ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2016-2017

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ

ΜΑΘΗΜΑ: ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΟ 3 - ΣΗΜΕΙΩΣΕΙΣ

Εντολές Προγραμματισμού του Φλοιού (Συνέχεια)

Εντολή case

Η σύνταξη της εντολής έχει ως εξής:

case word **in**
    [pattern [| pattern]...**)** commands **;;**]...
esac

Το παράδειγμα του μενού επιλογών γίνεται ως εξής:

Script 1

```
#!/bin/bash
# case-menu: a menu driven system information program
clear
echo "
Please Select:
1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "
case $REPLY in
        0) echo "Program terminated."
                exit
                ;;
        1) echo "Hostname: $HOSTNAME"
                uptime
                ;;
        2) df -h
                ;;
        3) if [[ $(id -u) -eq 0 ]]; then
                echo "Home Space Utilization (All Users)"
                du -sh /home/*
        else
                echo "Home Space Utilization ($USER)"
                du -sh $HOME
        fi
        ;;
        *) echo "Invalid entry" >&2
        exit 1
        ;;
esac
```

Κάποια παραδείγματα patterns παρουσιάζονατι στον ακόλουθο πίνακα:

Pattern	Description
a)	Matches if *word* equals *a*.
[[:alpha:]])	Matches if *word* is a single alphabetic character.
???)	Matches if *word* is exactly three characters long.
*.txt)	Matches if *word* ends with the characters *.txt*.
*)	Matches any value of *word*. It is good practice to include this as the last pattern in a case command to catch any values of *word* that did not match a previous pattern; that is, to catch any possible invalid values.

Εντολή for

Η σύνταξη της εντολής for έχει ως ακολούθως:

for variable [**in** words]; **do**
    commands
done

Παράδειγμα: Εύρεση του μεγαλύτερου αλφαριθμητικού σε ένα αρχείο.

Script 2

```
if [[ -r $1 ]]; then
        max_word=
        max_len=0
        for i in $(strings $1); do
                len=$(echo $i | wc -c)
                if (( len > max_len )); then
                        max_len=$len
                        max_word=$i
                fi
        done
        echo "$1: '$max_word' ($max_len characters)"
fi
```

Το ακόλουθο παραδειγμα εμφανίζει το μέγιστο μήκος αλφαριθμητικού για τα αρχεία που δίνονται ως παράμετροι στη γραμμή εντολής:

Script 3

```
#!/bin/bash
# longest-word2 : find longest string in a file
for i; do
        if [[ -r $i ]]; then
                max_word=
                max_len=0
                for j in $(strings $i); do
                        len=$(echo $j | wc -c)
                        if (( len > max_len )); then
                                max_len=$len
                                max_word=$j
                        fi
                done
                echo "$i: '$max_word' ($max_len characters)"
        fi
```

done

Ο φλοιός υποστηρίζει επίσης τη σύνταξη της εντολής που μοιάζει με τη σύνταξη της for στη γλώσσα C. Η μορφή της έχει ως εξής:

for ((expression1; expression2; expression3)); **do**
    commands
done

Παράδειγμα:

Script 4

```
#!/bin/bash
# simple_counter : demo of C style for command
for (( i=0; i<5; i=i+1 )); do
    echo $i
done
```

Εντολή while

Η σύνταξη της εντολής while έχει ως εξής:

while test-command; **do**
    commands;
done

Παράδειγμα: Μενού επιλογών με χρήση της while

Script 5

```
#!/bin/bash
# while-menu: a menu driven system information program
DELAY=2 # Number of seconds to display results
while [[ $REPLY != 0 ]]; do
    clear
    echo "
    Please Select:
    1. Display System Information
    2. Display Disk Space
    3. Display Home Space Utilization
    0. Quit
    "
    read -p "Enter selection [0-3] > "
    if [[ $REPLY =~ ^[0-3]$ ]]; then
        if [[ $REPLY == 1 ]]; then
            echo "Hostname: $HOSTNAME"
            uptime
            sleep $DELAY
        fi
        if [[ $REPLY == 2 ]]; then
            df -h
            sleep $DELAY
        fi
        if [[ $REPLY == 3 ]]; then
            if [[ $(id -u) -eq 0 ]]; then
                echo "Home Space Utilization (All Users)"
                du -sh /home/*
            else
```

```
                    echo "Home Space Utilization ($USER)"
                    du -sh $HOME
                fi
                sleep $DELAY
            fi
else
        echo "Invalid entry."
        sleep $DELAY
fi
done
echo "Program terminated."
```

Σημείωση: στις επαναληπτικές εντολές, η χρήση των **continue** και **break** είναι η ίδια όπως και στις άλλες γλώσσες προγραμματισμού.

Εντολή until

Η σύνταξη της εντολής until έχει ως εξής

```
until test-command; do
        commands
done
```

Παράδειγμα

Script 6

```
#!/bin/bash
# until-count: display a series of numbers
count=1
until [ $count -gt 5 ]; do
        echo $count
        count=$((count + 1))
done
echo "Finished."
```

Συναρτήσεις Φλοιού

Στο φλοιό bash μπορούμε να χρησιμοποιήσουμε συναρτήσεις όπως σε οποιαδήποτε άλλη γλώσσα προγραμματισμού. Η σύνταξη των συναρτήσεων έχει ως ακολούθως:

```
[function] name()
{
        Commands
        return
}
```

Το *[function]* είναι προαιρετικό. Ακολουθούν κάποια παραδείγματα χρήσης συναρτήσεων σε προγράμματα φλοιού.

Script 7

```
#! /bin/bash
function whoson()
{
        date
        echo "Users Currently Logged On"
        who
```

```
        return
}
whoson
```

Script 8

```
#!/bin/bash
function funct {
        echo "Step 2"
        return
}

# Main program starts here
echo "Step 1"
funct
echo "Step 3"
```

Στις συναρτήσεις μπορούμε να χρησιμοποιήσουμε τοπικές μεταβλητές. Η δήλωσή τους σε μια συνάρτηση γίνεται με τη χρήση της λέξης *local*. Στο επόμενο παράδειγμα γίνεται χρήση τόσο τοπικών όσο και καθολικών μεταβλητών.

Script 9

```
#!/bin/bash
# local-vars: script to demonstrate local variables
foo=0 # global variable foo
funct_1 () {
        local foo # variable foo local to funct_1
        foo=1
        echo "funct_1: foo = $foo"
}

funct_2 () {
        local foo # variable foo local to funct_2
        foo=2
        echo "funct_2: foo = $foo"
}

echo "global: foo = $foo"
funct_1
echo "global: foo = $foo"
funct_2
echo "global: foo = $foo"
```

Διαχείριση Πινάκων

Στον προγραμματισμό του φλοιού του Linux μπορούμε να χρησιμοποιήσουμε πίνακες οι οποίοι όμως δεν μπορεί να είναι πολυδιάστατοι. Ο ορισμός ενός πίνακα γίνεται με τους ακόλουθους τρόπους:

- declare –a <name>
- <name>=values

Παραδείγματα δηλώσεων και αρχικοποιήσεων ενός πίνακα:

- a[1]=foo

- declare –a a
- days=(Sun Mon Tue Wed Thu Fri Sat)
- days=([0]=Sun [1]=Mon [2]=Tue [3]=Wed [4]=Thu [5]=Fri [6]=Sat)

Το ακόλουθο παράδειγμα μας εμφανίζει για ένα κατάλογο που δίνουμε από το πληκτρολόγιο τα αρχεία που έχουν τροποποιηθεί κάθε ώρα της ημέρας.

Script 10

```
#!/bin/bash
# hours : script to count files by modification time
usage () {
        echo "usage: $(basename $0) directory"
}
# Check that argument is a directory
if [[ ! -d $1 ]]; then
        usage
        exit 1
fi
# Initialize array
for i in {0..23}; do
        hours[i]=0;
done
# Collect data
for i in $(stat -c %y "$1"/* | cut -c 12-13); do
        j=${i/#0}
        ((++hours[j]))
        ((++count))
done
# Display data
echo -e "Hour\tFiles\tHour\tFiles"
echo -e "----\t-----\t----\t-----"
for i in {0..11}; do
        j=$((i + 12))
        printf "%02d\t%d\t%02d\t%d\n" $i ${hours[i]} $j ${hours[j]}
done
printf "\nTotal files = %d\n" $count
```

Οι επόμενες εντολές δίνουν τη δυνατότητα διαχείρισης των περιεχομένων ενός πίνακα:

- **Προσπέλαση όλων των στοιχείων ενός πίνακα**

  ```
  animals=("a dog" "a cat" "a fish")
  for i in ${animals[*]}; do echo $i; done
  for i in ${animals[@]}; do echo $i; done
  for i in "${animals[*]}"; do echo $i; done
  for i in "${animals[@]}"; do echo $i; done
  ```

- **Πλήθος στοιχείων ενός πίνακα**

  ```
  a[100]=foo
  echo ${#a[@]} # number of array elements
  echo ${#a[100]} # length of element 100
  ```

- **Προσθήκη στοιχείων σε ένα πίνακα**

  ```
  foo=(a b c)
  ```

```
echo ${foo[@]}
foo+=(d e f)
echo ${foo[@]}
```

- **Ταξινόμηση στοιχείων ενός πίνακα**

Script 5

```
#!/bin/bash
# array-sort : Sort an array
a=(f e d c b a)
echo "Original array: ${a[@]}"
a_sorted=($(for i in "${a[@]}"; do echo $i; done | sort))
echo "Sorted array: ${a_sorted[@]}"
```

Εισαγωγή στη Διαχείριση Διεργασιών

Όταν εκκινεί το Λειτουργικό Σύστημα, ένα σύνολο λειτουργιών εκκινούν ως ξεχωριστές διεργασίες και καλεί ένα πρόγραμμα το οποίο ονομάζεται ***init***. Επίσης, ένας αριθμός από scripts τα οποία βρίσκονται στον κατάλογο **/etc** (init scripts) αναλαμβάνουν να εκτελέσουν τις λειτουργίες του συστήματος. Πολλά από τα προγράμματα του Λειτουργικού Συστήματος εκτελούνται ως 'δαίμονες' (daemons) τα οποία εκκινούν και εκτελούνται στο background, συνήθως, χωρίς αλληλεπίδραση με τους χρήστες.

Στο Linux μπορεί να δημιουργηθεί μια ιεραρχία διεργασιών με τη μορφή πατέρας – παιδί (parent – child). Ο πυρήνας κρατά συγκεκριμένες πληροφορίες για κάθε διεργασία όπως το ID κάθε διεργασίας (process ID – PID). Τα PIDs αποδίδονται σε αύξουσα σειρά με τη διεργασία init να έχει πάντα PID = 1. Επίσης, ο πυρήνας κρατά τη μνήμη που αποδίδεται σε κάθε διεργασία όπως και την 'ετοιμότητα' κάθε διεργασίας να συνεχίσει την εκτέλεσή της.

Για να δούμε πληροφορίες που σχετίζονται με τις διεργασίες που εκτελούνται στο Λειτουργικό Σύστημα μπορούμε να χρησιμοποιήσουμε την εντολή **ps**. Στο ακόλουθο παράδειγμα, η ps μας δείχνει ότι εκτελούνται δύο διεργασίες: ο φλοιός bash και η ίδια η εντολή ps.

```
kk@ubuntu:~$ ps
  PID TTY          TIME CMD
 2487 pts/0    00:00:00 bash
 3237 pts/0    00:00:00 ps
```

Η εντολή **ps x** μας δείχνει περισσότερες πληροφορίες σχετικά με τις διεργασίες που εκτελούνται από το Λειτουργικό σύστημα. Ο χαρακτήρας ? στη στήλη TTY μας λέει ότι οι συγκεκριμένες διεργασίες δεν έχουν ένα συγκεκριμένο τερματικό ελέγχου. Στη στήλη STAT μπορούμε να διακρίνουμε την κατάσταση κάθε διεργασίας. Ο επόμενος πίνακας συνοψίζει τις πιθανές καταστάσεις των διεργασιών.

STAT Code	Description
S	Sleeping. Usually waiting for an event to occur, such as a signal or input to become available.
R	Running. Strictly speaking, "runnable," that is, on the run queue either executing or about to run.
D	Uninterruptible Sleep (Waiting). Usually waiting for input or output to complete.
T	Stopped. Usually stopped by shell job control or the process is under the control of a debugger.
Z	Defunct or "zombie" process.
N	Low priority task, "nice."
W	Paging. (Not for Linux kernel 2.6 onwards.)
s	Process is a session leader.
+	Process is in the foreground process group.
l	Process is multithreaded.
<	High priority task.

Οι χρήστες που 'τρέχουν' τις διεργασίες παρουσιάζονται στα αποτελέσματα της εντολής **ps aux**. Ο ακόλουθος πίνακας παρουσιάζει τη σημασία της κάθε στήλης αποτελεσμάτων της ps aux.

Header	Meaning
USER	User ID. This is the owner of the process.
%CPU	CPU usage as a percent.
%MEM	Memory usage as a percent.
VSZ	Virtual memory size.
RSS	Resident Set Size. The amount of physical memory (RAM) the process is using in kilobytes.
START	Time when the process started. For values over 24 hours, a date is used.

Για τη διαχείριση διεργασιών χρησιμοποιούνται οι ακόλουθες εντολές:.

- **Εντολή top**

Η εντολή top παρουσιάζει πληροφορίες σχετικά με τις διεργασίες που εκτελούνται. Οι πληροφορίες ανανεώνονται κάθε 3 δευτερόλεπτα. Στην αρχή των αποτελεσμάτων παρουσιάζονται οι συνολικές πληροφορίες του συστήματος ενώ η λίστα των διεργασιών ξεκινά με αυτές τις διεργασίες που έχουν τη μεγαλύτερη δραστηριότητα. Ο ακόλουθος πίνακας παρουσιάζει τη σημασιολογία των πληροφοριών που εμφανίζει η εντολή top.

```
top - 14:59:20 up  6:30,  2 users,  load average: 0.07, 0.02, 0.00
Tasks: 109 total,   1 running, 106 sleeping,   0 stopped,   2 zombie
Cpu(s):  0.7%us,  1.0%sy,  0.0%ni, 98.3%id,  0.0%wa,  0.0%hi,  0.0%si
Mem:    319496k total,   314860k used,     4636k free,    19392k buff
Swap:   875500k total,   149128k used,   726372k free,   114676k cach

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6244 me        39  19 31752 3124 2188 S  6.3  1.0  16:24.42 trackerd
11071 me        20   0  2304 1092  840 R  1.3  0.3   0:00.14 top
 6180 me        20   0  2700 1100  772 S  0.7  0.3   0:03.66 dbus-dae
 6321 me        20   0 20944 7248 6560 S  0.7  2.3   2:51.38 multiloa
 4955 root      20   0  104m 9668 5776 S  0.3  3.0   2:19.39 Xorg
    1 root      20   0  2976  528  476 S  0.0  0.2   0:03.14 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migratio
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.72 ksoftirq
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.04 watchdog
    6 root      15  -5     0    0    0 S  0.0  0.0   0:00.42 events/0
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.06 khelper
   41 root      15  -5     0    0    0 S  0.0  0.0   0:01.08 kblockd/
   67 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
  114 root      20   0     0    0    0 S  0.0  0.0   0:01.62 pdflush
  116 root      15  -5     0    0    0 S  0.0  0.0   0:02.44 kswapd0
```

Row	Field	Meaning
1	`top`	Name of the program.
	`14:59:20`	Current time of day.
	`up 6:30`	This is called *uptime*. It is the amount of time since the machine was last booted. In this example, the system has been up for 6½ hours.
	`2 users`	Two users are logged in.
	`load average:`	*Load average* refers to the number of processes that are waiting to run; that is, the number of processes that are in a runnable state and are sharing the CPU. Three values are shown, each for a different period of time. The first is the average for the last 60 seconds, the next the previous 5 minutes, and finally the previous 15 minutes. Values under 1.0 indicate that the machine is not busy.
2	`Tasks:`	This summarizes the number of processes and their various process states.
	`0.7%us`	0.7% of the CPU is being used for *user processes*. This means processes outside of the kernel itself.
	`1.0%sy`	1.0% of the CPU is being used for *system* (kernel) processes.
	`0.0%ni`	0.0% of the CPU is being used by nice (low-priority) processes.
	`98.3%id`	98.3% of the CPU is idle.
	`0.0%wa`	0.0% of the CPU is waiting for I/O.
4	`Mem:`	Shows how physical RAM is being used.
5	`Swap:`	Shows how swap space (virtual memory) is being used.

- **Χαρακτήρας &**

Ο χαρακτήρας & χρησιμοποιείται για να θέσει την εκτέλεση μιας διεργασίας στο παρασκήνιο (background). Για παράδειγμα, δώστε την εντολή ***xlogo &*** και έπειτα την εντολή ***ps***.

- **Εντολή jobs**

Η εντολή jobs μας εμφανίζει τις διεργασίες που τρέχουν.

- **Εντολή fg**

Αφού δώσουμε την εντολή jobs, με την εντολή ***fg %<number>*** μπορούμε να επαναφέρουμε μια διεργασία στο προσκήνιο (foreground). Για να σταματήσουμε μια διεργασία που τρέχει στο προσκήνιο μπορούμε να πιέσουμε Ctrl-Z.

- **Εντολή bg**

Με την εντολή bg θέτουμε μια διεργασία για εκτέλεση στο παρασκήνιο (background).

- **Εντολή kill**

Με την εντολή kill τερματίζουμε μια διεργασία. Με την εντολή μπορούμε να στείλουμε και σήματα στις διεργασίες ως εξής: **kill <-signal> PID**. Αν δεν καθοριστεί κάποιο σήμα, τότε στέλνεται το σήμα TERM (terminate). Ο ακόλουθος πίνακας παρουσιάζει τα πιο συνηθισμένα σήματα:

Type	Name	Number	Generating condition
Not a real signal	EXIT	0	Exit because of exit command or reaching the end of the program (not an actual signal but useful in trap)
Hang up	SIGHUP or HUP	1	Disconnect the line
Terminal interrupt	SIGINT or INT	2	Press the interrupt key (usually CONTROL-C)
Quit	SIGQUIT or QUIT	3	Press the quit key (usually CONTROL-SHIFT-| or CONTROL-SHIFT-\)
Kill	SIGKILL or KILL	9	The kill builtin with the **–9** option (cannot be trapped; use only as a last resort)
Software termination	SIGTERM or TERM	15	Default of the kill command
Stop	SIGTSTP or TSTP	20	Press the suspend key (usually CONTROL-Z)
Debug	DEBUG		Executes ***commands*** specified in the trap statement after each command (not an actual signal but useful in trap)
Error	ERR		Executes ***commands*** specified in the trap statement after each command that returns a nonzero exit status (not an actual signal but useful in trap)

Η αναφορά στα σήματα μπορεί να γίνει είτε αριθμητικά είτε ονομαστικά χρησιμοποιώντας τα ονόματα που παρουσιάζονται στον παραπάνω πίνακα (προβάλλοντας τους χαρακτήρες SIG).
Παραδείγματα:
$ xlogo &
$ kill -INT 13601
[1]+ Interrupt xlogo
$ xlogo &
[1] 13608
$ kill -SIGINT 13608
[1]+ Interrupt xlogo

Άλλα είδη σημάτων είναι τα ακόλουθα:

Number	Name	Meaning
3	QUIT	Quit.
11	SEGV	Segmentation violation. This signal is sent if a program makes illegal use of memory; that is, it tried to write somewhere it was not allowed to.
20	TSTP	Terminal stop. This is the signal sent by the terminal when CTRL-Z is pressed. Unlike the STOP signal, the TSTP signal is received by the program but the program may choose to ignore it.
28	WINCH	Window change. This is a signal sent by the system when a window changes size. Some programs, like top and less, will respond to this signal by redrawing themselves to fit the new window dimensions.

Ο τερματισμός ή η αποστολή σημάτων σε πολλαπλές διεργασίες μπορεί να γίνει με τη χρήση της εντολής **killall <name>**.

Άλλες εντολές που αφορούν σε διεργασίες είναι οι ακόλουθες:

Command	Description
pstree	Outputs a process list arranged in a tree-like pattern showing the parent/child relationships between processes.
vmstat	Outputs a snapshot of system resource usage including memory, swap, and disk I/O. To see a continuous display, follow the command with a time delay (in seconds) for updates (e.g., vmstat 5). Terminate the output with CTRL-C.
xload	A graphical program that draws a graph showing system load over time.
tload	Similar to the xload program, but draws the graph in the terminal. Terminate the output with CTRL-C.