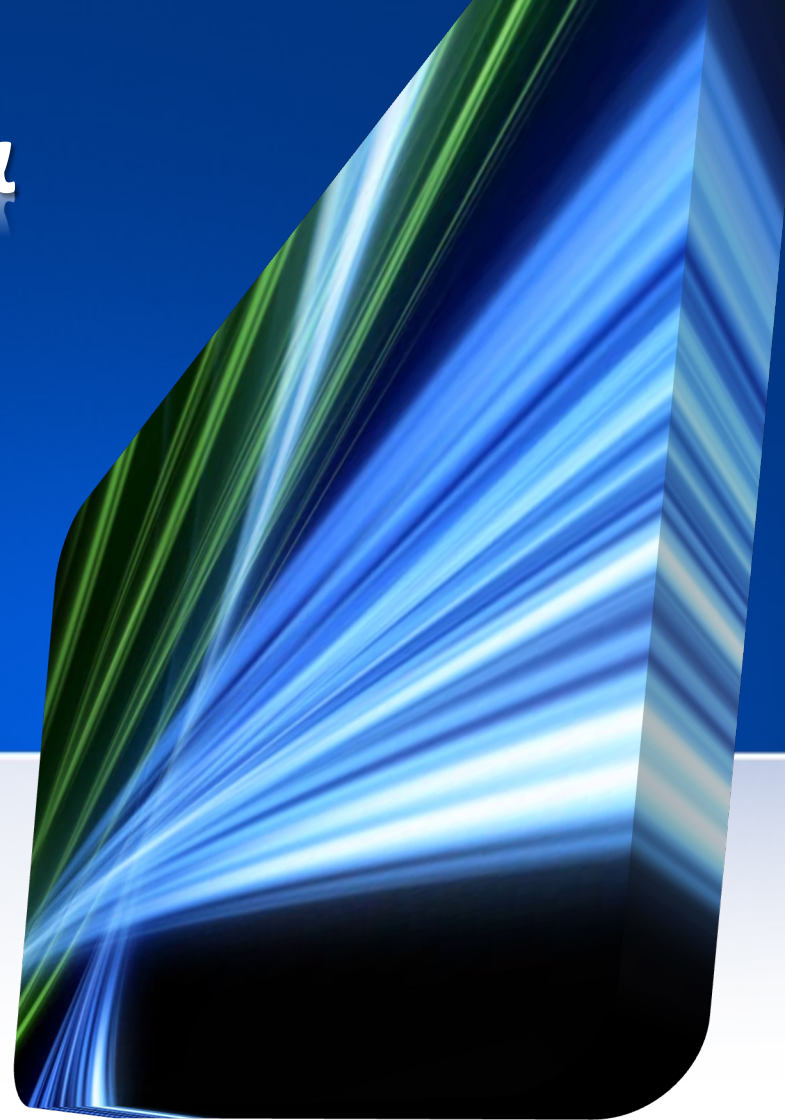


# Λειτουργικά Συστήματα

## 6ο εξάμηνο ΣΗΜΜΥ

### Ακ. έτος 2022-2023

## Εργαστηριακή Άσκηση 2



# Θεωρία Εργ. Άσκησης 2

## Σήματα



**Σήμα** (**signal**) είναι ένας τρόπος επικοινωνίας των διεργασιών.

Είναι μια ειδοποίηση που στέλνεται σε μια διεργασία προκειμένου να την ενημερώσει για κάποιο **γεγονός**.

Η διεργασία που λαμβάνει ένα σήμα **διακόπτει** την εκτέλεση της και αναγκάζεται να χειριστεί το σήμα **άμεσα**.

# Θεωρία Εργ. Άσκησης 2

## Σήματα

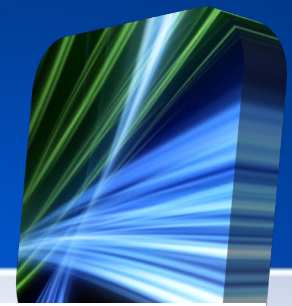


Κάθε σήμα έχει έναν ακέραιο αριθμό που το αντιπροσωπεύει (1,2, ...), καθώς επίσης και ένα συμβολικό όνομα που καθορίζεται συνήθως στο αρχείο `/usr/include/signal.h`

Με την εντολή "**kill -l**" βλέπουμε τον κατάλογο σημάτων που υποστηρίζονται από το σύστημά μας.

# Θεωρία Εργ. Άσκησης 2

## Λίστα Σημάτων



Signal	Value	Action	Comment
<b>SIGHUP</b>	1	Term	Hangup detected on controlling terminal or death of controlling process
<b>SIGINT</b>	2	Term	Interrupt from keyboard
<b>SIGQUIT</b>	3	Core	Quit from keyboard
<b>SIGILL</b>	4	Core	Illegal Instruction
<b>SIGABRT</b>	6	Core	Abort signal from <code>abort(3)</code>
<b>SIGFPE</b>	8	Core	Floating point exception
<b>SIGKILL</b>	9	Term	Kill signal
<b>SIGSEGV</b>	11	Core	Invalid memory reference
<b>SIGPIPE</b>	13	Term	Broken pipe: write to pipe with no readers
<b>SIGALRM</b>	14	Term	Timer signal from <code>alarm(2)</code>
<b>SIGTERM</b>	15	Term	Termination signal
<b>SIGUSR1</b>	30,10,16	Term	User-defined signal 1
<b>SIGUSR2</b>	31,12,17	Term	User-defined signal 2
<b>SIGCHLD</b>	20,17,18	Ign	Child stopped or terminated
<b>SIGCONT</b>	19,18,25	Cont	Continue if stopped
<b>SIGSTOP</b>	17,19,23	Stop	Stop process
<b>SIGTSTP</b>	18,20,24	Stop	Stop typed at terminal
<b>SIGTTIN</b>	21,21,26	Stop	Terminal input for background process
<b>SIGTTOU</b>	22,22,27	Stop	Terminal output for background process

# Θεωρία Εργ. Άσκησης 2

## Σήματα



Κάθε σήμα μπορεί να έχει έναν **χειριστή σήματος** (handler), ο οποίος είναι μια συνάρτηση που εκτελείται όταν η διεργασία λαμβάνει το συγκεκριμένο σήμα. Η συνάρτηση αυτή καλείται ασύγχρονα.

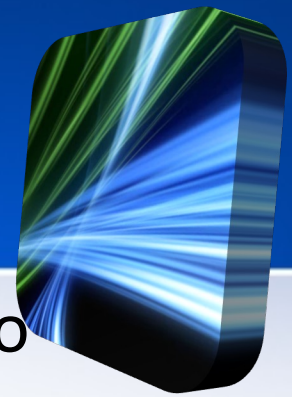
Όταν στέλνεται το σήμα σε μια διεργασία, το ΛΣ σταματά την εκτέλεση της διεργασίας, και "την αναγκάζει" να καλέσει την αντίστοιχη συνάρτηση χειρισμού αυτού του σήματος (handler).

Όταν η συνάρτηση ολοκληρώσει την εκτέλεσή της, η διεργασία συνεχίζει την εκτέλεση από το σημείο που βρισκόταν αμέσως πριν παραληφθεί το σήμα.

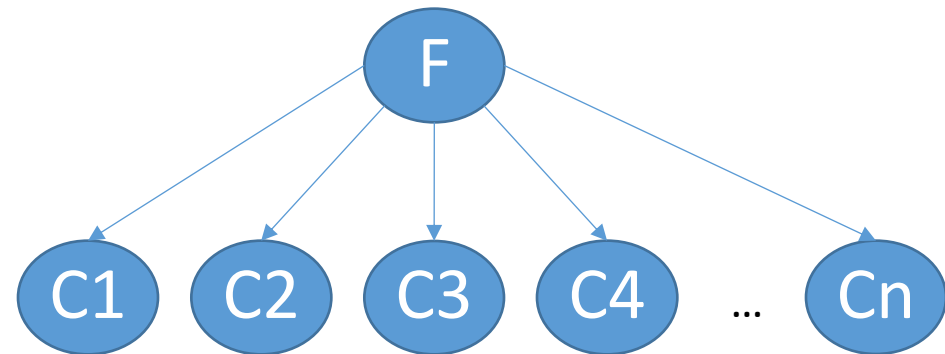


# Εργαστηριακή Άσκηση 2

## The gates are open



Να γραφτεί πρόγραμμα C σε περιβάλλον Linux, στο οποίο η διεργασία-πατέρας (F) δημιουργεί  $n$  διεργασίες-παιδιά (C1, C2, C3, C4 ... Cn) σύμφωνα με το παρακάτω δέντρο διεργασιών:



Ο χρήστης θα μπορεί να στέλνει σήματα **SIGUSR1**, **SIGUSR2**, **SIGTERM** στην διεργασία πατέρα και στις διεργασίες παιδιά. Ο χειρισμός αυτών των σημάτων θα εξηγηθεί αναλυτικά στις επόμενες διαφάνειες

# Εργαστηριακή Άσκηση 2

## The gates are open



Παράδειγμα εκτέλεσης προγράμματος

**`./gates tfttt`**

Το πρόγραμμα μπορεί να χρησιμοποιηθεί για τη διαχείριση ενός αριθμού πυλών (gates). Ο πατέρας δημιουργεί N διεργασίες παιδιά (το N είναι το μήκος του string "tfttt"). Κάθε μια διεργασία παιδί θα "διαχειρίζεται" την αντίστοιχη πύλη. Ο πατέρας δέχεται τον αριθμό των πυλών και την αρχική τους κατάσταση ως συμβολοσειρά που αποτελείται από χαρακτήρες «t» ή «f», π.χ. «ttffff» σημαίνει 5 πύλες, 2 ανοιχτές και 3 κλειστές.

# Εργαστηριακή Άσκηση 2

## The gates are open



Αρχικά, η διεργασία πατέρας δημιουργεί όλες τις διεργασίες παιδιά και τυπώνει το δέντρο διεργασιών που έχει δημιουργηθεί ώστε να φαίνονται τα PID τους.

```
$ ./gates ft
```

```
[PARENT/PID=100] Created child 0 (PID=101) and initial state 'f'
```

```
[PARENT/PID=100] Created child 1 (PID=102) and initial state 't'
```



# Εργαστηριακή Άσκηση 2

## The gates are open



Οι διεργασίες παιδιά κάθε 15 δευτερόλεπτα τυπώνουν το state τους (με alarm) και τον χρόνο που εκτελούνται

**[GATE=0/PID=101/TIME=0s] The gates are closed!**

**[GATE=1/PID=102/TIME=0s] The gates are open!**

**[GATE=0/PID=101/TIME=15s] The gates are closed!**

**[GATE=1/PID=102/TIME=15s] The gates are open!**

# Εργαστηριακή Άσκηση 2

## The gates are open



Οι διεργασίες παιδιά

- όταν λαμβάνουν σήμα SIGUSR2 τυπώνουν το state τους: state είναι η κατάσταση της πύλης (ανοιχτή/κλειστή, μέσω μιας απλής boolean τιμής) και ένας αριθμός (δευτερόλεπτα από την στιγμή εκκίνησης).
- όταν λαμβάνουν σήμα SIGUSR1 κάνουν flip το state.
- όταν λαμβάνουν σήμα SIGTERM τερματίζουν

Οι διεργασίες παιδιά μπορούν να λάβουν σήμα είτε από τον χρήστη από το terminal είτε από τον πατέρα

# Εργαστηριακή Άσκηση 2

## The gates are open



Η διεργασία πατέρας

- όταν λάβει σήμα SIGUSR1, στέλνει SIGUSR1 σε όλα τα παιδιά.
- όταν λάβει σήμα SIGUSR2, στέλνει SIGUSR2 σε όλα τα παιδιά.
- όταν λάβει σήμα SIGTERM, στέλνει SIGTERM σε όλα τα παιδιά.
- φροντίζει επίσης όλα τα παιδιά να είναι υγιή, οπότε ελέγχει για σήματα SIGCHLD και (α) αν κάποιο παιδί έχει σκοτωθεί, το κάνει wait και φτιάχνει καινούριο στη θέση του (β) αν κάποιο παιδί έχει σταματήσει, το κάνει resume. Θα πρέπει να υπάρχουν πάντα N παιδιά ενεργά, κάθε ένα για μια πύλη.

# Εργαστηριακή Άσκηση 2

## Υποδειγματική εκτέλεση



```
$ ./gates ft
```

```
[PARENT/PID=100] Created child 0 (PID=101) and initial state 'f'
```

```
[PARENT/PID=100] Created child 1 (PID=102) and initial state 't'
```

```
[GATE=0/PID=101/TIME=0s] The gates are closed!
```

```
[GATE=1/PID=102/TIME=0s] The gates are open!
```

```
[GATE=0/PID=101/TIME=15s] The gates are closed!
```

```
[GATE=1/PID=102/TIME=15s] The gates are open!
```

```
kill -SIGUSR1 102
```

```
[GATE=1/PID=102/TIME=24s] The gates are closed!
```

```
[GATE=0/PID=101/TIME=30s] The gates are closed!
```

```
[GATE=1/PID=102/TIME=30s] The gates are closed!
```

```
kill -SIGTERM 101
```

```
[PARENT/PID=100] Child 0 with PID=101 exited
```

```
[PARENT/PID=100] Created new child for gate 0 (PID 103) and initial state 'f'
```

```
[GATE=0/PID=103/TIME=0s] The gates are closed!
```

```
[GATE=1/PID=102/TIME=45s] The gates are closed!
```

```
[GATE=0/PID=103/TIME=15s] The gates are closed!
```

```
kill -SIGUSR1 103
```

```
[GATE=0/PID=103/TIME=21s] The gates are open!
```

```
[GATE=1/PID=102/TIME=60s] The gates are closed!
```

```
[GATE=0/PID=103/TIME=30s] The gates are open!
```

```
kill -SIGTERM 100
```

```
[PARENT/PID=100] Waiting for 2 children to exit
```

```
[PARENT/PID=100] Child with PID=103 terminated successfully with exit status code 0!
```

```
[PARENT/PID=100] Waiting for 1 children to exit
```

```
[PARENT/PID=100] Child with PID=102 terminated successfully with exit status code 0!
```

```
[PARENT/PID=100] All children exited, terminating as well
```

# Εργαστηριακή Άσκηση 2

## Απαραίτητες Υποδείξεις



- Έλεγχος ορθότητας ορισμάτων (`argc`, `argv`) και εκτύπωση κατάλληλου μηνύματος
- Έλεγχος σφαλμάτων για κάθε κλήση συστήματος
- Ο κώδικας των παιδιών είναι σε ξεχωριστό αρχείο `child.c`.  
Να γίνει χρήση `execv`
- Να δημιουργηθεί `makefile` που παράγει τα εκτελέσιμα

# Εργαστηριακή Άσκηση 2



Μια διεργασία μπορεί να στείλει ένα σήμα σε μια άλλη, με την κλήση **kill(pid, SIGNAL)** η οποία στέλνει το σήμα με όνομα **SIGNAL** στη διεργασία με process id ίσο με το όρισμα **pid**.

Για παράδειγμα με την εκτέλεση της εντολή **kill(3247, SIGCONT)** στέλνεται το σήμα **SIGCONT** στην διεργασία με pid = 3247



# Εργαστηριακή Άσκηση 2



Η εντολή αποστολής ενός σήματος από terminal είναι

**kill -*SIGNAL* pid**

όπου ***SIGNAL*** το όνομα του σήματος που θέλουμε να αποσταλεί και **pid** το process id της διεργασίας στην οποία θέλουμε να στείλουμε το σήμα. Για παράδειγμα η εντολή **kill -*SIGUSR1* 3216** στέλνει το σήμα ***SIGUSR1*** στην διεργασία με pid = 3216

# Εργαστηριακή Άσκηση 2



Η **waitpid()** χρησιμοποιείται για να περιμένει η καλούσα διεργασία αλλαγές κατάστασης σε μια διεργασία παιδί της και να λάβει πληροφορίες σχετικά με την διεργασία της οποίας η κατάσταση έχει αλλάξει

# Εργαστηριακή Άσκηση 2



Ορισμός **waitpid()**

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

- Το πεδίο **pid** ορίζει το pid της διεργασίας που περιμένουμε να αλλάξει κατάσταση (με τιμή ίση με -1 περιμένουμε οποιαδήποτε διεργασία παιδί)
- Το πεδίο **wstatus** είναι ο δείκτης στην μεταβλητή στην οποία μπορούμε να αποθηκεύσουμε την κατάσταση της διεργασίας παιδί
- Το πεδίο **options** ορίζει επιπλέον επιλογές

Περισσότερες πληροφορίες

<http://www.man7.org/linux/man-pages/man2/waitpid.2.html>

# Εργαστηριακή Άσκηση 2



Μια διεργασία μπορεί να ορίσει τον χειριστή (handler) κάθε σήματος που θα λάβει με τις συναρτήσεις:

- **sigaction()** (προτείνεται)
- **signal()**

Η κλήση **signal(SIGNAL, handler)** καθορίζει ότι όταν η καλούσα διεργασία λάβει σήμα με όνομα **SIGNAL** θα εκτελέσει τη συνάρτηση χειρισμού σήματος **handler**.

# Εργαστηριακή Άσκηση 2



Η κλήση συστήματος `sigaction()` χρησιμοποιείται για να αλλάξει τη δράση (action) που θα γίνει από μια διεργασία κατά την λήψη συγκεκριμένου σήματος.

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

- Η παράμετρος **signum** καθορίζει το σήμα και μπορεί να είναι οποιοδήποτε έγκυρο σήμα εκτός από SIGKILL και SIGSTOP
- Εάν η παράμετρος **act** είναι not NULL τότε θέτει το νέο sigaction για το δεδομένο signal
- Εάν η παράμετρος **oldact** είναι not NULL τότε εκεί αποθηκεύεται η προηγούμενη ορισμένη δράση για το δεδομένο signal

# Εργαστηριακή Άσκηση 2



Η παράμετρος **act** και **oldact** είναι δομές τύπου **struct sigaction**

```
struct sigaction {  
    void (*sa_handler)(int);           // καθορίζει τη δράση που πρέπει να συσχετιστεί με το signum  
    void (*sa_sigaction)(int, siginfo_t *, void *); // Εάν ορίζεται SA_SIGINFO στο sa_flags, τότε  
                                           // στο sa_sigaction (αντί του sa_handler) καθορίζεται η λειτουργία χειρισμού για το signum  
                                           // μην εκχωρήσετε τιμή και στα δύο: sa_handler και sa_sigaction  
    sigset_t sa_mask;                  // καθορίζει μια μάσκα σημάτων που πρέπει να αποκλειστούν  
    int sa_flags;                       // καθορίζει ένα σύνολο σημαιών που τροποποιούν τη συμπεριφορά του σήματος  
    void (*sa_restorer)(void);          // Το πεδίο δεν προορίζεται για application use  
};
```



# Εργαστηριακή Άσκηση 2



Παράδειγμα βασικής χρήσης **sigaction()**

<b>struct sigaction action;</b>	//δήλωση δομής struct sigaction
<b>action.sa_handler = myfunction;</b>	//ορισμός ονόματος συνάρτησης χειρισμού
<b>sigaction(SIGUSR1, &amp;action, NULL);</b>	//ορισμός χειριστή ( <b>myfunction</b> ) για το σήμα <b>SIGUSR1</b>

**Αποτέλεσμα:** όταν η διεργασία λάβει το σήμα **SIGUSR1** θα διακόψει την εκτέλεση της και θα καλέσει την συνάρτηση **myfunction()**

Περισσότερες πληροφορίες

<http://man7.org/linux/man-pages/man2/sigaction.2.html>

# Εργαστηριακή Άσκηση 2



Οι διεργασίες μπορούν να ορίσουν να εκτελούν μια εργασία μετά την πάροδο συγκεκριμένου χρόνου. Για να το πετύχουν αυτό οι διεργασίες μπορούν να κάνουν κλήση της συνάρτησης **alarm()**

Η συνάρτηση **alarm(unsigned int seconds)** θα προκαλέσει το σύστημα να παράγει ένα σήμα **SIGALRM**, προς τη διεργασία που την κάλεσε, μετά την παρέλευση του αριθμού των δευτερόλεπτων πραγματικού χρόνου που καθορίζονται από την παράμετρο **seconds**.

# Χρήσιμη αναφορά



<http://man7.org/linux/man-pages/man7/signal.7.html>