Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

4η Εργαστηριακή Άσκηση:

Χρονοδρομολόγηση

Λειτουργικά Συστήματα Υπολογιστών
6ο Εξάμηνο, 2019-2020

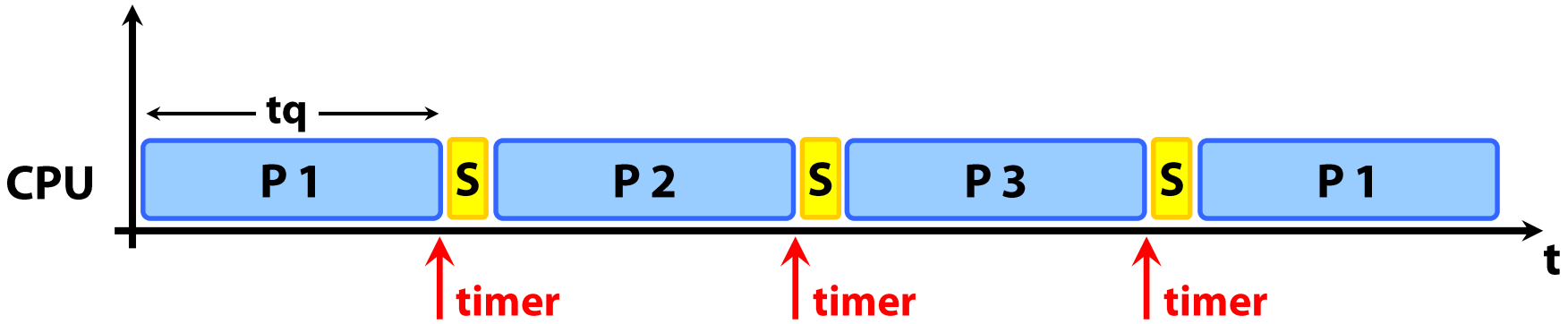
Σύνοψη

- Χρονοδρομολόγηση κυκλικής επαναφοράς (RR)
- Ζητούμενο 1: Χρονοδρομολογητής RR
  - Ασύγχρονη σχεδίαση, βασισμένη σε σήματα
  - Χειρισμός SIGALRM, SIGCHLD
- Ζητούμενο 2: Αλληλεπίδραση με φλοιό
  - Δυναμική δημιουργία και καταστροφή εργασιών
  - Επικοινωνία φλοιού – χρονοδρομολογητή
- Ζητούμενο 3: Χρονοδρομολόγηση με προτεραιότητες
  - Δύο κλάσεις προτεραιότητας: HIGH και LOW

Χρονοδρομολόγηση Κυκλικής Επαναφοράς

- Χρονοδρομολογητής Round-Robin (RR)
- Κυκλική ανάθεση *κβάντων χρόνου* (tq)
  - Για Ν διεργασίες: P_0, P_1, …, P_{N-1}, P_0, P_1, …, P_{N-1}, P_0, …

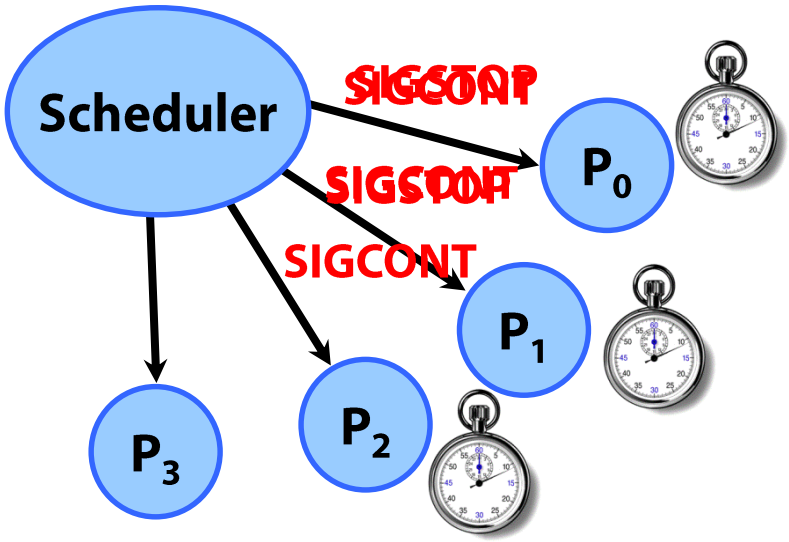
Καταμερισμός Χρόνου: η γενική ιδέα



- Ο υπολογιστικός χρόνος κατανέμεται ανάμεσα στις διεργασίες που είναι έτοιμες να τρέξουν (P1, P2, P3)
  - Κάθε διεργασία τρέχει για χρόνο <= του *κβάντου χρόνου* (time quantum)
- Το χρονοδρομολογητή ενεργοποιούν *διακοπές χρονιστή* (timer interrupts)

Ζητούμενο 1: Χρονοδρομολογητής RR

- Υλοποίηση ενός RR Scheduler
- Στο χώρο χρήστη
  - Μια γονική διεργασία (scheduler) κατανέμει τον υπολογιστικό χρόνο ανάμεσα σε διεργασίες-παιδιά
  - Εκκίνηση – παύση διεργασιών με σήματα **SIGCONT** και **SIGSTOP**
- Η τρέχουσα διεργασία διακόπτεται μετά από *tq* sec



Σχεδίαση Χρονοδρομολογητή (1)

- Ασύγχρονη σχεδίαση, βασισμένη στα σήματα **SIGALRM** και **SIGCHLD**.

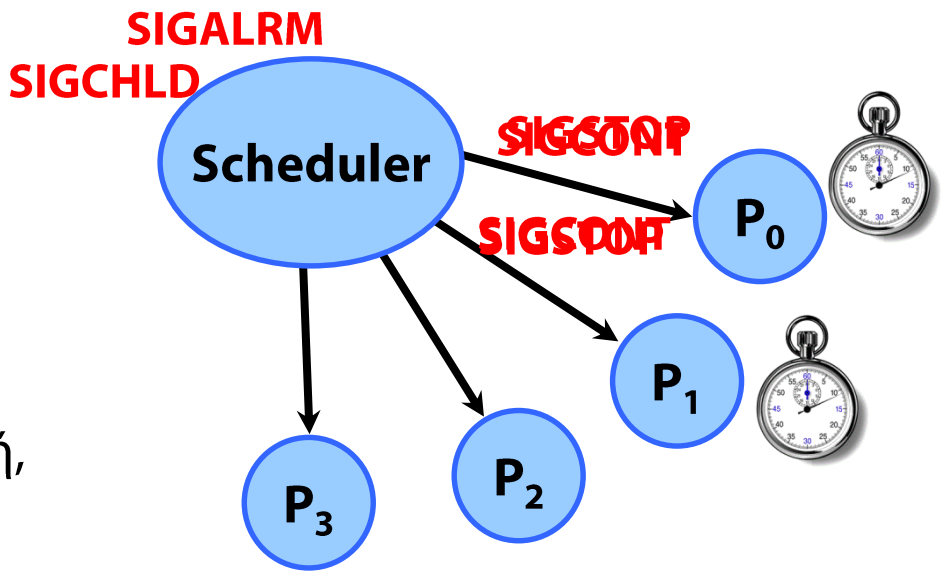
- Όταν εκπνεύσει το κβάντο χρόνου, σταμάτα την τρέχουσα διεργασία. →**SIGALRM**
- Όταν η τρέχουσα διεργασία σταματήσει, βρες την επόμενη και ενεργοποίησέ τη. → **SIGCHLD**

- Χρονιστής βασισμένος στην κλήση συστήματος **alarm()**.

Σχεδίαση Χρονοδρομολογητή (2)

```
sigalrm_handler()
{
        σταμάτα την
        τρέχουσα διεργασία.
}

sigchld_handler()
{
        διάλεξε την επόμενη,
        ρύθμισε τον χρονιστή,
        ενεργοποίησέ τη.
}
```




```
/* Κύριο πρόγραμμα */
while(pause())
        ;
```

Δομές Δεδομένων ΧΔ

- Δομή ανάλογη του Process Control Block
  - αριθμός εργασίας
  - PID εργασίας
  - όνομα εκτελέσιμου)

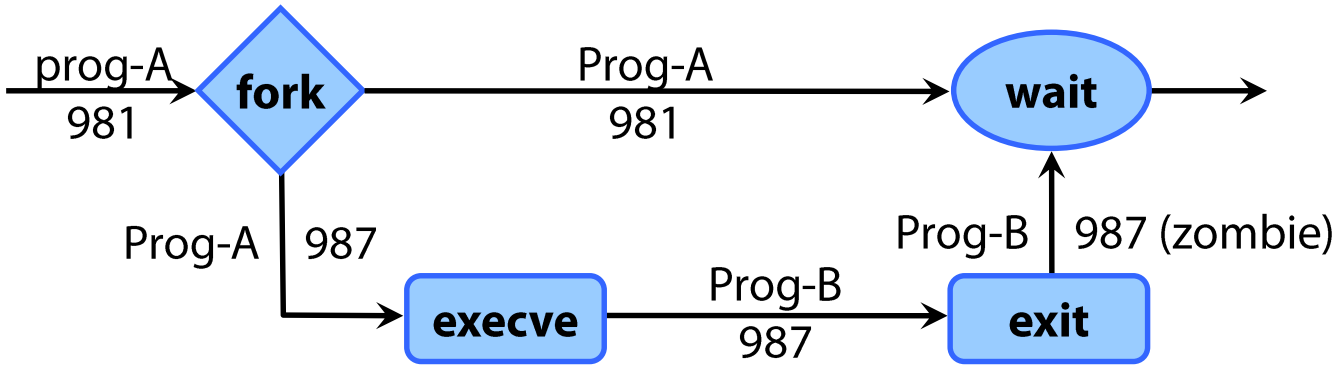
- Ουρά διεργασιών (συνδεδεμένη λίστα ή πίνακας)
  - Μπορείτε να θεωρήσετε ένα μέγιστο αριθμό εργασιών αν διευκολύνει την υλοποίηση
  - αλλά με λίστες σίγουρα είναι ευκολότερο μακροπρόθεσμα

Επισκόπηση Κώδικα

- Τι πρόγραμμα θα εκτελούν οι εργασίες;
  - Σας δίνεται **prog.c**: εκτυπώνει NMSG μηνύματα, με τυχαία καθυστέρηση ανάμεσά τους.
- Δημιουργία διεργασιών με **fork()** + **execve()**
  - δίνεται **execve-example.c**.
- Σκελετός Χρονοδρομολογητή
  - Στο χώρο χρήστη, βασίζεται στα SIGARLM / SIGCHLD
  - δίνεται **scheduler.c**.

Δημιουργία στο μοντέλο του UNIX: fork()

- Όλες οι διεργασίες προκύπτουν με fork() [*σχεδόν* όλες]
  - Ίδιο πρόγραμμα με γονική διεργασία, αντίγραφο χώρου μνήμης, κληρονομεί ανοιχτά αρχεία, συνδέσεις, δικαιώματα πρόσβασης
- Αντικατάσταση προγράμματος διεργασίας: execve()
- Η γονική διεργασία ενημερώνεται για το θάνατο του παιδιού με wait() → συλλογή τιμής τερματισμού (exit status)
  - Μέχρι τότε, παιδί που έχει καλέσει την exit() είναι *zombie*
  - Αν ο γονέας πεθάνει πρώτα, η διεργασία γίνεται παιδί της init (PID = 1), που κάνει συνεχώς wait()



Σήματα στο UNIX (1)

- Αποστολή (**kill()**, **raise()**)

  Παράδειγμα:

```
if (kill(pid, SIGUSR1) < 0) {
        perror("kill");
        exit(1);
}
```

- Χειρισμός (**signal()**, με SIG_IGN, SIG_DFL ή handler)

  Παράδειγμα:

```
void sighandler(int signum)
{
        got_sigusr1 = 1;
}

if (signal(SIGUSR1, sighandler) < 0) {
        perror("could not establish SIGUSR1 handler");
        exit(1);
}
```

Σήματα στο UNIX (2)

- Αναξιόπιστα
  - Τι θα γίνει αν έρθουν πολλά σήματα;
    - Η συνάρτηση χειρισμού θα τρέξει από *1* έως *n* φορές
  - Τι θα γίνει αν το σήμα έρθει ενώ η συνάρτηση χειρισμού εκτελείται;
- Race conditions: αυτό θα δουλέψει;

Παράδειγμα:

```
void sighandler(int signum)
{
        got_sigusr1 = 1;
}
. . .
got_sigusr = 0;
while (!got_sigusr1)
        pause(); /* Αναμονή έως ότου ληφθεί κάποιο σήμα */
```

Σήματα στο UNIX (3)

- Η signal() δεν είναι φορητή
- Ο handler ακυρώνεται όταν εκτελείται (System V)
  - και πρέπει να επανεγκατασταθεί
  - ή όχι… BSD. Στο Linux; εξαρτάται… libC vs. kernel
- Καλύτερη, φορητή λύση: **sigaction()**

Παράδειγμα:

```
struct sigaction sa;
sigset_t sigset;

sa.sa_handler = sigchld_handler;
sa.sa_flags = SA_RESTART;
sigemptyset(&sigset);
sa.sa_mask = sigset;
if (sigaction(SIGCHLD, &sa, NULL) < 0) {
        perror("sigaction");
        exit(1);
}
```

SIGCHLD

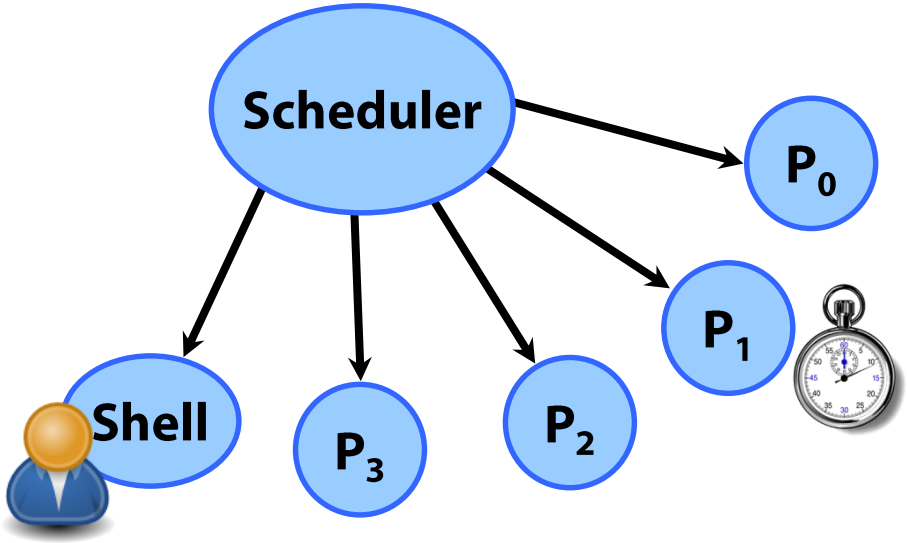
- SIGCHLD: ένα παιδί άλλαξε κατάσταση
  - Πέθανε κανονικά
  - τερματίστηκε από σήμα
  - έχει σταματήσει λόγω σήματος (SIGTSTP, SIGSTOP)
- Επιτρέπει στη γονική διεργασία να κάνει waitpid() *ασύγχρονα*, όταν χρειάζεται
  - Κάτι συμβαίνει σε ένα παιδί
  - Ο πατέρας λαμβάνει SIGCHLD
  - Εκτελεί waitpid()
    - Ιδανικά: πολλές φορές, με WNOHANG

Κώδικας: παράδειγμα χειρισμού SIGCHLD

```
void sigchld_handler(int signum)
{
    for (;;) {
            p = waitpid(-1, &status, WUNTRACED | WNOHANG);
            if (p < 0) {
                    perror("waitpid");
                    exit(1);
            }
            if (p == 0)
                break;
            explain_wait_status(p, status);

            if (WIFEXITED(status) || WIFSIGNALED(status))
                /* A child has died */
            if (WIFSTOPPED(status))
                /* A child has stopped due to SIGSTOP/SIGTSTP, etc */
    }
}
```

Ζητούμενο 2: Αλληλεπίδραση με φλοιό



- Ο φλοιός αλληλεπιδρά με το *χρήστη*
  - Εντολές για *δυναμική* δημιουργία και καταστροφή εργασιών
  - **e**(xec), **k**(ill), **p**(rint queue), **q**(uit)
- Ο φλοιός *χρονοδρομολογείται* μαζί με τις υπόλοιπες διεργασίες
- Ο φλοιός δίνεται και δεν επιτρέπεται *καμία αλλαγή* σε αυτόν!

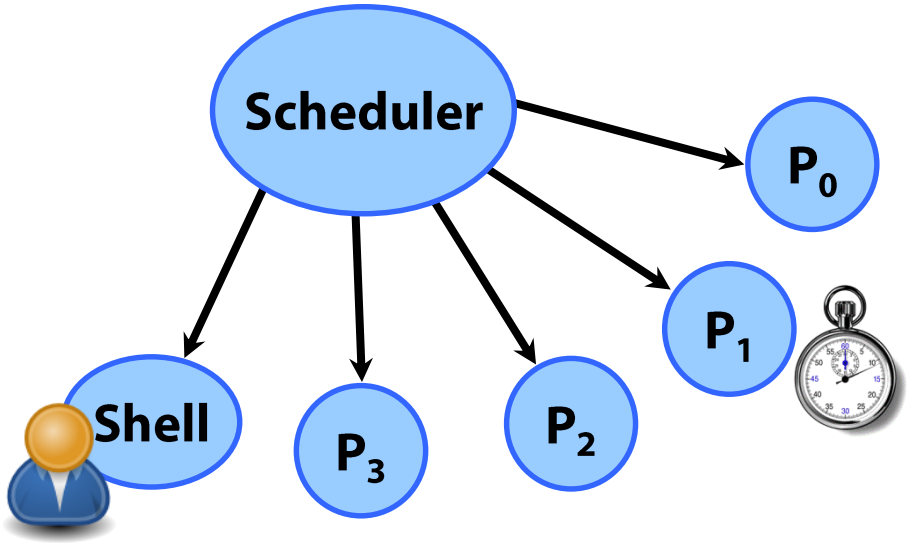
Επισκόπηση Κώδικα

- Πώς επικοινωνεί ο φλοιός με το χρονοδρομολογητή;
  - Δίνονται **request.h**, **shell.c**
- Σκελετός Χρονοδρομολογητή
  - Ξεκινάει το πρόγραμμα του φλοιού
    - Εγκαθιστά το μηχανισμό επικοινωνίας μαζί του
  - Μπαίνει σε βρόχο εξυπηρέτησης αιτήσεων φλοιού
  - δίνεται **scheduler-shell.c**
- Τι μένει;
  - Να γίνει το **scheduler-shell.c** κανονικός ΧΔ
  - Συναρτήσεις που υλοποιούν τις αιτήσεις του φλοιού

Ζητούμενο 2: Αλληλεπίδραση με φλοιό

```
sigalrm_handler()
{
        …
}

sigchld_handler()
{
        …
}
```



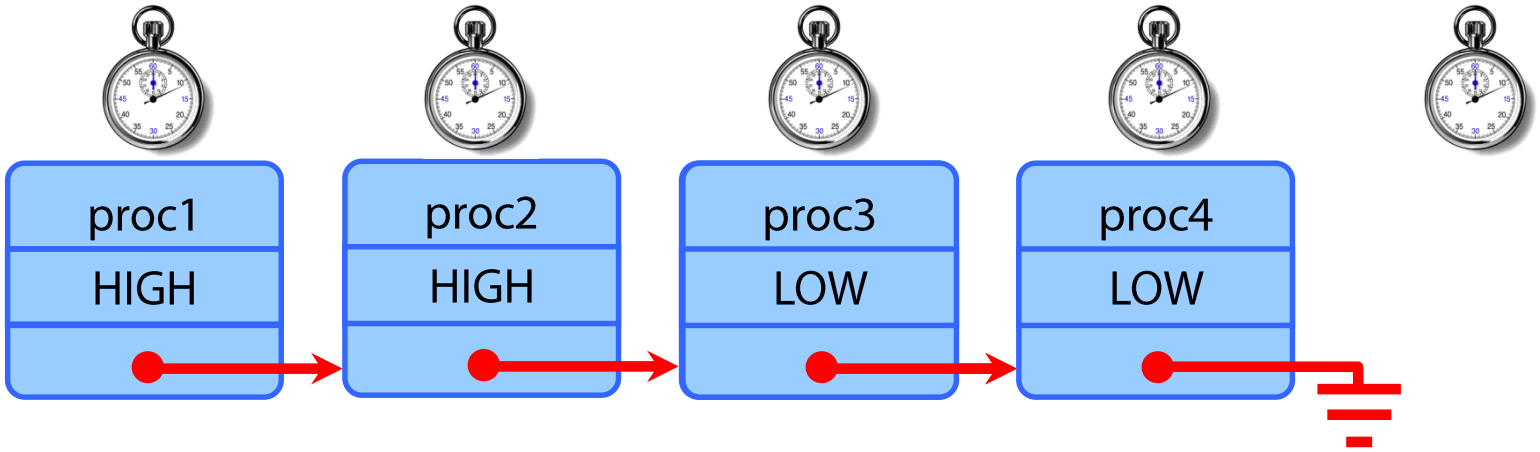

```
/* Κύριο πρόγραμμα */
for (;;) {
        read_request_from_shell();
        process_shell_request();
        write_reply_to_shell();
}
```

Ζητ. 3: Χρονοδρομολόγηση με προτεραιότητες

- Δύο κλάσεις προτεραιότητας
  - HIGH – LOW
  - Κάθε διεργασία ανήκει σε μία από αυτές
  - Μέσα στην ίδια κλάση γίνεται RR

- Χρονοδρομολόγηση με προτεραιότητες
  - όσο υπάρχουν διεργασίες HIGH, επιλέγονται πάντα αυτές προς χρονοδρομολόγηση
  - *ή αλλιώς*: Δεν τρέχει ποτέ LOW διεργασία, αν υπάρχει HIGH στην ουρά

Υλοποίηση ουράς με προτεραιότητες



- Όσο υπάρχουν διεργασίες HIGH
  - δρομολογούνται μόνο αυτές
- Εισαγωγή νέας διεργασίας
  - έχει προτεραιότητα LOW
  - σε ποια θέση μπαίνει στην ουρά;
- Αλλαγή προτεραιότητας διεργασίας
  - Αλλαγή θέσης στην ουρά (HIGH: μπροστά, LOW: μετά τις HIGH)

Ζητήματα Υλοποίησης

- Δύο νέες εντολές στο φλοιό
  - **h**(igh priority), **l**(ow priority)
- Ο χρονοδρομολογητής επεκτείνεται
  - Υλοποίηση εξυπηρέτησης των δύο νέων αιτήσεων
  - Επιλογή με βάση προτεραιότητες
    - επέκταση του PCB με πεδίο prio
    - μεταβολές στη λογική του χρονοδρομολογητή
  - Η εντολή **p**(rint queue) περιλαμβάνει προτεραιότητες

Hands-on!

user@host:~$

Ερωτήσεις;

και στη λίστα:

OS@lists.cslab.ece.ntua.gr

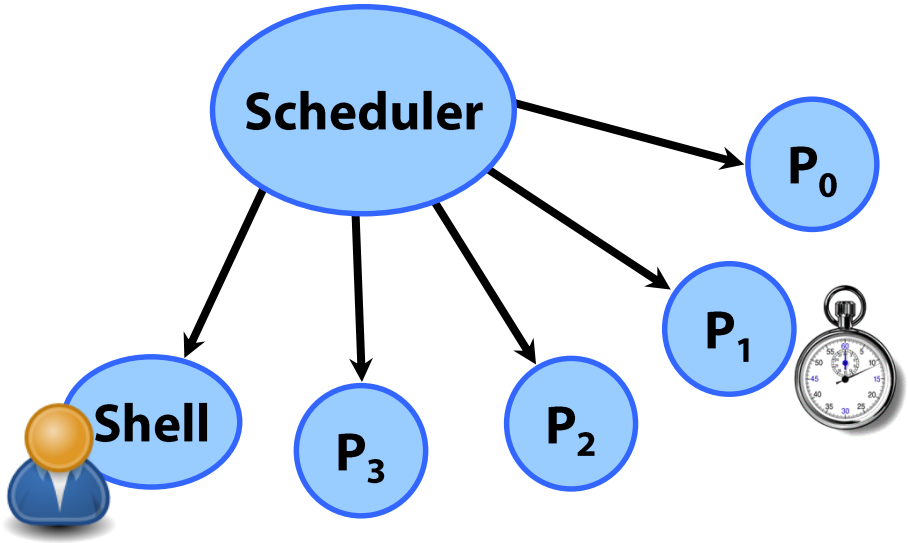
Πλεονάζοντα Πράγματα

Τ Ε Λ Ο Σ

Πλεονάζοντα Πράγματα

```
sigalrm_handler()
{
        …
}

sigchld_handler()
{
        …
}
```



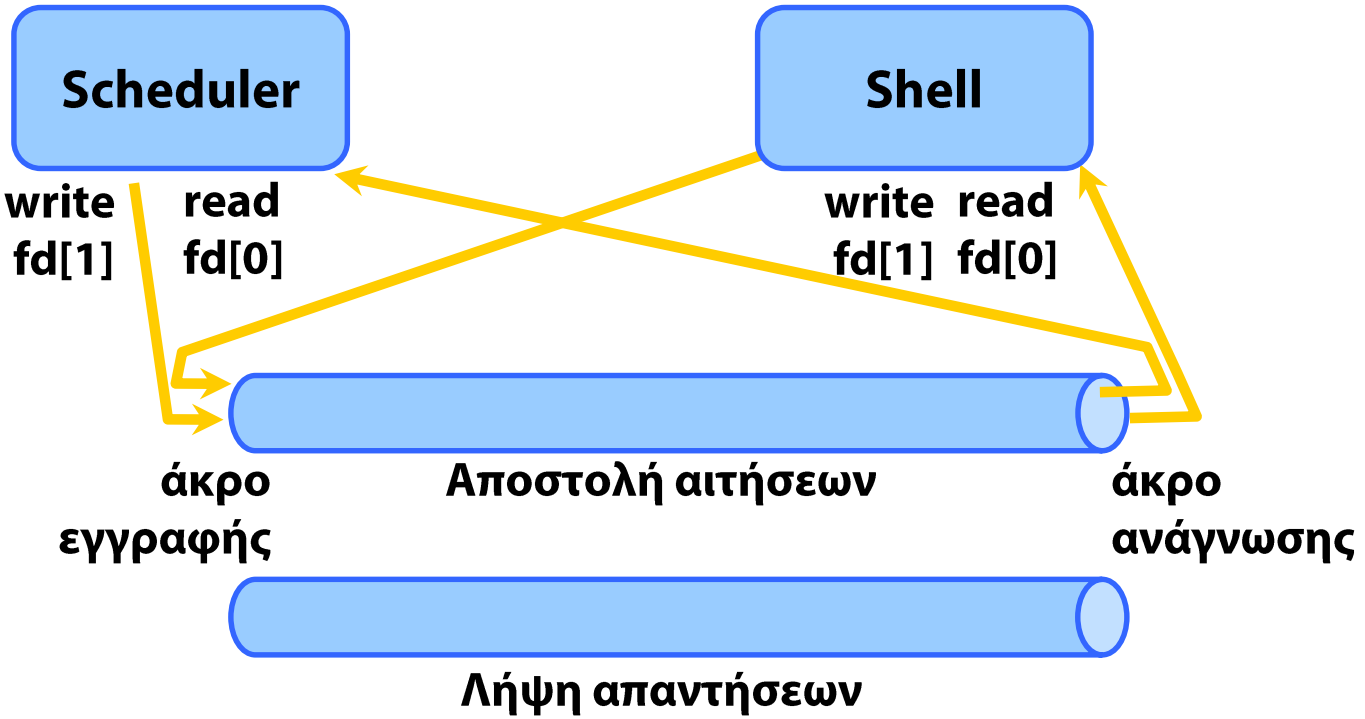

```
/* Κύριο πρόγραμμα */
for (;;) {
        read_request_from_shell();
        process_shell_request();
        write_reply_to_shell();
}
```

Ζητήματα Υλοποίησης (2)

- Πώς θα ξεχωρίζουν τα μηνύματα του κάθε προγράμματος;
  - Διαφορετικό εκτελέσιμο, εκτυπώνει **argv[0]**
- Αντιγραφή του εκτελέσιμου **prog** στα **prog1**, **prog2**, …
- Ή σύνδεση:

```
$ ln -s prog prog1
$ ln -s prog prog2
$ ls -l prog*
-rwxr-xr-x 1 oslabg01 oslab  9177 2014-12-08 23:22 prog*
lrwxrwxrwx 1 oslabg01 oslab  4 2014-12-08 23:22 prog1 -> prog*
lrwxrwxrwx 1 oslabg01 oslab  4 2014-12-08 23:22 prog2 -> prog*
...
```

Επικοινωνία φλοιού – χρονοδρομολογητή (1)



- Δύο σωληνώσεις
  - μία για αποστολή αιτήσεων από φλοιό προς χρονοδρομολογητή
  - μία για λήψη απαντήσεων σε κάθε αίτηση
- Ο φλοιός είναι **άλλο** πρόγραμμα: πώς μαθαίνει τους περιγραφητές;

Ζητήματα Υλοποίησης (3)

- Ο φλοιός μαθαίνει τους περιγραφητές αρχείων για τις σωληνώσεις από ορίσματα της γραμμής εντολών
  - Αν οι δύο σωληνώσεις είναι οι [3, 5] και [7, 9]:
  - ο φλοιός έχει το «5» και το «7» ως ορίσματα στη γραμμή εντολών (**execve()**)
- Ο φλοιός πρέπει να χρονοδρομολογείται *μαζί* με τις υπόλοιπες εργασίες