N12 - teme lab02

BAJCSI Elias-Robert

2026-02-17 Tue

1 Sa se scrie un script shell care aduna numerele date ca parametrii in linia de comanda

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} NUM1 NUM2" >&2
    echo "Prints the sum of NUM1 and NUM2." >&2
}

# Check args
[ "$#" -ne 2 ] && { usage; exit 1; }

echo $(( $1 + $2 ))
```

2 Sa se scrie un script shell care scrie numerele in ordine descrescatoare incepand de la n dat ca parametru folosind while.

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} NUM1" >&2
    echo "Prints numbers from NUM1 to 0, in descending order." >&2
    echo "The number must be positive." >&2
}

# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

num=$1;
while [ "$num" -ge 0 ]; do
    echo $num
    num=$(( "$num" - 1 ))
done
```

3 Sa se inverseze cifrele unui numar (254 -> 452).

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} NUM1" >&2
    echo "Prints numbers in reverse order." >&2
}

# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

n=$1
rev=0

while [ "$n" -gt 0 ]; do
    rev=$(( rev * 10 + n % 10 ))
    n=$(( n / 10 ))
done

echo "$rev"
```

4 Scrieti un script care afiseaza data curenta, ora, numele utilizatorului si directorul curent

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0}" >&2
    echo "Prints current datetime, user and current directory" >&2
}

# Check args
[ "$#" -ne 0 ] && { usage; exit 1; }

echo "Current datetime: $(date)"
echo "Logged user name: $(whoami)"
echo "Current working directory $(pwd)"
```

5 Scrieti un script care calculeaza suma cifrelor unui numar dat ca parametru

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} NUM1" >&2
    echo "Prints the sum of the figures from NUM1." >&2
}
```

```sh
# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

n=$1
rev=0

while [ "$n" -gt 0 ]; do
    sum=$(( sum + n % 10 ))
    n=$(( n / 10 ))
done

echo "$sum"
```

6 Scrieti un script care determina daca o comanda contine caracterul "*"

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} ARG1 ..." >&2
    echo "Checks if the provided arguments contain a \`*' character." >&2
}

# Check args
[ "$#" -eq 0 ] && { usage; exit 1; }

for s in "$@"; do
    case $s in
        *"*"*) echo "Found \`*' character." && exit 0;;
    esac
done

echo "No \`*' character found."
```

7 Scrieti un script care afiseaza numerele lui Fibonacci

```sh
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} NUM1" >&2
    echo "Prints the first NUM1 numbers of the Fibonacci series." >&2
}

# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

a=0
b=1
```

```
i=0

while [ "$i" -lt "$1" ]; do
    tmp=$b
    b=$(( a + b ))
    a=$tmp
    i=$(( i + 1 ))
done

echo "$a"
```

8 Scrieti un script care transforma litere mari in litere mici pentru nume de fisiere primite ca parametru.

```
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} SOURCE" >&2
    echo "Renames a file name to it's lowercase variant." >&2
}

# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

# Check if file exist
[ ! -f "${1}" ] && { echo "File doesn't exist!" >&2; exit 1; }

# Only lower the basename
mv "${1}" $(dirname "$1")/$(basename "$1" | tr '[:upper:]' '[:lower:]')
```

9 Scrieti un script care transforma litere mari in litere mici pentru nume de fisiere primite ca parametru.

```
#!/usr/bin/env sh

usage() {
    echo "Usage: ${0} SOURCE" >&2
    echo "Renames a file name to it's lowercase variant." >&2
}

# Check args
[ "$#" -ne 1 ] && { usage; exit 1; }

# Check if file exist
[ ! -f "${1}" ] && { echo "File doesn't exist!" >&2; exit 1; }

# Only lower the basename
mv "${1}" $(dirname "$1")/$(basename "$1" | tr '[:upper:]' '[:lower:]')
```

10 Quiz

10.1 Ce moduri de operare s-au intalnit in evolutia sistemelor de calcul?

- Batch — programele erau încărcate și executate fără intervenția utilizatorului, unul câte unul.
- Interactiv (time-sharing) — mai mulți utilizatori pot interacționa cu sistemul în mod simultan, partajând timpul procesorului central.
- Multiprogramare — mai multe programe stau în memorie și CPU alternează între ele pentru a crește eficiența.
- RT (Real Time) — sistemul răspunde la evenimente într-un timp STRICT.

10.2 Descrieti pe scurt principiul multiprogramarii si motivul introducerii acestuia.

Multiprogramarea reprezintă păstrarea mai multor program în memorie în același timp, astfel încât procesorul să poată executa un program la altul când așteaptă I/O.

Acesta s-a introdus deoarece CPU-ul este mult mai rapid decât dispozitivele de intrare/ieșire. Așteptarea după I/O reprezintă eficiență redusă.

10.3 Care este deosebirea dintre modul kernel si modul utilizator?

10.3.1 Mod Kernel

- Spațiu de adresare este Kernel space – acces direct la hardware și resurse
- Are access complet la CPI, memorie, dispozitive
- Orice eroare poate panica sau bloca sistemul

10.3.2 Mod User

- Spațiul de adresare este User space – acces restricționat
- Access limitat, prin apeluri la kernel
- Eroarea afectează doar procesul curent și copii

10.4 Care dintre urmatoarele operatii sunt permise doar in modul kernel?

1. **mascarea tuturor intreruperilor**
2. citirea ceasului
3. apel de operare kernel
4. **fixarea orei sistem**
5. **translatarea adreselor de memorie**

10.5 Explicati pe scurt pasii de intrare a unui program in modul kernel

- Procesul rulează în mod utilizator și face un apel de sistem.
- CPU comută în mod kernel și salvează contextul procesului curent.
- Kernelul preia controlul și execută funcția solicitată.
- La terminarea operației, kernelul restabilește contextul și revine în modul utilizator.

10.6 Definitia sistemul de operare contine doau moduri de abordare a acestuia. Care sunt acestea si ce semnificatie au?

10.6.1 Monolitic

- Toate funcțiile kernelului rulează într-un singur spațiu, fără izolare între module.
- Dezavantaj: erorile pot afecta tot sistemul (Exemplu: Intel iwlwifi every single f***ing update)
- Exemplu: Linux, BSD

10.6.2 Microkernel

- Kernelul minimal rulează doar funcțiile esențiale (planificare, comunicare, I/O minim)
- Restul serviciilor rulează în user space
- Avantaj: mai sigur și mai ușor de extins
- Dezavantaj: performanță mai mică datorită apelurilor între spații
- Exemplu: GNU Hurd *:3*