Πανεπιστήμιο Δυτικής Μακεδονίας
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα

Ενότητα 8:Διεργασίες και νήματα (δημιουργία, καταστροφή, συγχρονισμός).

Επιβλέπων: Δασυγένης Μηνάς
Παυλίδου Ελένη

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Λειτουργικών Συστημάτων
http://arch.ece.uowm.gr/courses/os/

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 στο Πανεπιστήμιο 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.

Η συνάρτηση time() (1/3)

`time_t time(time_t *tloc)`

- Επιστρέφει τον ημερολογιακό χρόνο από την 00:00:00 GMT, Jan. 1, 1970 (γνωστή ως 'the epoch', GMT=Greenwitch Mean Time), μετρούμενο σε δευτερόλεπτα.
- Αν ο δείκτης `tloc` δεν είναι NULL, η τιμή που επιστρέφει η συνάρτηση επίσης αποθηκεύεται στη θέση που δείχνει ο `tloc`.
- Σε περίπτωση επιτυχίας η συνάρτηση `time()` επιστρέφει το χρόνο, ενώ σε περίπτωση αποτυχίας `-1`. Ο τύπος `time_t` ορίζεται ως `long int` στη βιβλιοθήκη `time.h`.

Η συνάρτηση time() (2/3)

- Εκτός από την απλή δομή που ορίζει το χρόνο σε δευτερόλεπτα υπάρχουν πολλές διαφορετικές δομές χρόνου ανάλογα με τη χρήση, π.χ. τοπικός ημερολογιακός χρόνος (ημέρα, μήνας, θερινή ώρα, ζώνη ώρας κλπ) ή ώρα μεγάλης ακρίβειας, (σε microsec ή σε nanosec).

  Έτσι έχουμε, λόγου χάρη τη συνάρτηση:

  `struct tm *localtime(const time_t *time)`

  δέχεται ως όρισμα τον ημερολογιακό χρόνο σε δευτερόλεπτα και επιστρέφει ένα δείκτη σε μια αναλυτική δομή χρόνου `tm` που ορίζεται στη βιβλιοθήκη `time.h` και περιλαμβάνει, δευτερόλεπτα, λεπτά, ώρες, ημέρα, μήνα, έτος, ημέρα της εβδομάδας, ημέρα του χρόνου ζώνη ώρας κ.α.

Η συνάρτηση time() (3/3)

- Οι συναρτήσεις:

```
char *ctime(const time_t *time)
char *asctime(const tm *time)
```

μεταφράζουν το χρόνο από τύπο `time_t` ή δομή `tm` στη γνωστή αναγνώσιμη συμβολοσειρά που εμφανίζει η εντολή `date` του UNIX/Linux. Αντίθετα η συνάρτηση

```
struct tm *getdate(const char *string)
```

μεταφράζει μια συμβολοσειρά ημερομηνίας σε δομή tm. Οι συναρτήσεις ορίζονται στο `time.h`.

Η συνάρτηση vfork() (1/3)

- Ίδια χρήση με τη `fork`, διαφορετική σημασιολογία για πιο αποτελεσματική υλοποίηση όταν πρόκειται να εκτελεσθεί άμεσα `exec`.
  - Η σημασιολογία της `vfork` είναι περίεργη, και ειδικότερα στα σύγχρονα συστήματα με τεχνικές copy-on-write καλό είναι να αποφεύγεται.

- Η `vfork` δημιουργεί τη νέα διεργασία *χωρίς* να αντιγράφει τον χώρο διευθύνσεων.

  - Η θυγατρική διεργασία εκτελείται στον χώρο διευθύνσεων της γονικής, μέχρις ότου καλέσει την `exec` ή την `exit`.
  - Ο γονέας αναστέλλεται μέχρις ότου η θυγατρική διεργασία καλέσει την `exec` ή την `exit`.

Η συνάρτηση vfork() (2/3)

❖ Παράδειγμα:

```
#include <sys/types.>
#include <unistd.h>
 int glob = 6;
/* μεταβλητή στα αρχικοποιημένα δεδομένα */
int main(void)
{
 int  var;
 /* τοπική μεταβλητή στη στοίβα */
      pid_t pid;
      var = 88;
      printf("before vfork\n");
/* χωρίς εκκένωση κανονικής εξόδου */
```

Η συνάρτηση vfork() (3/3)

…συνέχεια παραδείγματος

```
if ((pid = vfork())< 0)
  perror("vfork error");
  else if (pid == 0) { /* θυγατρική διεργασία */
     glob++;    /* τροποποίηση μεταβλητών γονέα */
     var++;
      printf("pid=%d, glob = %d, var =
%d\n", getpid(), glob, var);
          _exit(0); /* Προσοχή: μόνο _exit, όχι exit! */
     }
    printf("pid = %d, glob = %d, var =
%d\n", getpid(), glob, var);
     exit(0);
```

Η συνάρτηση pthread_create() (1/5)

- Αρχικά, το κύριο πρόγραμμα - `main()`- αποτελείται από ένα μόνο προκαθορισμένο νήμα. Όλα τα υπόλοιπα νήματα πρέπει να δημιουργηθούν ξεχωριστά από τον προγραμματιστή.

- Η συνάρτηση `pthread_create` δημιουργεί ένα καινούριο νήμα και το κάνει εκτελέσιμο. Αυτή η ρουτίνα μπορεί να κληθεί όσες φορές θέλει ο προγραμματιστής μέσα στον κώδικα του.

Η συνάρτηση pthread_create() (2/5)

- Παράμετροι της `pthread_create`:

- **`thread`**: Ένα μοναδικό αναγνωριστικό (ID) για το καινούριο νήμα που επιστρέφεται από την υπορουτίνα. Είναι τύπου `pthread_t`.

- **`attr`**: Ένα αντικείμενο ιδιοτήτων το οποίο μπορεί να χρησιμοποιηθεί για να ορίσει ιδιότητες των νημάτων. Ένα τέτοιο αντικείμενο είτε δημιουργείται από το χρήστη, πριν τη δημιουργία του νήματος, ή μπορεί να χρησιμοποιηθεί το NULL για τις προκαθορισμένες τιμές ιδιοτήτων.

Η συνάρτηση pthread_create() (3/5)

- **`start_routine`**: Η παράμετρος *`start_routine`* είναι η διεύθυνση της ρουτίνας που θα εκτελέσει το νήμα. Είναι δείκτης τύπου `void`.

- **`rg`**: Η παράμετρος *`arg`* είναι η διεύθυνση της δομής των παραμέτρων για τη ρουτίνα *`start_routine`*. Είναι δείκτης τύπου `void`.

❖ Ο μέγιστος αριθμός νημάτων που μπορεί να δημιουργηθεί από μια διεργασία εξαρτάται από την υλοποίηση.
❖ Εφόσον δημιουργηθούν τα νήματα, μπορούν να δημιουργήσουν άλλα νήματα.

Η συνάρτηση pthread_create() (4/5)

- Η ρουτίνα `pthread_create()` επιτρέπει στον προγραμματιστή να περάσει μία παράμετρο στην ρουτίνα εκκίνησης του νήματος. Σε περιπτώσεις όπου πρέπει να περαστούν πολλές παράμετροι, αυτός ο περιορισμός εύκολα καταρρίπτεται δημιουργώντας μια δομή που περιέχει όλες τις παραμέτρους και μετά περνώντας έναν δείκτη σε αυτή τη δομή κατά την ρουτίνα εκκίνησης του νήματος.

- Όλες οι παράμετροι πρέπει να περαστούν κατ' αναφορά με τον τύπο `(void *)`.

Η συνάρτηση pthread_create() (5/5)

- Η συνάρτηση `pthread_create` έχει τις ακόλουθες παραμέτρους :

1. Ταυτότητα (αριθμός) νήματος – τιμή που επιστρέφει η συνάρτηση.
2. Παράμετροι νήματος – attributes, NULL για προκαθορισμένες παραμέτρους.
3. Συνάρτηση εκκίνησης – η συνάρτηση που θα εκτελεστεί μόλις δημιουργηθεί το νέο νήμα.
4. Παράμετροι συνάρτησης – οι μεταβλητές που θα περαστούν στη συνάρτηση μόλις εκτελεστεί.

Η συνάρτηση pthread_exit() (1/4)

➢ Υπάρχουν πολλοί τρόποι με τους οποίους μπορεί ένα Pthread να τερματιστεί :

- Το νήμα επιστρέφει από την αρχική ρουτίνα (η κύρια ρουτίνα για το αρχικό νήμα).
- Το νήμα καλεί την υπορουτίνα `pthread_exit`.
- Το νήμα ακυρώνεται από ένα άλλο νήμα μέσω της ρουτίνας `pthread_cancel`.
- Τερματίζει ολόκληρη η διεργασία εξαιτίας της κλήσης σε μία από τις υπορουτίνες `exec` ή `exit`.

Η συνάρτηση pthread_exit() (2/4)

- Η συνάρτηση `pthread_exit` χρησιμοποιείται για να τερματιστεί ένα νήμα. Τυπικά, η ρουτίνα `pthread_exit()` καλείται αφού ένα νήμα έχει ολοκληρώσει την δουλειά του και δεν χρειάζεται να υπάρχει άλλο.

- Αν η συνάρτηση `main()` τερματίσει πριν τα νήματά της, και τερματίσει με την `pthread_exit()`, τα άλλα νήματα θα συνεχίσουν να εκτελούνται. Διαφορετικά, θα τερματιστούν αυτόματα όταν τερματιστεί και η `main()`.

Η συνάρτηση pthread_exit() (3/4)

- Ο προγραμματιστής μπορεί να προσδιορίσει μια κατάσταση τερματισμού, η οποία αποθηκεύεται σαν ένας κενός δείκτης για οποιοδήποτε νήμα μπορεί να ενωθεί με το νήμα που το καλεί.

- Καθαρισμός: η ρουτίνα `pthread_exit()` δεν κλείνει αρχεία, όποια αρχεία ανοίχτηκαν μέσα στο νήμα θα μείνουν ανοιχτά μέχρι να τερματιστεί το νήμα.

Η συνάρτηση pthread_exit() (4/4)

- Στις ρουτίνες που τερματίζουν φυσιολογικά, μπορεί συχνά να κληθεί η `pthread_exit()` και να τερματιστεί - εκτός κ αν, φυσικά, θέλουν να επιστρέψουν έναν κώδικα. Ωστόσο στη `main()`, υπάρχει ένα σίγουρο πρόβλημα αν η `main()` ολοκληρώσει πριν ολοκληρώσουν τα νήματά της. Αν δεν κληθεί η `pthread_exit()` όταν ολοκληρώσει η `main()`,η διεργασία (και όλα τα νήματα) θα τερματιστούν. Καλώντας την `pthread_exit()` στη `main()`, η διεργασία και όλα τα νήματά της θα κρατηθούν ζωντανά ακόμα κ αν όλος ο κώδικας της `main()` έχει εκτελεστεί.

Η συνάρτηση pthread_attr() (1/2)

- Εξ' ορισμού, ένα νήμα δημιουργείται με συγκεκριμένες ιδιότητες. Ορισμένες από αυτές τις ιδιότητες μπορούν να τροποποιηθούν από τον προγραμματιστή διαμέσου του αντικειμένου ιδιοτήτων του νήματος.

- Η συνάρτηση `pthread_attr_init` και η `pthread_attr_destroy` χρησιμοποιούνται για να αρχικοποιήσουν/καταστρέψουν το αντικείμενο ιδιοτήτων του νήματος.

Η συνάρτηση pthread_attr() (2/2)

- Για να δημιουργηθεί ένα νήμα ικανό για ένωση ή όχι, χρησιμοποιείται η παράμετρος attr στην συνάρτηση `pthread_create()`. Τα 4 τυπικά βήματα είναι:

- Ορισμός μιας μεταβλητής τύπου `pthread_attr_t` για ένα νήμα.
- Αρχικοποίηση της παραμέτρου με την συνάρτηση `pthread_attr_init()`.
- Ορισμός της κατάστασης ένωσης της παραμέτρου με την συνάρτηση `pthread_attr_setdetachstate()`.
- Μετά το τέλος, γίνεται αποδέσμευση των πηγών των βιβλιοθηκών από την παράμετρο με την συνάρτηση `pthread_attr_destroy()`.

Η συνάρτηση pthread_self() (1/3)

- `pthread_t pthread_self(void);`
  Επιστρέφει την ταυτότητα του τρέχοντος νήματος.

- Παράδειγμα:

```
pthread_t pthread_self(void);
int pthread_equal (pthread_t tid1,
pthread_t tid2);
```

```
#include <pthread.h>
pthread_t tid, tid1, tid2;
int res;
tid = pthread_self();
res = pthread_equal(tid1, tid2);
```

Η συνάρτηση pthread_self() (2/3)

Το `pthread_t` είναι 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);
```

➢ Εάν υπάρχουν δυο διεργασίες Α, Β τότε αυτές μπορεί να έχουν νήματα με τα ιδία ThreadIDs.

Η συνάρτηση pthread_self() (3/3)

- Κάθε νήμα χαρακτηρίζεται, όπως και οι διεργασίες από ένα ThreadID.

- Το ThreadID έχει νόημα μόνο στην εμβέλεια μιας διεργασίας.

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

  - Επιστρέφει το threadID του νήματος που κάνει την κλήση.

Η συνάρτηση pthread_join() (1/3)

- 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`**.

Η συνάρτηση pthread_join() (2/3)

❖ 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.

```
int pthread_detach(pthread_t thread);
```

Η συνάρτηση pthread_join() (3/3)

```
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`.

Η συνάρτηση fflush() (1/3)

- Αν έχουμε την εντολή:

```
scanf("%d",&x);
scanf("%d",&y);
```

και ο χρήστης εισάγει 15 20 30, τότε διαβάζεται το 15 και ανατίθεται στη x, αγνοείται το κενό και διαβάζεται το 20 και ανατίθεται στην y. Οι υπόλοιποι χαρακτήρες παραμένουν στην `stdin` για επόμενο διάβασμα. Πρέπει πάντα να περιμένουμε ότι ενδέχεται να έχουν απομείνει χαρακτήρες στην `stdin` μετά από κάποιο διάβασμα. Ειδικά ο χαρακτήρας `\n` απομένει στην είσοδο μετά από διάβασμα με την `scanf`. Για τον λόγο αυτό χρειαζόμαστε τη συνάρτηση `fflush`.

Η συνάρτηση fflush() (2/3)

- Το πρωτότυπο της συνάρτησης `fflush` είναι:

```
int fflush(FILE *fp)
```

και έχει οριστεί στο αρχείο βιβλιοθήκης: `stdio.h`

- Η συνάρτηση «αδειάζει» την `stdin` από τυχόν χαρακτήρες που έχουν απομείνει σε αυτήν.

Η συνάρτηση fflush() (3/3)

❖ Παράδειγμα:

```
 #include <stdio.h>
     main()
     {
int x,y,z; char c; printf("Dwse enan tripsifio
arithmo:");
     scanf("%1d%1d%1d", &x,&y,&z);
printf("Epelekse M-Monades,D-Dekades,E-
Ekatontades: ");
     scanf("%c",&c);
     …
     }
```

Τέλος Ενότητας

Πανεπιστήμιο Δυτικής Μακεδονίας