Σήματα
(Signals)

Τι είναι ένα σήμα;

- Το σήμα είναι μια **ειδοποίηση** που στέλνεται στην διεργασία που εκτελεί ένα πρόγραμμα
- Μπορεί να προκύψει για διάφορους λόγους, π.χ. εξαίρεση αριθμητικής πράξης, τερματισμός διεργασίας παιδιού, αποστολή σήματος από άλλη διεργασία ...

- Τα σήματα έχουν μόνο έναν **κωδικό/τύπο** που φανερώνει τον λόγο για τον οποίο δημιουργήθηκαν
- Ένα σήμα **δεν** περιέχει (επιπλέον) πληροφορία
  - είναι μήνυμα με τύπο αλλά χωρίς περιεχόμενο

- Για κάθε σήμα υπάρχει **προκαθορισμένος** αυτόματος χειρισμός από το λειτουργικό σύστημα

Σήματα σφάλματος

- `SIGFPE`: εσφαλμένη αριθμητική λειτουργία (Ε)
- `SIGILL`: άκυρη εντολή (Ε)
- `SIGSYS`: άκυρη κλήση συστήματος (Ε)
- `SIGBUS`: άκυρη διεύθυνση σε επίπεδο υλικού (Ε)
- `SIGSEGV`: άκυρη αναφορά μνήμης (Ε)
- `SIGPIPE`: γράψιμο σε αγωγό χωρίς αναγνώστη (Τ)

Αυτόματες/προεπιλεγμένες ενέργειες (αν δεν γίνει χειρισμός από το πρόγραμμα):
Π: παράβλεψη
Τ: τερματισμός
Ε: Τ + άλλες ενέργειες, π.χ., δημιουργία core dump
Α: αναστολή/σταμάτημα εκτέλεσης
Σ: συνέχιση εκτέλεσης (μετά από αναστολή)

Σήματα ελέγχου κατάστασης διεργασίας

- `SIGCHLD`: αλλαγή κατάστασης παιδιού (Π)
- `SIGSTOP`: αναστολή εκτέλεσης διεργασίας (Α)
- `SIGCONT`: συνέχιση εκτέλεσης διεργασίας (Σ)
- `SIGTSTP`: αναστολή εκτέλεση από τερματικό (Α)
- `SIGTTIN`: ανάγνωση από το παρασκήνιο (Α)
- `SIGTTOU`: γράψιμο από το παρασκήνιο (Α)

Αυτόματες/προεπιλεγμένες ενέργειες (αν δεν γίνει χειρισμός από το πρόγραμμα):
Π: παράβλεψη
Τ: τερματισμός
Ε: Τ + άλλες ενέργειες, π.χ., δημιουργία core dump
Α: αναστολή εκτέλεσης
Σ: συνέχιση εκτέλεσης (μετά από αναστολή)

Σήματα διακοπής/τερματισμού κλπ

- `SIGINT`: διακοπή από το πληκτρολόγιο (Τ)
- `SIGQUIT`: εγκατάλειψη από το πληκτρολόγιο (Ε)
- `SIGHUP`: κλείσιμο γραμμής/τερματικού (Τ)
- `SIGTERM`: τερματισμός – συνθετικά (Τ)
- `SIGKILL`: θανάτωση – συνθετικά (Τ)
- `SIGUSR1/2`: σήματα εφαρμογής (Τ)

Αυτόματες/προεπιλεγμένες ενέργειες (αν δεν γίνει χειρισμός από το πρόγραμμα):
Π: παράβλεψη
Τ: τερματισμός
Ε: Τ + άλλες ενέργειες, π.χ., δημιουργία core dump
Α: αναστολή εκτέλεσης
Σ: συνέχιση εκτέλεσης (μετά από αναστολή)

Σύγχρονα και ασύγχρονα σήματα

- **Σύγχρονα** σήματα: προκύπτουν ως αποτέλεσμα μιας ενέργειας που επιχείρησε η ίδια η διεργασία (εκτέλεση εντολής ή κλήσης συστήματος)
- Π.χ., εξαίρεση αριθμητικής πράξης (`SIGFPE`), αναφορά σε άκυρη θέση μνήμης (`SIGSEGV`), άκυρη εντολή μηχανής (`SIGILL`) ...

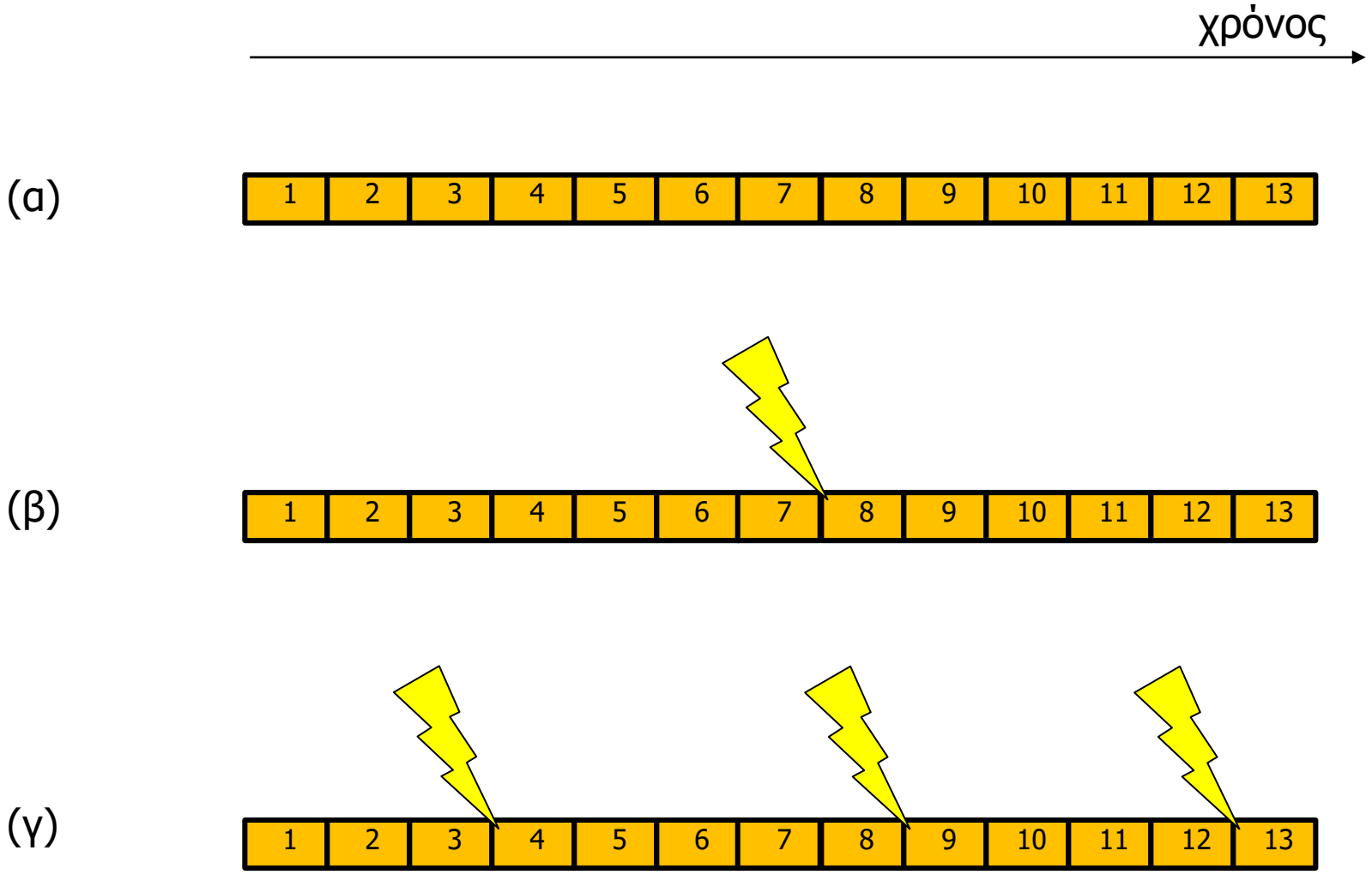
- **Ασύγχρονα** σήματα: προκύπτουν χωρίς να τα έχει προκαλέσει η ίδια η διεργασία με άμεσο τρόπο
- Π.χ., διακοπή εκτέλεσης (`SIGINT`), τερματισμός ή αλλαγή κατάστασης διεργασίας παιδιού (`SIGCHLD`), αναστολή διεργασίας (`SIGSTOP`) ...

Διακοπή ροής εκτέλεσης διεργασίας

- Τα σήματα **διακόπτουν** την κανονική ροή εκτέλεσης της διεργασίας

- Τα σύγχρονα σήματα διακόπτουν την εκτέλεση στο σημείο της εντολής / κλήσης συστήματος που προκάλεσε την δημιουργία του σήματος

- Τα ασύγχρονα σήματα μπορεί να διακόψουν την εκτέλεση **ανά πάσα στιγμή**
  - **δεν** γνωρίζουμε **πόσες φορές** μια διεργασία θα λάβει ένα ασύγχρονο σήμα, **ούτε το σημείο** της εκτέλεσης όπου θα βρίσκεται όταν λάβει το σήμα


χρόνος
(α)
1 2 3 4 5 6 7 8 9 10 11 12 13
(β)
1 2 3 4 5 6 7 8 9 10 11 12 13
(γ)
1 2 3 4 5 6 7 8 9 10 11 12 13

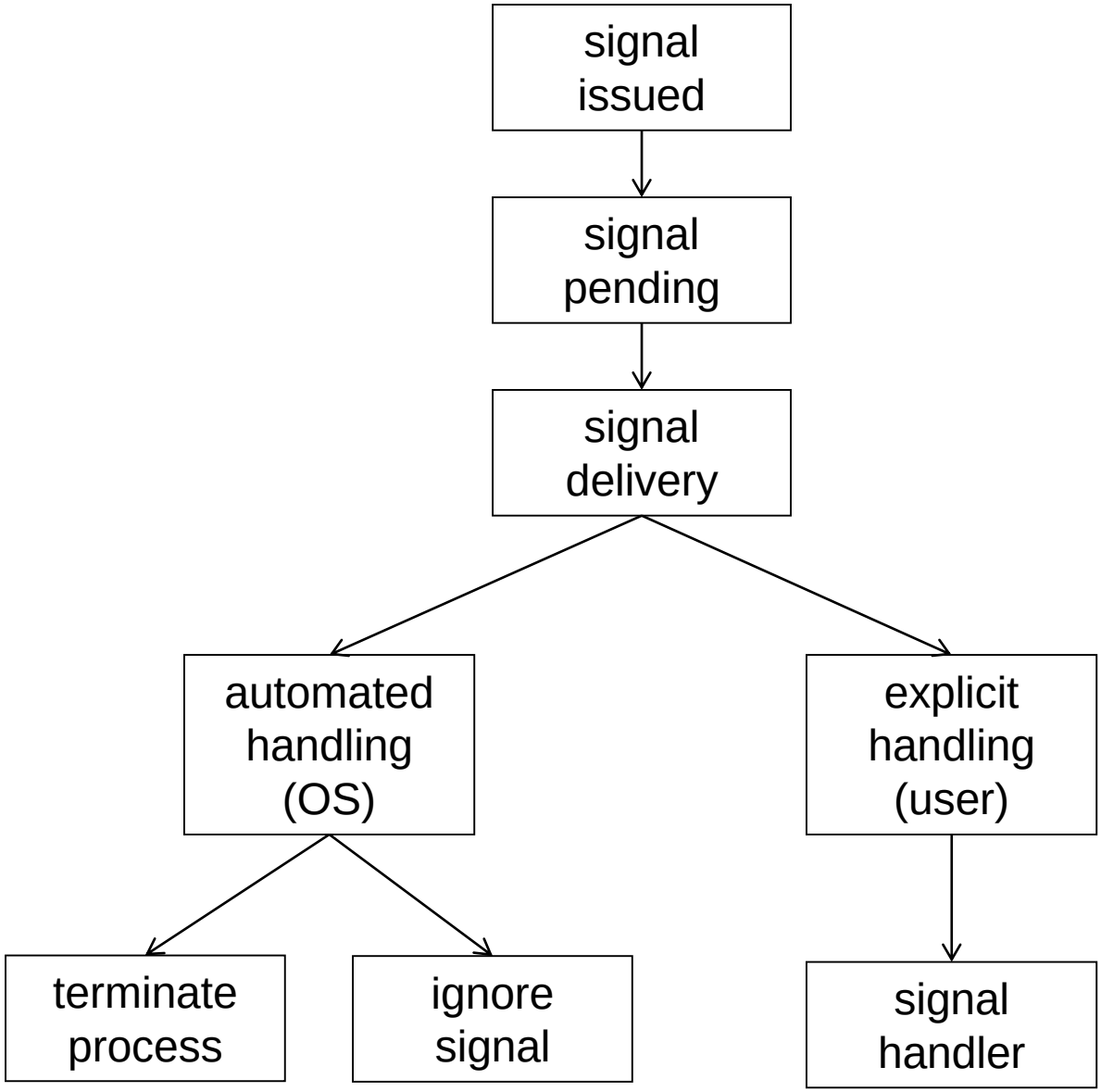
Κύκλος ζωής ενός σήματος

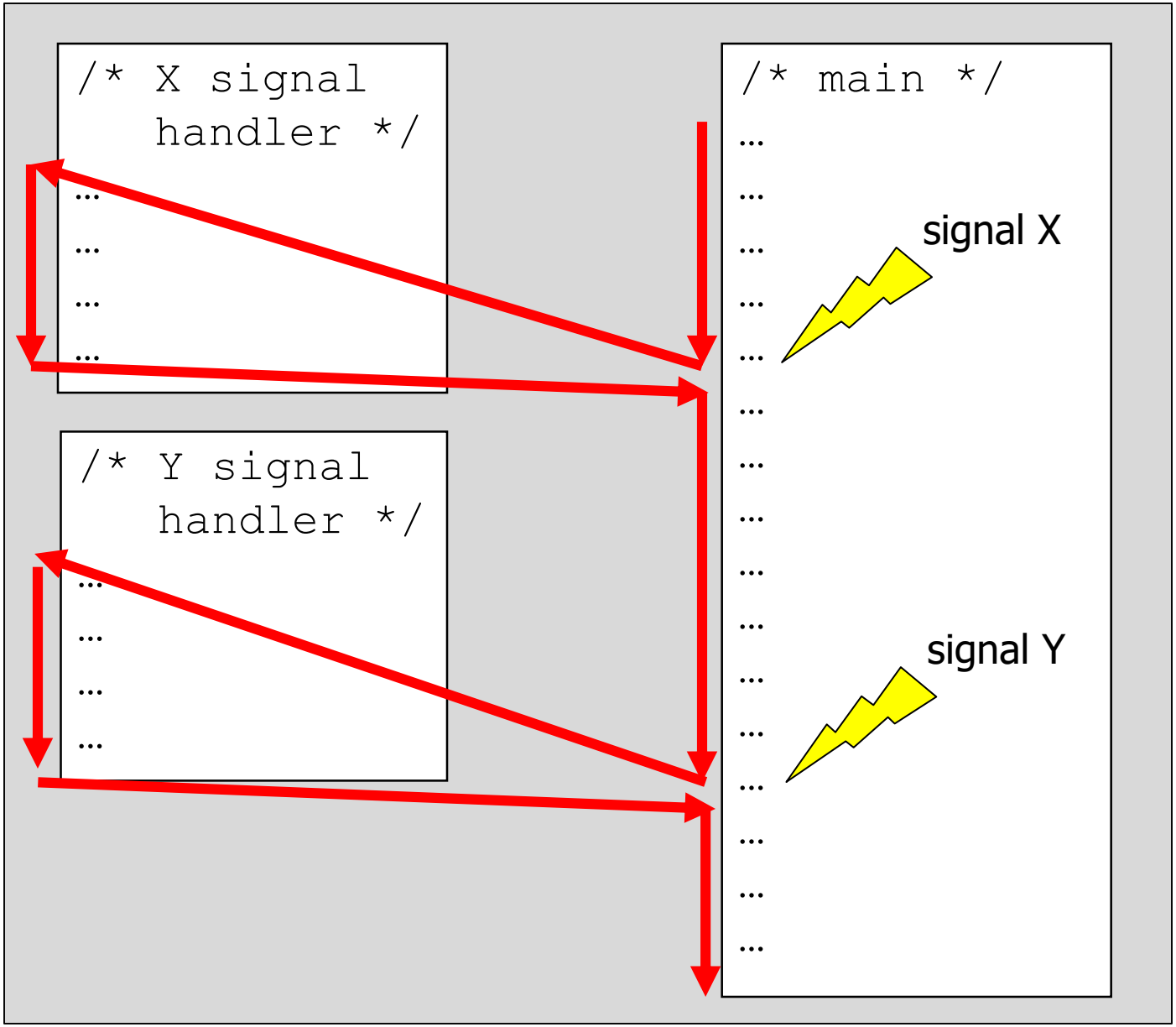
- **Γέννηση:** δημιουργία/αποστολή σήματος
- **Μπλοκάρισμα:** αναστολή παράδοσης/χειρισμού του σήματος
- **Παράδοση/παραλαβή:** χρονική στιγμή που γίνεται επεξεργασία του σήματος στο πλαίσιο της διεργασίας
- **Χειρισμός:** διαδικασία επεξεργασίας του σήματος

Χειρισμός σήματος

- Το ΛΣ ορίζει **αυτόματες προκαθορισμένες** ενέργειες χειρισμού για κάθε τύπο σήματος
  - **παράβλεψη** σήματος (π.χ., για `SIGCHLD`)
  - **αναστολή** εκτέλεσης διεργασίας (π.χ., για `SIGSTOP`)
  - **συνέχεια** εκτέλεσης διεργασίας (π.χ., για `SIGCONT`)
  - **τερματισμός** διεργασίας (π.χ., για `SIGINT`)

- Ένα πρόγραμμα μπορεί να **«παγιδέψει»** ένα σήμα, και να εισάγει **δικές του ενέργειες χειρισμού**
  - αυτόματος προκαθορισμένος χειρισμός σήματος (`SIG_DFL`)
  - αυτόματη παράβλεψη σήματος (`SIG_IGN`)
  - ρητός χειρισμός σήματος από ρουτίνα εφαρμογής


signal issued
signal pending
signal delivery
automated handling (OS)
explicit handling (user)
terminate process
ignore signal
signal handler


/* X signal
handler */
/* Y signal
handler */
/* main */
signal X
signal Y

Καθορισμός ενεργειών χειρισμού σήματος

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

- Καθορίζει ενέργειες χειρισμού για το σήμα `signum`
- Οι **νέες** ενέργειες προσδιορίζονται στην δομή `act`
- Οι **παλιές** ενέργειες χειρισμού αποθηκεύονται στην δομή `oact` (για μπορεί να γίνει επαναφορά της προηγούμενης κατάστασης)
- Μπορεί να καθοριστούν διαφορετικές ενέργειες για κάθε ξεχωριστό σήμα
- Οι ενέργειες για `SIGKILL` και `SIGSTOP` **δεν** αλλάζουν

Δομή ενεργειών χειρισμού σήματος

```
struct sigaction {
        void (*sa_handler)(int);
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_sigaction)(int,siginfo_t*,void *);
};
```

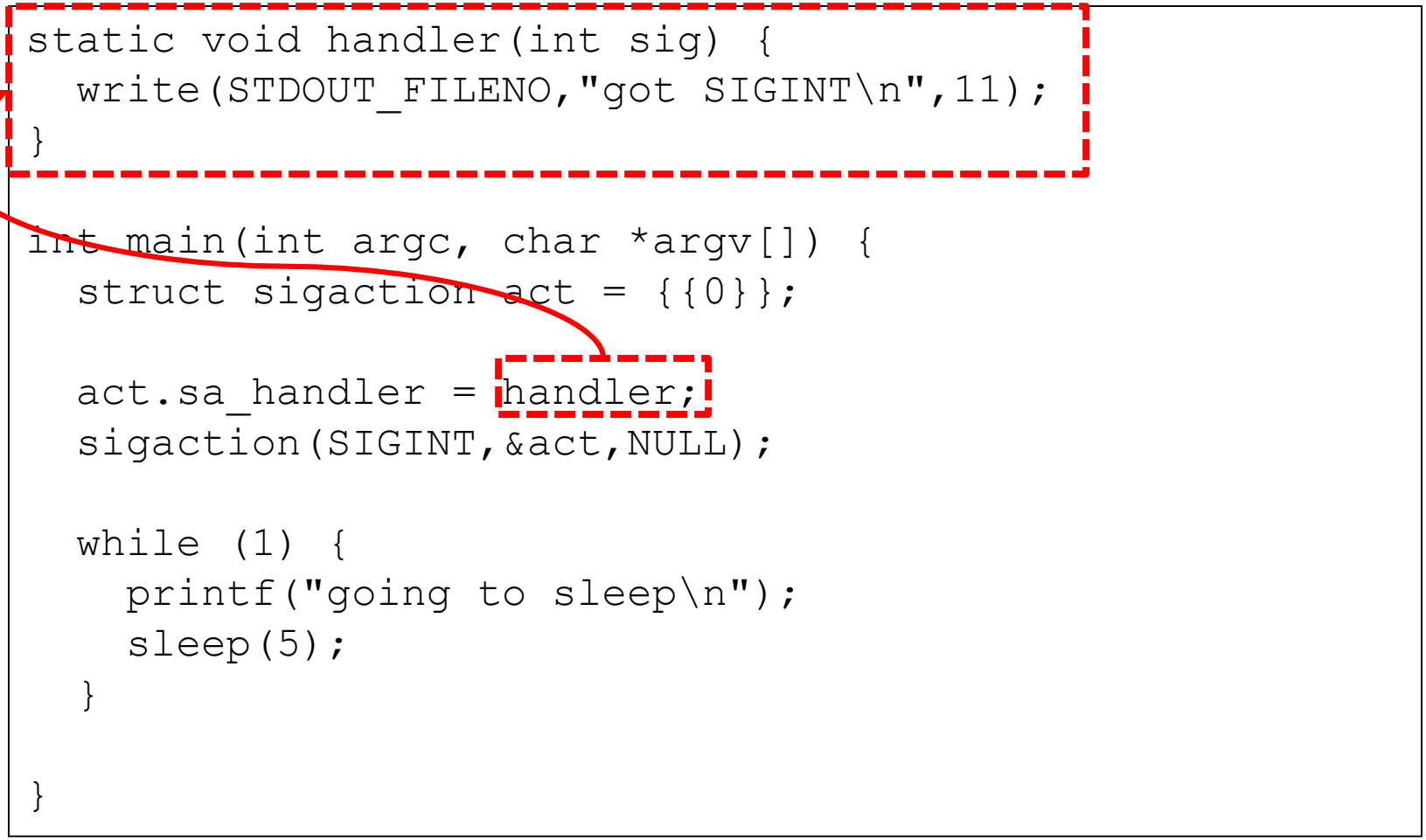
- Στο `sa_handler` ανατίθεται ο **χειριστής σήματος** (`SIG_IGN` ή `SIG_DFL` ή δείκτης σε συνάρτηση)
  - το `sa_sigaction` είναι για σήματα πραγματικού χρόνου
- Το `sa_mask` καθορίζει τα σήματα που **μπλοκάρονται** όταν εκτελείται ο χειριστής σήματος (πλέον του σήματος που προκάλεσε τον χειρισμό του σήματος, και όσων άλλων σημάτων είναι ήδη μπλοκαρισμένα)
- Το `sa_flags` καθορίζει επιπλέον ιδιότητες
  - π.χ. `SA_RESTART` για επανεκτέλεση κλήσεων συστήματος

```
int main(int argc, char *argv[]) {
  struct sigaction act = {{0}};

  act.sa_handler = SIG_IGN; // ignore signal
  sigaction(SIGINT,&act,NULL);

  while (1) {
    printf("going to sleep\n");
    sleep(5);
  }

}
```


static void handler(int sig) {
write(STDOUT_FILENO,"got SIGINT\n",11);
}
int main(int argc, char *argv[]) {
struct sigaction act = {{0}};
act.sa_handler = handler;
sigaction(SIGINT,&act,NULL);
while (1) {
printf("going to sleep\n");
sleep(5);
}
}

Αναμονή παραλαβής/χειρισμού σήματος

```
int sleep(int secs);
```

- Μπλοκάρει μέχρι να γίνει χειρισμός ενός σήματος ή μετά την πάροδο του χρονικού διαστήματος `secs`
  - επιστρέφει 0 αν η διεργασία δεν διακόπηκε από σήμα
  - διαφορετικά, το χρονικό διάστημα που απέμεινε

```
int pause();
```

- Μπλοκάρει μέχρι να γίνει χειρισμός ενός σήματος
  - επιστρέφει `-1` με την `errno` ίση με `EINTR`

- Οι `pause` και `sleep` επιστρέφουν **μόνο** για σήματα που χειρίζεται ρητά το πρόγραμμα

```
static void handler(int sig) {
  write(STDOUT_FILENO,"got SIGINT\n",11);
}

int main(int argc, char *argv[]) {
  struct sigaction act = {{0}};

  act.sa_handler = handler;
  sigaction(SIGINT,&act,NULL);

  while (1) {
    printf("going to pause\n");
    pause();
  }

}
```

Διακοπή κλήσεων συστήματος

- Η παράδοση και χειρισμός ενός ασύγχρονου σήματος μπορεί να **διακόψει** κάποιες **κλήσεις συστήματος**
- Συνήθως, αυτές που **μπλοκάρουν** την διεργασία

- Εφόσον γίνει ρητός χειρισμός του σήματος από το πρόγραμμα, η κλήση συστήματος που διακόπηκε **αποτυγχάνει** με την `errno` να έχει τιμή `EINTR`
  - όπως ακριβώς και η `pause`

- Αν αυτό δεν είναι επιθυμητό, χρησιμοποιούμε την `siginterrupt` ή την επιλογή `SA_RESTART` ...

```
static void handler(int sig) {
  write(STDOUT_FILENO,"got SIGINT\n",11);
}

int main(int argc, char *argv[]) {
  int n; char data[N];
  struct sigaction act = {{0}};

  act.sa_handler = handler;
  act.sa_flags = SA_RESTART;
  sigaction(SIGINT,&act,NULL);

  while (1) {
    printf("reading from stdin\n");
    n = read(STDIN_FILENO,data,N-1);
    if (n > 0) { datas[n] = '\0'; printf("%s\n",data); }
    else if (n < 0) { perror("read"); }
    else { printf("end of input\n"); break; }
  }
  return(0);
}
```

Σχεδιασμός χειριστών σημάτων

- Ο χειριστής ενός ασύγχρονου σήματος μπορεί να εκτελεστεί ανά πάσα στιγμή, **διακόπτοντας** την κανονική ροή εκτέλεσης του προγράμματος
- Ένας χειριστής σήματος μπορεί να διακοπεί με παρόμοιο τρόπο, για να εκτελεστεί ένας άλλος χειριστής σήματος
- Υπάρχουν (σχετικά λίγες) συγκεκριμένες λειτουργίες που είναι **ασφαλείς** για (ασύγχρονα) σήματα
  - μπορούν να χρησιμοποιηθούν μέσα σε χειριστές σημάτων
- Αν ένας χειριστής σήματος χρησιμοποιεί καθολικές μεταβλητές, τότε αυτές πρέπει να δηλώνονται ως `volatile sig_atomic_t`
  - διαφορετικά μπορεί να προκύψουν προβλήματα ασυνέπειας

Μπλοκάρισμα σημάτων

- Η παράδοση και ο αντίστοιχος χειρισμός σημάτων μπορεί να είναι ανεπιθύμητα σε κάποιες περιπτώσεις
- Τα σήματα μπορεί να **μπλοκαριστούν**
- Μέσω της **μάσκας μπλοκαρίσματος**

- Τα μπλοκαρισμένα σήματα **δεν παραδίδονται** στην διεργασία για χειρισμό, αλλά παραμένουν σε **εκκρεμότητα** μέχρι να ξεμπλοκαριστούν
- Ο χειρισμός ενός σήματος που βρίσκεται σε εκκρεμότητα γίνεται **όταν** αυτό ξεμπλοκαριστεί

- Τα `SIGKILL` και `SIGSTOP` **δεν** μπλοκάρονται

Λειτουργίες συνόλων σημάτων

- `sigset_t`
  - σύνολο σημάτων
- `int sigemptyset(sigset_t *set)`
  - αρχικοποίηση κενού συνόλου
- `int sigfillset(sigset_t *set)`
  - αρχικοποίηση γεμάτου συνόλου
- `int sigaddset(sigset_t *set, int sig)`
  - προσθήκη σήματος στο σύνολο
- `int sigdelset(sigset_t *set, int sig)`
  - αφαίρεση σήματος από το σύνολο
- `int sigismember(const sigset_t *set,int sig)`
  - έλεγχος για το αν το σήμα ανήκει στο σύνολο

Καθορισμός μάσκας μπλοκαρίσματος

```
int sigprocmask(int how,
                const sigset_t *set,
                sigset_t *oset);
```

- Η `how` προσδιορίζει τον τρόπο αλλαγής της μάσκας μπλοκαρίσματος, με βάση το σύνολο σημάτων `set`
  - `SIG_BLOCK`: μπλοκάρισμα των σημάτων του `set`
  - `SIG_UNBLOCK`: ξεμπλοκάρισμα των σημάτων του `set`
  - `SIG_SETMASK`: αντικατάσταση της μάσκας με το `set`
- Στο `oset` αποθηκεύεται η παλιά τιμή της μάσκας μπλοκαρίσματος – για να μπορεί να αποκατασταθεί
- Σε περίπτωση που ξεμπλοκαριστεί ένα σήμα που εκκρεμεί, τότε γίνεται και ο **χειρισμός** του σήματος

Εξέταση εκκρεμών σημάτων

```
int sigpending(sigset_t *set);
```

- Αποθηκεύει στο `set` τα σήματα που έχουν **σταλεί** στην διεργασία και βρίσκονται σε **εκκρεμότητα**
- Επιστρέφεται πληροφορία για το **αν** ένα σήμα εκκρεμεί, **όχι** για το **πόσες** φορές αυτό έχει σταλεί
- Το λειτουργικό **δεν** καταγράφει το πόσες φορές έχει σταλεί ένα μπλοκαρισμένο σήμα, **ούτε** γίνεται επανειλημμένος χειρισμός του σήματος όταν αυτό ξεμπλοκαριστεί

```
static void handler(int sig) {
  write(STDOUT_FILENO,"got SIGINT\n",11);
}

int main(int argc, char *argv[]) {
  sigset_t s1,s2;
  struct sigaction act = {{0}};

  act.sa_handler = handler;
  sigaction(SIGINT,&act,NULL);

  sigemptyset(&s1); sigaddset(&s1,SIGINT);

  while (1) {
    sigprocmask(SIG_BLOCK,&s1,NULL);
    sleep(10);
    sigpending(&s2);
    if (sigismember(&s2,SIGINT)) {
      printf("SIGINT is pending\n");
    }
    sigprocmask(SIG_UNBLOCK,&s1,NULL);
    sleep(10);
  }
}
```

Μπλοκάρισμα και παράβλεψη σήματος

- Το μπλοκάρισμα ενός σήματος έχει νόημα μόνο αν το πρόγραμμα επιθυμεί «τελικά» να παραλάβει και να χειριστεί το σήμα (ξεμπλοκάροντας το)

- Αν η παραλαβή και ο χειρισμός ενός σήματος είναι γενικότερα **ανεπιθύμητα**, είναι συνήθως καλύτερο και απλούστερο το σήμα να παραβλεφθεί εντελώς
- Αυτό μπορεί να γίνει εγκαθιστώντας/καθορίζοντας τον ειδικό χειριστή σήματος `SIG_IGN`

Αλλαγή μάσκας και αναμονή σήματος

```
int sigsuspend(const sigset_t *sigmask);
```

- Αντικαθιστά την μάσκα μπλοκαρίσματος με το σύνολο σημάτων `sigmask` **και** μπλοκάρει μέχρι να παραληφθεί ένα **μη μπλοκαρισμένο** σήμα για το οποίο υπάρχει ρητός **χειρισμός** από το πρόγραμμα

- Όταν παραληφθεί ένα σήμα, αφού γίνει χειρισμός του σήματος, **αποκαθίσταται** η παλιά μάσκα, και η κλήση επιστρέφει με `-1` και `errno` ίση με `EINTR`

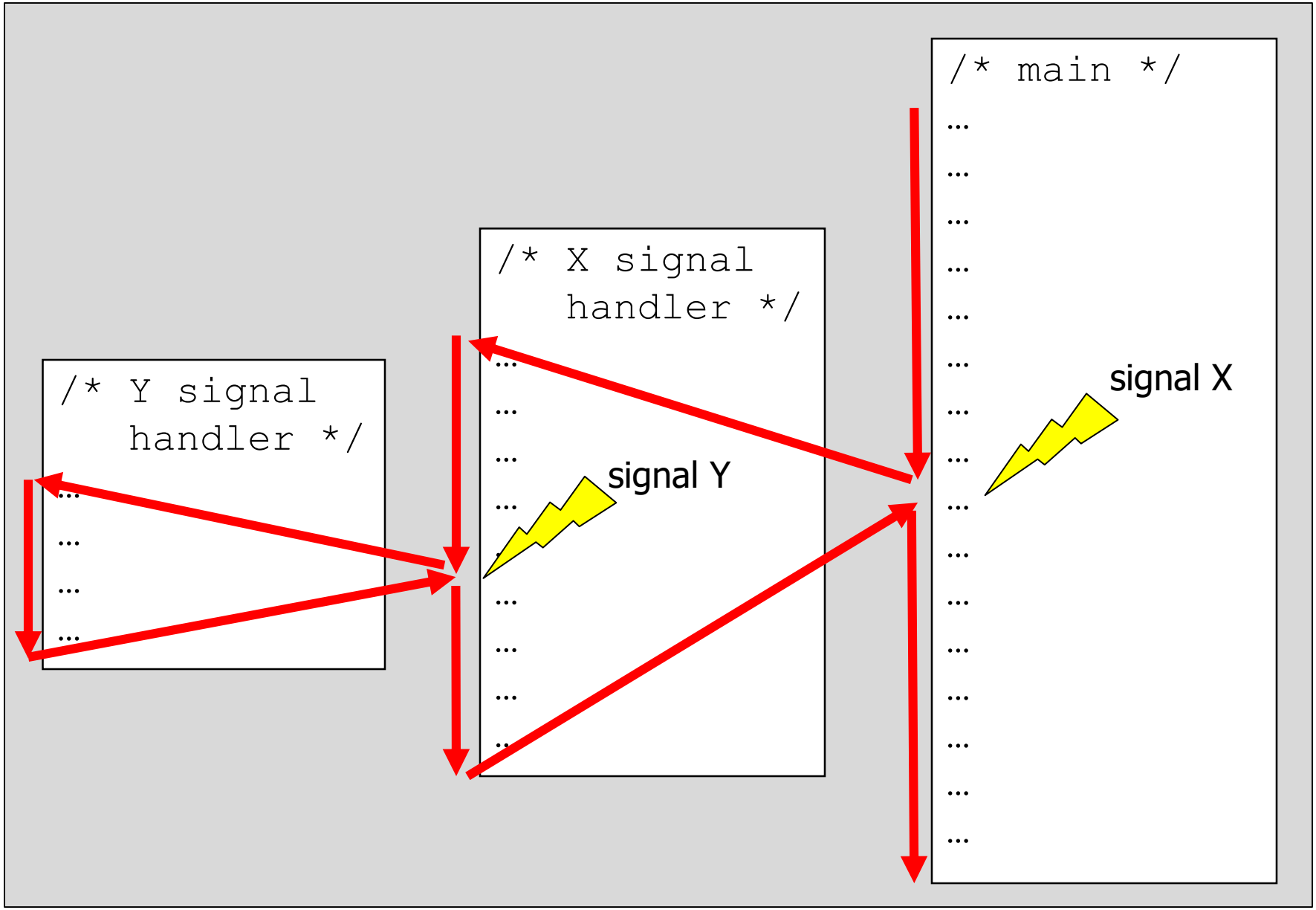
Αναμονή για εκκρεμή σήματα

```
int sigwait(const sigset_t *set,
            int *signum);
```

- Μπλοκάρει μέχρι κάποιο από τα σήματα στο `set` βρεθεί σε **εκκρεμότητα**
- Το εκκρεμές σήμα γίνεται **αποδεκτό** και αποθηκεύεται στην `signum`
- Παρότι **δεν** γίνεται χειρισμός του σήματος, το σήμα που επιστρέφεται θεωρείται **παραδομένο**
- Αν υπάρχουν περισσότερα εκκρεμή σήματα, επιστρέφεται ένα από αυτά (απροσδιόριστη επιλογή)
- Η `sigwait` επιτρέπει τον **ελεγχόμενο** χειρισμό σημάτων μέσα από τον **συμβατικό** κώδικα, χωρίς την εγκατάσταση χειριστών σημάτων

Διακοπή χειριστή σήματος

- Και ο χειριστής σήματος του προγράμματος είναι κώδικας η εκτέλεση του οποίου μπορεί να διακοπεί από σήματα – αν τύχει να προκύψουν κατά την εκτέλεση του χειριστή σήματος
- Αν δεν είναι επιθυμητό ο χειριστής σήματος να διακοπεί από τον χειρισμό ενός άλλου σήματος, το συγκεκριμένο σήμα πρέπει να μπλοκαριστεί κατά την εκτέλεση του χειριστή σήματος
  - βλέπε `sa_mask` της δομής χειρισμού σήματος
- Κατά σύμβαση, το σήμα που προκάλεσε τον χειρισμό σήματος παραμένει μπλοκαρισμένο κατά την εκτέλεση του αντίστοιχου χειριστή σήματος
  - και αυτό μπορεί να αλλάξει με κατάλληλα options


/* main */
signal X
/* X signal handler */
signal Y
/* Y signal handler */

Σήματα και `fork` / `exec`

- Η διεργασία παιδί που δημιουργείται μέσω `fork`
  - κληρονομεί **τους χειρισμούς σημάτων** του γονέα
  - κληρονομεί **την μάσκα μπλοκαρίσματος**
  - **δεν** κληρονομεί **τα εκκρεμή σήματα** του γονιού (αρχίζει την εκτέλεση της **χωρίς** εκκρεμή σήματα)

- Όταν μια διεργασία αλλάζει κώδικα μέσω `exec`
  - για τα σήματα που η καθορισμένη ενέργεια είναι `SIG_DFL` ή `SIG_IGN`, αυτή **παραμένει** (το `SIGCHLD` μπορεί να γίνει `SIG_DFL` από `SIG_IGN`, ανάλογα με την υλοποίηση)
  - τα σήματα με χειριστή **γυρίζουν** σε `SIG_DFL` – γιατί;
  - η μάσκα μπλοκαρίσματος **παραμένει** ως έχει
  - τα εκκρεμή σήματα **παραμένουν** ως έχουν

Παραγωγή/αποστολή σήματος

```
int kill(pid_t pid, int signum);
```

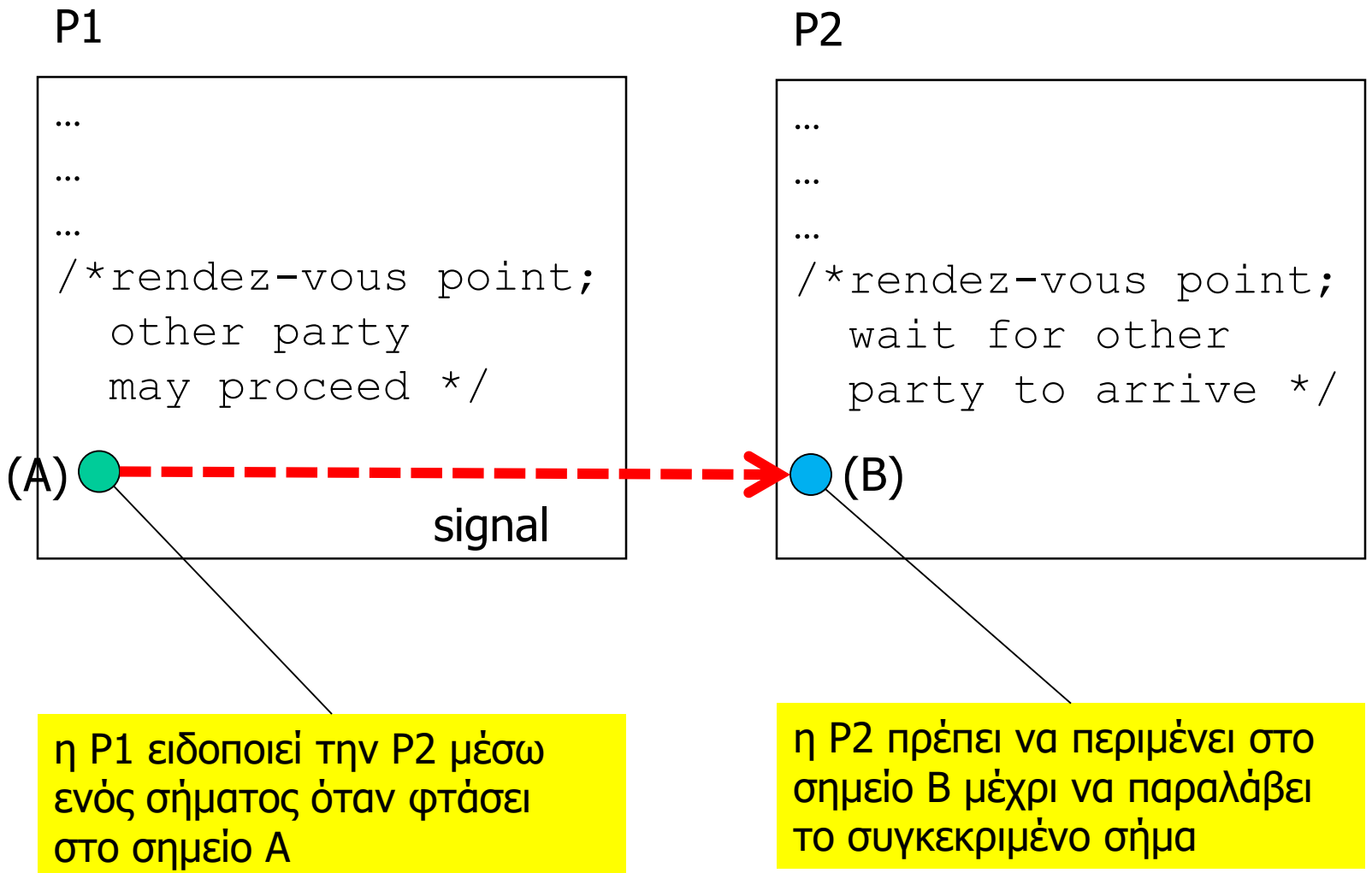
- `pid > 0`: στέλνει το σήμα `signum` στην διεργασία με αναγνωριστικό `pid`
- `pid == 0`: στέλνει το σήμα `signum` στην ομάδα διεργασιών το αναγνωριστικό της οποίας είναι το ίδιο με το αναγνωριστικό ομάδας του αποστολέα
- `pid <-1`: στέλνει το σήμα `signum` στην ομάδα διεργασιών το αναγνωριστικό της οποίας είναι ίσο με την απόλυτη τιμή του `pid`
- `pid == -1`: στέλνει το σήμα `signum` σε όλες τις διεργασίες για τις οποίες έχει άδεια ο αποστολέας

Συγχρονισμός διεργασιών με σήματα

- Τα σήματα μπορεί να χρησιμοποιηθούν για να υλοποιηθεί **συγχρονισμός** μεταξύ διεργασιών

- Παράδειγμα: ασύμμετρο ραντεβού
- Η διεργασία P2 πρέπει να **περιμένει** προτού συνεχίσει την εκτέλεση της, **μέχρι** η διεργασία P1 να ολοκληρώσει μια διαδικασία (τα αποτελέσματα της οποίας πιθανώς επηρεάζουν την διεργασία P2)

- **Υλοποίηση με σήματα:** η P1 στέλνει σήμα στην P2, και η P2 περιμένει να λάβει το σήμα για να συνεχίσει την εκτέλεση της


P1
P2
…
…
…
/*rendez-vous point;
other party
may proceed */
…
…
…
/*rendez-vous point;
wait for other
party to arrive */
(A)
(B)
signal
η P1 ειδοποιεί την P2 μέσω ενός σήματος όταν φτάσει στο σημείο Α
η P2 πρέπει να περιμένει στο σημείο Β μέχρι να παραλάβει το συγκεκριμένο σήμα

```
int main(int argc, char *argv[]) {
  pid_t pid; sigset_t s; int signum;

  sigemptyset(&s); sigaddset(&s,SIGUSR1);
  sigprocmask(SIG_BLOCK,&s,NULL);

  if (!(pid=fork())) {
    sigwait(&s,&signum); // wait for SIGUSR1
    printf("this should print second\n");
    return(0);
  }

  printf("this should print first\n");
  kill(pid,SIGUSR1);
  return(0);
}
```

Περισσότερα για την `waitpid`

- Η `waitpid` μπορεί να χρησιμοποιηθεί και για την **αναμονή για αλλαγή κατάστασης εκτέλεσης** διεργασιών – μέσα από αντίστοιχα `options`
- `WUNTRACED/WCONTINUED`: σταμάτημα/ξεκίνημα
- Αντίστοιχα, μπορεί να γίνει έλεγχος της κατάστασης που αποθηκεύεται στην διεύθυνση `status`
- `WIFSTOPPED()`: `true` αν η θυγατρική διεργασία ανέστειλε την εκτέλεση της (χωρίς να έχει πεθάνει)
- `WSTOPSIG()`: το σήμα που οδήγησε στην αναστολή της εκτέλεσης της θυγατρικής διεργασίας
- `WIFCONTINUED()`: `true` αν η θυγατρική διεργασία συνεχίζει την εκτέλεση της λόγω `SIGCONT`

```
int main(int argc, char *argv[]) {
  int pid,status,i;

  pid = fork();
  if (pid == 0) {
    for (i=0; i<20; i++) {
      printf("going to sleep\n");
      sleep(5);
    }
    return(42);
  }

  while (1) {
    waitpid(pid,&status,WUNTRACED|WCONTINUED);
    if (WIFEXITED(status)) {
      printf("returned %d\n",WEXITSTATUS(status)); break;
    }
    else if (WIFSIGNALED(status)) {
      printf("terminated %d\n",WTERMSIG(status)); break;
    }
    else if (WIFSTOPPED(status)) {
      printf("stopped %d\n",WSTOPSIG(status));
    }
    else if (WIFCONTINUED(status)) {
      printf("continued\n");
    }
  }
  return(0);
}
```

Ξυπνητήρια

```
int setitimer(int which,
              const struct itimerval *nval,
              struct itimerval *oval);
```

- `ITIMER_REAL`: μετράει τον πραγματικό χρόνο, στην λήξη στέλνεται το σήμα `SIGALRM`
- `ITIMER_VIRTUAL`: μετράει τον χρόνο εκτέλεσης της διεργασίας, στην λήξη στέλνεται `SIGVTALRM`
- `ITIMER_PROF`: μετράει τον χρόνο εκτέλεσης της διεργασίας και τον χρόνο εκτέλεσης του συστήματος για λογαριασμό της διεργασίας, στην λήξη στέλνεται το σήμα `SIGPROF`
- Στο `oval` επιστρέφονται οι τρέχουσες ρυθμίσεις

Δομή χρονισμού ξυπνητηριού

```
struct itimerval {
    struct timeval it_interval; /*nxt val*/
    struct timeval it_value;    /*cur val*/
};
struct timeval {
    time_t tv_sec;         /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
```

- Το πεδίο `it_value` καθορίζει/επιστρέφει τον χρόνο που απομένει, και το πεδίο `it_interval` την τιμή επαναφοράς για τις επόμενες επαναλήψεις

```
static void handler(int sig) {
  write(STDOUT_FILENO,"got SIGALRM\n",12);
}

int main(int argc, char *argv[]) {
  int n; char data[N];
  struct sigaction act = { {0} };
  struct itimerval t = { {0} };

  act.sa_handler = handler;
  sigaction(SIGALRM,&act,NULL);

  t.it_value.tv_sec = atoi(argv[1]);
  t.it_value.tv_usec = 0;
  t.it_interval.tv_sec = atoi(argv[2]);
  t.it_interval.tv_usec=0;

  setitimer(ITIMER_REAL,&t,NULL);

  while (1) {
    n = read(STDIN_FILENO,data,N-1);
    if (n > 0) { data[n] = '\0'; printf("read: %s\n",data); }
    else if (n == 0) { printf("end of input\n"); break; }
    else if (errno != EINTR) { perror("read"); break; }
    else { printf("interrupted, retrying\n"); }
  }
  return(0);
}
```