




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

2^η Εργαστηριακή Άσκηση:

Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Λειτουργικά Συστήματα Υπολογιστών
6ο Εξάμηνο, 2020-2021

2^η άσκηση – Σύνοψη



◆ Διαχείριση διεργασιών

- ➔ Δημιουργία δεδομένου δέντρου διεργασιών
 - με `fork()` / `wait()`
- ➔ Δημιουργία αυθαίρετου δέντρου διεργασιών
 - με βάση αρχείο εισόδου που το περιγράφει

◆ Διαδιεργασιακή επικοινωνία

- ➔ με σήματα, `SIGSTOP` / `SIGCONT`
- ➔ με σωληνώσεις (UNIX pipes)

Θεωρία - Μηχανισμοί



- ◆ Χρήση του make, Makefile
- ◆ Διαχείριση διεργασιών στο UNIX
 - ➔ `fork()`, `wait()`, `waitpid()`, `explain_wait_status()`
- ◆ Σήματα στο UNIX
 - ➔ `kill()`, `signal()`, `sigaction()`, race conditions
- ◆ Χειρισμός του σήματος SIGCHLD
 - ➔ Ασύγχρονη ειδοποίηση γονικής διεργασίας
 - ➔ non-blocking `waitpid()`
- ◆ Διαδιεργασιακή επικοινωνία με UNIX pipes

Makefile

```
$ cat Makefile
```

```
# a simple Makefile
```

```
CC = gcc
```

```
CFLAGS = -Wall -O2
```

```
all: fork-example
```

```
fork-example: fork-example.o proc-common.o
```

```
<Tab> $(CC) -o fork-example fork-example.o proc-common.o
```

```
proc-common.o: proc-common.c proc-common.h
```

```
<Tab> $(CC) $(CFLAGS) -o proc-common.o -c proc-common.c
```

```
fork-example.o: fork-example.c proc-common.h
```

```
<Tab> $(CC) $(CFLAGS) -o fork-example.o -c fork-example.c
```

```
clean:
```

```
<Tab> rm -f fork-example proc-common.o fork-example.o
```


```
$ make
```

```
gcc -Wall -O2 -o fork-example.o -c fork-example.c
```

```
gcc -Wall -O2 -o proc-common.o -c proc-common.c
```

```
gcc -o fork-example fork-example.o proc-common.o
```

2^η άσκηση – Σύνοψη



◆ Διαχείριση διεργασιών

➔ Δημιουργία δεδομένου δέντρου διεργασιών

- με `fork()` / `wait()`

➔ Δημιουργία αυθαίρετου δέντρου διεργασιών

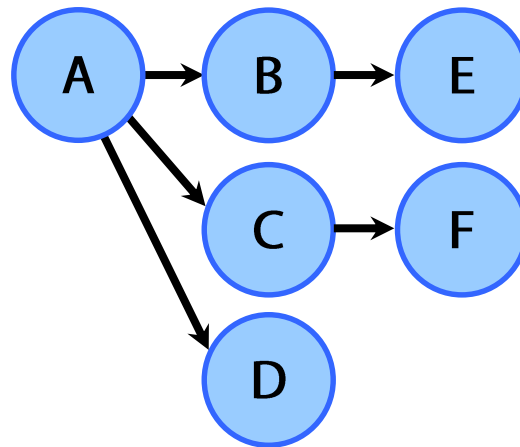
- με βάση αρχείο εισόδου που το περιγράφει

◆ Διαδιεργασιακή επικοινωνία

➔ με σήματα, `SIGSTOP` / `SIGCONT`

➔ με σωληνώσεις (UNIX pipes)

Δέντρο διεργασιών



Δημιουργία στο μοντέλο του UNIX: fork()

Δεδομένα :

p = -1 mypid = ?

Κείμενο:

```
pid_t p, mypid;
```

→ ...

→ p = fork(); p = 981; errno

→ if (p < 0) { = ENOMEM

 → perror("fork");

 → exit(1);

} else if (p == 0) {

 mypid = getpid();

 child();

} else {

 → father();

}

PID=981

Δεδομένα :

p = 0 mypid = 987

Κείμενο:

```
pid_t p, mypid;
```

→ ...

→ p = fork(); p = 0

→ if (p < 0) {

 perror("fork");

 exit(1);

} else if (p == 0) {

 → mypid = getpid();

 → child();

} else {

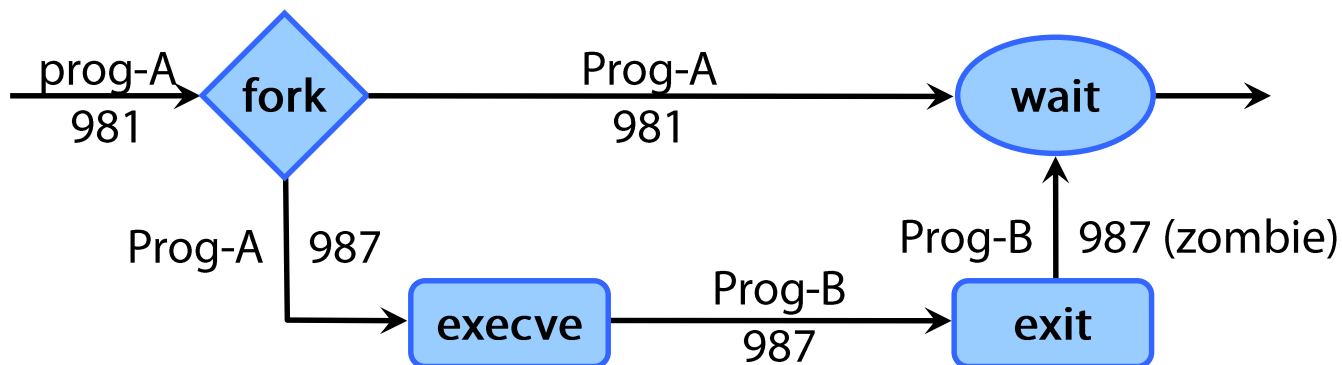
 father();

}

PID=987

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()



wait() / waitpid()

- ◆ Για κάθε **fork()** πρέπει να γίνει ένα **wait()**
- ◆ **wait(&status)**
 - ➔ Μπλοκάρει έως οποιοδήποτε παιδί πεθάνει
- ◆ Το **status** κωδικοποιεί πώς πέθανε η διεργασία
 - ➔ Κανονικά (**exit()**), λόγω κάποιου σήματος (**SIGTERM**, **SIGKILL**)
- ◆ Χρήσιμες μακροεντολές για την ερμηνεία του **status**
 - ➔ **WIFEXITED()**, **WEXITSTATUS()**, **WIFSIGNALED()**, **WTERMSIG()**
 - ➔ σας δίνεται η **explain_wait_status()**
- ◆ Μια πιο ευέλικτη **wait()**: **waitpid()**
 - ➔ Περιμένει για αλλαγή κατάστασης συγκεκριμένου ή οποιουδήποτε PID διεργασίας-παιδιού
 - ➔ Συμπεριφορά ελεγχόμενη από flags (**WNOHANG**, **WUNTRACED**)

explain_wait_status()

```
void explain_wait_status(pid_t pid, int status)
{
    if (WIFEXITED(status))
        fprintf(stderr, "Child with PID = %ld terminated normally, exit status = %d\n",
                (long)pid, WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        fprintf(stderr, "Child with PID = %ld was terminated by a signal, signo = %d\n",
                (long)pid, WTERMSIG(status));
    else if (WIFSTOPPED(status))
        fprintf(stderr, "Child with PID = %ld has been stopped by a signal, signo = %d\n",
                (long)pid, WSTOPSIG(status));
    else {
        fprintf(stderr, "%s: Internal error: Unhandled case, PID = %ld, status = %d\n",
                __func__, (long)pid, status);
        exit(1);
    }
    fflush(stderr);
}
```

Παράδειγμα:

```
pid = wait(&status);
explain_wait_status(pid, status);
if (WIFEXITED(status) || WIFSIGNALED(status))
    --processes_alive;
```

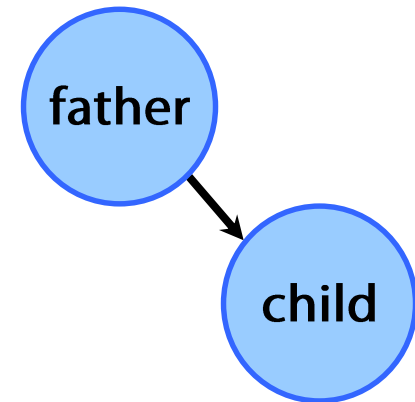
Κώδικας: παράδειγμα fork() / wait()

```
void child(void)
{
    compute(10000);
    exit(7);
}


int main(void)
{
    pid_t p;
    int status;

    p = fork();
    if (p < 0) {
        perror("fork");
        exit(1);
    }
    if (p == 0) {
        child(); exit(1);
    }

    p = wait(&status);
    explain_wait_status(p, status);
    return 0;
}
```



2^η άσκηση – Σύνοψη



◆ Διαχείριση διεργασιών

- ➔ Δημιουργία δεδομένου δέντρου διεργασιών
 - με `fork()` / `wait()`
- ➔ Δημιουργία αυθαίρετου δέντρου διεργασιών
 - με βάση αρχείο εισόδου που το περιγράφει

◆ Διαδιεργασιακή επικοινωνία

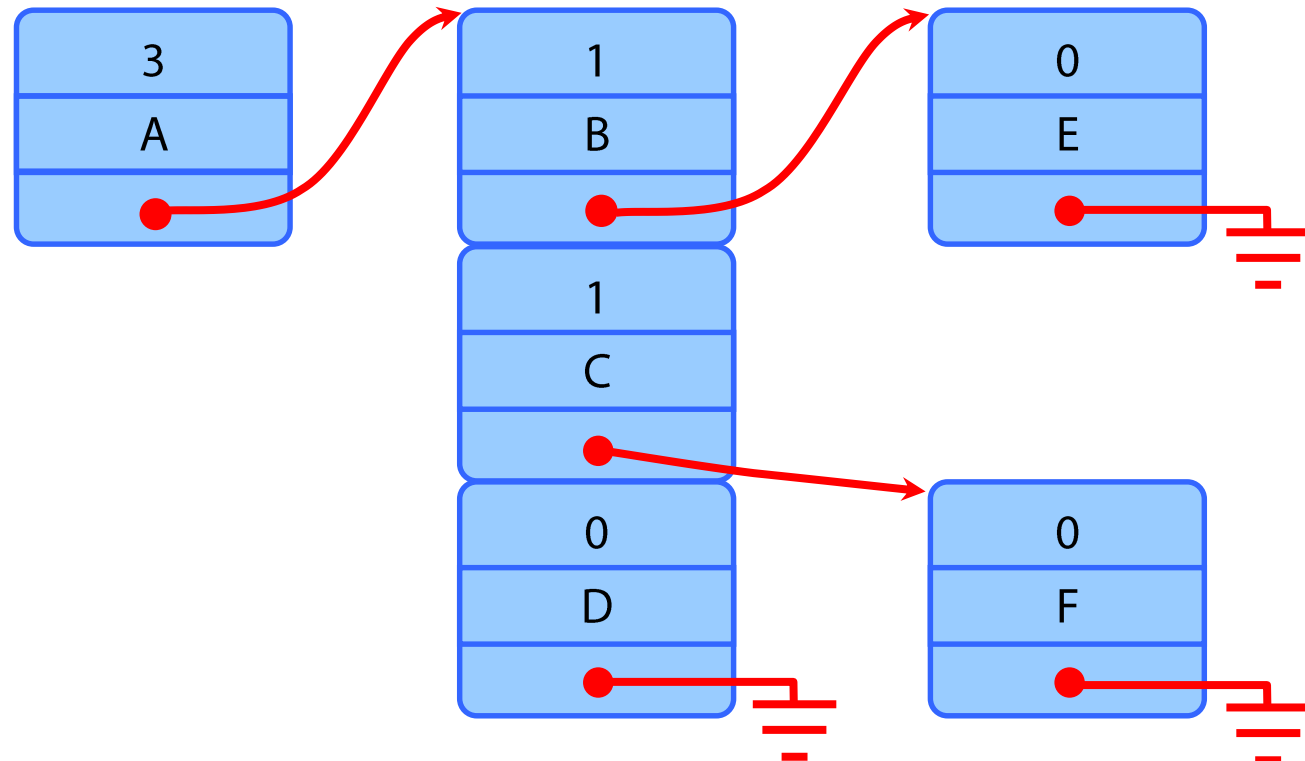
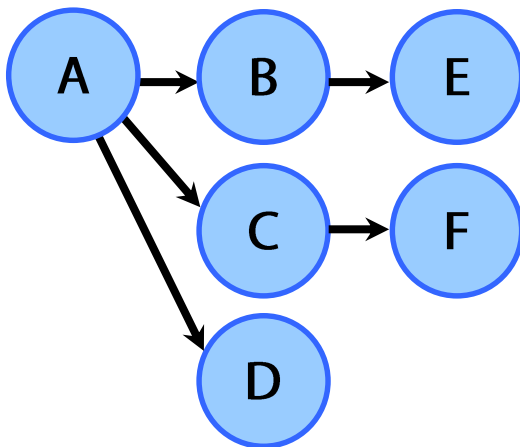
- ➔ με σήματα, `SIGSTOP` / `SIGCONT`
- ➔ με σωληνώσεις (UNIX pipes)

Βοηθητικές συναρτήσεις - παραδείγματα

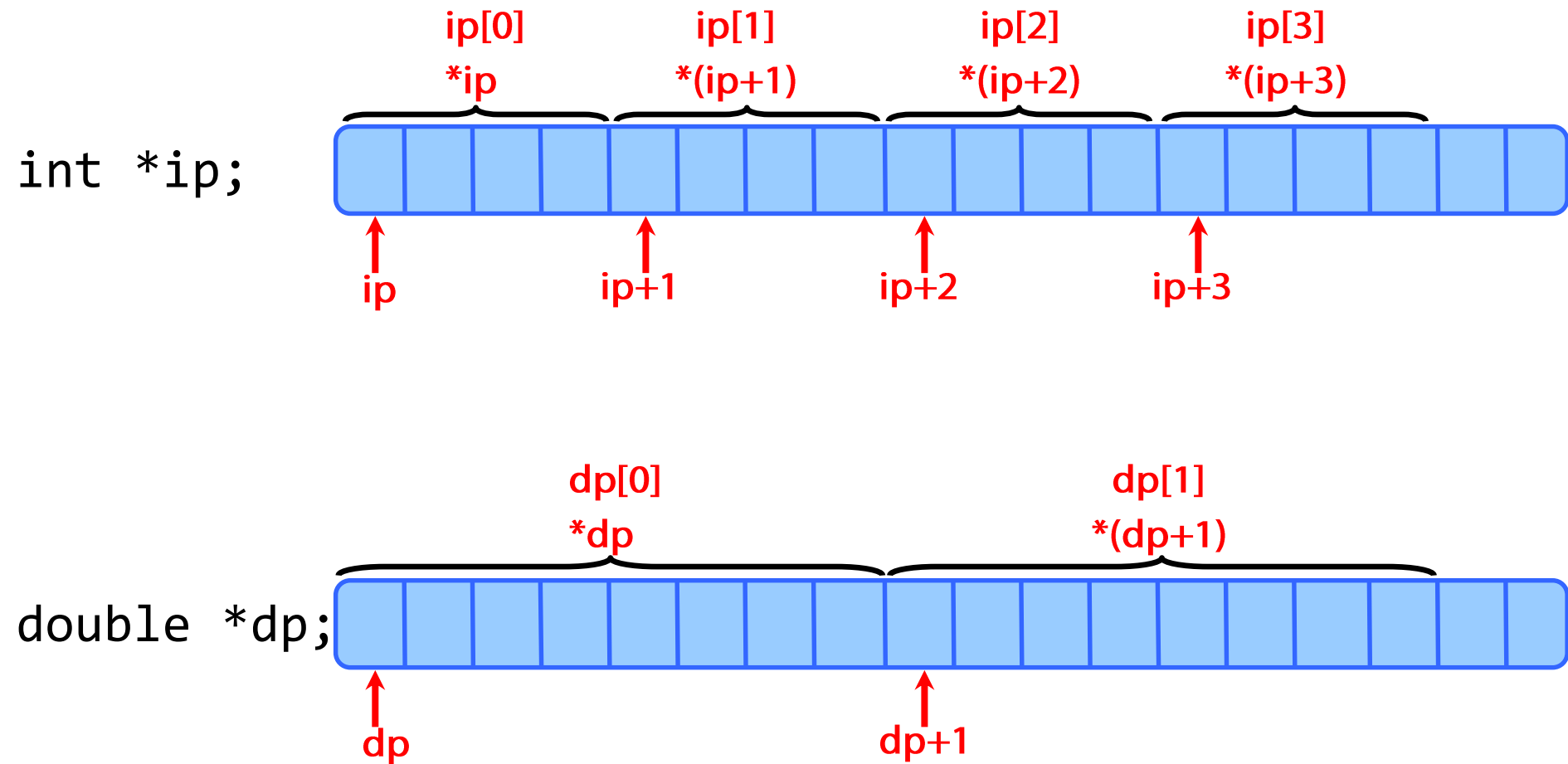
- ◆ `proc-common.{h,c}`, `fork-example.c`, `pstree-this.c`
 - ➔ `change_pname()`
 - αλλαγή ονόματος διεργασίας
 - ➔ `explain_wait_status()`
 - διαγνωστικό μήνυμα για status της `wait()`
 - ➔ `wait_for_ready_children()`
 - αναμονή αναστολής παιδιών με `SIGSTOP`
- ◆ `tree.{h,c}` και `tree-example.c`
 - ➔ `get_tree_from_file()`
 - ανάγνωση δέντρου από αρχείο, επιστροφή `struct tree_node`
 - ➔ `print_tree()`

Η δομή struct tree node

```
struct tree_node {  
    unsigned    nr_children;  
    char        name[NODE_NAME_SIZE];  
    struct tree_node *children;  
};
```

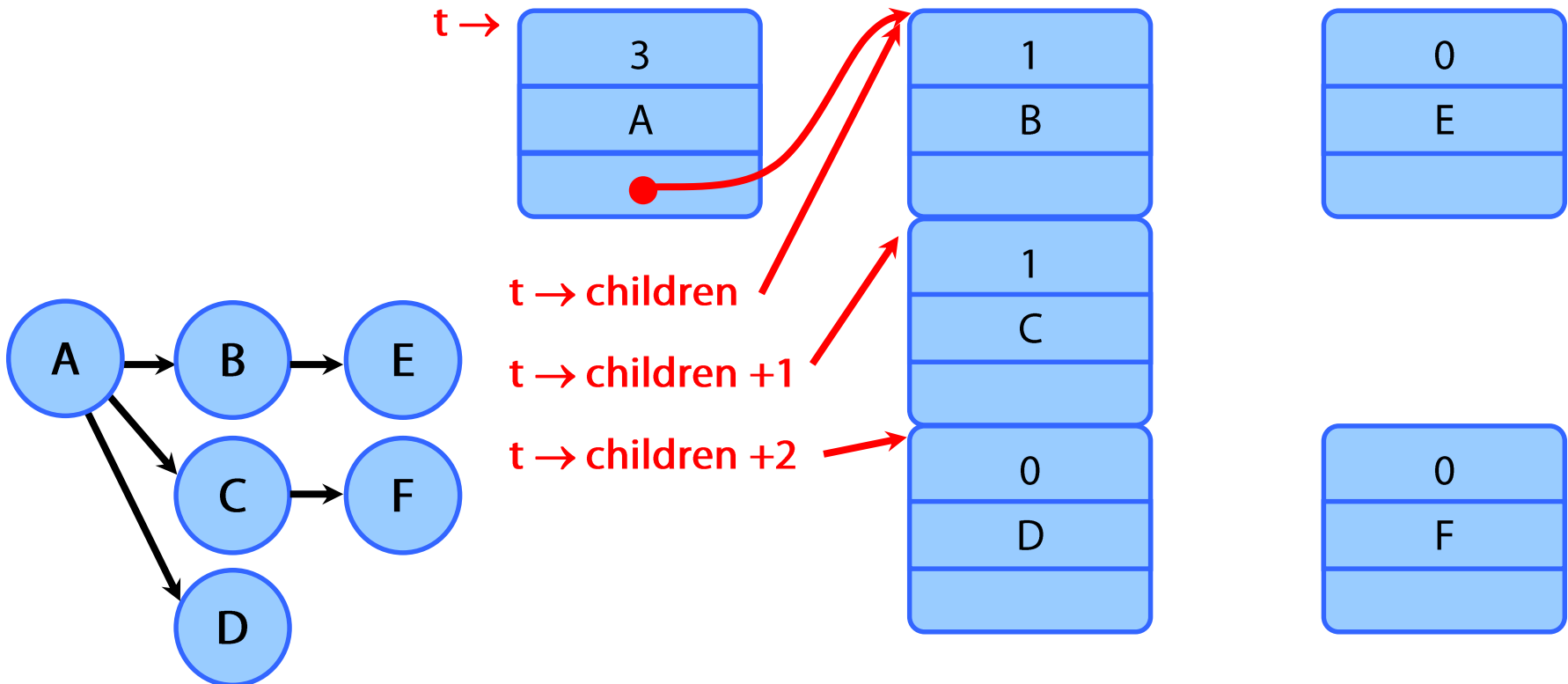


Αριθμητική με δείκτες



Η δομή struct tree node

```
struct tree_node {  
    unsigned    nr_children;  
    char        name[NODE_NAME_SIZE];  
    struct tree_node *children;  
};
```

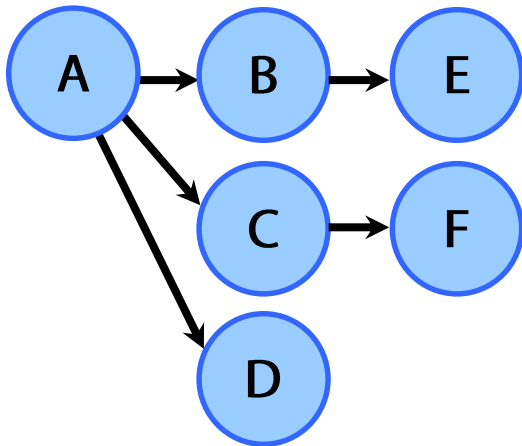


Αναδρομική διάσχιση: print_tree()

```
static void
__print_tree(struct tree_node *root, int level)
{
    int i;
    for (i=0; i<level; i++)
        printf("\t");
    printf("%s\n", root->name);


    for (i=0; i < root->nr_children; i++){
        __print_tree(root->children + i, level + 1);
    }
}

void
print_tree(struct tree_node *root)
{
    __print_tree(root, 0);
}
```



Διάσχιση κατά βάθος
(depth-first search)

2^η άσκηση – Σύνοψη



◆ Διαχείριση διεργασιών

- ➔ Δημιουργία δεδομένου δέντρου διεργασιών
 - με `fork()` / `wait()`
- ➔ Δημιουργία αυθαίρετου δέντρου διεργασιών
 - με βάση αρχείο εισόδου που το περιγράφει

◆ Διαδιεργασιακή επικοινωνία

- ➔ με σήματα, `SIGSTOP` / `SIGCONT`
- ➔ με σωληνώσεις (UNIX pipes)

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_sigusr1 = 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);  
}
```

Σήματα στο UNIX (4)

◆ Χρήσιμες εντολές

➔ kill, ps, pstree, killall, grep

\$ kill -l

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
. . .			

\$ ps -ef | grep oslabg01|grep bash

```
oslabg01  4277  4276  0 12:17 pts/0    00:00:00 bash
```

\$ kill -TERM 4277

\$ kill -9 4277

\$ killall -9 bash

◆ Όλες οι λεπτομέρειες στα manual pages

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)
{
    pid_t p;
    int status;

    /*
     * Something has happened to one of the children.
     * We use waitpid() with the WUNTRACED flag, instead of wait(), because
     * SIGCHLD may have been received for a stopped, not dead child.
     *
     * A single SIGCHLD may be received if many processes die at the same time.
     * We use waitpid() with the WNOHANG flag in a loop, to make sure all
     * children are taken care of before leaving the handler.
     */

    do {
        p = waitpid(-1, &status, WUNTRACED | WNOHANG);
        if (p < 0) {
            perror("waitpid");
            exit(1);
        }
        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... */
    } while (p > 0);
}
```


2^η άσκηση – Σύνοψη



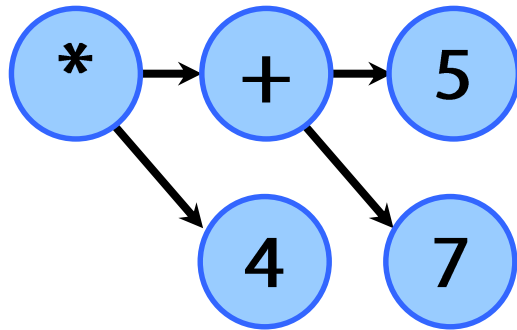
◆ Διαχείριση διεργασιών

- ➔ Δημιουργία δεδομένου δέντρου διεργασιών
 - με `fork()` / `wait()`
- ➔ Δημιουργία αυθαίρετου δέντρου διεργασιών
 - με βάση αρχείο εισόδου που το περιγράφει

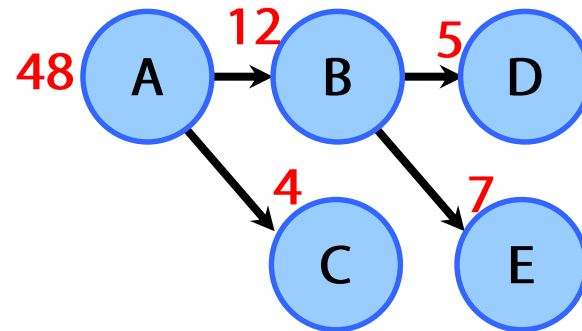
◆ Διαδιεργασιακή επικοινωνία

- ➔ με σήματα, `SIGSTOP` / `SIGCONT`
- ➔ με σωληνώσεις (UNIX pipes)

Αριθμητική έκφραση ως δέντρο

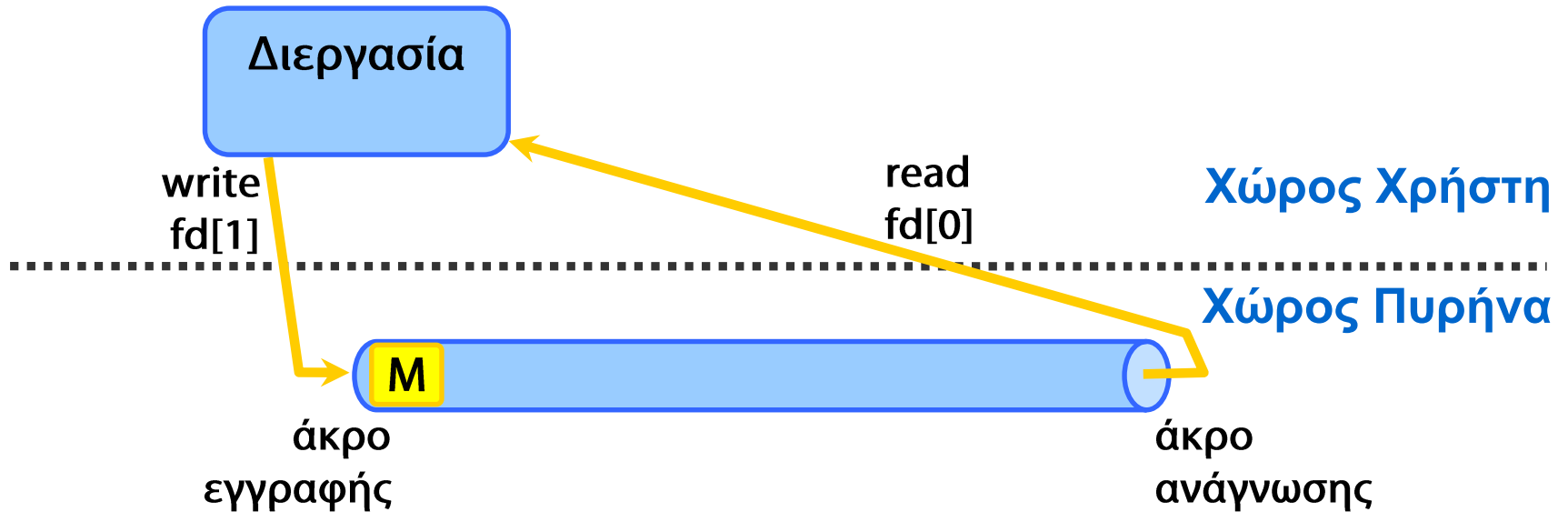


$$4 * (5 + 7)$$



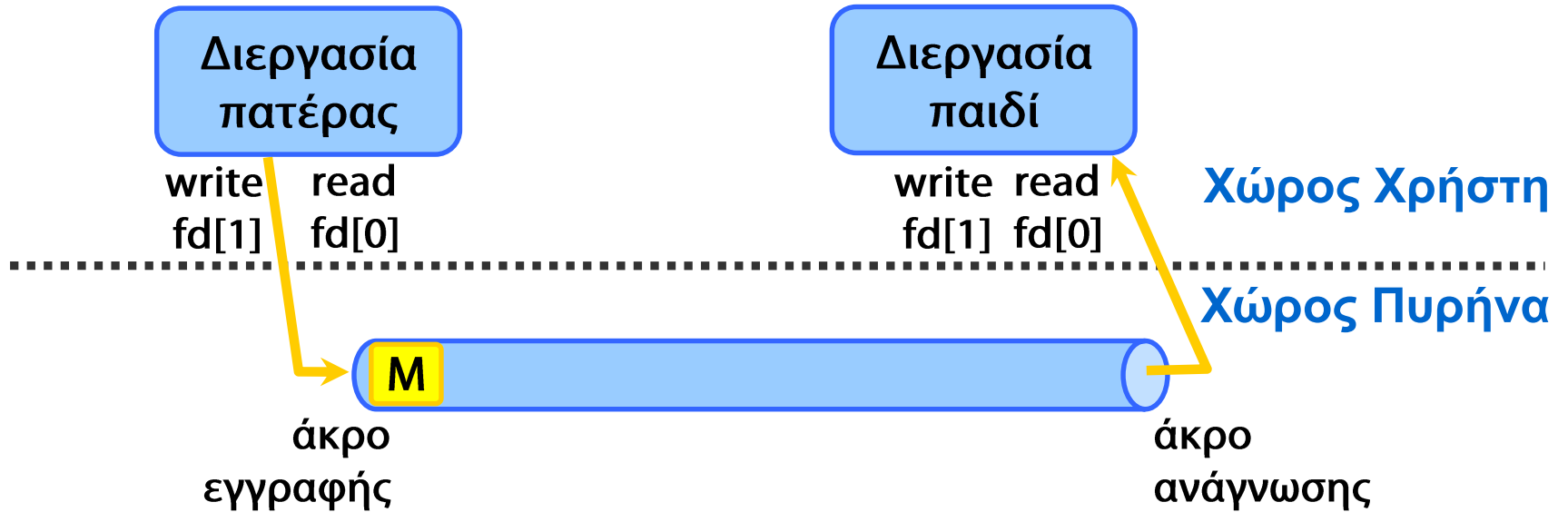
Ανάγκη για επικοινωνία
παιδί → πατέρας

Σωληνώσεις στο UNIX (1)



- ◆ Ένας από τους βασικότερους μηχανισμούς στο UNIX
- ◆ Μονόδρομη μεταφορά δεδομένων
- ◆ Από το άκρο εγγραφής στο άκρο ανάγνωσης
 - `pipe(fd);`
 - Δημιουργία με `pipe()`, επικοινωνία με `write()` και `read()`
 - `write(fd[1], &num1, sizeof(num1));`
 - Αν η σωλήνωση είναι άδεια; → η `read()` μπλοκάρει
 - `read(fd[0], &num2, sizeof(num2));`

Σωληνώσεις στο UNIX (2)



- `pipe(fd);`
- `fork();`
- ... ο πατέρας κλείνει το άκρο ανάγνωσης
- ...το παιδί κλείνει το άκρο εγγραφής

Κώδικας: παράδειγμα IPC με UNIX pipes

```
double value;
int pfd[2];
pid_t p;

if (pipe(pfd) < 0) {
    perror("pipe");
    exit(1);
}

p = fork();
if (p < 0) {
    perror("fork");
    exit(1);
} else if (p == 0) {
    if (read(pfd[0], &value, sizeof(value)) != sizeof(value)) {
        perror("read from pipe");
        exit(1);
    }
    printf("child received value: value = %f\n", value);
    exit(0);
} else {
    compute_value(&value);
    if (write(pfd[1], &value, sizeof(value)) != sizeof(value)) {
        perror("write to pipe");
        exit(1);
    }
    exit(0);
}
```

Ερωτήσεις;



και στη λίστα:

OS@lists.cslab.ece.ntua.gr