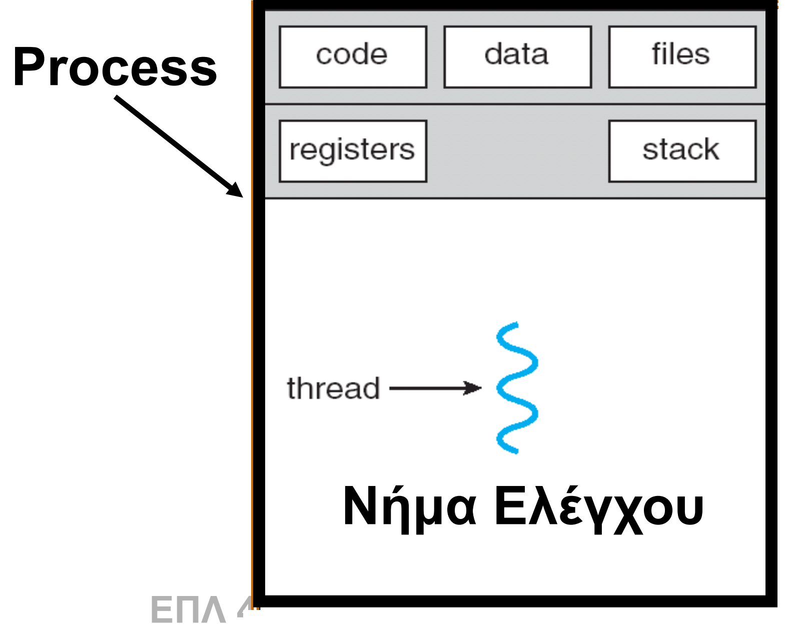


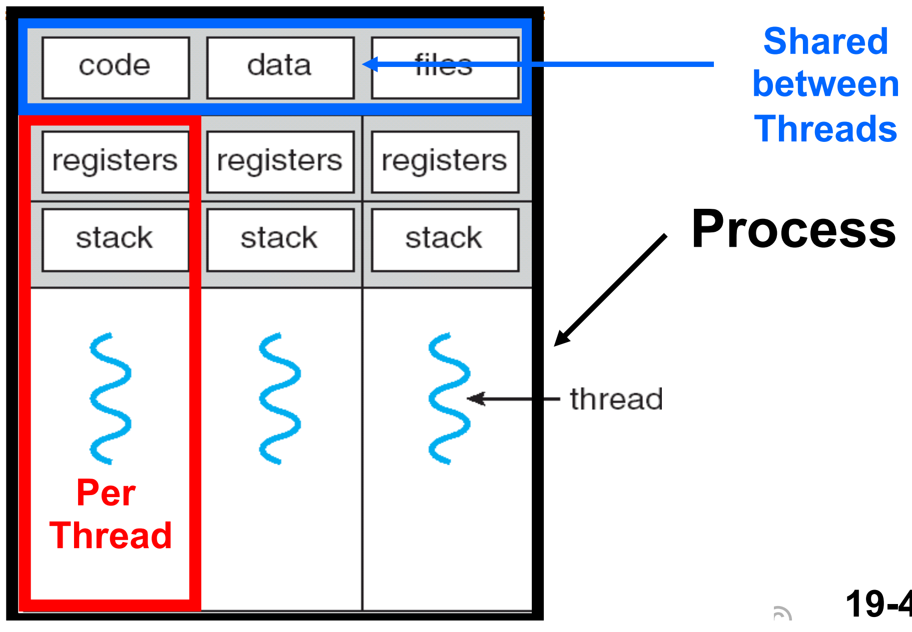
Α. Εισαγωγή στα Νήματα

- Μέχρι τώρα υποθέταμε ότι η βασική μονάδα χρονοδρομολόγησης στον επεξεργαστή ήταν η **διεργασία** (η οποία έχει **1 νήμα ελέγχου**).
- Τα περισσότερα σύγχρονα Λειτουργικό Σύστημα (Λ.Σ.) ωστόσο υποστηρίζουν πολλαπλά **νήματα ελέγχου ανά διεργασία.**
- **Νήματα - Threads (lightweight processes)**
  - **Η βασική μονάδα που χρονοδρομολογείται από το Λ.Σ. είναι το Νήμα** (και όχι η Διεργασία!)
  - Ένα ή περισσότερα νήματα μπορούν να εκτελούνται στο πλαίσιο μιας διεργασίας.
  - Το Λ.Σ. εκτελεί πολύ γρηγορότερα την εναλλαγή νημάτων από ότι την εναλλαγή διεργασιών

Α. Αναπαράσταση Νημάτων

– Ένα νήμα κατέχει προσωπικά: ένα ThreadID, ένα PC (Program Counter) & Καταχωρητές, και Στοίβα.
– Ένα νήμα διαμοιράζεται με τις υπόλοιπες διεργασίες το text segment (code), global variables, heap, και τους file descriptors.





A. Αναπαράσταση Νήματα

* http://lkml.indiana.edu/hypermail/linux/kernel/9608/0191.html

- Για να βρούμε τα threads μιας διεργασίας στο UNIX χρησιμοποιούμε ps –efm (e: all processes, f: full listing, m: show threads)

$ps -ef | grep tomcat # tomcat: servlet daemon (apache's extension για java servlets)

```
UID          PID   PPID   C STIME TTY TIME      CMD
tomcat4      792      1   1 19:01 ?   00:00:32 /usr/java/j2sdk1.4.1_02/bin/java
```

$ps -efm | grep tomcat

```
UID          PID   PPID   C STIME TTY TIME      CMD
tomcat4      792      1   0 19:01 ?   00:00:14 /usr/java/j2sdk1.4.1_02/bin/java
tomcat4      797    792   0 19:01 ?   00:00:00 /usr/java/j2sdk1.4.1_02/bin/java
.... (εδώ έχουμε ακόμα 111 threads με διαφορετικό PID) ....
tomcat4      988    797   0 19:01 ?   00:00:00 /usr/java/j2sdk1.4.1_02/bin/java
tomcat4      989    797   0 19:01 ?   00:00:00 /usr/java/j2sdk1.4.1_02/bin/java
```

- **Γιατί τα threads φαίνονται να έχουν διαφορετικό Process ID?***
  - Στο Linux (όχι σε άλλα UNIX), τα threads παρουσιάζονται σαν νέα processes στο process table του πυρήνα (i.e., tasks (threads|processes))
  - Αυτό συμβαίνει διότι στο Linux η εντολή δημιουργίας ενός thread υλοποιείται μέσω της κλήσης συστήματος **clone()**
    - Η clone επιτρέπει σε μια διεργασία να **μοιραστεί συγκεκριμένα συστατικά** της με άλλες διεργασίες (αυτό το χρειαζόμαστε για να δημιουργηθεί η έννοια του Νήματος).
    - Συγκεκριμένα, η **clone() μοιράζει** το **memory space**, τον πίνακα των **File Descriptors**, τον πίνακας των **signal handlers**, κτλ με μια νέα διεργασία. Επομένως στο Linux ισχύει ότι MAXTHREADS == MAXPROCESSES
    - Η **fork()** από την άλλη, **κλωνοποιεί ολόκληρη την διεργασία**

Α. Αναπαράσταση Νημάτων

- Τα λογισμικά συστήματα και πακέτα τα οποία χρησιμοποιείτε σε Unix ή Windows είναι συνήθως πολυνηματικά (multithreaded)

Windows Task Manager

File Options View Help

Applications | Processes | Performance | Networking

Image Name	PID	CPU	CPU Time	Mem Usage	Threads	I/O R
taskmgr.exe	3960	01	0:00:01	4,780 K	3	
avgupsvc.exe	3612	00	0:00:25	676 K	3	5
inetinfo.exe	3036	00	0:00:00	1,692 K	4	
wmpnetwk.exe	2996	00	0:00:01	1,196 K	14	
svchost.exe	2752	00	0:00:00	200 K	8	
acrotray.exe	2644	00	0:00:01	356 K	2	
bash.exe	2524	00	0:00:00	544 K	3	1
cmd.exe	2488	00	0:00:00	0 K	1	
iexplore.exe	2152	00	0:00:03	5,400 K	9	
explorer.exe	1964	00	0:10:55	30,312 K	19	136
wowexec.exe		00	0:00:01		1	
ntvdm.exe	1780	00	0:00:00	1,172 K	2	7
spoolsv.exe	1652	00	0:00:41	4,296 K	15	12
PcfMgr.exe	1564	00	0:00:10	2,524 K	5	
Adobelm_Cleanup...	1544	00	0:00:00	252 K	2	
svchost.exe	1432	00	0:00:00	3,696 K	18	
svchost.exe	1400	00	0:00:02	1,212 K	6	
svchost.exe	1312	00	0:04:46	14,512 K	69	666
HKWnd.exe	1268	00	0:00:01	884 K	3	
wmpnscfg.exe	1228	00	0:00:00	892 K	6	
svchost.exe	1152	00	0:00:22	1,860 K	10	
svchost.exe	1072	00	0:00:02	1,488 K	5	
ati2evxx.exe	1060	00	0:00:05	648 K	4	
lsass.exe	900	00	0:01:07	1,232 K	18	186
services.exe	888	00	0:01:28	1,968 K	16	
winlogon.exe	844	00	0:00:24	4,124 K	20	1
csrss.exe	804	00	0:05:07	3,040 K	12	2,663
jusched.exe	788	00	0:00:00	72 K	1	
ctfmon.exe	784	00	0:00:02	1,088 K	1	
smss.exe	748	00	0:00:00	88 K	3	
msimn.exe	700	00	0:00:16	30,068 K	12	56
zlclient.exe	652	00	0:00:47	4,736 K	6	63
HKServ.exe	624	00	0:00:01	1,420 K	4	
type32.exe	580	00	0:00:19	212 K	4	4
atiptaxx.exe	564	00	0:00:10	912 K	2	

Show processes from all users

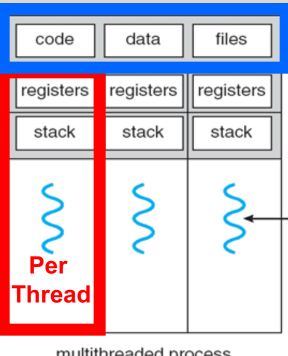
End Process

Processes: 43 | CPU Usage: 21% | Commit Charge: 654M / 2014M

Π.χ. ένας browser έχει ένα thread το οποίο ανακτά από το Διαδίκτυο την ιστοσελίδα που ζητήσατε ενώ ένα άλλο νήμα παράλληλα διεκπεραιώνει την μορφοποίηση στην οθόνη.

Αριστερά βλέπουμε ότι ο iexplore.exe (#2151) (Internet Explorer σε Windows) χρησιμοποιεί την συγκεκριμένη στιγμή 9 νήματα.

Β. Πλεονεκτήματα Νημάτων



1. **Responsiveness (Βελτίωση Χρόνου Απόκρισης Εφαρμογών)**
   - Χρήσιμο σε διαδραστικές εφαρμογές (GUIs), όπου μπορούμε να **«αποκρύψουμε»** μεγάλες **καθυστερήσεις** προερχόμενες από **I/Os**.
   - Σημειώστε ότι και ο ίδιος ο πυρήνας χρησιμοποιεί πολλαπλά threads (π.χ. έλεγχος διαθέσιμης μνήμης). | cat /proc/cpuinfo
2. **Resource Sharing (Διαμοιρασμός Πόρων)**
   - Ένα thread **διαμοιράζεται** όλα τα **δεδομένα** (ακόμα και η προσωπική **στοίβα** του είναι προσβάσιμη) με τα υπόλοιπα νήματα της ίδιας διεργασίας.
   - Επομένως **δεν χρειάζεται** κάποιος μηχανισμός **IPC** (shared memory, pipes, fifo, etc) για να μπορούν δυο νήματα να **μοιράζονται πόρους** (δομές δεδομένων, μεταβλητές, κτλ)

Β. Πλεονεκτήματα Νημάτων

3. **Economy: Οικονομία Δημιουργίας και Χρονοδρομολόγησης**:
   - Το fork() είναι πολύ ακριβό (γενικά η δέσμευση μνήμης είναι ακριβή).
   - Για παράδειγμα στο Solaris η **δημιουργία** μιας διεργασίας είναι **30 φορές πιο αργή** από την δημιουργία ενός νήματος,
   - Επίσης η εναλλαγή διεργασιών (**context switching)** είναι **5 φορές πιο αργή από την «εναλλαγή» νημάτων**

4. **Multiprocessing: Μέγιστη Αξιοποίηση Πολυεπεξεργαστών**
   - Σε συστήματα 1 επεξεργαστή τα νήματα εκτελούνται **ψευδό-παράλληλα,** ενώ ένα νήμα κάνει block (π.χ., I/O) ένα άλλο εκτελείται.
   - Σε συστήματα Ν επεξεργαστών τα νήματα εκτελούνται πραγματικά παράλληλα, Π.χ. Όλα τα λειτουργικά συστήματα υποστηρίζουν σήμερα **SMP (symmetric multiprocessing)**
     - 2 ή περισσότεροι επεξεργαστές σε κοινή μνήμη (π.χ. Intel's Xeon, Core Duo, Intel Quad Core, Intel Itanium, AMD Opteron)
     - Τεχνολογία **Hyper-threading** (SMT): multiple logical cores πάνω από 1 physical core

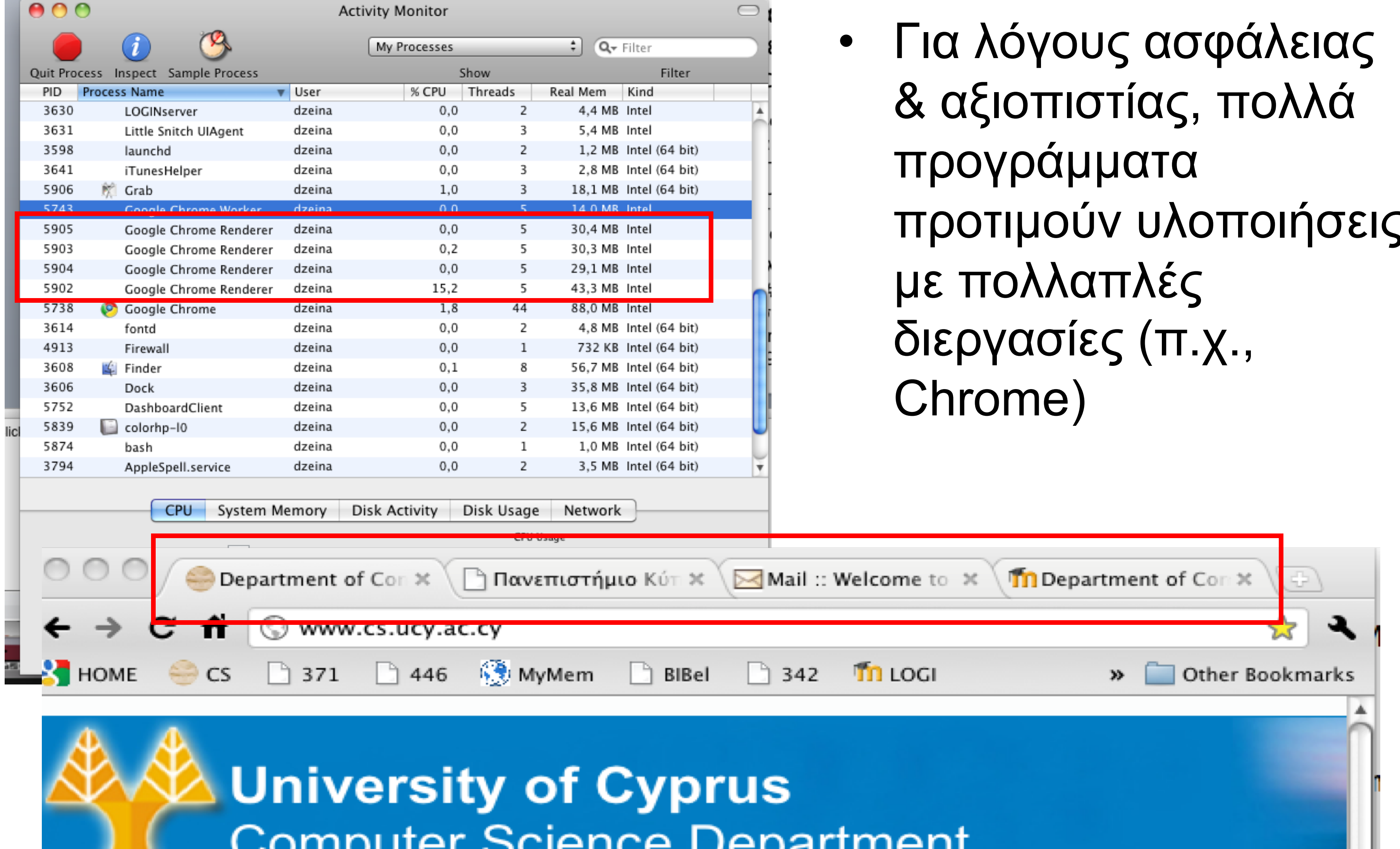
Dual CPU Core Chip
CPU Core and L1 Caches
CPU Core and L1 Caches
Bus Interface and L2 Caches

Β. Μειονεκτήματα Νημάτων

Όχι παρά πολλά μειονεκτήματα!

- **Ανάπτυξη Κώδικα:** Η ανάπτυξη κώδικα γίνεται σημαντικά πιο ακριβή λόγω προβλημάτων **συγχρονισμού** και **διαμοιρασμού κοινόχρηστης μνήμης** μεταξύ των νημάτων μιας διεργασίας.
- **Αποσφαλμάτωση:** Γίνεται πιο δύσκολη διότι είναι πιο δύσκολο να ελέγξουμε την ασύγχρονη ροή εκτέλεσης (παρόλο που εργαλεία όπως **gdb** υποστηρίζουν debugging με threads).
  - NPTL Trace tool (requires patching and recompilation of gcc)
  - (http://nptltracetool.sourceforge.net/).

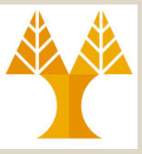
Β. Μειονεκτήματα Νημάτων



PID	Process Name	User	% CPU	Threads	Real Mem	Kind
3630	LOGINserver	dzeina	0,0	2	4,4 MB	Intel
3631	Little Snitch UIAgent	dzeina	0,0	3	5,4 MB	Intel
3598	launchd	dzeina	0,0	2	1,2 MB	Intel (64 bit)
3641	iTunesHelper	dzeina	0,0	3	2,8 MB	Intel (64 bit)
5906	Grab	dzeina	1,0	3	18,1 MB	Intel (64 bit)
5743	Google Chrome Worker	dzeina	0,0	5	14,0 MB	Intel
5905	Google Chrome Renderer	dzeina	0,0	5	30,4 MB	Intel
5903	Google Chrome Renderer	dzeina	0,2	5	30,3 MB	Intel
5904	Google Chrome Renderer	dzeina	0,0	5	29,1 MB	Intel
5902	Google Chrome Renderer	dzeina	15,2	5	43,3 MB	Intel
5738	Google Chrome	dzeina	1,8	44	88,0 MB	Intel
3614	fontd	dzeina	0,0	2	4,8 MB	Intel (64 bit)
4913	Firewall	dzeina	0,0	1	732 KB	Intel (64 bit)
3608	Finder	dzeina	0,1	8	56,7 MB	Intel (64 bit)
3606	Dock	dzeina	0,0	3	35,8 MB	Intel (64 bit)
5752	DashboardClient	dzeina	0,0	5	13,6 MB	Intel (64 bit)
5839	colorhp-I0	dzeina	0,0	2	15,6 MB	Intel (64 bit)
5874	bash	dzeina	0,0	1	1,0 MB	Intel (64 bit)
3794	AppleSpell.service	dzeina	0,0	2	3,5 MB	Intel (64 bit)

- Για λόγους ασφάλειας & αξιοπιστίας, πολλά προγράμματα προτιμούν υλοποιήσεις με πολλαπλές διεργασίες (π.χ., Chrome)

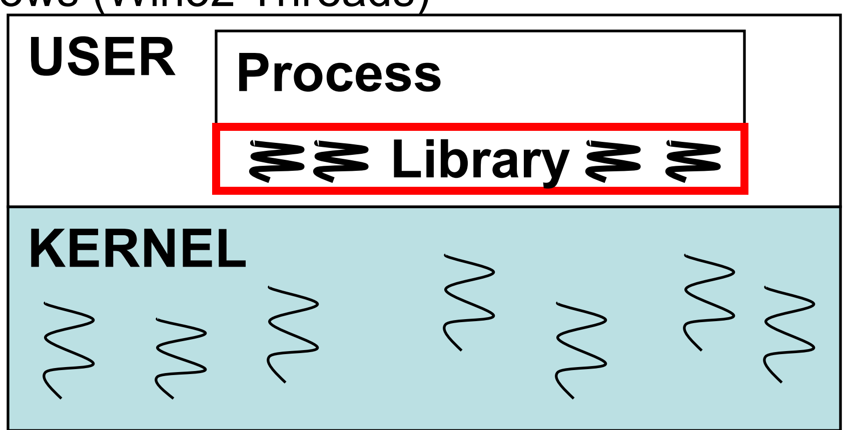
Γ. Είδη Νημάτων



- Υπάρχουν δυο είδη νημάτων
  - **User-Level Threads (στο user space)**
  - **Kernel-Level Threads (στο kernel space)**
- Η διαχείριση των **User-Level Threads** (δημιουργία, καταστροφή, etc) γίνεται μέσω βιβλιοθηκών στο user-space (επομένως είναι αποδοτική - δεν γίνονται system calls!)

Γ. Είδη Νημάτων

- Οι πιο γνωστές βιβλιοθήκες για **User-Level Threading**
  - **POSIX Pthreads**, τα οποία θα μελετήσουμε.
  - **Win32 Threads**, δείτε την σελίδα παρουσιάσεων του μαθήματος
  - **Java Threads**, τα προγράμματα JAVA εκτελούνται από το Java Virtual Machine το οποίο εκτελεί μια μορφή ενδιάμεσου κώδικα (το bytecode).
    - Το JVM υλοποιεί το Multithreading ανάλογα με το ΛΣ δηλ. σε Unix (Pthreads) ενώ σε Windows (Win32 Threads)



- Όλα τα σύγχρονα ΛΣ υποστηρίζουν και **Kernel-Threads** (Windows XP, Linux, MacOSX, Solaris, Unix) για την διαχείριση των πόρων του συστήματος.

Ε. Η Βιβλιοθήκη <pthread.h>

- Τώρα θα μελετήσουμε την POSIX Threads **<pthread.h>,** την πιο διαδεδομένη βιβλιοθήκη για προγραμματισμό νημάτων στο UNIX.
- Για να μεταγλωττίσουμε μια εφαρμογή, η οποία χρησιμοποιεί συναρτήσεις αυτής της βιβλιοθήκης, πρέπει να συμπεριλάβουμε την:
  #include <pthread.h>
- Για την μεταγλώττιση πρέπει να χρησιμοποιηθεί η επιλογή (option) του GCC –lpthread, δηλ.

  gcc –o <filename> <filename>.c –lpthread

Δημιουργία Νημάτων pthread_create()

- Όπως αναφέραμε πριν, κατά την διάρκεια δημιουργίας μιας Διεργασίας υπάρχει ακριβώς ένα Νήμα Ελέγχου (Thread-Control), δηλ Process = 1 Thread-Process
- ***Η συνάρτηση βιβλιοθήκης pthread_create δημιουργεί ένα <u>καινούργιο</u> νήμα που εκτελεί την συνάρτηση thread_f.***

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

Επιστρέφει 0 σε επιτυχία ή τον κωδικό λάθους

- **pthread_t *thread :** Το ThreadID του νήματος που δημιουργείται.
- **pthread_attr_t *attr :** Μπορούμε να ορίσουμε κάποια attr για το νήμα που δημιουργείται (προς το παρόν το αφήνουμε NULL και θα δούμε αργότερα αυτά τα attributes).
- **void * (* *thread_f*) (void *) :** Δείκτης σε συνάρτηση που παίρνει ως όρισμα οτιδήποτε (void *) και επιστρέφει οτιδήποτε (void *). Αυτή η συνάρτηση εκτελείται με την δημιουργία του νήματος.
  - Σημειώστε ότι στην fork() η εκτέλεση μιας διεργασίας συνεχίζει από το σημείο του fork() ενώ εδώ η εκτέλεση του νήματος συνεχίζει από το thread_f).
- **void *arg** : Παράμετρος της συνάρτησης του νήματος (μόνο μια). Εάν θέλετε να περάσετε περισσότερες τοποθετήσετε τις σε ένα *struct* και περάσετε εδώ την διεύθυνση του struct.
  - Εναλλακτικά δηλώσετε τις παραμέτρους *globally* (συνιστάται να αποφεύγετε όμως).

Παράδειγμα: Hello Thread

```
#include <pthread.h> /* Pthread related functions*/
#include <string.h> /* For strerror */
#include <stdio.h> /* For printf */
#include <errno.h> /* For errno variable */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
void *runner1(); /* function prototype of thread's code */
char *name; // global variable accessible by all threads!

int main(int argc, char *argv[]) {
    int err;    // error code
    pthread_t tid;  // Thread ID

    if (err = pthread_create( &tid, NULL, &runner1, NULL)) {
                perror2("pthread_create", err);  exit(1);
     }
    printf("ProcessID:%d created ThreadID:%u\n", getpid(), (unsigned int)tid);
    // wait for child to finish – We will replace the sleep() with pthread_join() in a few slides
    sleep(1);     printf("Name: %s\n", name);     return 0;
}

void *runner1() {
     printf("Hello: I am the child thread!\n");
     name = (char *)malloc(50);   strcpy(name,"ThreadWorker");
     pthread_exit( 0 );
}
```

Ανοικτά αρχεία (fopen), δεσμευμένη μνήμη (malloc), κτλ, παραμένουν και μετά τον τερματισμό του νήματος. Επομένως πρέπει να αποδεσμεύουμε αυτό τον χώρο ρητά προτού τερματιστούμε το νήμα!

```
$./hellothread
Hello: I am the child thread!
ProcessID:1596 created
ThreadID:6685000
Name: ThreadWorker
```

Ε. Η Βιβλιοθήκη <pthread.h>

Διαχείριση Λαθών

- Οι συναρτήσεις **βιβλιοθήκης pthread** δεν θέτουν την τιμή **errno** σε περίπτωση λάθους (για λόγους καλύτερης δομής).
  - Σημειώστε ότι η μεταβλητή αυτή είναι στη στοίβα (συνεπώς είναι προσωπική για κάθε νήμα)
- Συνεπώς, δεν μπορεί να χρησιμοποιηθεί η συνάρτηση **perror(char *)** για την εκτύπωση του διαγνωστικού μηνύματος.
  - Μπορείτε ωστόσο να χρησιμοποιείτε την **perror** μέσα σε ένα νήμα για άλλους λογούς, πχ. εάν το νήμα εκτελεί κάποια συνάρτηση που θέτει το errno (malloc(), open(), κτλ).
- Σε περίπτωση λάθους σε κλήσεις βιβλιοθήκης νημάτων θα χρησιμοποιούμε την συνάρτηση βιβλιοθήκης strerr
  - **char *strerr(int errnum)** Απαίτηση: #include <string.h>
  - Επιστρέφει μια συμβολοσειρά που περιγράφει το λάθος που αντιστοιχεί στον κωδικό λάθους errnum.
  - Π.χ. Εάν δημιουργήσετε 1000 threads θα πάρετε το μήνυμα «Cannot allocate memory»
  - Στα προγράμματα μας θα κάνουμε define την perror2 ως εξής:

```
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
```

Τερματισμός Νημάτων pthread_exit(), pthread_cancel()

- Εάν ένα νήμα καλέσει την συνάρτηση **exit(),** τότε αυτόματα διακόπτεται η διεργασία (και όλα τα νήματα).
- Για διακοπή **συγκεκριμένου** νήματος έχουμε τις επιλογές:
  - A) Κάνουμε **return((void *) exitcode);** μέσα στο main του νήματος **ή**
  - B) Καλούμε την συνάρτηση **pthread_exit()** μέσα στο νήμα το οποίο τερματίζει το νήμα που κάνει την κλήση, **ή**
  - C) Ένα άλλο νήμα (στην ιδία διεργασία) διακόπτει ρητά το νήμα. (με την συνάρτηση **int pthread_cancel(pthread_t tid));**

#include <pthread.h>

void pthread_exit(void *retval);

Επιστρέφει void. Το retval δηλώνει τον κωδικό λάθους

- Εάν καλέσει κανείς pthread_exit() στο τέλος του main(), τότε το συγκεκριμένο thread περιμένει μέχρι να τερματίσουν όλα τα threads την εκτέλεση τους.
- **Προσοχή:** Το **Garbage Collection** των ανοικτών αρχείων, δεσμευμένης μνήμης (αυτά είναι στο κοινόχρηστο heap) πρέπει να γίνεται πριν τον τερματισμό του νήματος από τον προγραμματιστή.
  - Αυτό επειδή ένα νήμα-γονέα δεν γνωρίζει τι δεσμεύει ένα νήμα-παιδιού
  - Όταν ολοκληρωθεί ένα πρόγραμμα βέβαια αποδεσμεύονται όλα αυτόματα

Αναγνώριση Νημάτων pthread_self()

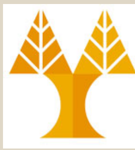
- Κάθε νήμα χαρακτηρίζεται, όπως και οι διεργασίες από ένα **ThreadID**.
- Το **ThreadID** έχει νόημα μόνο στην **εμβέλεια μιας διεργασίας**.
- Επομένως εάν υπάρχουν δυο διεργασίες Α, Β τότε αυτές μπορεί να έχουν νήματα με τα ιδία ThreadIDs.

```
#include <pthread.h>
pthread_t pthread_self(void);
```

  - Επιστρέφει το threadID του νήματος που κάνει την κλήση
- Το pthread_t είναι ένας unsigned int (printf("%u", tid))
- Για να συγκρίνουμε δυο threadIDs χρησιμοποιούμε την πιο κάτω συνάρτηση:

```
pthread_t tid1, tid2; int ret = pthread_equal(tid1, tid2);
```

Αναγνώριση Νημάτων pthread_self()



```
#include <pthread.h> /* Pthread related functions*/
#include <string.h> /* For strerror */
#include <stdio.h> /* For fopen, printf */
#include <errno.h> /* For errno variable */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
#define THREADS 5
void *runner1(); /* function prototype of thread's code */

int main(int argc, char *argv[]) {
   int err, i;    // error code and loop counter
   pthread_t tid[THREADS];  /* thread id  table*/

   for (i=0; i<THREADS; i++) {
           /* create thread */
          if (err = pthread_create( &tid[i], NULL, &runner1, NULL)) {
               perror2("pthread_create", err);  exit(1);
          }
          printf("ProcessID:%d created ThreadID:%u\n", getpid(),
                 (unsigned int)tid[i]);
   }
   // wait for child to finish – We will replace the sleep() with
   // pthread_join() in a few slides
   sleep(1);
   return 0;
}

void *runner1() {
   printf("I am thread %u\n", pthread_self());    pthread_exit( 0 );
}
```

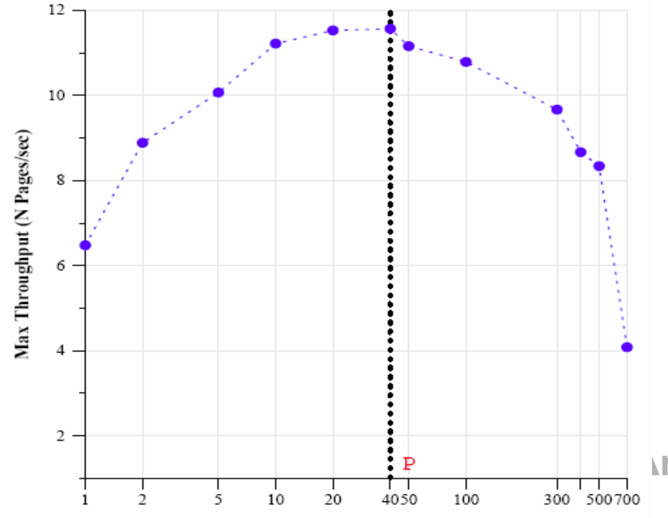
Δημιουργία 5 threads όπου κάθε thread εκτυπώνει το TID και το thread-γονέας το PID και το TID του thread που δημιούργησε.

Αποτέλεσμα Εξόδου
(μπορεί να είναι με αυθαίρετη σειρά)

```
ProcessID:20261 created ThreadID:3086691232
ProcessID:20261 created ThreadID:3076201376
ProcessID:20261 created ThreadID:3065711520
ProcessID:20261 created ThreadID:3055221664
ProcessID:20261 created ThreadID:3044731808
I am thread 3086691232
I am thread 3076201376
I am thread 3065711520
I am thread 3055221664
I am thread 3044731808
```

Αριθμός Νημάτων ανά Διεργασία

- Το πιο κάτω γράφημα δείχνει το Throughput (ανακτημένες ιστοσελίδες / δευτερόλεπτο) ενός πολυνηματικού crawler.
- Το γράφημα δείχνει οι ο βέλτιστος αριθμός νημάτων στην συγκεκριμένη εφαρμογή είναι 40 νήματα.
- **Η απόδοση με 700 νήματα είναι χειρότερη από αυτή με 1 μονό thread!**
- Επομένως πρέπει να είμαστε προσεκτικοί με τον αριθμό νημάτων που χρησιμοποιούμε σε μια εφαρμογή εφόσον η αλόγιστη χρήση νημάτων θα οδηγήσει σε μείωση της επίδοσης ενός συστήματος
  - Αυτό εφόσον ο περισσότερος χρόνος θα αναλώνεται σε εναλλαγή νημάτων.



Σημειώστε ότι ο crawler είναι μια τυπική εφαρμογή με πολλά I/O operations.

Συνεπώς η χρήση νημάτων επιβάλλεται!

Ορφανά Νήματα?

- Η σειρά εκτέλεσης των νημάτων είναι **αυθαίρετη**.
  - Τόσο μεταξύ νημάτων-παιδιών, όσο και σε σχέση με το νήμα-γονέα που δημιούργησε το νήμα-παιδί.
- Επομένως, θα υπάρχουν περιπτώσεις όπου ο γονέας θα **ολοκληρώσει** προτού τα παιδιά του **προλάβουν** να εκτελεστούν.

- Άραγε αυτό δημιουργεί, όπως τις διεργασίες την έννοια του **ορφανού νήματος?**
  - **Απάντηση: ΌΧΙ.** Αν **ολοκληρώσει** το **νήμα-γονέας** τότε **τερματίζει** αναγκαστικά και το **νήμα-παιδί** (θα δούμε στη συνέχεια ένα τρόπο αναμονής των παιδιών με την phtread_join())
- Το θέμα αυτό παρουσιάζεται με ένα παράδειγμα στην επόμενη διαφάνεια.

Παράδειγμα: Ορφανά Νήματα?

```
#include <pthread.h> /* Pthread related functions*/
#include <string.h> /* For strerror */
#include <stdio.h> /* For fopen, printf */
#include <errno.h> /* For errno variable */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
#define THREADS 5
void *runner1(); /* function prototype of thread's code */

int main(int argc, char *argv[]) {
   int err, i;    // error code and loop counter
   pthread_t tid[THREADS];  /* thread id  table*/

   for (i=0; i<THREADS; i++) {
            /* create thread */
          if (err = pthread_create( &tid[i], NULL, &runner1, NULL)) {
                perror2("pthread_create", err);  exit(1);
          }
          printf("ProcessID:%d created ThreadID:%u\n", getpid(), (unsigned int)tid[i]);
   }
  // Εδώ το νήμα-γονέα τερματίζει – και τα νήματα-παιδιά δεν πρόλαβαν να εκτελεστούν.
  return 0;
}

void *runner1() {
   sleep(3); // Εδώ κάνουμε sleep 3 sec για να ολοκληρώσει ο γονέας προτού προλάβει το νήμα να εκτελεστεί
   printf("Hello!");    pthread_exit( 0 );
}
```

Ολοκληρωση γονεα πριν τα νηματα-παιδι

Αποτέλεσμα Εξόδου

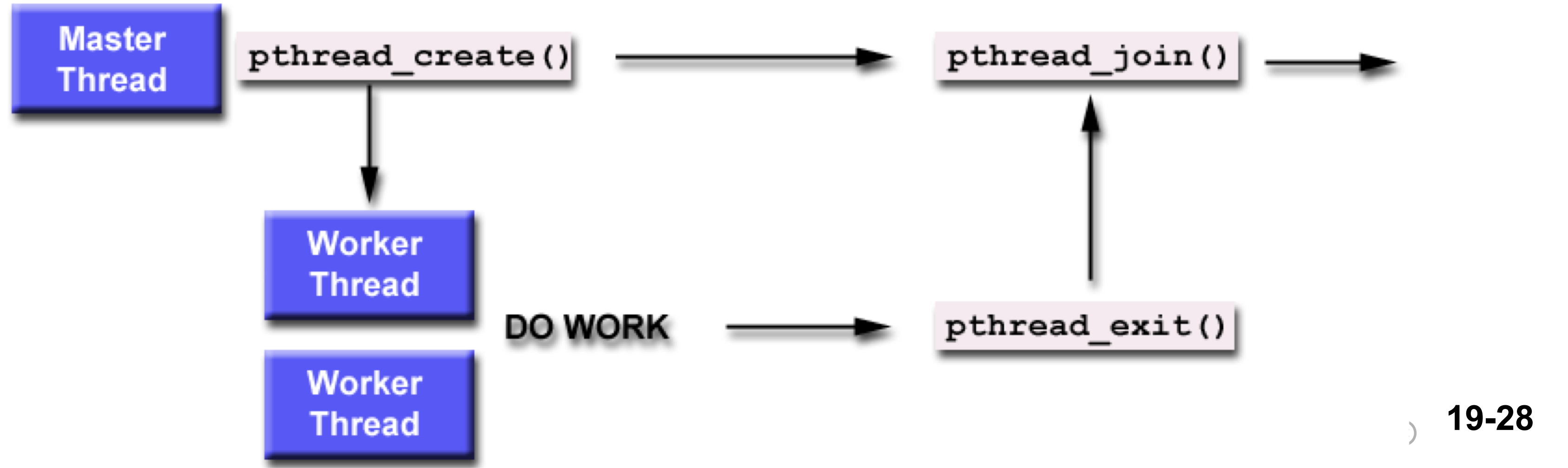
ProcessID:20261 created ThreadID:3086691232
ProcessID:20261 created ThreadID:3076201376
ProcessID:20261 created ThreadID:3065711520
ProcessID:20261 created ThreadID:3055221664
ProcessID:20261 created ThreadID:3044731808

Επομένως ΚΑΝΕΝΑ νήμα παιδί δεν πρόλαβε να εκτελεστεί ☹.

Επομένως θέλουμε κάποιο μηχανισμό για να κάνουμε wait() τα νήματα παιδιά

Αναμονή Νημάτων pthread_join()

- Όταν δημιουργείται ένα thread, τότε αυτό είναι εξ' ορισμού **προσαρτημένο (attached)** στο νήμα-γονέα.
- Αυτό σημαίνει ότι για να αποδεσμεύσει τους πόρους το παιδί (την στοίβα, text) πρέπει ο γονέας (ή κάποιο άλλο νήμα) να λάβει τον κωδικό εξόδου του νήματος με την **pthread_join** (επόμενη διαφάνεια)



Αναμονή Νημάτων pthread_join()

int pthread_join(pthread_t thread, void *retaddr)

Επιστρέφει 0 σε επιτυχία ή κωδικό λάθους.

- Περιμένει τον τερματισμό του **προσαρτημένου νήματος** με ταυτότητα **thread**.
- Έχει αντίστοιχη λειτουργία με την συνάρτηση **waitpid()** που είδαμε στις διεργασίες.
- Ο κωδικός εξόδου του νήματος που τερμάτισε, όπως δόθηκε από την **pthread_exit** του νήματος-παιδιού επιστρέφεται στο ***retaddr**.

Αναμονή Νημάτων pthread_join()

```
#include <pthread.h> /* Pthread related functions*/
#include <string.h> /* For strerror */
#include <stdio.h> /* For fopen, printf */
#include <errno.h> /* For errno variable */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
#define THREADS 2
void *runner1(); /* function prototype of thread's code */

int main(int argc, char *argv[]) {
   int err, i, status;          // error code, loop counter and exit status
   pthread_t tid[THREADS];  /* thread id  table*/

   for (i=0; i<THREADS; i++) {
            /* create thread */
          if (err = pthread_create( &tid[i], NULL, &runner1, NULL)) {
               perror2("pthread_create", err);  exit(1);
          }
          printf("ProcessID:%d created ThreadID:%u\n", getpid(), (unsigned int)tid[i]);
   }
   for (i=0; i<THREADS; i++) {
          if (err = pthread_join( tid[i], (void **) &status)) {  /* wait for thread to end */
               perror2("pthread_join", err);  exit(1);
          }
          printf("Thread %d exited with code %d\n", tid[i], status);
   }
return 0;
}

void *runner1() {    printf("I am thread %u\n", pthread_self());    pthread_exit( (void *)47); }
```

Αναμονή και εκτύπωση κωδικού εξόδου του νήματος. (με αυθαίρετη σειρά)

I am thread 6685000 **// νήμα-παιδί**
ProcessID:2428 created ThreadID:6685000
I am thread 6686216 **// νήμα-παιδί**
ProcessID:2428 created ThreadID:6686216
Thread 6685000 exited with code 47
Thread 6686216 exited with code 47

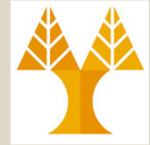
Παράδειγμα 1

Υπολογισμός του $\sum_{i=1}^{n} i$ με ένα νήμα

Κάνοντας χρήση **ενός (1) νήματος** και της βιβλιοθήκης pthread στην C, να γράψετε ένα πρόγραμμα το οποίο δεδομένου του **n** να υπολογίζει το άθροισμα των πρώτων **n** θετικών ακεραίων (χωρίς χρήση της κλειστής μορφής του αθροίσματος $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$)

Παράδειγμα 1

Υπολογισμός του $\sum_{i=1}^{n} i$ με ένα νήμα



```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e)) /* Error printing function */

int sum; /* this data is shared by the two threads*/
void *runner(void *param); /* function prototype of thread's code */

int main(int argc, char *argv[]) {
        int err;        // error code
        pthread_t           tid;  /* thread id */
        pthread_attr_t    attr;   /* set of thread attributes */
        pthread_attr_init ( &attr ); /* get default thread attributes */

        /* create thread */
        if (err = pthread_create( &tid, &attr, &runner, argv[1])) {
                perror2("pthread_create", err);            exit(1);
        }

        if (err = pthread_join( tid, NULL)) { /* wait for thread to end */
                perror2("pthread_join", err);              exit(1);
        }
        printf("sum = %d\n", sum );
}
```

n

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

    for ( i = 1; i <= upper; i++ )
        sum += i;

    pthread_exit( 0 );
}
```

Παράδειγμα 2

Σύγκριση Απόδοσης Υπολογισμού $\sum_{i=1}^{n} i$

Λύσετε το Παρ. 1 ($\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$)
με M νήματα και συγκρίνετε τον
χρόνο εκτέλεσης για
N=1,000,000,000 με χρήση M=1
και M=10 νήματα.

Παράδειγμα 2

Υπολογισμός του $\sum_{i=1}^{n} i$ με Ν νήματα

```
#include <pthread.h> /* Pthread related functions*/
#include <string.h> /* For strerror */
#include <stdio.h> /* For fopen, printf */
#include <errno.h> /* For errno variable */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
#define THREADS 10

void *runner(); /* function prototype of thread's code */

//  unsigned long long (64 bits) i.e., (0 to 18,446,744,073,709,551,615)
typedef struct {
    unsigned long long low;      // data “passed” from parent-thread to child-thread
    unsigned long long high;     // data “passed” from parent-thread to child-thread
    unsigned long long mysum;  // data “passed” from “child-thread” to  parent-thread
} BOUNDS;

int main(int argc, char *argv[]) {
   int err, status;      // error code, loop counter and exit status
        pthread_t tid[THREADS];  /* thread id  table*/
        BOUNDS bounds[THREADS];
        unsigned long long sum = 0; /* final (global) sum value */
        unsigned long long i;       /* thread id  table*/
        unsigned long long N=1000000000;
        N/=THREADS;
```

Παράδειγμα 2

Υπολογισμός του $\sum_{i=1}^{n} i$ με N νήματα

```
if (THREADS == 1) { // Σε αυτή τη περίπτωση δεν καλούμε τη pthread_create() αλλά εκτελούμε τον
                    // runner απευθείας από το νήμα-γονέα.
        bounds[0].low = 1;  bounds[0].high = N;  bounds[0].mysum = 0;
        runner((void *) &bounds[0]);
}
else {          // Δημιουργία N νημάτων. Κάθε νήμα τροφοδοτείτε με μια δομή bounds
        for (i=0; i<THREADS; i++) {
                bounds[i].low = i*N;
                bounds[i].high = bounds[i].low + N;
                bounds[i].mysum = 0;

                /* create thread */
                if (err = pthread_create( &tid[i], NULL, &runner, (void *) &bounds[i])) {
                        perror2("pthread_create", err);  exit(1);
                }
        }
        // Αναμονή Ολοκλήρωσης των N Νημάτων
        for (i=0; i<THREADS; i++) {
                if (err = pthread_join( tid[i], (void **) &status)) {  /* wait for thread to end */
                        perror2("pthread_join", err);  exit(1);
                }
        }
}
// Υπολογισμός τελικού αθροίσματος
 for (i=0; i<THREADS; i++) {
        sum += bounds[i].mysum;     printf("Sum[%lld]=%lld\n", i, bounds[i].mysum);
 }
printf("Final Sum is : %lld\n", sum);
return 0;
}
```

Παράδειγμα 2

Υπολογισμός του $\sum_{i=1}^{n} i$ με N νήματα

```
void *runner(void *param) {
    int i,j;
    BOUNDS *bounds = (BOUNDS *) param;
    printf("[%lld,%lld]\n", bounds->low+1, bounds->high);
    for ( i = bounds->low+1; i <=bounds->high; i++ )
            bounds->mysum += i;
    pthread_exit( 0 );  // το νήμα-γονέα περιμένει στο pthread_join()
}
```

- Εκτελούμε το πρόγραμμα 10 φορές για και τυπώνουμε τον μέσο όρο.
- Χρησιμοποιούμε την μηχανή των εργαστηρίων cs4030.in (1 core with Hyperthreading - 2 logical cores) 3.6GHz με την Native Posix Thread Library.

Wall clock

Με 1 νήμα
$time padd
real 0m7.3127s
user 0m7.3117s
sys 0m0.0013s

Με 10 νήματα
$time padd
real 0m8.5703s
user 0m17.1011s
sys 0m0.004s

Παρατηρούμε ότι ο χρόνος αυξήθηκε!
Εάν ωστόσο είχαμε μια εφαρμογή όπου τα νήματα έκαναν I/O, τότε θα ήταν πολύ γρηγορότερα τα δέκα νήματα

Τα νήματα δεν αφιερώνουν καθόλου χρόνο στον πυρήνα (δηλ., system (kernel) time σχεδόν 0)

Παράδειγμα 3

Κάνοντας χρήση της βιβλιοθήκης **pthread** στην C, να γράψετε ένα πρόγραμμα το οποίο να δημιουργεί **ένα αριθμό από νήματα** (το οποίο δίδεται σαν παράμετρος στο πρόγραμμα), **καθένα από τα οποία να καθυστερεί να τερματίσει ένα τυχαίο αριθμό δευτερολέπτων**.

Παράδειγμα 3

```
#include <stdio.h> /* For printf */
#include <string.h> /* For strerror */
#include <stdlib.h> /* For srandom and random */
#include <pthread.h> /* For threads */
#define MAX_SLEEP 10 /* Maximum sleeping time in seconds */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e)) /* Error message printing function */
void *sleeping(void *); /* Forward declaration */

int main(int argc, char *argv[]) {
        int n, i, sl, err;
        pthread_t *tids;
        if (argc > 1) n = atoi(argv[1]); /* Make integer */
        else exit(0);

        if (n > 50) { /* Avoid too many threads */
            printf("Number of threads should be up to 50\n"); exit(0);
        }
        // Δημιουργία πίνακα από n pthread_t αντικείμενα.
        if ((tids = (pthread_t *) malloc(n * sizeof(pthread_t))) == NULL) {
            perror("malloc"); exit(1);
        }
```

Παράδειγμα 3

Συνέχεια από προηγουμένη σελίδα…

```
srandom((unsigned int) time(NULL)); /* Initialize generator */

for (i=0 ; i<n ; i++) {
    sl = random() % MAX_SLEEP + 1; /* [1..MAX_SLEEP] */
    if (err = pthread_create(tids+i, NULL, sleeping, (void *) sl)) {
        /* Create a thread */
        perror2("pthread_create", err); exit(1);
    }
}

for (i=0 ; i<n ; i++)
    if (err = pthread_join(*(tids+i), NULL)) {
        /* Wait for thread termination */
        perror2("pthread_join", err); exit(1);
    }
printf("all %d threads have terminated\n", n);
}
```

Δεδομένα Εξόδου Προγράμματος

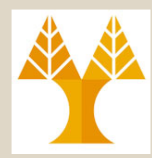
```
$threadsleep  3
thread 6685016 sleeping 1 seconds ...
thread 6686232 sleeping 6 seconds ...
thread 6686368 sleeping 4 seconds ...
thread 6685016 awakening
thread 6686368 awakening
thread 6686232 awakening
all 3 threads have terminated
```

```
// Causes thread to sleep for arg seconds
void *sleeping(void *arg)
{
    int sl = (int) arg;
    printf("thread %d sleeping %d seconds ...\n",
               pthread_self(), sl);
    sleep(sl); /* Sleep a number of seconds */
    printf("thread %d awakening\n",
               pthread_self());
    pthread_exit(0);
}
```

Απόσπαση (Detachment) Νημάτων pthread_detach()

- Εάν δεν **παραληφθεί** ο **κωδικός εξόδου** από τον γονέα (ή κάποιο άλλο νήμα) τότε δημιουργείται ένα “zombie”-νήμα (όπως στις διεργασίες!), το οποίο κατακρατεί άσκοπα τους δεσμευμένους πόρους.
- Εάν θέλουμε να ολοκληρώσει το νήμα οποτεδήποτε αυτό θέλει τότε μπορούμε να το **αποσπάσουμε (detach)** με την χρήση της συνάρτησης pthread_detach() (είτε μέσα στον γονέα ή μέσα στο παιδί).
- Τα περισσότερα νήματα που δημιουργούμε συνήθως είναι αποσπασμένα.

Απόσπαση Νημάτων pthread_detach()



- **Η συνάρτηση βιβλιοθήκης pthread_detach επιτρέπει σε ένα νήμα-παιδί να ολοκληρώσει την εκτέλεση του χωρίς να χρειάζεται να γίνει join από κάποιο άλλο νήμα-γονέα.**

int pthread_detach(pthread_t thread);
Επιστρέφει 0 σε επιτυχία ή τον κωδικό λάθους.

- Η συνάρτηση μετατρέπει το νήμα με ταυτότητα thread από **προσαρτημένο (attached)** σε **αποσπασμένο (detached).**
- Το detached νήμα απελευθερώνει άμεσα τους δεσμευμένους πόρους μετά την ολοκλήρωση της εκτέλεσης του.
- Επομένως ο γονέας δεν χρειάζεται να εκτελέσει **pthread_join**. Εάν το καλέσει τότε θα πάρει μήνυμα λάθους EINVAL.

Παράδειγμα 4

```
#include <stdio.h> /* For printf */
#include <string.h> /* For strerror */
#include <pthread.h> /* For threads */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))
/* Error message printing function */
void *thread_f(void *); /* Forward declaration */

int main()
{
        pthread_t thr;
        int err, arg = 29;
        if (err = pthread_create(&thr, NULL, thread_f, (void *) &arg)) {/* New thread */
            perror2("pthread_create", err); exit(1);
        }

        if (err = pthread_join( thr, NULL)) {  /* wait for thread to end */
            perror2("pthread_join", err);  exit(1);
        }
        printf("I am original thread %d and I created thread %u\n", pthread_self(), thr);
        pthread_exit(0);
}

void *thread_f(void *argp) /* Thread function */
{
        int err;
        if (err = pthread_detach(pthread_self())) {/* Detach thread */
            perror2("pthread_detach", err);          exit(1);
        }
        printf("I am thread %u and I was called with argument %d\n",pthread_self(), *(int *) argp);
        return;
}
```

Το νήμα-παιδί δεν χρειάζεται να επιστρέψει κάτι στο νήμα-γονέα

$./detach
I am thread -1208788080 and I was called with argument 29
pthread_join: Invalid argument

(a)

X

(b)