ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΕΡΓΑΣΙΕΣ

Πολυεπεξεργασία

- Διεργασία (process)
- Νήμα (thread)
- Εργασία (task/job)

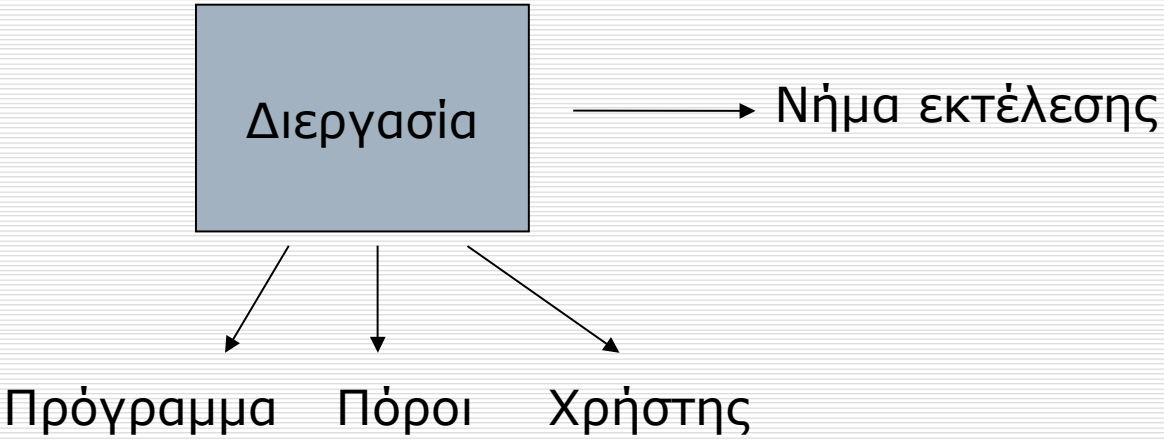
Διεργασίες

- Διεργασία είναι μια (συγκεκριμένη) εκτέλεση κάποιου προγράμματος για λογαριασμό κάποιου χρήστη.

- Σε μια διεργασία αναγνωρίζουμε
  - Το *πρόγραμμα* της διεργασίας
  - Τον χρήστη-*ιδιοκτήτη*
  - Τους *πόρους* της διεργασίας

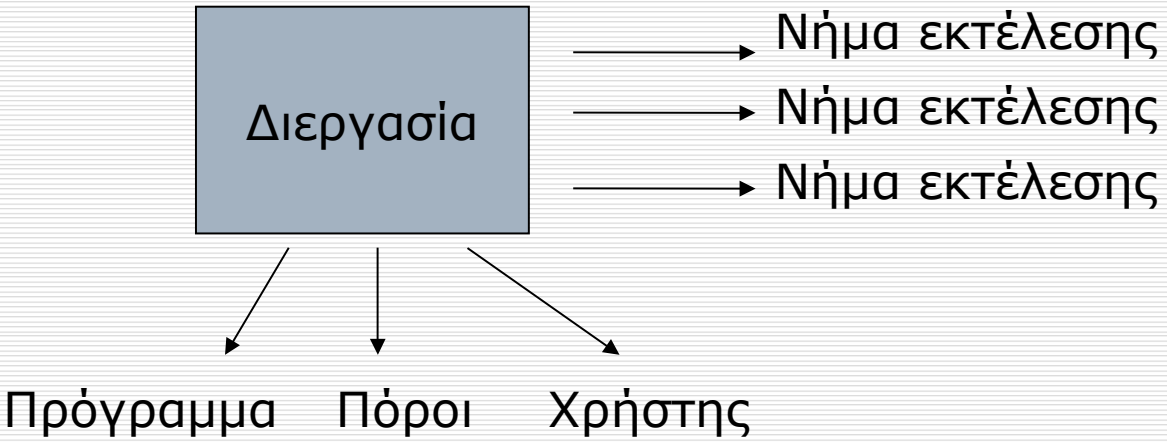
Νήματα

- Βασική έννοια: νήμα εκτέλεσης (thread of execution)
  - Stack
  - Instruction pointer
  - State (registers)

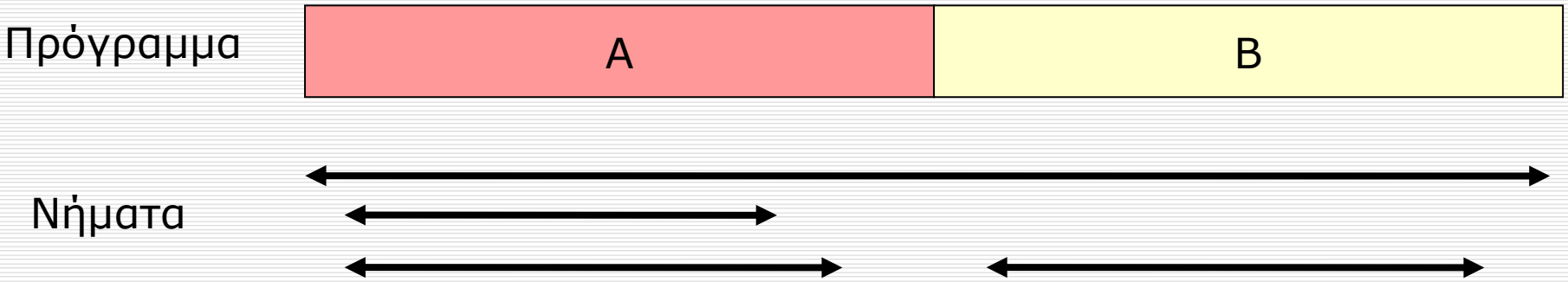


Πολυνηματική επεξεργασία

- Μια διεργασία μπορεί να έχει πολλαπλά νήματα.
- Κάθε νήμα ανήκει σε μία διεργασία.
- Νήμα = πόρος της διεργασίας.



Νήματα και διεργασίες



- Σε μιά διεργασία
  - Ένα πρόγραμμα σε κάθε χρονική στιγμή
  - Ένα ή περισσότερα νήματα

Σημ:

- Το πρόγραμμα μιας διεργασίας μπορεί να αλλάξει.

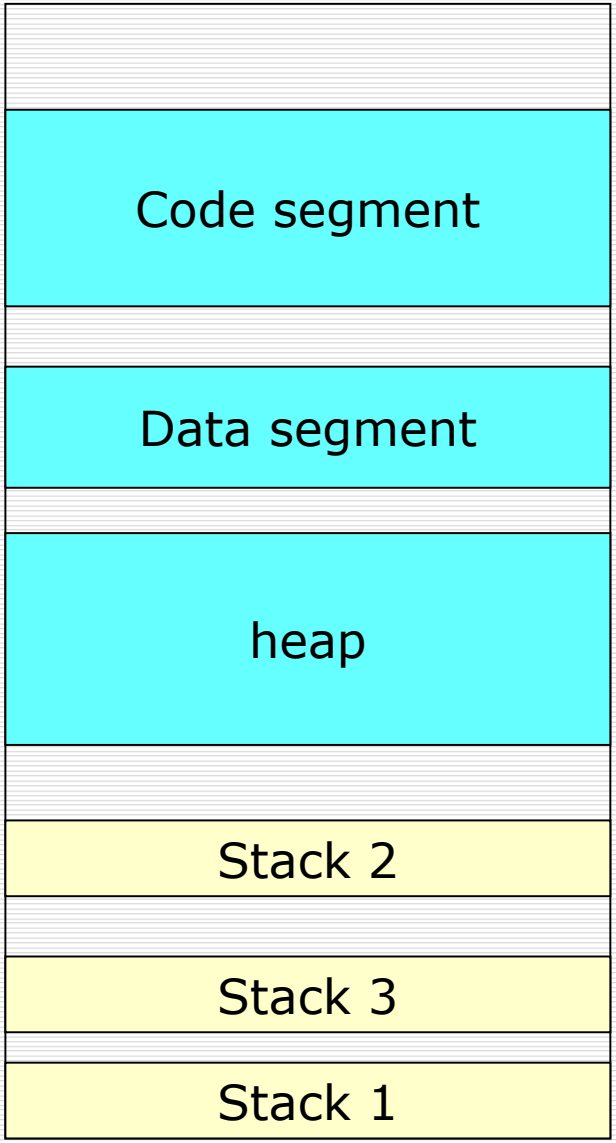
Χρήση νημάτων

- Παραλληλισμός: χρήση πολλαπλών CPU από την ίδια διεργασία
  - Πχ. εκτέλεση ενός μεγάλου αριθμητικού υπολογισμού με πίνακες
- Εκτέλεση του ίδιου κώδικα σε πολλά νήματα ταυτόχρονα
  - Πχ. ένας web server που εξυπηρετεί πολλαπλούς browsers.
- Πολυπλεξία I/O
  - Η διεργασία να μπορεί να επικαλύπτει I/O και υπολογισμούς (πχ. Interactive image processing – photoshop).

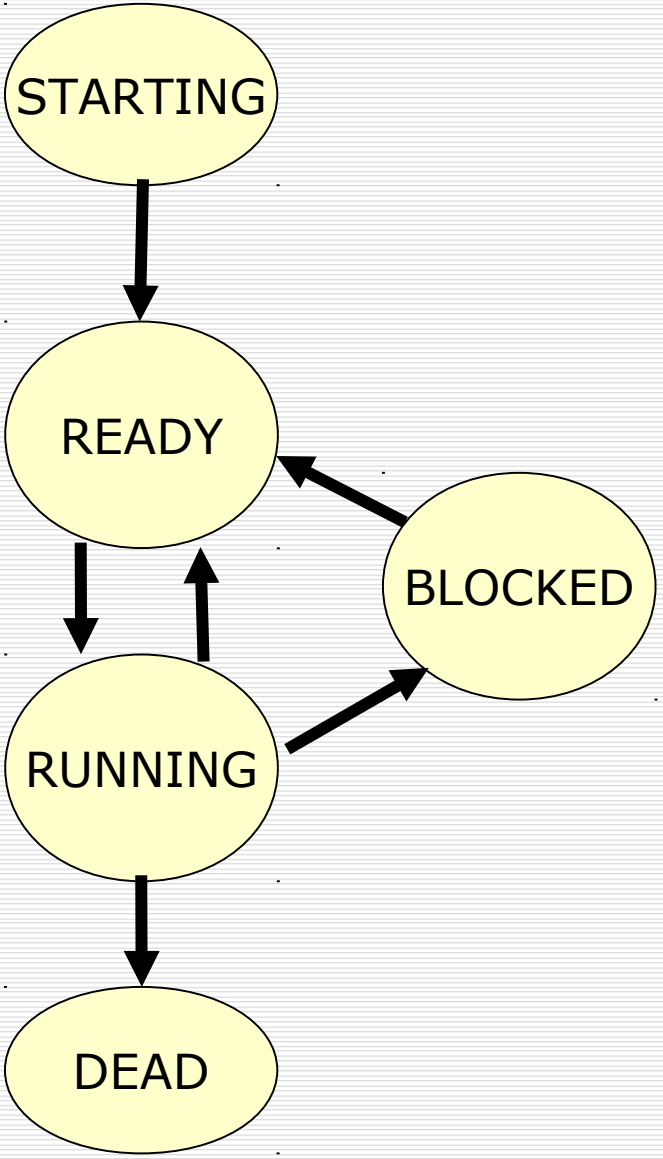
Υλοποίηση νημάτων

- Ένα stack για κάθε νήμα.
- Νήματα επικοινωνούν με κοινές μεταβλητές.

- Context νήματος
  - Καταχωρητές
  - (Το TLB δεν αλλάζει)



Καταστάσεις διεργασίας (μονονηματικής)

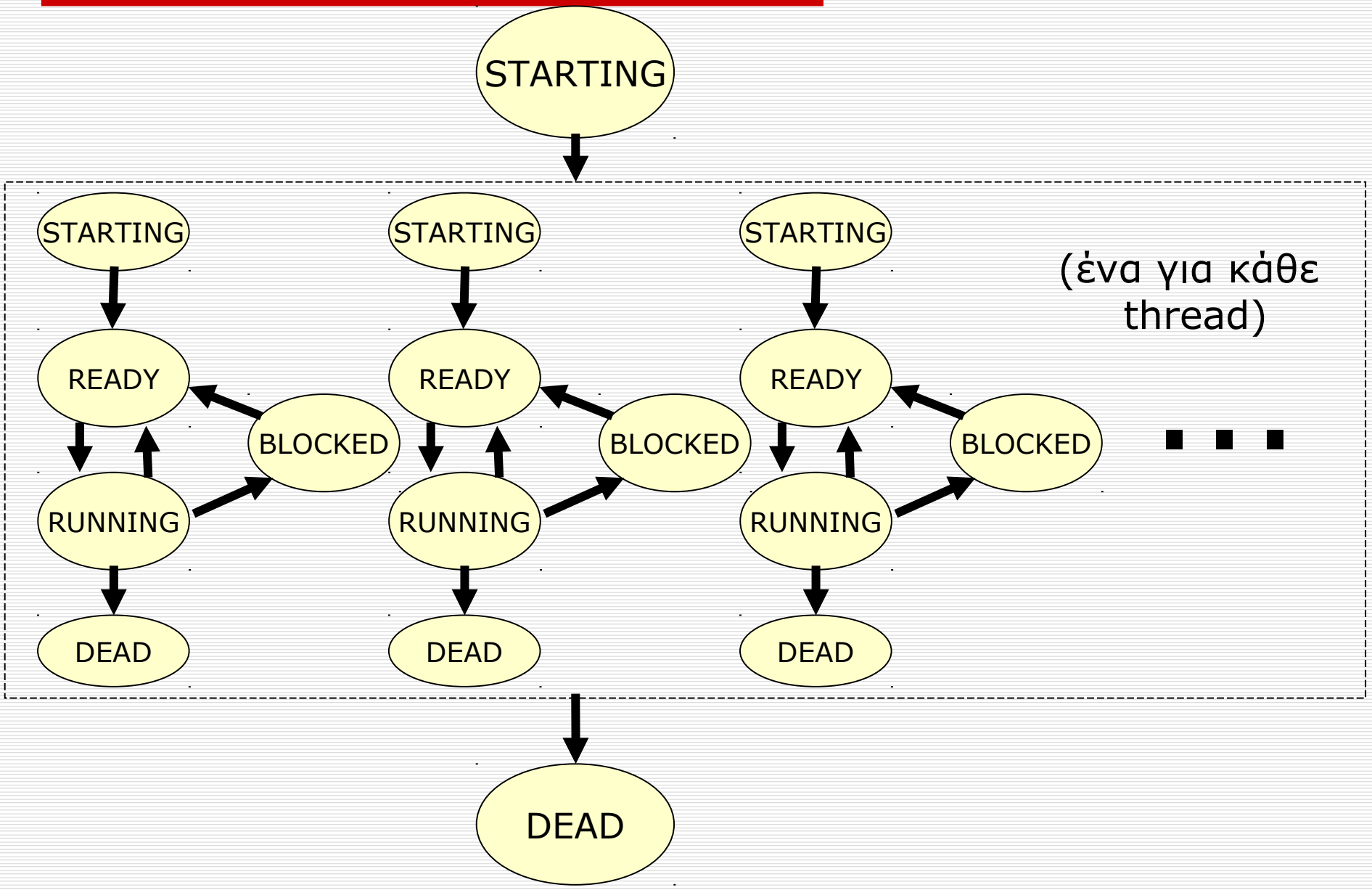


- STARTING: Η διεργασία δημιουργήθηκε αλλά δεν είναι ακόμη έτοιμη για εκτέλεση.
- READY: Είναι έτοιμη για εκτέλεση αλλά δεν εκτελείται.
- RUNNING: Εκτελείται (αυτή τη στιγμή) σε κάποιο CPU.
- BLOCKED: Δεν είναι έτοιμη για εκτέλεση (πχ. περιμένει για I/O)
- DEAD: Έχει τερματιστεί (γιατί είναι ακόμη στη μνήμη?)

Δρομολόγηση νημάτων

- Δύο είδη νημάτων
  - **User-level threads**: υπεύθυνη η ίδια η διεργασία για δρομολόγησή τους
    - Ο πυρήνας δεν γνωρίζει την ύπαρξή τους
    - Συχνά, non-preemptive δρομολόγηση
    - ΔΕΝ εκμεταλλεύονται πολλαπλά CPUs.
  - **System-level threads**: υπεύθυνο το ΛΣ για τη δρομολόγηση τους.
    - Εκμεταλλεύονται πολλαπλά CPUs
    - Scheduler του ΛΣ: κατανέμει CPU(s) στα threads των διεργασιών.
  - Κάθε user-level thread εκτελείται από κάποιο system-level thread.

Καταστάσεις διεργασίας (πολυνηματικής)



Διεργασίες στα WindowsNT

- Αντικείμενα: Εργασίες (jobs), διεργασίες (processes), νήματα (threads), ίνες (fibers).
- Job: Ένα σύνολο από processes που μοιράζονται ρυθμίσεις.
- Process: Συλλογή από πόρους. Περιέχει ¸ 1 threads.
- Thread: (system thread) Δρομολογείται από τον πυρήνα του ΛΣ.
- Fiber: user thread. Δρομολογείται από κάποιο thread.

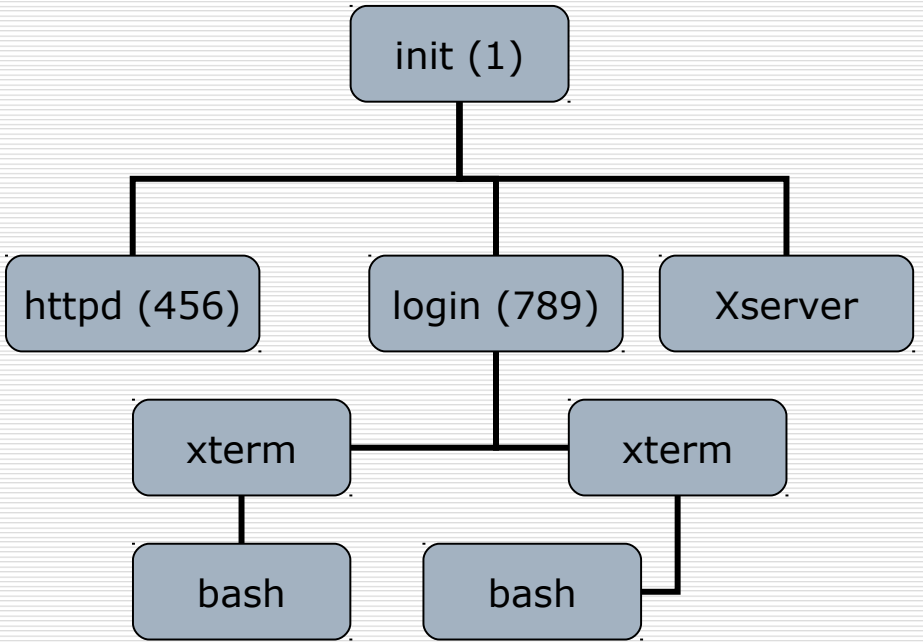
Διεργασίες στο Unix

- Αντικείμενα: διεργασίες (processes), νήματα (threads).
- Διάφορα πρότυπα threads:

Πακέτο	Περιγραφή
Pthreads	POSIX standard. Πλέον portable.
DECthreads	Από τα πρώτα πακέτα. Μέρος του DEC RPC.
Pth	Portable user-level threads.
Solaris LWPs	Ώριμη υλοποίηση, hybrid system-user threads με καλή υποστήριξη SMP.

Ιεραρχία διεργασιών

- Κάθε διεργασία χαρακτηρίζεται από έναν ακέραιο, το PID (Process ID – Αριθμός Ταυτότητας Διεργασίας).
  - Δεν αλλάζει ποτέ
- Κάθε διεργασία έχει μια άλλη διεργασία ως γονέα (parent). Αρα,
  - Μπορεί να αλλάξει (πότε?)
- Οι διεργασίες αποτελούν δέντρο.
- Ρίζα: διεργασία **init** με pid=1.



System calls για διεργασίες

Κλήση	Περιγραφή
`pid_t fork()`	Δημιουργεί μια νέα διεργασία, που είναι αντίγραφο της καλούσας διεργασίας.
`int execve(prog,argv,envp)`	Αλλάζει το εκτελέσιμο πρόγραμμα της τρέχουσας διεργασίας.
`pid_t wait(status)` `pid_t waitpid(pid,status,opts)`	Ελέγχει και (πιθανώς) περιμένει μέχρι μια διεργασία να τερματίσει.
`void _exit(status)`	Τερματίζει την τρέχουσα διεργασία.
`pid_t getpid()`	Επιστρέφει PID τρέχουσας διεργασίας.
`pid_t getppid()`	Επιστρέφει γονέα τρέχουσας διεργασίας.

Δημιουργία διεργασίας

- Γίνεται ΜΟΝΟΝ από τη fork.
- Η fork καλείται μια φορά και επιστρέφει δυο φορές:
  - Στη διεργασία που την κάλεσε (γονέας της νέας διεργασίας) επιστρέφει το PID της νέας διεργασίας-παιδιού.
  - Στη νέα διεργασία επιστρέφει (pid_t)0.
- Η νέα διεργασία κληρονομεί:
  - Αντίγραφο της μνήμης (κώδικα + δεδομένα).
  - Όλους τους ανοιχτούς file descriptors (αρχεία, sockets, pipes κλπ).
  - Ιδιοκτήτη και άδειες εκτέλεσης, περιβάλλον (environment), όρια πόρων (resource limits), current directory, τερματικό ελέγχου ...
- ΔΕΝ κληρονομεί
  - File locks
  - Pending signals (?).

Εκτέλεση προγράμματος

- ΜΟΝΟΝ από την execve.

```
execve(char* filename,char* argv[], char* envp[])
```

- Αντικαθιστά το τρέχον πρόγραμμα με ένα νέο
  - Φορτώνεται από το δίσκο.
  - Ξεκινά εκτέλεση από τη συνάρτηση main του προγράμματος, όπου περνά παραμέτρους argv.
  - Περιβάλλον το envp.
  - ΔΕΝ ΑΛΛΑΖΕΙ ΔΙΕΡΓΑΣΙΑ (pid, ppid, χρήστης κλπ).
- Το νέο πρόγραμμα κληρονομεί:
  - Τα ανοιχτά file descriptors (ίσως... close_on_exex).
  - Τερματικό ελέγχου, τρέχον directory, resource limits, file locks, …

Τερματισμός διεργασίας

- Κανονικά: με την _exit

```
void exit(int status)
```

- Έκτακτα: σφάλμα εκτέλεσης. Με signals (δες παρακάτω...)
- Η exit δεν επιστρέφει ποτέ.
- Status: η τερματισθείσα διεργασία "επιστρέφει" μια *τιμή εξόδου* τύπου **int** που μπορεί να διαβαστεί από το γονέα της.
  - Σύμβαση: status == 0 σημαίνει κανονική εκτέλεση, !=0 σημαίνει πρόβλημα. Αλλά το ΛΣ αδιαφορεί.

Ανάγνωση τιμής εξόδου

- Με χρήση (από τον γονέα) της waitpid:

```
pid_t waitpid(pid_t pid, int* status, int options)
```

- Εμπλέκεται (is blocked) μέχρι να μπορεί να επιστρέψει.
- pid:
  - -1: επέστρεψε για οποιοδήποτε τερματισμένο παιδί
  - >0: επέστρεψε για το συγκεκριμένο παιδί-pid
  - <-1 ή 0: επέστρεψε για διεργασία σε συγκεκριμένο process group (??)
- status: αποθηκεύεται εκεί η τιμή εξόδου (αν !=0)
- options: 0 ή WNOHANG (οπότε δεν εμπλέκεται).

Έλεγχος διεργασιών - signals

- Signals = Interrupts για διεργασίες
- Ασύγχρονη εκτέλεση.
- Υλοποίηση μέσω stack.
- Πηγές:
  - Πυρήνας
  - Άλλες διεργασίες

Έννοια	**Αντιστοιχία**
Interrupt	Signal
Interrupt handler	Signal handler
Interrupt vector	Sigaction(…)
Interrupt mask	Signal mask

Κυριότερα signals (POSIX)

signal	Περιγραφή
SIGHUP	Διακοπή σύνδεσης με το τερματικό ελέγχου
SIGINT	Πλήκτρο διακοπής (<Ctrl>-C) από πληκτρολόγιο
SIGCHLD	Τερματισμός διεργασίας-παιδιού
SIGILL	Illegal instruction
SIGABRT	Κλήση abort()
SIGFPE	Floating Point Exception (πχ διαίρεση με 0)
SIGKILL	Άμεσος τερματισμός διεργασίας
SIGSEGV	Παράνομη πρόσβαση στη μνήμη
SIGALRM	Ξυπνητήρι (συνεργάζεται με την κλήση alarm(...))
SIGTERM	Σήμα τερματισμού διεργασίας
SIGSTOP	Σήμα σταματήματος διεργασίας
SIGCONT	Σήμα συνέχισης διεργασίας

Ρουτίνες χειρισμού signals (POSIX)

`struct sigaction` ▫ `handler` ▫ `mask` ▫ `flags`	**Handler** - pointer to function, SIG_DFL, SIG_IGN. **Mask** – signals masked when handler called **Flags** – ONESHOT, NOMASK, RESTART, ONSTACK, …
`sigaction(int num, newaction, oldaction)`	Θέτει το νέο sigaction για το δεδομένο signal
`sigprocmask(how, set, oldset)`	Block και unblock διάφορα signals
`sigpending(set)`	Εξετάζει ποια signals (που μπλοκάρονται) είναι σε αναμονή (pending)
`kill(pid, sig)`	Στέλνει το sig στο process με το pid.
`sigsuspend(maskset)`	Εμπλέκεται και περιμένει για κάποια συγκεκριμένα signals.

Παράδειγμα: Τερματισμός παιδιού

- Διεργασία Α ο γονέας, διεργασία Β το παιδί.
- Τερματισμός Β -> signal SIGCHLD στην Α

```
void sigchld_handler(int sig) {
   pid_t pid;
   int status;
   pid=wait(&status);
   printf("Child %d exit, status
   %d\n",pid,status);
}
...
struct sigaction ca;
ca.handler = sigchld_handler;
sigaction(SIGCHLD, &ca, NULL);
```

Άλλες περιπτώσεις

- Αν ο γονέας τερματίσει πρίν το παιδί καλέσει την exit:
  - Νέος γονέας η init (pid == 1)
  - Η init θα καλέσει τη wait
- Αν το παιδί τερματίσει πρίν ο γονέας καλέσει τη waitpid:
  - Το παιδί είναι στην κατάσταση DEAD.
  - Διατηρείται στον πίνακα διεργασιών για να διαβαστεί το exit status.
  - zombie

Είδη διεργασιών

- Χρήστη
- Δαίμονες (daemons)
  - Non-interactive διεργασίες (χωρίς τερματικό ελέγχου).
  - Προσφέρουν υπηρεσίες (services), πχ. ο web server, print server, κλπ.
- Πυρήνα (kernel processes)
  - Βρίσκονται συνέχεια εντός του πυρήνα
    - Πώς; πχ. εκτελούν μόνιμα κάποιο ειδικά φτιαγμένο system call
  - Εκτελούν βασικές λειτουργίες διαχείρισης
    - Init, idle (swapper), κλπ.
    - Βελτιστοποίηση χρήσης μνήμης
    - Υλοποίηση NFS (Network File System)
    - Διαχείριση ενέργειας (power management, πχ. σε laptops)
    - ...

Βασική υλοποίηση διεργασιών

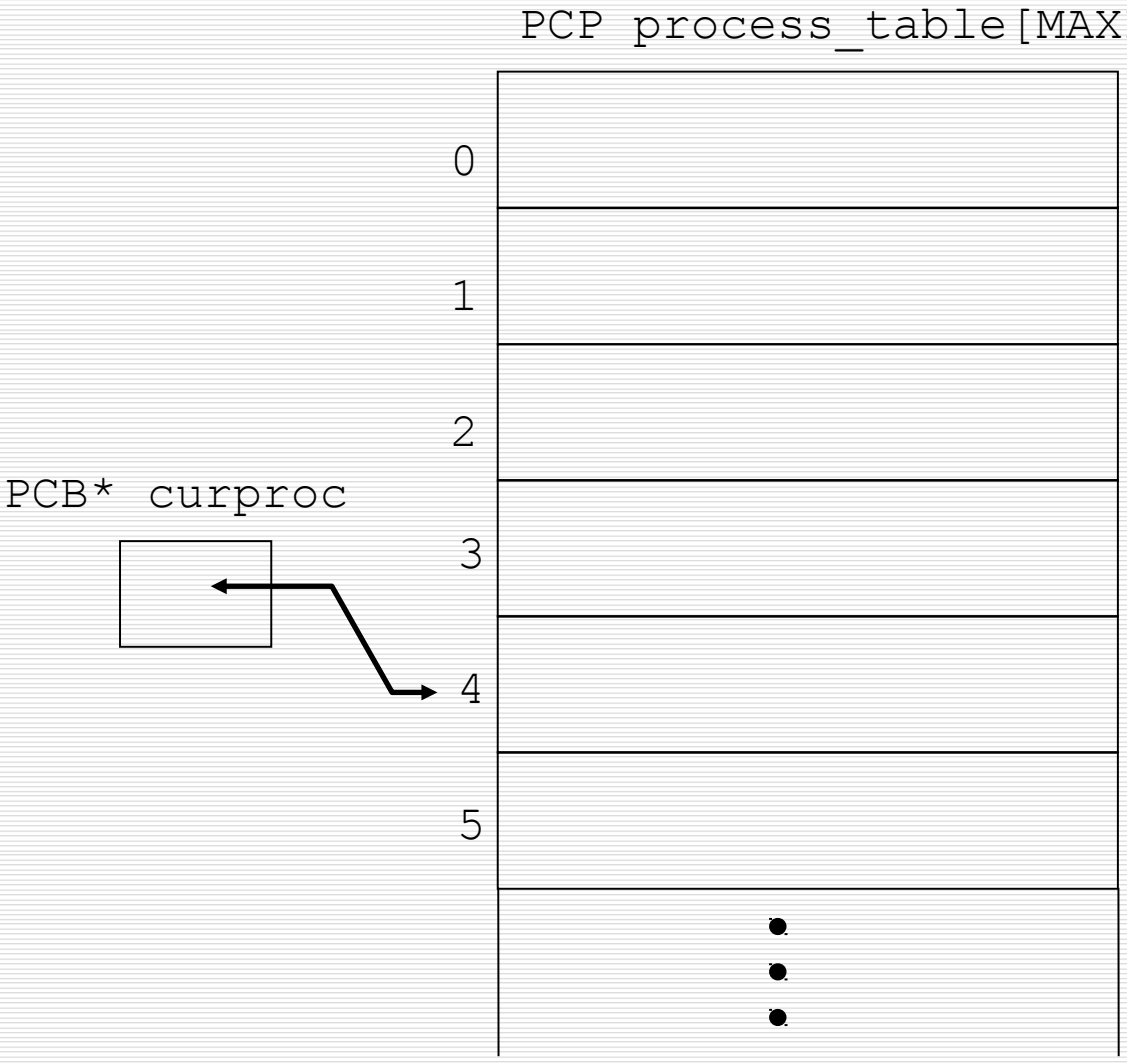
- Process table (πίνακας διεργασιών)
  - Ένα array από PCB (Process Control Block).

```
struct PCB {
    Pid_t pid;
    …
};

#define MAXPROC 4096

PCB process_table[MAXPROC];
```

Πίνακας διεργασιών

- Παράμετροι διεργασίας
  - Χρήστης (ομάδες χρηστών).
  - Process ID, parent, job id, …
  - Χρόνος εκτέλεσης, κύκλοι CPU που χρησιμοποιήθηκαν κλπ. όρια εκτέλεσης
  - Signal handlers, mask
- Παράμετροι ανα thread
  - Context
  - Thread state
  - Προτεραιότητα και άλλες παράμετροι δρομολόγησης
- Παράμετροι μνήμης και Ε/Ε
  - Διευθύνσεις text, data, stack segments.
  - Ιδεατή μνήμη
  - Root directory, current directory
  - File descriptors

Scheduler - curproc




```
/* Σχηματική υλοποίηση
   scheduler */

void schedule() {

    PCB* oldproc;


    oldproc = curproc;

    curproc = select_next();


    if(curproc != oldproc)

        swap(curproc->context,

             oldproc->context);

}
```

System call execution

Παράδειγμα: getpid(), getppid(),

```
struct PCB {
  pid_t  pid, parentid;
  …
};

…

pid_t getpid() {
    return curproc->pid;
}
pid_t getppid() {
    return curproc->parentid;
}
```

Υλοποίηση signals

- Δύο ειδών: σύγχρονα και ασύγχρονα.
- **Σύγχρονα signals**: τα λαμβάνει η διεργασία όταν δεν μπορεί να προχωρήσει. πχ. SIGSEGV, SIGILL, SIGKILL κλπ
  - Έρχονται από το kernel, οφείλονται σε interrupt
- **Ασύγχρονα signals**:
  - Κατά την αποστολή στη διεργασία Α:
    - Το signal καταγράφεται ως pending (*επικείμενο*) στο PCB της διεργασίας Α (που το λαμβάνει).
- Κατά την επόμενη δρομολόγηση της διεργασίας Α (ασύγχρονα signals) ή άμεσα (σύγχρονα signals) :
  - Ελέγχεται η μάσκα, αν δεν επιτρέπει εκτέλεση, συνεχίζει η κανονική ροή της διεργασίας.
  - Δημιουργείται στο stack της διεργασίας το κατάλληλο stack frame και καλείται ο signal handler.
  - Στην περίπτωση πολλών νημάτων, χρησιμοποιείται κάποιο από αυτά (η επιλογή του είναι περίπλοκη).

Προγραμματισμός threads

- POSIX Threads (pthreads)
  - POSIX: Portable Operating System Interface
  - http://www.opengroup.org
  - Unix spec: 1003.1 (το λεγόμενο POSIX.1 standard)
    - http://www.unix.org/version3/

- Τα pthreads είναι ένα API για τη γλώσσα C.
- Όλες οι δηλώσεις:

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

PThreads: Βασικοί τύποι

Τύπος	**Περιγραφή**
pthread_t	thread id: χαρακτηρίζει κάποιο thread.
pthread_attr_t	Αντικείμενο που χρησιμοποιείται για να παραμετροποιήσει κάποιο thread.
pthread_key_t	Per-thread data: χρησιμοποιείται για αποθήκευση τιμών ανά νήμα.
pthread_mutex_t	Δομές συγχρονισμού: θα μελετηθούν αργότερα.
pthread_cond_t	
pthread_rwlock_t	

PThreads: Χειρισμός threads

Συνάρτηση	**Περιγραφή**

`int pthread_create(t, a, func, arg)`
`pthread_t* t;`
`const pthread_attr_t* a`
`void* (*func)(void*);`
`void* arg;`
	Ξεκινά νέο thread, με attributes `a` (μπορεί να είναι NULL) εκτελώντας την `func` με όρισμα `arg`. Το thread id αποθηκεύεται στο `t`.

`int pthread_exit(val)` `void* val;`	Τερματίζει το τρέχον thread επιστρέφοντας `val`.

`int pthread_join(t, vp)`
`pthread_t* t;`
`void** vp;`
	Περιμένει το `t` να τερματίσει, αποθηκεύει στο `vp` την τιμή επιστροφής.

`pthread_t pthread_self()`	Επιστρέφει το id του τρέχοντος thread.
`int pthread_equal(t1, t2)` `pthread_t t1, t2;`	Συγκρίνει δύο thread ids.

Παράδειγμα

```
void* foo(void* arg) {
    printf("Hello %s, I am a thread\n", (char*)arg);
    return arg;  // το ίδιο με pthread_exit(arg)
}
…

pthread_t t;
static char* name = "vsam";
pthread_create(&t, NULL, foo, name);

… // do other stuff

pthread_join(t, NULL);  // ignore return value
```

Λεπτομέρειες για threads

- Threads και η **fork()**
  - Όταν γίνει **fork()**, κληρονομούνται τα threads του πατέρα?
    - Απάντηση: ΟΧΙ!!!!
  - Πολλά προβλήματα (εκτός αν το παιδί καλέσει **exec()**)
  - Threaded libraries ???
- Threads και signals
  - Όταν έρθει κάποιο signal, ποιο thread θα το εκτελέσει?
  - Απάντηση: περίπλοκο. (**`pthread_kill`**)
  - Ένα signal vector για όλη τη διεργασία
  - Ένα signal mask ανα thread (**`pthread_sigmask`**)
- Cancellation
  - **pthread_cancel**: Προσπαθεί να τερματίσει «βίαια» ένα άλλο thread. ΠΟΛΥ ΠΡΟΒΛΗΜΑΤΙΚΟ.