ΣΗΜΑΤΑ (SIGNALS)

Λειτουργικά Συστήματα - Εργαστήριο

Πάνος Παπαδόπουλος

Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική
Πανεπιστήμιο Θεσσαλίας

ΘΕΩΡΙΑ

ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

- **Signal** είναι ένα **software interrupt** που στέλνεται σε ένα process.
- Κάθε **signal** έχει μοναδικό **id** που φανερώνει τον λόγο για τον οποίο δημιουργήθηκε.
- Ένα **signal δεν** περιέχει (επιπλέον) πληροφορία.
- `#include <signal.h>` (Ορισμός των ονομάτων των signals)
- **GNU C Library**
  http://www.gnu.org/software/libc/manual/html_node

PROGRAM ERROR SIGNALS

Τα ακόλουθα **signals** παράγονται όταν ένα **σοβαρό σφάλμα** προγράμματος ανιχνεύεται από το λειτουργικό σύστημα. Σε γενικές γραμμές, αυτά τα signals είναι ενδείξεις ότι το πρόγραμμά έχει να σταματήσει να λειτουργεί, και δεν υπάρχει συνήθως κανένας τρόπος να συνεχίσει την εκτέλεσή του.

- **SIGFPE:** εσφαλμένη αριθμητική λειτουργία.
- **SIGILL:** άκυρη εντόλη.
- **SIGSEGV:** προσπάθεια για διάβασμα/γράψιμο εκτός μνήμης προγράμματος.
- **SIGABRT:** σφάλμα που ανιχνεύεται από το ίδιο το πρόγραμμα και αναφέρεται καλώντας την **abort()**.

TERMINATION SIGNALS

Τα ακόλουθα **signals** χρησιμοποιούνται για να τον **τερματισμό** ενός **process**. Έχουν διαφορετικά ονόματα, επειδή χρησιμοποιούνται για ελαφρώς διαφορετικούς σκοπούς.

- **SIGTERM:** γενικό signal για τον τερματισμό ενός process. Μπορεί να γίνει **blocked**, **handled** και **ignored**.
- **SIGINT:** αποστολή signal τερματισμού ενός process από το πληκτρολόγιο (**Ctrl+C**).
- **SIGQUIT:** παρόμοια λειτουργία με το SIGINT, εκτός από το τρόπο αποστολής (**Ctrl+**). Επίσης παράγει core dump όταν τερματίζει το process.
- **SIGKILL:** προκαλεί **άμεσο** τερματισμό ενός process. **Δεν** μπορεί να γίνει **blocked**, **handled** ή **ignored**.

PROCESS CONTROL SIGNALS

Τα ακόλουθα **signals** χρησιμοποιούνται για να τη **διαχείριση** ενός **process**.

- **SIGCHLD:** αυτό το signal αποστέλλεται σε parent process όταν ένα από τα child processes του **τερματίζει** ή **σταματά**.
- **SIGCONT:** αυτό το signal αποστέλλεται σε process για να **συνεχίσει** την εκτέλεσή του.
- **SIGSTOP:** αυτό το signal αποστέλλεται σε process για να **σταματήσει** την εκτέλεσή του.
- **SIGTSTP:** αυτό το signal αποστέλλεται από το πληκτρολόγιο (**Ctrl+Z**) σε process για να **σταματήσει** την εκτέλεσή του.

USER-DEFINE SIGNALS

Αυτά τα **signals** χρησιμοποιούνται για διάφορους άλλους σκοπούς. Σε γενικές γραμμές, δεν θα επηρεάσουν το πρόγραμμα αν δεν ορίσουμε ρητά την ενέργεια που θα κάνουν.

- **SIGUSR1:** προορίζεται για να το χρησιμοποιήσετε με όποιον τρόπο θέλετε. Είναι χρήσιμο για την απλή επικοινωνία μεταξύ διεργασιών, αν γραφεί ένας handler για τη διαχείριση αυτού του signal.

- **SIGUSR2:** λειτουργεί όπως και το SIGUSR1.

ΣΥΓΧΡΟΝΑ-ΑΣΥΓΧΡΟΝΑ SIGNALS

- **Σύχρονα signals:** προκύπτουν ως αποτέλεσμα μιας ενέργειας που που επιχείρησε το ίδιο το process (*SIGSEGV*, *SIGILL*, ...).

- **Ασύχρονα signals:** προκύπτουν χωρίς να τα έχει προκαλέσει το ίδιο το process (*SIGINT*, *SIGQUIT*, *SIGCONT*, ...).

ΔΙΑΚΟΠΗ ΕΚΤΕΛΕΣΗΣ ΕΝΟΣ PROCESS

- Τα **signals** διακόπτουν την εκτέλεση ενός process.
- Τα **σύγχρονα signals** διακόπτουν την εκτέλεση **ακριβώς** στο σημείο του προγράμματος όπου εκτελέστηκε η εντολή που προκάλεσε το signal.
- Τα **ασύγχρονα signals** μπορούν να διακόψουν την εκτέλεση **ανά πάσα στιγμή**.
- **Δε** γνωρίζουμε το **πόσες φορές** ένα process θα λάβει ένα ασύγχρονο signal, **ούτε** γνωρίζουμε **το σημείο** της εκτέλεσης που βρίσκεται ένα process όταν λάβει το ασύγχρονο σήμα.

SIGNAL HANDLERS

Το λειτουργικό σύστημα ορίζει **αυτόματες/προεπιλεγμένες** ενέργειες χειρισμού για κάθε signal.

- **παράβλεψη** του signal (π.χ. *SIGCHLD*).
- **αναστολή** εκτέλεσης του process (π.χ. *SIGSTOP*).
- **συνέχεια** εκτέλεσης του process (π.χ. *SIGCONT*).
- **τερματισμός** του process (π.χ. *SIGTERM*).

SYSTEM CALL SIGACTION() (1/2)

- #include <signal.h> (Δήλωση του **sigaction()**)
- Prototype : int **sigaction**(int **signum**, const struct **sigaction** ***act**, struct **sigaction** ***oldact**)
- Το **sigaction()** χρησιμοποιείται για την αλλαγή των ενεργειών του signal **signum**.
- Το **signum** προσδιορίζει οποιοδήποτε έγκυρο signal εκτός των **SIGKILL** και **SIGSTOP**.
- Οι νέες ενέργειες προσδιορίζονται στη δομή **act**.
- Οι παλιές ενέργειες αποθηκεύονται στη δομή **oldact** (για να μπορεί να γίνει επαναφορά της προηγούμενης κατάστασης).

SYSTEM CALL SIGACTION() (2/2)

```
1 struct sigaction {
2   void     (*sa_handler)(int);
3   void     (*sa_sigaction)(int, siginfo_t *, void *);
4   sigset_t   sa_mask;
5   int        sa_flags;
6   void     (*sa_restorer)(void);
7 };
```

- Στο **sa_handler** ανατίθεται ο **handler** του signal και μπορεί να είναι:
  - SIG_IGN
  - SIG_DFL
  - δείκτης σε συνάρτηση
- Το **sa_mask** καθορίζει τα signals που **μπλοκάρονται** όταν εκτελείται ο signal handler.
- Το **sa_flags** καθορίζει επιπλέον ιδιότητες.

CODE 1

```
1  int main (int argc, char *argv[])
2  {
3      struct sigaction act = { {0} };
4
5      act.sa_handler = SIG_IGN; // ignore signal
6      sigaction(SIGINT, &act, NULL);
7
8      while (1) {
9          printf("sleep for 5 secs ...\ n");
10         sleep(5);
11     }
12
13     return 0;
14 }
```

CODE 2

```
1  static void handler(int sig)
2  {
3      printf("\nCaught signal: %d\n", sig);
4  }
5
6  int main (int argc, char *argv[])
7  {
8      struct sigaction act = { {0} };
9
10     act.sa_handler = &handler;
11     sigaction(SIGINT, &act, NULL);
12
13     while (1) {
14         printf("sleep for 5 secs ...\ n");
15         sleep(5);
16     }
17
18     return 0;
19 }
```

CODE 3

```
1  static void handler(int sig)
2  {
3      printf("\nCaught signal: %s\n", strsignal(sig));
4  }
5
6  int main (int argc, char *argv[])
7  {
8      struct sigaction act = { {0} };
9
10     act.sa_handler = &handler;
11     sigaction(SIGINT, &act, NULL);
12     sigaction(SIGQUIT, &act, NULL);
13
14     while (1) {
15         printf("sleep for 5 secs ...\ n");
16         sleep(5);
17     }
18
19     return 0;
20 }
```

SIGNAL HANDLER

- Ο handler ενός ασύγχρονου signal μπορεί να εκτελεστεί ανά πάσα στιγμή, **εμβόλιμα** στον κώδικα του προγράμματος.

- Ένας signal handler μπορεί να διακοπεί με παρόμοιο τρόπο για να εκτελεστεί κάποιος άλλος signal handler.

- Αν ένας signal handler χρησιμοποιεί καθολικές μεταβλητές, τότε αυτές θα πρέπει να δηλώνονται ως **volatile sig_atomic_t** διαφορετικά μπορεί να προκύψουν προβλήματα ασυνέπειας.

CODE 4

```
1  volatile sig_atomic_t counter = 1;
2
3  static void handler(int sig)
4  {
5      printf("\nCaught signal(%d) %d times.\n", sig, counter);
6      counter++;
7  }
8
9  int main (int argc, char *argv[])
10 {
11     struct sigaction act = { {0} };
12
13     act.sa_handler = &handler;
14     sigaction(SIGINT, &act, NULL);
15
16     while (1) {}
17
18     return 0;
19 }
```

KILL() FUNCTION (1/2)

- #include <signal.h> (Δήλωση της συνάρτησης **kill()**)
- #include <sys/types.h> (Δήλωση του τύπου δεδομένων pid_t)
- Prototype : int kill(pid_t pid, int sig)
- **Αποστολή** ενός **signal** σε ένα **process** ή σε group από process.
- Επιστρέφει **0** αν εκτελεστεί επιτυχώς, αλλιώς επιστρέφει **-1**.

KILL() FUNCTION (2/2)

- Το **pid** μπορεί να πάρει τις παρακάτω τιμές :
  - **< -1 :** η kill() στέλνει το signal **sig** σε όλα τα processes των οποίων το **process group id** είναι ίδιο με το |**pid**|.
  - **-1 :** η kill() στέλνει το signal **sig** σε **όλα** processes.
  - **0 :** η kill() στέλνει το signal **sig** σε όλα τα processes των οποίων το **process group id** είναι ίδιο με αυτό του process που την κάλεσε.
  - **> 0 :** η kill() στέλνει το signal **sig** στο process με **process id** είναι ίδιο με το **pid**.

CODE 5

```
1  int main (int argc, char *argv[])
2  {
3      int count;
4
5      count = 3;
6      while (count > 0) {
7          printf("exit in %d sec(s)\n", count);
8          sleep(1);
9          count--;
10     }
11
12     printf("message before kill()\n");
13     kill (getpid(), SIGINT);
14     printf("message after kill ()\n");
15
16     return 0;
17 }
```

ΑΣΚΗΣΕΙΣ

ΑΣΚΗΣΕΙΣ (1/2)

1. Να γραφεί πρόγραμμα σε C που θα τρέχει συνεχώς και θα αγνοεί το signal **SIGTSTP**.

2. Να γραφεί πρόγραμμα σε C που θα τρέχει συνεχώς, θα αγνοεί το signal **SIGTSTP** και θα τυπώνει "**Received SIGTSTP**".

3. Να γραφεί πρόγραμμα σε C που θα τρέχει συνεχώς. Όποτε δέχεται ένα signal **SIGINT** θα αυξάνει έναν counter. Όταν ο counter πάρει την τιμή 5 θα τερματίζεται το πρόγραμμα.

ΑΣΚΗΣΕΙΣ (2/2)

4. Να γραφεί πρόγραμμα σε C που θα δημιουργεί ένα νέο **process**. Το process θα τρέχει συνεχώς και τυπώνει κάθε 2 δευτερόλεπτα "**Luke, I Am Your Father**"". Θα πρέπει να υλοποιήσετε έναν **signal handler** που θα κάνει **pause** και **continue** το process που δημιουργήθηκε. Θα πρέπει να τυπώνονται κατάλληλα μηνύματα κατά την αλλαγή κατάστασης του process.