Προγραμματισμός συστημάτων UNIX/POSIX

Σήματα (signals)

Σήματα (signals)

- Τα σήματα είναι «διακοπές» λογισμικού (software interrupts) οι οποίες διακόπτουν την κανονική λειτουργία μίας διεργασίας.
- Προκαλούνται από διάφορες συνθήκες, π.χ. κάποιο πρόβλημα στο hardware, κάποια παράνομη λειτουργία (π.χ. διαίρεση δια μηδέν ή προσπέλαση σε θέση μνήμης που δεν επιτρέπεται) ή και από τον χρήστη (π.χ. πάτημα CTRL-C).
- Τα περισσότερα σήματα προκαλούν τον άμεσο τερματισμό της διεργασίας, εκτός και αν έχει προβλεφτεί τρόπος «διαχείρισής» τους από την ίδια την εφαρμογή.
- Υπάρχουν πολλά διαφορετικά είδη σημάτων, π.χ.

```
SIGINT 2 /* interrupt */,  SIGQUIT 3 /* quit */,
SIGILL 4 /* illegal instruction */, SIGKILL 9 /* hard kill */,
SIGALRM 14 /* alarm clock */,  SIGCHLD 20 /* to parent on child exit */
```

Οι «φάσεις» ενός σήματος

- **Signal generation**
  Εφόσον συμβεί κάποιο γεγονός (π.χ. διαίρεση δια 0) τότε ***παράγεται*** ένα σχετικό σήμα (π.χ. SIGFPE)
- **Signal delivery**
  Το σήμα ***παραδίδεται*** στη διεργασία που πρέπει
- **Signal action**
  Με την παράδοση, προκαλείται κάποια ***ενέργεια*** (π.χ. τερματίζει τη διεργασία)
- **Pending signal**
  Μεταξύ της στιγμής που παράγεται και της στιγμής που παραδίδεται στη διεργασία, το σήμα βρίσκεται σε φάση ***εκκρεμότητας*** (συνήθως αμελητέος χρόνος).

- Τα μόνα πράγματα που μπορεί να κάνει μία διεργασία είναι:
  - Να αλλάξει την ενέργεια (action) που προκαλείται από το σήμα
  - Να κρατάει ένα σήμα σε κατάσταση εκκρεμότητας επ' αόριστο, «**μπλοκάροντάς**» το (**blocked**). Το σήμα θα παραδοθεί όταν και εφόσον η διεργασία το «**ξεμπλοκάρει**» (**unblocked**)
  - … κι αυτά, όχι για όλους τους τύπους σημάτων.

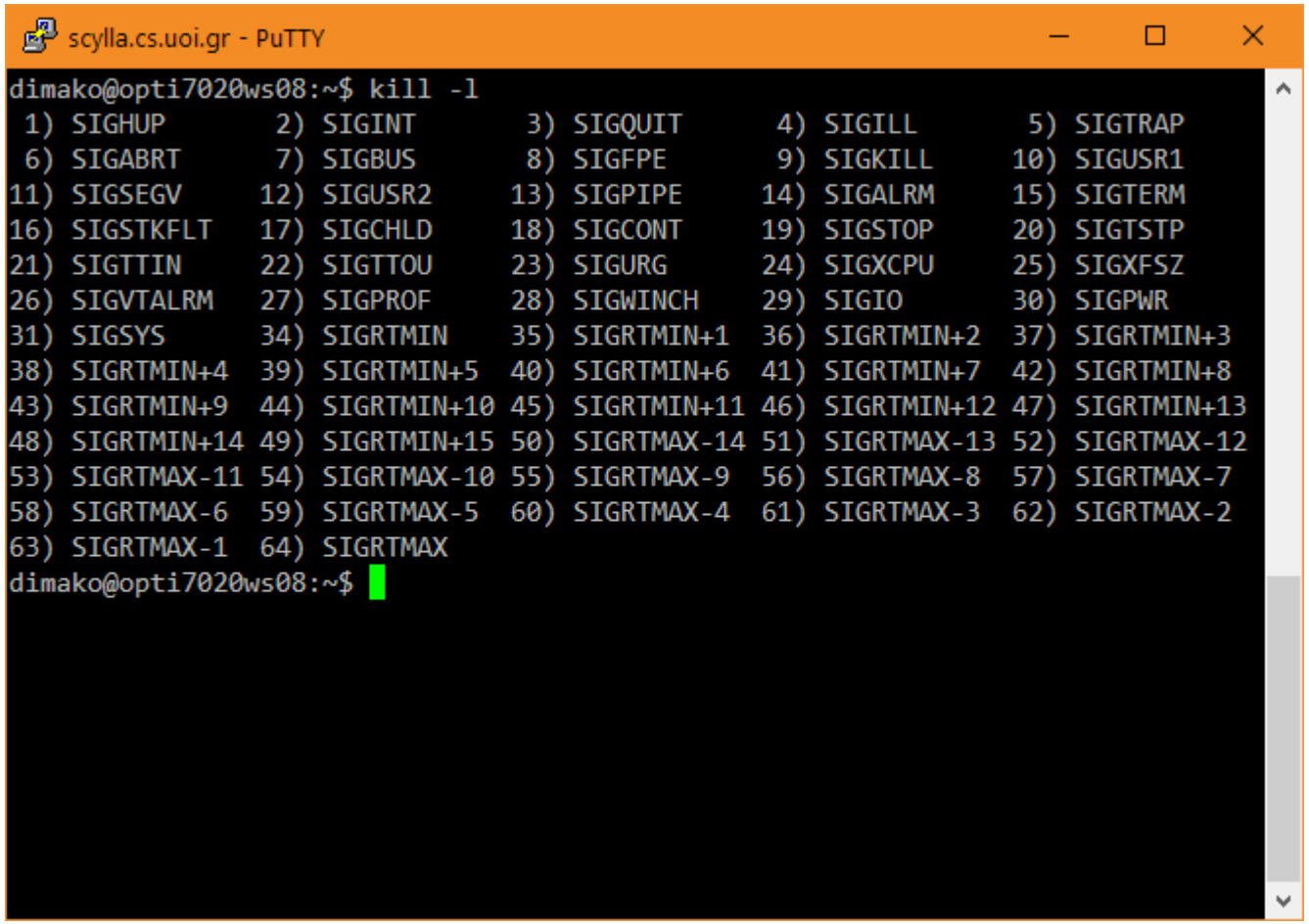
Παραδείγματα από σήματα

ΑΙΤΙΟ	ΣΗΜΑ	DEFAULT ACTION	ΜΠΛΟΚΑΡΕΤΑΙ;	ΑΛΛΑΖΕΙ Η ΕΝΕΡΓΕΙΑ;
Αριθμητικό σφάλμα (διαίρεση δια 0, υπερχείλιση κλπ)	SIGFPE	Τερματισμός διεργασίας	ΝΑΙ	ΝΑΙ
Πρόσβαση σε μνήμη που δεν κατέχει	SIGSEGV	Τερματισμός διεργασίας	ΝΑΙ	ΝΑΙ
Ο χρήστης πάτησε CTRL-C	SIGINT	Τερματισμός διεργασίας	ΝΑΙ	ΝΑΙ
$ kill -9	SIGKILL	Τερματισμός διεργασίας	ΟΧΙ	ΟΧΙ
Τερματίζει μία θυγατρική διεργασία	SIGCHLD	Αγνοείται	ΝΑΙ	ΝΑΙ

- Για περισσότερα:
  `$ man signal` ή
  `$ man 7 signal`

Μια λίστα από τα διαθέσιμα σήματα

- Τα κλασικά και συνηθισμένα σήματα είναι μέχρι το 31 (SIGSYS).
- Από το 32 και μετά είναι τα λεγόμενα σήματα πραγματικού χρόνου (Real Time Signals) που δεν μας αφορούν εδώ.



Πώς **μπλοκάρω/ξεμπλοκάρω** ένα σήμα;

- Κάθε διεργασία διαθέτει ένα σύνολο από σήματα που μπλοκάρει (και άρα δεν επιτρέπει στο σύστημα να της παραδίδονται), το λεγόμενο ***signal mask***.
  - Αν παραχθεί κάποιο από τα σήματα που είναι στο signal mask, δεν θα της παραδοθεί και άρα δεν θα προκληθεί καμία ενέργεια από αυτό.
  - Το σήμα είναι pending. Θα παραδοθεί όταν (αν) η διεργασία το ξεμπλοκάρει.

- Για να μπλοκάρουμε ένα σήμα, αρκεί να το βάλουμε στο signal mask και για να το ξεμπλοκάρουμε αρκεί να το βγάλουμε από αυτό το σύνολο.

- Το signal mask αλλάζει με τη συνάρτηση: [*επιστρέφει < 0 αν αποτύχει*]

```
#include <signal.h>
int sigprocmask(int how, sigset_t *newmask, sigset_t *oldmask);
```

όπου το `newmask` είναι το νέο σύνολο, ενώ στο `oldmask` θα επιστραφεί το παλαιό (αν `oldmask`≠`NULL`). Το how καθορίζει πώς θα γίνει η αλλαγή:

- `SIG_SETMASK`: το νέο signal mask είναι ακριβώς αυτό που δίνεται στο `newmask`
- `SIG_BLOCK`: πρόσθεσε στο υπάρχον signal mask ότι έχει το `newmask`
- `SIG_UNBLOCK`: αφαίρεσε από το υπάρχον mask ότι έχει το `newmask`

sigprocmask() – συνέχεια

- Το `sigset_t` είναι μία δομή που καταγράφει ποια σήματα μπλοκάρονται.
- Οι συναρτήσεις που μπορούν να προσθαφαιρέσουν σήματα στη δομή αυτή είναι οι εξής:

`sigemptyset(sigset_t *set)`	Άδειασε το σύνολο. Κανένα σήμα δεν θα μπλοκαριστεί.
`sigfillset(sigset_t *set)`	Βάλε όλα τα σήματα μέσα στο σύνολο. Όλα μπλοκάρονται.
`sigaddset(sigset_t *set, int sigid)`	Πρόσθεσε το σήμα `sigid` στο σύνολο (θα μπλοκάρεται)
`sigdelset(sigset_t *set, int sigid)`	Αφαίρεσε το σήμα `sigid` από το σύνολο (ξεμπλοκάρεται)
`sigismember(sigset_t *set, int sigid)`	Έλεγξε αν το σήμα `sigid` είναι στο σύνολο

Παράδειγμα χρήσης

- Έλεγχος αν η τρέχουσα signal mask περιέχει το SIGINT και αν ναι, το ξεμπλοκάρουμε:

```
#include <signal.h>
...

int main() {
  sigset_t newmask, oldmask;

  ...

  sigprocmask(SIG_SETMASK, NULL, &oldmask);    /* Just get the current mask */
  if (sigismember(&oldmask, SIGINT)) {         /* Check if SIGINT is blocked */
    sigemptyset(&newmask);                     /* Create an empty set */
    sigaddset(&newmask, SIGINT);               /* Add the SIGINT signal */
    sigprocmask(SIG_UNBLOCK, &newmask, NULL);  /* Remove from current mask */
  }

  ...
}
```

Πώς αλλάζω την **ενέργεια** που θα προκληθεί;

- Προκειμένου να προσδιορίσουμε την ενέργεια που θα προκληθεί όταν και αν μας παραδοθεί κάποιο σήμα, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση `sigaction()`: [*επιστρέφει < 0 αν αποτύχει*]

```
#include <signal.h>
int sigaction(int sigid,
              struct sigaction *newact,
              struct sigaction *oldact);
```

όπου το `sigid` είναι το σήμα που μας ενδιαφέρει, και `newact` είναι μία δομή που περιγράφει τη νέα ενέργεια που θέλουμε να προκαλείται. Εφόσον το `oldact` δεν είναι NULL, κατά την επιστροφή θα περιέχει την περιγραφή της παλιάς ενέργειας.

- Οι ενέργειες που μπορούμε να ορίσουμε (μέσω του `newact`) είναι μία από τις παρακάτω τρεις:
  - Να αγνοηθεί το σήμα (και άρα να μην προκαλέσει τίποτε)
  - Να γίνει η ενέργεια που έχει εξ ορισμού το σύστημα (default action), όποια και να είναι αυτή
  - Να κληθεί μία δική μας συνάρτηση, κάτι που είναι γνωστό ως *διαχειριστής σήματος (signal handler)*

Καθορίζοντας την ενέργεια

Η δομή που περιγράφει την ενέργεια έχει ως εξής:

```
struct sigaction {
  void (*sa_handler)(int); /* SIG_DFL, SIG_IGN ή handler δικό μας */
  sigset_t sa_mask;        /* Επιπρόσθετα σήματα προς μπλοκάρισμα */
  int sa_flags;            /* 0 συνήθως */
  ...
};
```

Λεπτομέρειες:

1. Το πεδίο `sa_handler` είναι δείκτης σε συνάρτηση δική μας (διαχειριστής σήματος) ή του δίνουμε την τιμή `SIG_DFL` (ώστε να εκτελεστεί η default action του συστήματος) ή του δίνουμε την τιμή `SIG_IGN` (ώστε να αγνοηθεί το σήμα).
2. Η συνάρτηση που ορίζουμε ως διαχειριστή θα έχει ως μοναδικό όρισμα την ταυτότητα του σήματος.

Καθορίζοντας την ενέργεια (συνέχεια)

Η δομή που περιγράφει την ενέργεια έχει ως εξής:

```
struct sigaction {
  void (*sa_handler)(int); /* SIG_DFL, SIG_IGN ή handler δικό μας */
  sigset_t sa_mask;        /* Επιπρόσθετα σήματα προς μπλοκάρισμα */
  int sa_flags;            /* 0 συνήθως */
  ...
};
```

Λεπτομέρειες (συνέχεια):

3. Εφόσον εκτελεστεί η ενέργεια (π.χ. ο δικός μας handler), *το σήμα που μας ενδιαφέρει θα είναι μπλοκαρισμένο μέχρι την ολοκλήρωσή της*, οπότε και το σήμα θα ξεμπλοκαριστεί αυτόματα.
4. Αν μας ενδιαφέρει, προσωρινά, όσο εκτελείται η ενέργεια, να μπλοκαριστούν επιπρόσθετα σήματα, αυτά πρέπει να τα συμπεριλάβουμε στη μάσκα `sa_mask`. Με το τέλος της ενέργειας, αυτά τα σήματα ξεμπλοκάρονται αυτόματα.

Παράδειγμα διαχειριστή σήματος

- Η παρακάτω εφαρμογή δεν μπορεί να τερματιστεί με CTRL-C διότι έχει ορίσει ένα χειριστή για το σήμα SIGINT:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void handlectrlc(int sigid) {
  printf("You cannot stop me with CTRL-C!\n");
}

int main() {
  struct sigaction sact;

  sact.sa_handler = handlectrlc; /* Our handler to catch CTRL-C */
  sigemptyset(&sact.sa_mask);    /* No other signal to block */
  sact.sa_flags = 0;             /* Nothing special, must be 0 */
  if (sigaction(SIGINT, &sact, NULL) < 0)
    perror("could not set action for SIGINT");

  /* Loop for ever */
  while(1) {
    sleep(1);    /* Let it sleep for a while */
  }
  return 0;
}
```

Σημειώσεις (I): η συνάρτηση signal()

- Ο ορισμός ενός διαχειριστή σήματος γινόταν με πιο απλό (αλλά πλέον *ξεπερασμένο* τρόπο) με την κλήση `signal()`.

```
void (*signal(int sig, void (*func)(int)))(int);
```

  Η οποία βασικά λέει στη `signal()` ότι αν συμβεί το σήμα `sig`, θα πρέπει να κληθεί η συνάρτηση `func`. Αν όλα πάνε καλά η `signal()` επιστρέφει δείκτη προς τον παλαιό handler, αλλιώς `SIG_ERR`.

  - Η `signal` πρέπει να αποφεύγεται και να χρησιμοποιείται η `sigaction`, έστω κι αν φαίνεται πιο πολύπλοκη.

- Μέσα στον διαχειριστή, πρέπει να αποφεύγονται «επικίνδυνες» κλήσεις συστήματος (ακόμα και η `printf()` πρέπει να αποφεύγεται) και γενικότερα τα πράγματα πρέπει να γίνονται πολύ προσεκτικά – ψάξτε να μελετήσετε τις λεπτομέρειες και τους λόγους για αυτό.

Σημειώσεις (II): σήματα ως διαδιεργασιακή επικοινωνία

- Μία διεργασία μπορεί να δημιουργήσει και να στείλει ένα σήμα σε μία άλλη (δεν επιτρέπονται όλα).

```
#include <signal.h>
int kill(pid_t pid, int signal);
```

- Υπάρχουν τα SIGUSR1 και SIGUSR2.
- Με αυτό τον τρόπο, αν η διεργασία που το λαμβάνει έχει ορίσει τον αντίστοιχο διαχειριστή, είναι σαν να έχουμε «επικοινωνήσει» στέλνοντάς της έναν (συγκεκριμένο) ακέραιο αριθμό.
- Τα σήματα δεν μπορούν να χρησιμοποιηθούν για ανταλλαγή δεδομένων – το μόνο που μπορεί να «στείλει» ένα σήμα είναι ο εαυτός του και τίποτε άλλο.
- Περισσότερο για συγχρονισμό παρά επικοινωνία.