Πανεπιστήμιο Πελοποννήσου
Τμήμα Επιστήμης και Τεχνολογίας Τηλεπικοινωνιών

Λειτουργικά Συστήματα – Προγραμματισμός Συστήματος

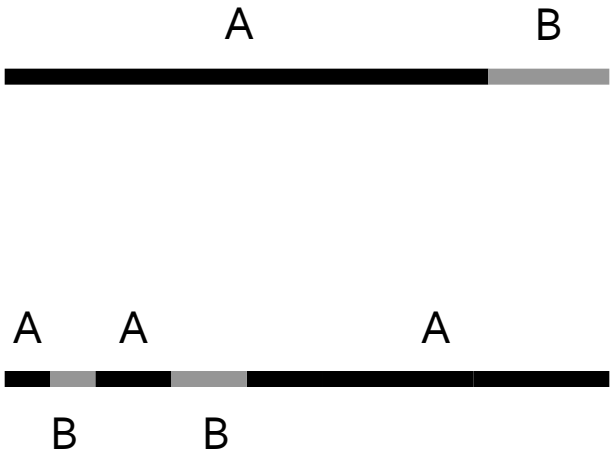
Διεργασίες και Νήματα

Διεργασίες

- **Διεργασία (process)**: ένα πρόγραμμα σε κατάσταση εκτέλεσης
- Το "διά ταύτα":
  - Επικάλυψη λειτουργίας
  - Πολυπρογραμματισμός
  - Ψευδοπαραλληλία
  - Βελτίωση ταχύτητας

Βελτίωση Ταχύτητας?

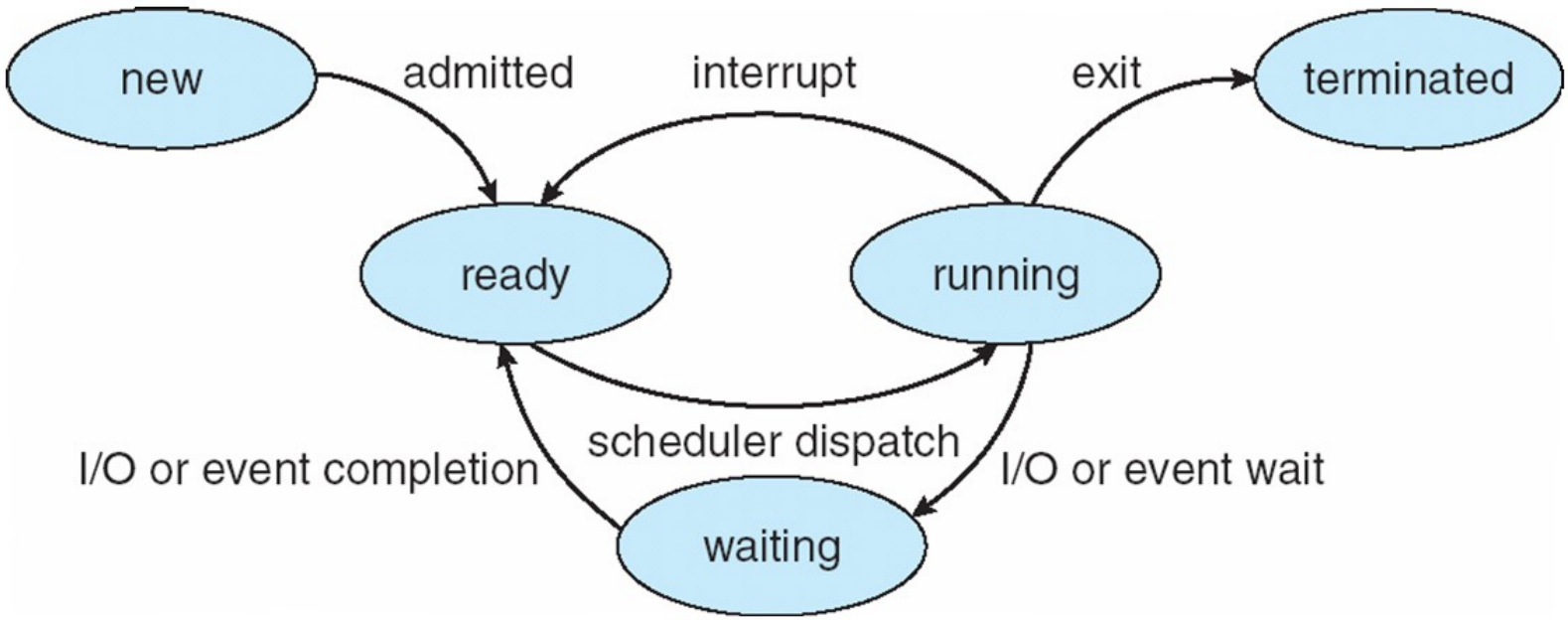
- Δύο διεργασίες:
  - Α: διάρκεια εκτέλεσης 100 sec
  - Β: διάρκεια εκτέλεσης 10 sec
- Σειριακή εκτέλεση:
  - Χρόνος ολοκλήρωσης Α 100 sec
  - Χρόνος ολοκλήρωσης Β 110 sec
  - Μέσος χρόνος: **105 sec**
- (Ψευδο)παράλληλη εκτέλεση:
  - Χρόνος ολοκλήρωσης Α 110 sec
  - Χρόνος ολοκλήρωσης Β 40 sec
  - Μέσος χρόνος: **75 sec**



Καταστάσεις Διεργασίας

- **Νέα (new)**: η διεργασία δημιουργείται
- **Εκτέλεση (running)**: η διεργασία εκτελείται σε κάποιον επεξεργαστή
- **Αναμονή (waiting)**: η διεργασία αναμένει κάποιο συμβάν (απενεργοποιημένη)
- **Ετοιμότητα (ready)**: η διεργασία αναμένει να της δοθεί (από το λειτουργικό σύστημα) χρόνος σε κάποιον επεξεργαστή για τη συνέχιση της εκτέλεσής της
- **Τερματισμός (terminated)**: η διεργασία έχει ολοκληρώσει την εκτέλεσή της

Διάγραμμα Καταστάσεων Διεργασίας

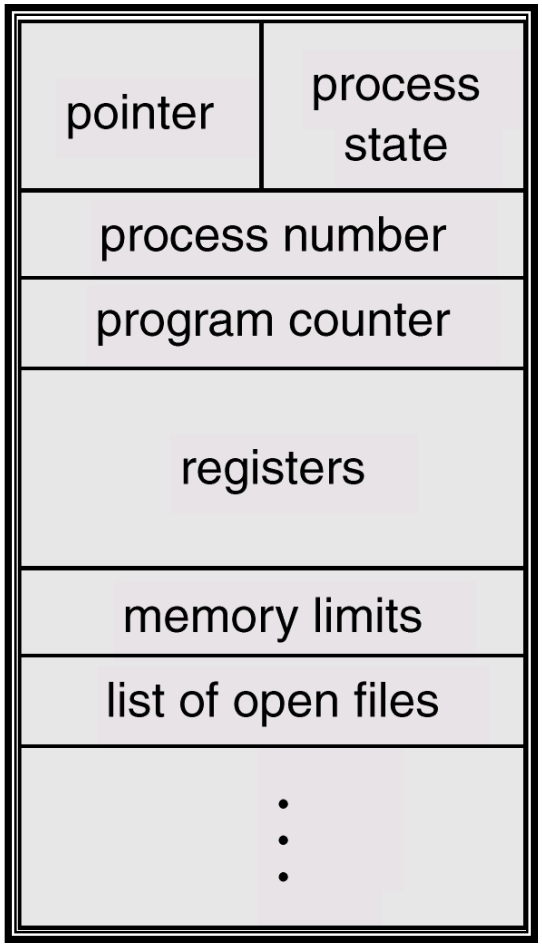


- Σε κάθε επεξεργαστή, μόνο μία διεργασία μπορεί να βρίσκεται σε κατάσταση εκτέλεσης κάθε στιγμή!
- Πολλές όμως σε κατάσταση αναμονής ή ετοιμότητας...

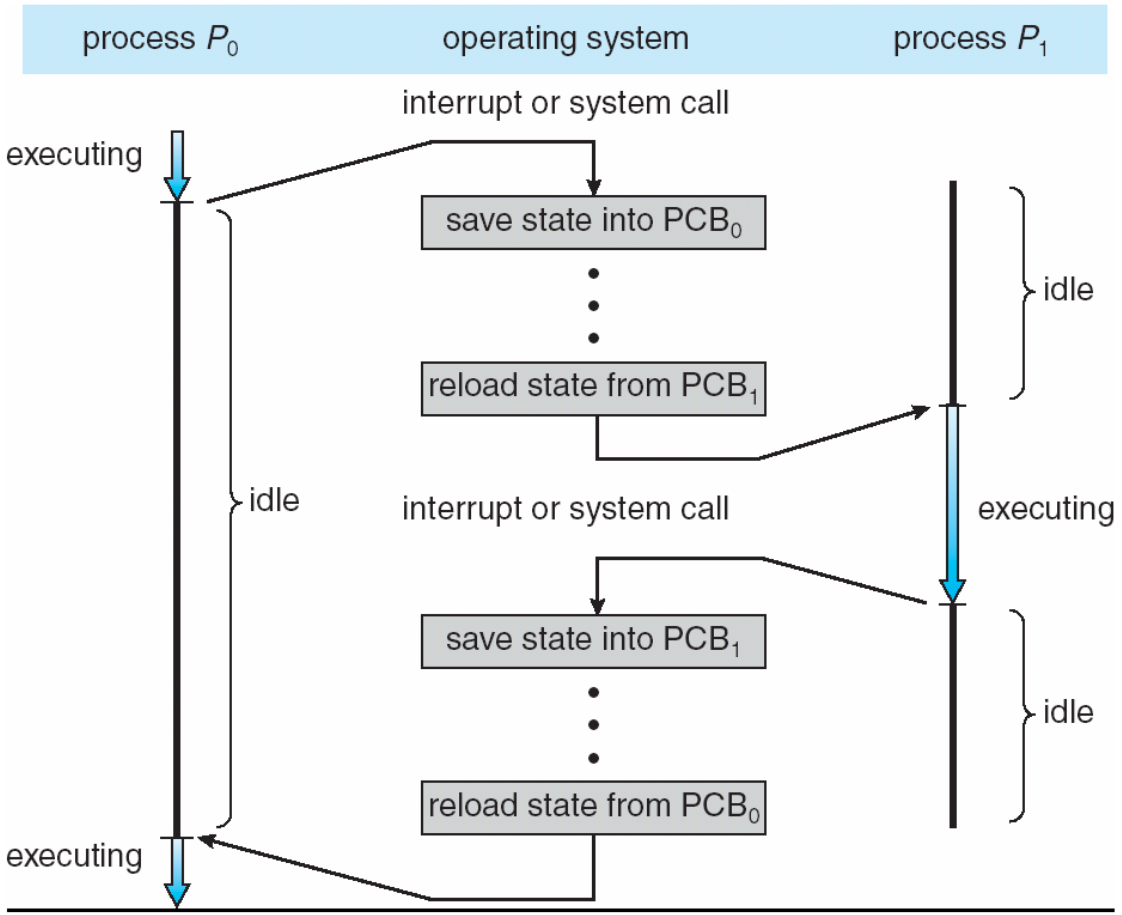
Πίνακας Ελέγχου Διεργασίας Ι

- Ο Πίνακας Ελέγχου Διεργασίας (Process Control Block – PCB) περιέχει βασικές πληροφορίες που συσχετίζονται με την κάθε διεργασία:
  - Κατάσταση διεργασίας (process state)
  - Μετρητής εντολών προγράμματος (program counter)
  - Καταχωρητές της ΚΜΕ (CPU registers)
  - Πληροφορίες για το χρονοπρογραμματισμό της ΚΜΕ (CPU scheduling information)
  - Πληροφορίες διαχείρισης μνήμης (memory-management information)
  - Πληροφορία διαχείρισης (accounting information)
  - Πληροφορίες κατάστασης εισόδου/εξόδου (I/O status information)

Πίνακας Ελέγχου Διεργασίας ΙΙ



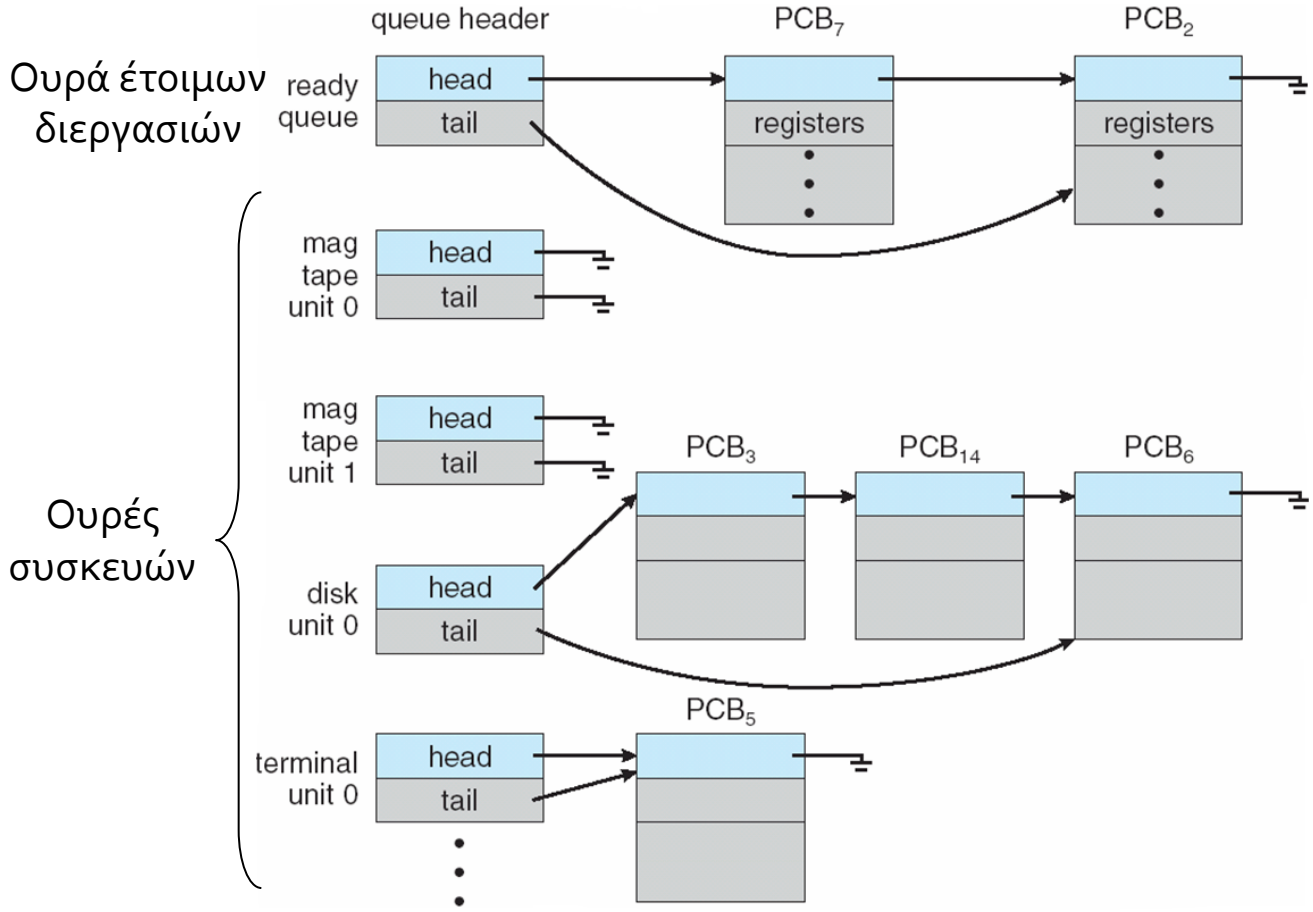
Εναλλαγή CPU Μεταξύ Διεργασιών



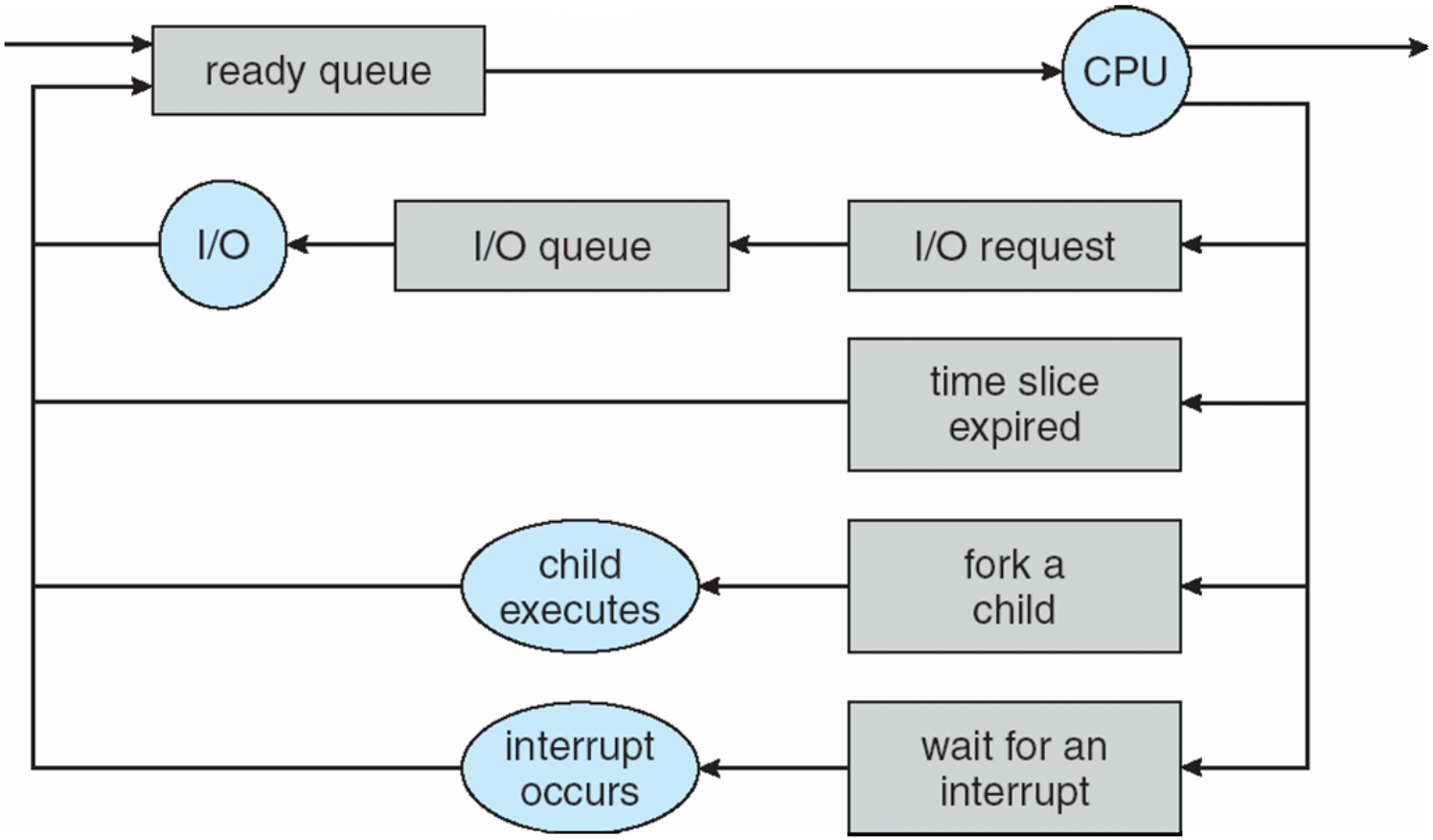
Ουρές Χρονοπρογραμματισμού Διεργασιών

- Ουρά εργασιών (job queue)
- Ουρά έτοιμων διεργασιών (ready queue)
- Ουρές συσκευών (device queue)
- Συνήθης αναπαράσταση: διαγράμματα ουρών (queueing diagrams)

Ουρές Έτοιμων Διεργασιών & Συσκευών



Αναπαράσταση Χρονοπρογραμματισμού (Διάγραμμα Ουρών)



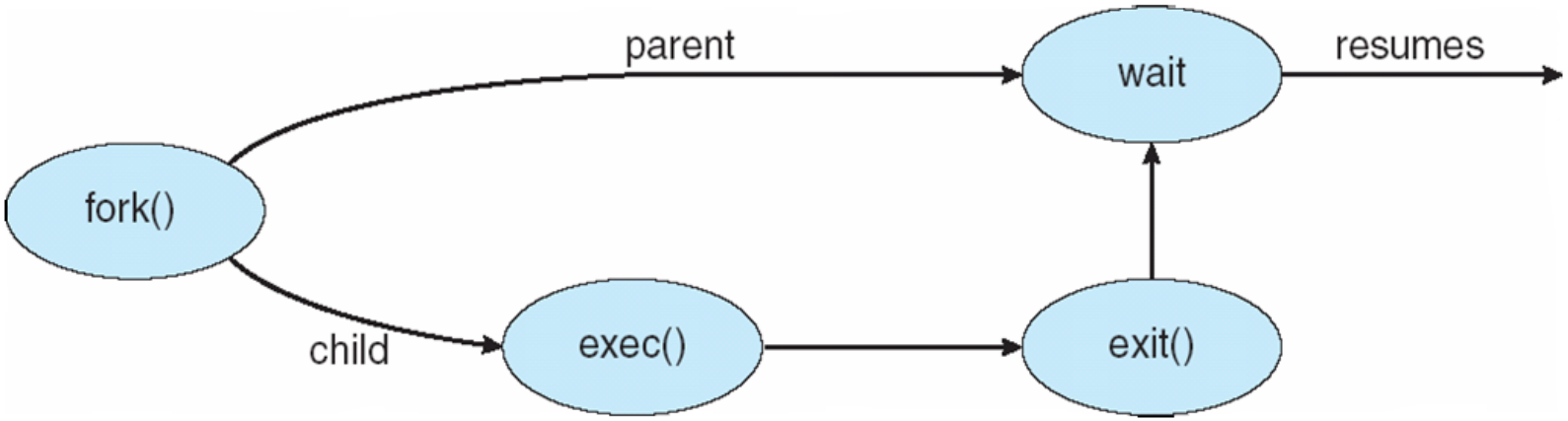
Δημιουργία Διεργασίας Ι

- Γονική διεργασία (parent process) δημιουργεί θυγατρικές διεργασίες (children processes)
- Περιπτώσεις διαμοιρασμού πόρων:
  - Γονική και θυγατρικές διεργασίες μοιράζονται όλους τους πόρους
  - Οι θυγατρικές διεργασίες διαθέτουν μέρος των πόρων της γονικής διεργασίας
  - Δεν υφίσταται διαμοιρασμός πόρων
- Περιπτώσεις εκτέλεσης:
  - Ταυτόχρονη εκτέλεση γονικής και θυγατρικών διεργασιών
  - Η γονική διεργασία μπαίνει σε κατάσταση αναμονής μέχρι την ολοκλήρωση των θυγατρικών διεργασιών

Δημιουργία Διεργασίας ΙΙ

- Στο UNIX:
  - fork() → κλωνοποίηση
  - Αμέσως μετά: exec() → αντικατάσταση χώρου μνήμης με νέο πρόγραμμα
  - Η fork() δεν απαιτεί παραμέτρους
- Στα Windows:
  - CreateProcess()
  - Απαιτεί τουλάχιστον 10 παραμέτρους!

Δημιουργία Διεργασίας ΙΙΙ



Παράδειγμα Δημιουργίας Διεργασίας στο UNIX

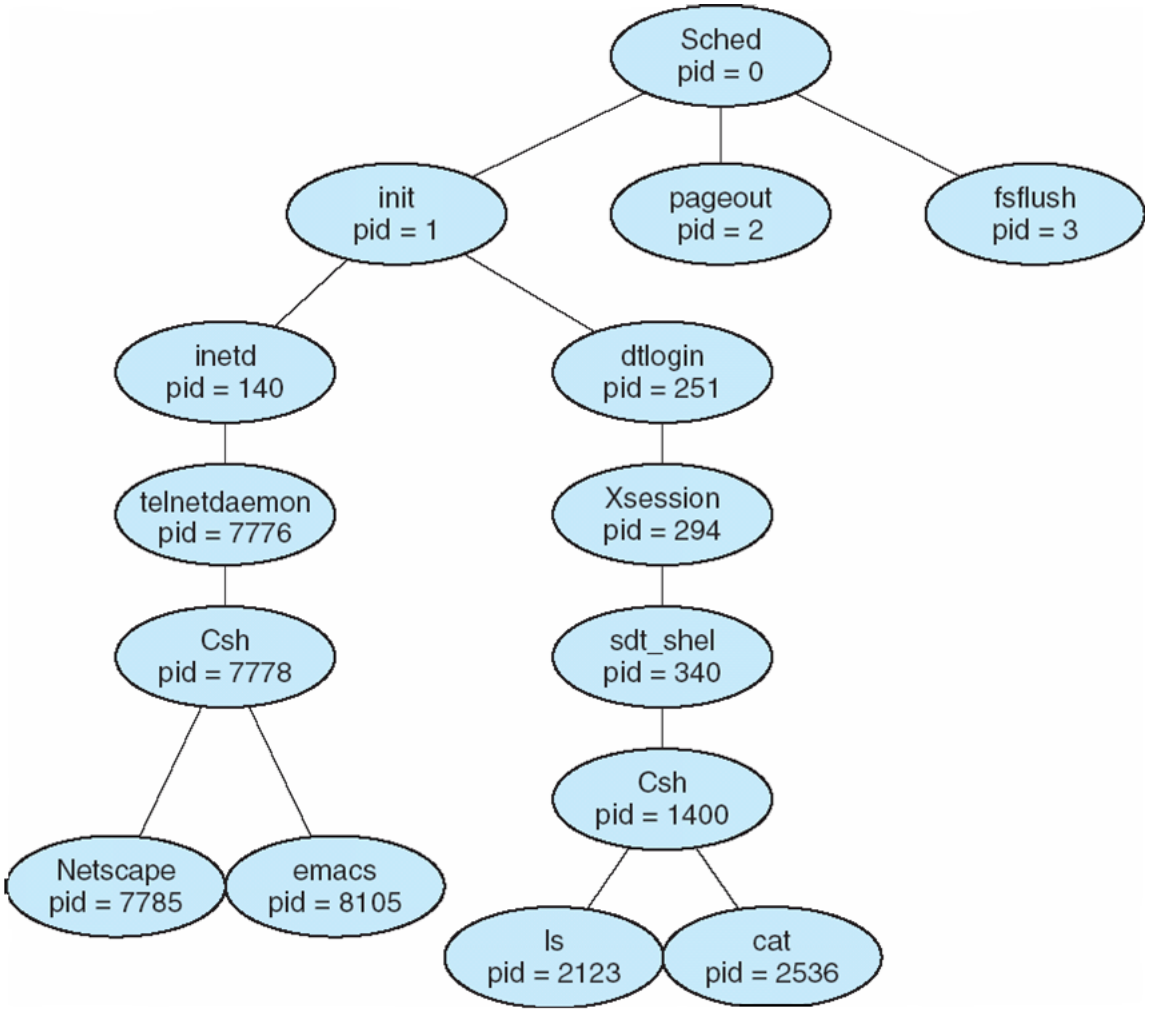
```
int main()
{
     pid_t  pid;

     /* fork another process */
     pid = fork();

     if (pid < 0)
     { /* error occurred */
            fprintf(stderr, "Fork Failed");
            exit(-1);
     }
     else if (pid == 0)
     { /* child process */
            execlp("/bin/ls", "ls", NULL);
     }
     else /* pid > 0 */
     { /* parent process */
            /* parent will wait for the child to complete */
            printf("I created the process with PID = %d.\n", pid);
            wait(NULL);
            printf("Child Complete");
            exit(0);
     }
}
```

Όπως είπαμε, fork() → κλωνοποίηση
Άρα, το if θα εκτελεστεί
ΚΑΙ από τη γονική διεργασία
ΚΑΙ από τη θυγατρική διεργασία!

Η θυγατρική διεργασία "βλέπει" ότι η fork() επέστρεψε 0...

... ενώ η γονική διεργασία λαμβάνει ως αποτέλεσμα της fork() το PID της θυγατρικής διεργασίας

Δέντρο Διεργασιών στο UNIX



Τερματισμός Διεργασίας

- Διαφορετικοί λόγοι:
  - Κανονική έξοδος
    - Η διεργασία ολοκλήρωσε το έργο της και τερματίζει
  - Τερματισμός λόγω λάθους
  - Τερματισμός εξαιτίας "μοιραίου" λάθους
  - Τερματισμός έπειδή κάποια άλλη διεργασία τη "σκότωσε"
    - Εντολή kill στο UNIX
    - Εντολή TerminateProcess() στα Windows

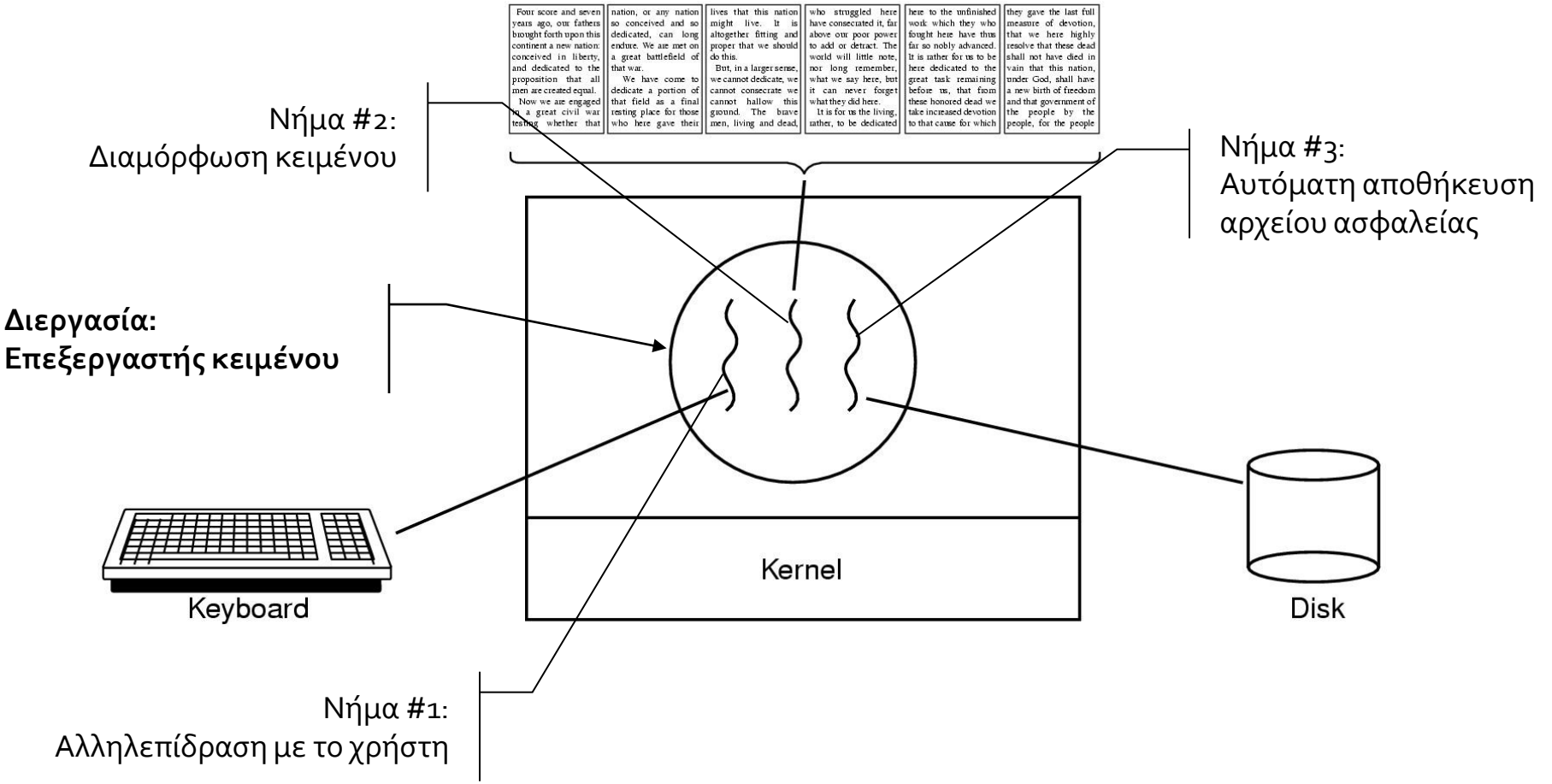
Νήματα

- Εναλλακτικός όρος (ενδεικτικός της λειτουργίας τους…): ελαφρές διεργασίες (lightweight processes – LWPs
- Σε πολλές εφαρμογές υπάρχουν δραστηριότητες οι οποίες εκτελούνται παράλληλα
  - Παράδειγμα?
- Τα νήματα μιας διεργασίας ανήκουν πάντα στον ίδιο χρήστη
- Λειτουργούν (περίπου) όπως οι διεργασίες αλλά στερούνται ανεξαρτησίας:
  - έχουν τα ίδια δικαιώματα πρόσβασης σε πόρους
  - μοιράζονται τους ίδιους πόρους (δεδομένα, κώδικα, ανοικτά αρχεία, σήματα, … ακόμα και το χρόνο της CPU)
  - μοιράζονται την ίδια μνήμη!

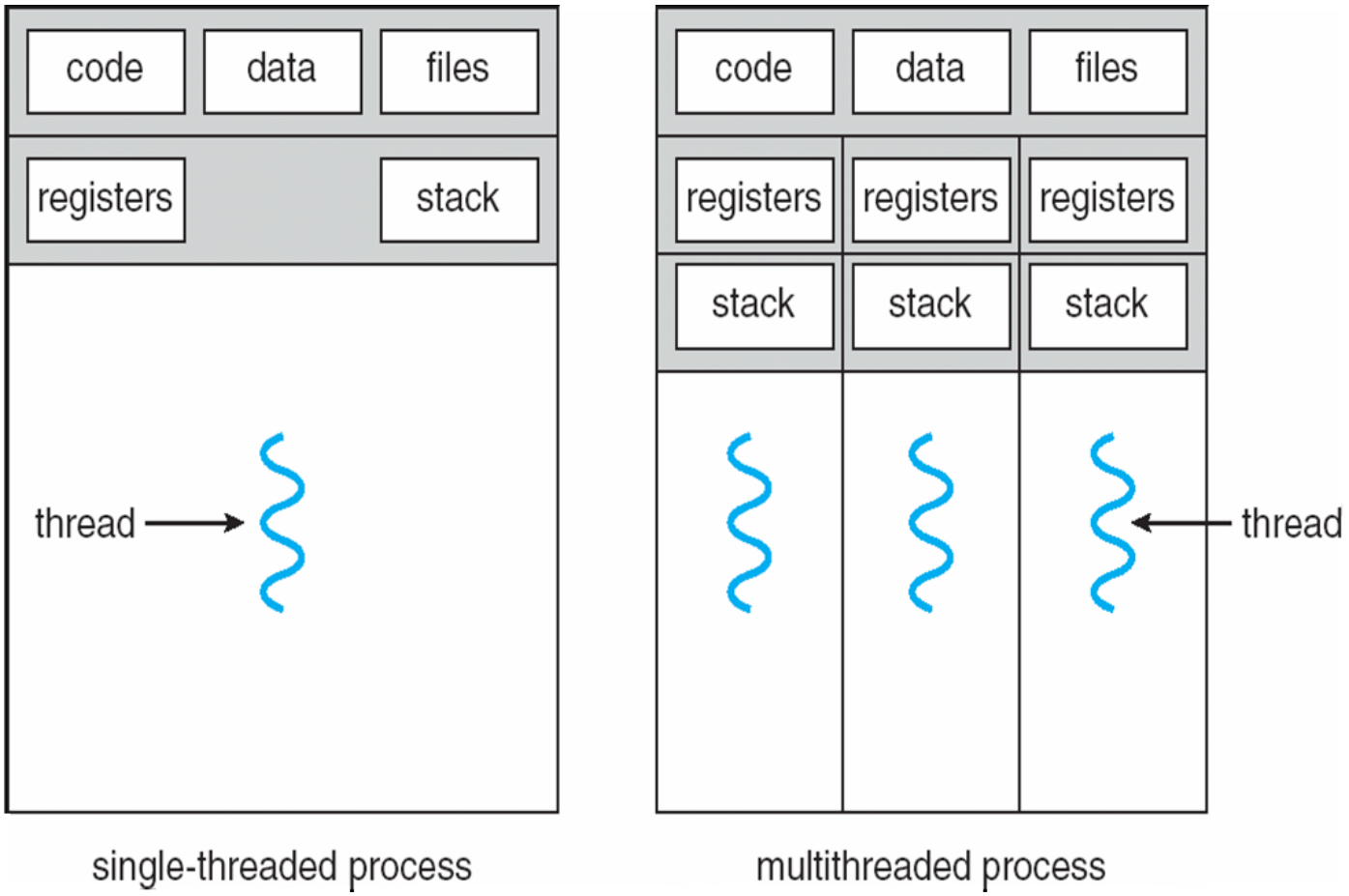
Νήματα

- Πλεονεκτήματα:
  - Ικανότητα απόκρισης
  - Διαμοιρασμός πόρων
  - Οικονομία
  - Κλιμάκωση
- Γιατί νήματα και όχι διεργασίες?
  - Η δημιουργία μίας διεργασίας είναι χρονοβόρα και απαιτεί πόρους

Παράδειγμα


Four score and seven years ago, our fathers brought forth upon this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal.
Now we are engaged in a great civil war testing whether that
nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war.
We have come to dedicate a portion of that field as a final resting place for those who here gave their
lives that this nation might live. It is altogether fitting and proper that we should do this.
But, in a larger sense, we cannot dedicate, we cannot consecrate we cannot hallow this ground. The brave men, living and dead,
who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here.
It is for us the living, rather, to be dedicated
here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which
they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people by the people, for the people
Νήμα #2:
Διαμόρφωση κειμένου
Νήμα #3:
Αυτόματη αποθήκευση
αρχείου ασφαλείας
Διεργασία:
Επεξεργαστής κειμένου
Kernel
Keyboard
Disk
Νήμα #1:
Αλληλεπίδραση με το χρήστη

Μονονηματικές και Πολυνηματικές Διεργασίες



Νήματα στο POSIX
(δηλαδή στο UNIX…)

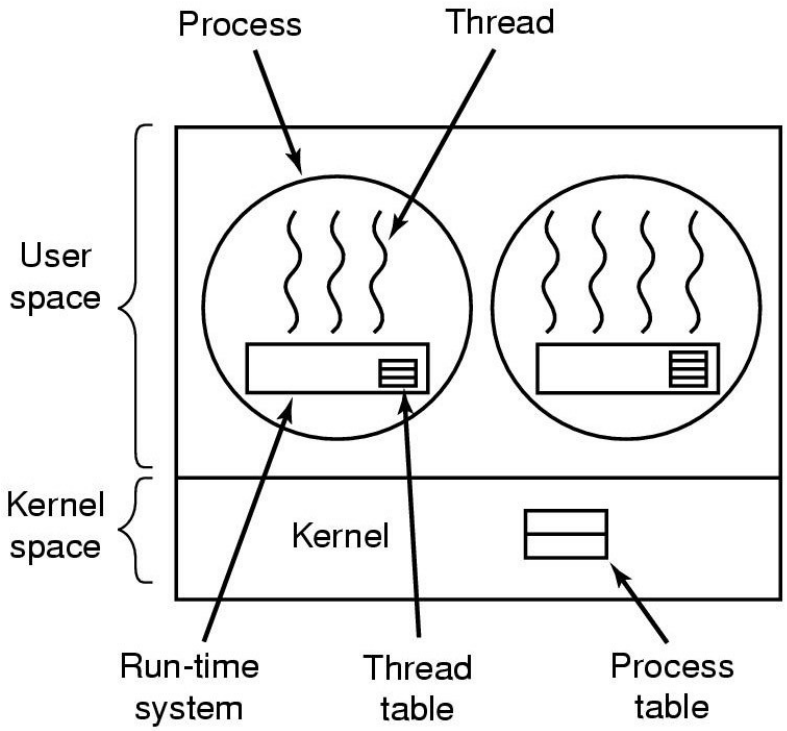
- Ακολουθούν το πρότυπο IEEE 1003.1c (1995)
  το οποίο ορίζει τη βιβλιοθήκη **Pthreads** για τη διαχείριση των νημάτων
- Κάθε νήμα του προτύπου Pthreads χαρακτηρίζεται από:
  - Αριθμητικό αναγνωριστικό (identifier)
  - Καταχωρητές
  - Διάφορα άλλα χαρακτηριστικά
    - μέγεθος στοίβας, παράμετροι χρονοπρογραμματισμού, …
- Βασικές κλήσεις Pthreads:
  - **`pthread_create`**: δημιουργία
  - **`pthread_exit`**: τερματισμός
  - **`pthread_join`**: αναμονή για έξοδο άλλου νήματος
  - **`pthread_yield`**: "ευγενική παραχώρηση" της CPU

Παράδειγμα

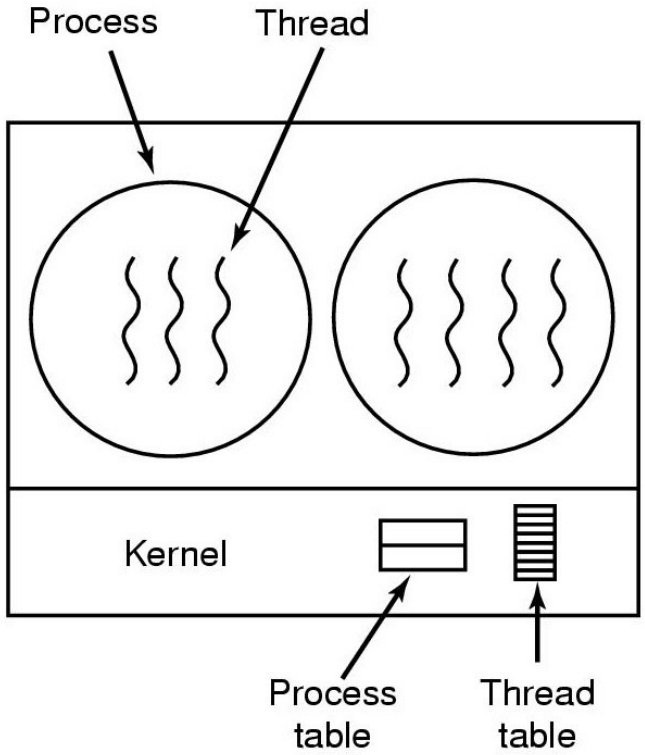
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 5

void *PrintHello(void *tid)
{
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;
    for(t=0;t<NUM_THREADS;t++)
    {
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if(rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Δείκτης προς τη λειτουργία που το νέο νήμα θα εκτελέσει

Νήματα Χρήστη & Νήματα Πυρήνα

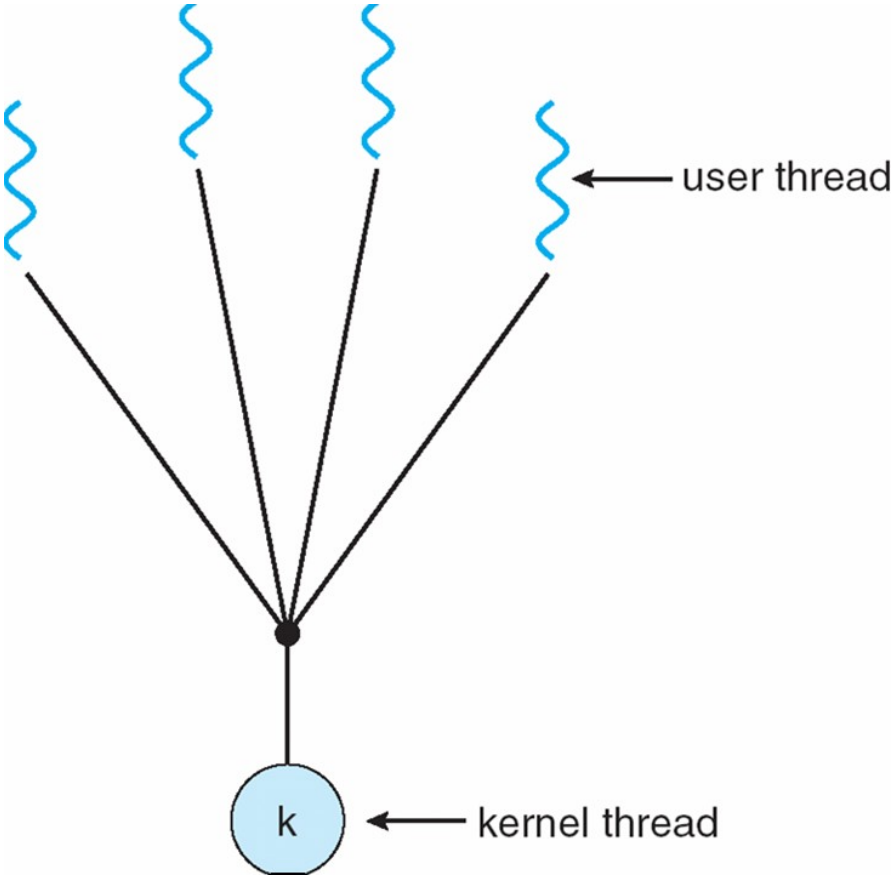


Νήματα στο χώρο του χρήστη



Νήματα πυρήνα

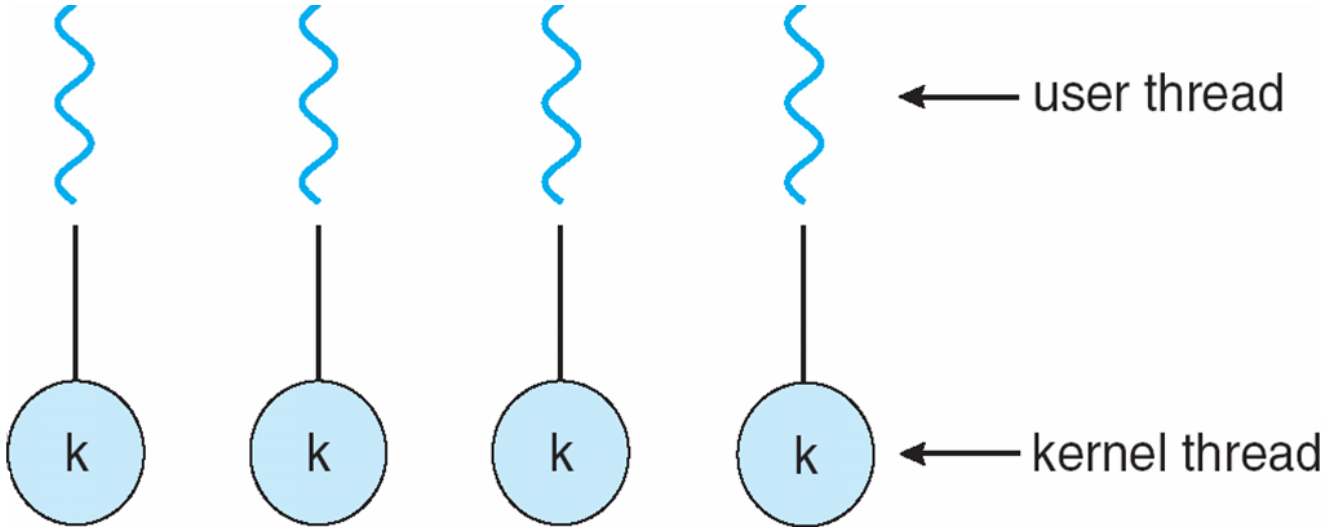
Μοντέλα Πολυνημάτωσης Ι



- Μοντέλο "πολλά-προς-ένα"
- Η βάση είναι τα νήματα χρήστη: πολλά νήματα επιπέδου χρήστη αντιστοιχούν σε ένα νήμα πυρήνα
- Χρησιμοποιείται σε συστήματα που δεν υποστηρίζουν νήματα πυρήνα
- Μεγάλο μειονέκτημα:

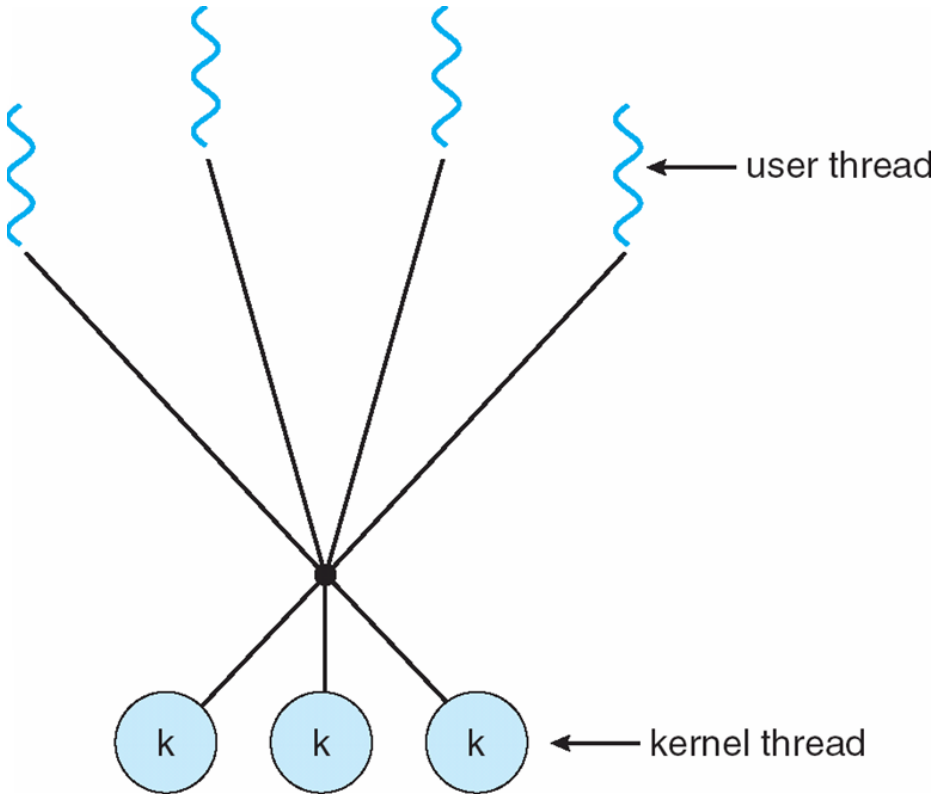
  μία λάθος ενέργεια μπορεί να τα μπλοκάρει όλα!
- Δεν υποστηρίζονται πολλαπλοί επεξεργαστές

Μοντέλα Πολυνημάτωσης II



- Μοντέλο "ένα-προς-ένα"
- Κάθε νήμα χρήστη αντιστοιχεί σε ένα νήμα πυρήνα
- Δυνατότητα παράλληλης εκτέλεσης σε πολλές CPU
- Χρησιμοποιείται σε Linux, Windows

Μοντέλα Πολυνημάτωσης ΙΙΙ



- Μοντέλο "πολλά-προς-πολλά"
- Πολυπλεξία νημάτων
- Επιτρέπει σε πολλά νήματα χρήστη να αντιστοιχιστούν σε πολλά νήματα πυρήνα
- Επιτρέπει στο λειτουργικό σύστημα να δημιουργήσει επαρκή αριθμό νημάτων πυρήνα
- Παραλλαγή:

  μοντέλο "δύο επιπέδων"
  - Παρόμοιο, αλλά: επιτρέπει σε ένα νήμα χρήστη να είναι δεσμευμένο σε κάποιο νήμα πυρήνα
  - π.χ., HPUX

Νήματα & Διεργασίες: Παράδειγμα

- Αριθμοί Fibonacci: η σειρά αριθμών 0, 1, 1, 2, 3, 5,…
- Ο κάθε αριθμός προκύπτει ως το άθροισμα των δύο προηγούμενων της σειράς
- Τυπικά:

  $Fib_0 = 0$

  $Fib_0 = 1$

  $Fib_n = Fib_{n-1} + Fib_{n-2}$
- Το ζητούμενο: πρόγραμμα που λαμβάνει από τη γραμμή εντολών το επιθυμητό πλήθος των αριθμών Fibonacci, τους υπολογίζει και τους παρουσιάζει στην έξοδο
- Επίλυση με διεργασίες & νήματα

Επίλυση με Διεργασίες Ι

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
    if (argc != 2)
          exit(0);

    pid_t pid;
    int i, a, b, fib;

    int n = atoi(argv[1]);

    /* fork another process */
    pid = fork();

    if (pid < 0)
    { /* error occurred */
          fprintf(stderr, "Fork Failed\n");
          exit(-1);
    }
```

Επίλυση με Διεργασίες II

```
    else if (pid == 0)
    { /* child process */
          if (n == 1)
                printf("0\n");
          else if (n == 2)
                printf("0, 1\n");
          else if (n > 2)
          {
                a = 0;
                b = 1;
                printf("0, 1,");
                for (i = 3; i < n; i++)
                {
                      fib = a + b;
                      printf("%d,",fib);
                      a = b;
                      b = fib;
                }
                printf("%d\n",a+b);
          }
    }
    else /* parent process */
    {
          wait(NULL);
          exit(0);
    }
}
```

Η θυγατρική διεργασία υπολογίζει τους αριθμούς.

Η γονική διεργασία είναι ουσιαστικά διακοσμητική! Απλώς περιμένει τη θυγατρική διεργασία να κάνει όλη τη βρωμοδουλειά...

Επίλυση με Νήματα POSIX I

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

#define MAX_SIZE 256

int fibs[MAX_SIZE];

void *runner(void *param) /* the thread */
{
     int i;
     int upper = atoi(param);

     if (upper== 0) pthread_exit(0);
     else if (upper == 1) fibs[0] = 0;
     else if (upper== 2)
     {
            fibs[0] = 0;
            fibs[1] = 1;
     }
     else
     { // sequence > 2
            fibs[0] = 0;
            fibs[1] = 1;

            for (i = 2; i < upper; i++)
                        fibs[i] = fibs[i-1] + fibs[i-2];
     }
     pthread_exit(0);
}
```

Ο κώδικας που θα εκτελέσει το θυγατρικό νήμα μόλις δημιουργηθεί (βλ. επόμενη διαφάνεια)

Επίλυση με Νήματα POSIX II

```
int main(int argc, char *argv[])
{
     int i;
     pthread_t tid; /* the thread identifier */
     pthread_attr_t attr; /* set of attributes for the thread */

     if (argc != 2)
     {
            fprintf(stderr,"usage: a.out <integer value>\n");
            return -1;
     }

     if (atoi(argv[1]) < 0)
     {
            fprintf(stderr,"Argument %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);

     for (i = 0; i < atoi(argv[1]); i++)
            printf("%d\n", fibs[i]);
}
```

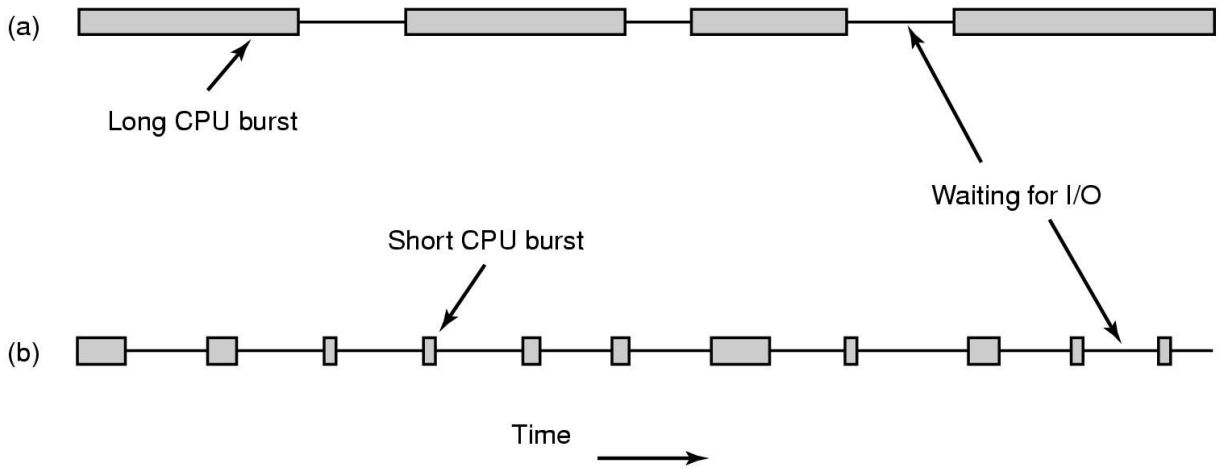
Δημιουργία νήματος

Το γονικό νήμα περιμένει τον υπολογισμό των αριθμών από το θυγατρικό νήμα. Κατόπιν, τυπώνει το αποτέλεσμα.

Χρονοπρογραμματισμός Διεργασιών

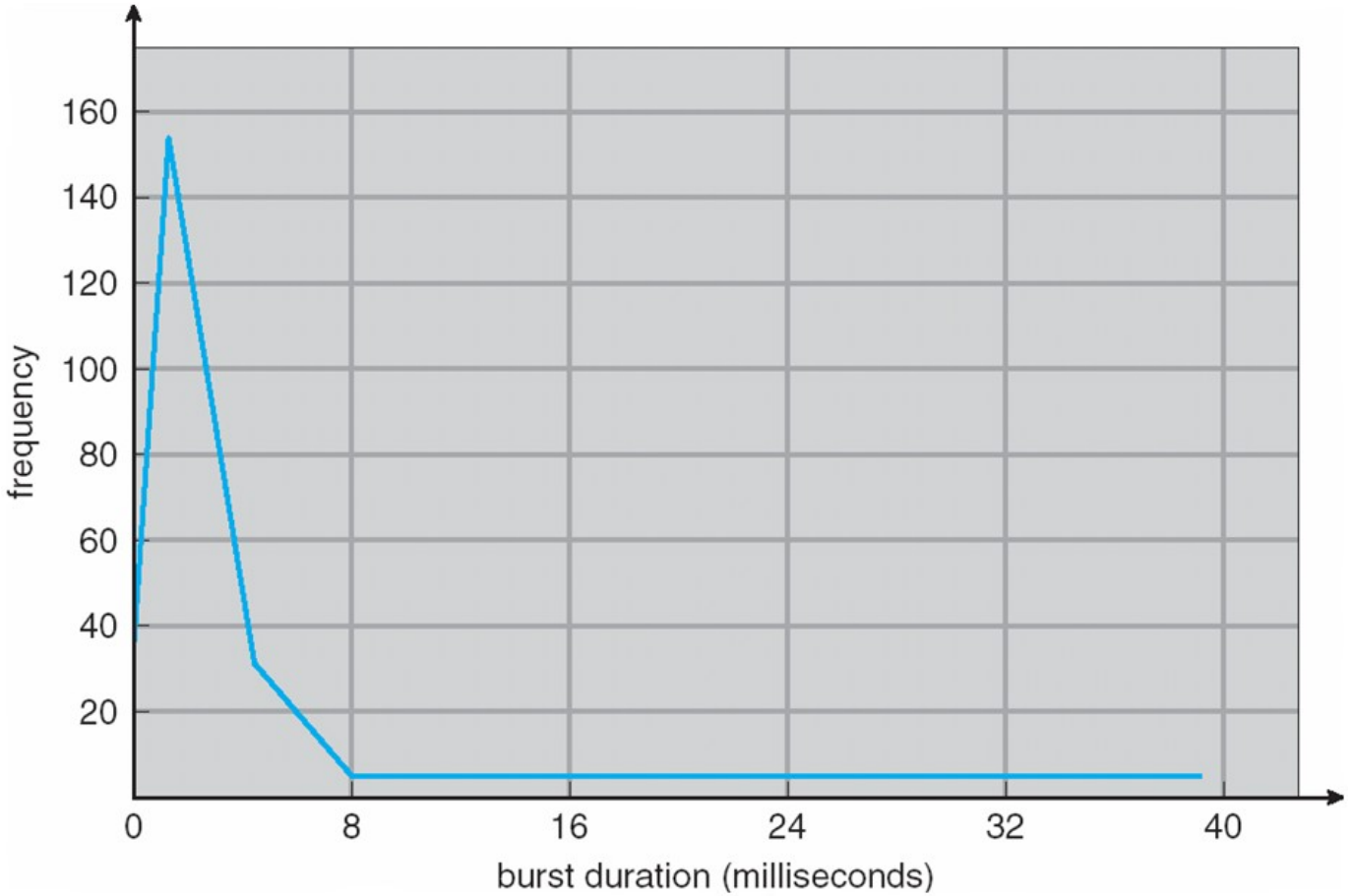
- Οι διεργασίες σε κατάσταση ετοιμότητας ανταγωνίζονται για τον έλεγχο της CPU
- **Χρονοπρογραμματιστής (scheduler)**: το τμήμα του λειτουργικού συστήματος που επιλέγει τη διεργασία που θα εκτελεστεί σε κάποια CPU
- **Αλγόριθμος χρονοπρογραμματισμού**: ο υποκείμενος αλγόριθμος
- Παρόμοια περίπτωση: χρονοπρογραμματισμός νημάτων
- Ενδιαφέρουσα περίπτωση: η ύπαρξη πολλών CPU

Συμπεριφορά Διεργασιών



- Κατηγορίες διεργασιών:
  - Εξαρτημένες από τη CPU (compute-bound)
  - Εξαρτημένες από είσοδο-έξοδο (I/O-bound)

"Ξεσπάσματα" CPU (CPU burst)



Περιπτώσεις Χρονοπρογραμματισμού

- Χρονοπρογραμματισμός CPU λαμβάνει χώρα στις εξής περιπτώσεις:
  - Μετάβαση διεργασίας από κατάσταση εκτέλεσης σε κατάσταση αναμονής
  - Μετάβαση διεργασίας από κατάσταση εκτέλεσης σε κατάσταση ετοιμότητας
  - Μετάβαση διεργασίας από κατάσταση αναμονής σε κατάσταση ετοιμότητας
  - Τερματισμός διεργασίας
- Τύποι αλγορίθμων:
  - Προεκτοπιστικοί αλγόριθμοι (preemptive): εκτέλεση μίας διεργασίας μέχρι κάποιο προκαθορισμένο χρονικό διάστημα
  - Μη προεκτοπιστικοί αλγόριθμοι (non-preemptive): εκτέλεση μίας διεργασίας "όσο θέλει" ή μέχρι να μπλοκαριστεί

Κριτήρια Βελτιστοποίησης

- Μεγιστοποίηση χρόνου χρήσης CPU
- Μεγιστοποίηση ρυθμού διεκπεραίωσης
  - Πλήθος διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου
- Ελαχιστοποίηση χρόνου ολοκλήρωσης
  - Η εγωιστική άποψη μίας διεργασίας…
- Ελαχιστοποίηση χρόνου αναμονής
- Ελαχιστοποίηση χρόνου απόκρισης
  - Πόσο γρήγορα μία διεργασία θα δώσει αποτελέσματα (παρόλο που η εκτέλεσή της συνεχίζεται…)

Χρονοπρογραμματισμός με Βάση τη Σειρά Άφιξης (FCFS)

- First-Come First-Served (FCFS) ή αλλιώς First-In First-Out (FIFO)
- Εκτέλεση διεργασιών ανάλογα με τη σειρά άφιξής τους
- Μη προεκτοπιστική λειτουργία:
  - Μία διεργασία: εκτέλεση μέχρι τερματισμού
  - Περισσότερες διεργασίες: έκτέλεση της καθεμίας μέχρι να ζητήσει Ε/Ε – στη συνέχεια εισαγωγή στο τέλος της ουράς
- Πλεονεκτήματα:
  - Απλότητα
- Μειονεκτήματα:
  - Η απόδοση εξαρτάται από τη σειρά άφιξης – πρόβλημα όταν μία "μεγάλη" διεργασία καταφθάσει νωρίς
  - "Convoy effect"… όπως ένα φορτηγό στην ανηφόρα…

Παράδειγμα FCFS I

- Θεωρούμε τις διεργασίες:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Εάν η σειρά άφιξης είναι P_1, P_2, P_3:

- Χρόνοι αναμονής: P_1 = 0, P_2 = 24, P_3 = 27
- Μέσος χρόνος αναμονής: (0 + 24 + 27)/3 = 17

Παράδειγμα FCFS II

- Εάν για τις ίδιες διεργασίες η σειρά άφιξης είναι P_2, P_3, P_1:

- Χρόνοι αναμονής: $P_1 = 6$, $P_2 = 0$, $P_3 = 3$
- Μέσος χρόνος αναμονής: (6 + 0 + 3)/3 = 3
- Πολύ καλύτερα!
  Αποφυγή του "convoy effect"

Χρονοπρογραμματισμός "Πρώτα η Συντομότερη" (SJF)

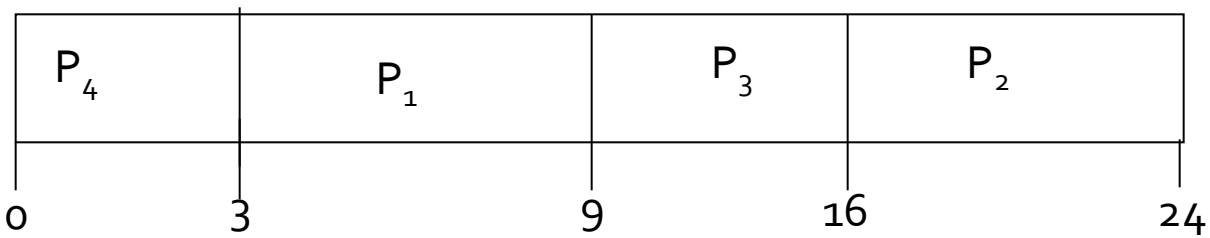
- Shortest Job First (SJF) ή αλλιώς Shortest Time to Completion First (STCF)
- Σύνοψη λειτουργίας:
  - Συσχετισμός κάθε διεργασίας με το μέγεθος του επόμενου ξεσπάσματος CPU
  - Προτεραιότητα στη συντομότερη διεργασία
- Ικανοποιητική λειτουργία!
  - Ελάχιστος μέσος χρόνος αναμονής για δεδομένες διεργασίες
- Δυσκολία: η γνώση της διάρκειας του επόμενου ξεσπάσματος

Παράδειγμα SJF

- Θεωρούμε τις διεργασίες:

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- Χρονοπρογραμματισμός:



- Μέσος χρόνος αναμονής: (3 + 16 + 9 + 0) / 4 = 7