

POSIX processes

Παράλληλα και κατανεμημένα συστήματα

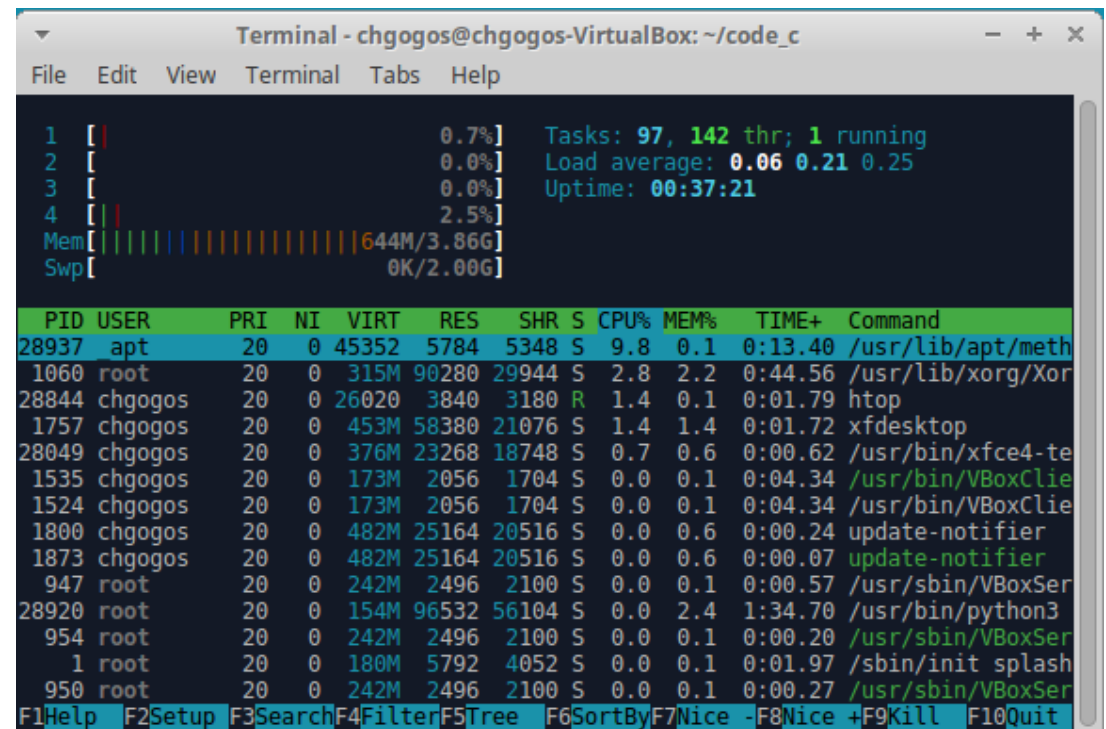
Τμήμα Μηχανικών Πληροφορικής ΤΕ

ΤΕΙ Ηπείρου

Γκόγκος Χρήστος @2018

Διεργασίες (processes)

- Μια διεργασία (process) είναι ένα πρόγραμμα σε εκτέλεση
- Μια διεργασία εκτελείται ανεξάρτητα και απομονωμένα από τις άλλες διεργασίες
- Μια διεργασία δεν μπορεί να προσπελάσει απευθείας δεδομένα άλλων διεργασιών
- Οι πόροι της διεργασίας όπως η μνήμη και ο χρόνος που της διατίθεται στη Κεντρικής Μονάδας Επεξεργασίας αποφασίζονται από το Λειτουργικό Σύστημα (ΛΣ)



The screenshot shows a terminal window titled "Terminal - chgogos@chgogos-VirtualBox: ~/code_c". The terminal displays system statistics and a process list. The statistics include: Tasks: 97, 142 thr; 1 running; Load average: 0.06 0.21 0.25; Uptime: 00:37:21; Mem: 644M/3.86G; Swp: 0K/2.00G. The process list is a table with columns: PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
28937	apt	20	0	45352	5784	5348	S	9.8	0.1	0:13.40	/usr/lib/apt/meth
1060	root	20	0	315M	90280	29944	S	2.8	2.2	0:44.56	/usr/lib/xorg/Xor
28844	chgogos	20	0	26020	3840	3180	R	1.4	0.1	0:01.79	htop
1757	chgogos	20	0	453M	58380	21076	S	1.4	1.4	0:01.72	xfdesktop
28049	chgogos	20	0	376M	23268	18748	S	0.7	0.6	0:00.62	/usr/bin/xfce4-te
1535	chgogos	20	0	173M	2056	1704	S	0.0	0.1	0:04.34	/usr/bin/VBoxClie
1524	chgogos	20	0	173M	2056	1704	S	0.0	0.1	0:04.34	/usr/bin/VBoxClie
1800	chgogos	20	0	482M	25164	20516	S	0.0	0.6	0:00.24	update-notifier
1873	chgogos	20	0	482M	25164	20516	S	0.0	0.6	0:00.07	update-notifier
947	root	20	0	242M	2496	2100	S	0.0	0.1	0:00.57	/usr/sbin/VBoxSer
28920	root	20	0	154M	96532	56104	S	0.0	2.4	1:34.70	/usr/bin/python3
954	root	20	0	242M	2496	2100	S	0.0	0.1	0:00.20	/usr/sbin/VBoxSer
1	root	20	0	180M	5792	4052	S	0.0	0.1	0:01.97	/sbin/init splash
950	root	20	0	242M	2496	2100	S	0.0	0.1	0:00.27	/usr/sbin/VBoxSer

POSIX (Portable Operating System Interface for Unix)

- Το POSIX είναι ένα σύνολο προδιαγραφών που καθορίζει (μεταξύ άλλων) τον τρόπο με τον οποίο οι εφαρμογές επικοινωνούν με το ΛΣ (API = Application Programming Interface)
- Στόχος του POSIX είναι να διευκολύνει την ανάπτυξη εφαρμογών που μπορούν να μεταφέρονται από ένα ΛΣ τύπου Unix σε ένα άλλο ΛΣ τύπου Unix
- Το POSIX έχει καθοριστεί από την IEEE Computer Society και αποτελεί πρότυπο ANSI και ISO

fork()

- Η κλήση της `fork()` δημιουργεί μια διεργασία παιδί που είναι ακριβές αντίγραφο της διεργασίας γονέα
- Τόσο η διεργασία γονέα όσο και η διεργασία παιδί συνεχίζουν την εκτέλεση του κώδικα με την εντολή αμέσως μετά το σημείο στο οποίο κλήθηκε η `fork()`
- Η `fork()` κατά την κλήση της επιστρέφει διαφορετικές τιμές στη διεργασία γονέα και στη διεργασία παιδί
 - στο γονέα η `fork()` επιστρέφει το process id του παιδιού
 - στο παιδί η `fork()` επιστρέφει το μηδέν

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

fork_example0.c

```
// gcc -Wall fork_example0.c -o fork_example0  
int main() {  
    int x = 5;  
    printf("A. value of x is %d by %d\n", x, getpid());  
    pid_t chid = fork();  
    if (chid == 0)  
        x--;          // child process  
    else  
        x++;          // parent process  
    printf("B. value of x is %d by %d\n", x, getpid());  
}
```

```
A. value of x is 5 by 5132  
B. value of x is 6 by 5132  
B. value of x is 4 by 5133
```

exec

- Οι συναρτήσεις exec αντικαθιστούν το πρόγραμμα το οποίο εκτελείται σε μια διεργασία με ένα άλλο
- Όταν μια διεργασία καλεί τη συνάρτηση exec η διεργασία σταματά να εκτελείται και ξεκινά την εκτέλεση του νέου προγράμματος από την αρχή
- Η οικογένεια συναρτήσεων exec έχει διάφορες συναρτήσεις:
 - execl
 - execl_e
 - execlp
 - execv
 - execve
 - execvp

<https://stackoverflow.com/questions/5769734/what-are-the-different-versions-of-exec-used-for-in-c-and-c>

Παράδειγμα κλήσης της `exec1`

`fork_example1.c`

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
    int x = 5;
    printf("A. value of x is %d\n", x);
    pid_t chid=fork();
    printf("B. value of x is %d\n", x);
    if (chid==0){
        x++;
        printf("C. child process %d having parent %d, value of x is %d\n",
            getpid(), getppid(), x);
        sleep(2);
        execl("/bin/echo", "echo", "hello", NULL);
    } else {
        x--;
        printf("D. parent process %d with child %d, value of x is %d\n",
            getpid(), chid, x);
        wait(NULL);
    }
    printf("E. value of x is %d\n", x);
}
```

A. value of x is 5
B. value of x is 5
D. parent process 4927 with child 4928, value of x is 4
B. value of x is 5
C. child process 4928 having parent 4927, value of x is 6
hello
E. value of x is 4

Παράδειγμα επικοινωνίας μέσω pipelines

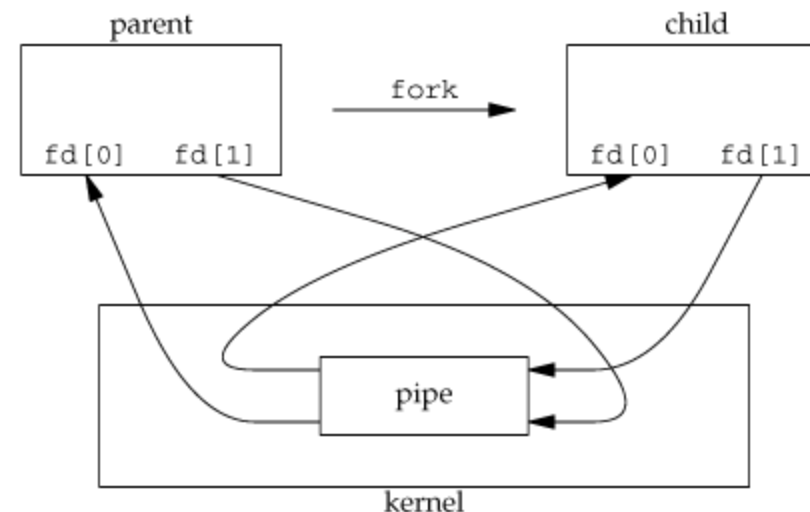
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define BUFFER_SIZE 25
int main() {
    char write_msg[BUFFER_SIZE] = "Greetings";
    char read_msg[BUFFER_SIZE];
    int fd[2];
    pipe(fd);
    pid_t pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Fork failed");
        return 1;
    }
    if (pid > 0) { // parent process
        close(fd[0]);
        write(fd[1], write_msg, strlen(write_msg) + 1);
        close(fd[1]);
    } else { // child process
        close(fd[1]);
        read(fd[0], read_msg, BUFFER_SIZE);
        printf("child process got data %s from parent process\n", read_msg);
        close(fd[0]);
    }
}
```

fork_example2.c

fd[0] είναι το άκρο ανάγνωσης, ενώ fd[1] είναι το άκρο εγγραφής

child process got data Greetings from parent process

- Η δημιουργία ενός pipeline γίνεται με τη κλήση της συνάρτησης pipe() που λαμβάνει ως όρισμα έναν πίνακα 2 ακεραίων και αν δεν υπάρξει σφάλμα κατά την κλήση της τότε επιστρέφει 2 file descriptors μέσω των οποίων γίνεται η επικοινωνία
- Αν ο γονέας θέλει να στείλει δεδομένα στο παιδί θα πρέπει ο γονέας να κλείσει το fd[0] και το παιδί να κλείσει το fd[1]
- Αν ο γονέας θέλει να λάβει δεδομένα από το παιδί θα πρέπει ο γονέας να κλείσει το fd[1] και το παιδί να κλείσει το fd[0]



Παράδειγμα με Posix διεργασίες

```
#include <unistd.h>
#include <stdlib.h>
#include <assert.h>
#include "collatz.h"

int main(int argc, char **argv) {
    int n = atoi(argv[1]);
    assert(n > 0);

    pid_t pid = fork();
    if (pid == 0) {
        collatz(n + 1);
    } else {
        wait(NULL);
        collatz(n);
    }
    return 0;
}
```

fork_example3.c

```
#include <stdio.h>

void collatz(int n) {
    printf("%d ", n);
    while (n != 1) {
        if (n % 2 == 0)
            n = n / 2;
        else
            n = 3 * n + 1;
        printf("%d ", n);
    }
    printf("\n");
}
```

collatz.c

```
void collatz(int);
```

collatz.h

all: fork_example3

run:

./fork_example3 56

fork_example3: collatz.o fork_example3.o

gcc collatz.o fork_example3.o -o fork_example3

collatz.o: collatz.c

gcc -c collatz.c

fork_example3.o: fork_example3.c

gcc -c fork_example3.c

clean:

rm *.o fork_example3

makefile

\$ make

\$ make run

./fork_example3 56

57 172 86 43 130 65 196 98 49 148 74 37 112 56 28 14 7 22 11 34 17

52 26 13 40 20 10 5 16 8 4 2 1

56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Αναφορές

- Operating System Concepts 9th Edition by Avi Silberschatz, Peter Baer Galvin, Greg Gagne <http://codex.cs.yale.edu/avi/os-book/OS9/>
- Linux fork() introduction by DrBFRaser <https://www.youtube.com/watch?v=9seb8hddeK4>
- Linux Exec System Call by DrBFRaser <https://www.youtube.com/watch?v=mj2VjcOXXs4>
- Fork and Exec Linux Programming by DrBFRaser <https://www.youtube.com/watch?v=l64ySYHmMmY>
- Linux inter process communications using pipes <http://tldp.org/LDP/lpg/node11.html>