Laborator 2 – Sisteme de Operare

Dezvoltarea unui proiect software complex

In zilele actuale proiectele software devin din ce in ce mai complexe. Programatorii sunt organizati in echipe pentru a grabi dezvoltarea produsului. Acesta trebuie sa indeplineasca anumite functionalitati cerute de catre client si sa fie livrat in timpul stabilit. Odata cu cresterea dimensiunii aplicatilor au aparut si utilitare de dezvoltare pentru a facilita managementul acestora.

Un **repository** este un loc unde sunt tinute fisierele sursa si istoria proiectului. In majoritatea cazurilor, repository-ul se afla pe un server si este foarte bine protejat. Pierderea unui repository poate sa fie o catastrofa.

Sistemele de control al versiunii sunt programe care tin codul sursa si toata istoria modificarilor asupra lui. Folosirea unui astfel de sistem este o practica aproape universala in industria software.

Exemple de astfel de sisteme sunt GIT sau SCCS.

Automatizarea compilarii – utilitarul make

De obicei, intr-un repository nu se afla programe compilate. Procesul de compilare a unui produs software complex poate dura uneori ore. Astfel, recompilarea integrala a produsului la fiecare modificare adusa unui modul nu este o optiune viabila.

Make este un utilitar care permite automatizarea si eficientizarea sarcinilor de compilare, linking si altele. Acesta rezolva dependentele intre fisierele sursa pentru a eficientiza procesul. Make rezolva problema executiei unor actiuni in functie de dependentele intre ele. O actiune este executata doar daca actiunile de care depinde au fost executate.

Utilitarul make foloseste un fisier de configurare denumit **Makefile**. Un astfel de fisier contine reguli si comenzi de automatizare.

Sintaxa unei reguli:

```
target :  prerequisites
<tab>    command
```

Unde:

- **target** - este, de obicei, fisierul care se va obtine prin rularea comenzii command. .
- **prerequisites** - reprezinta dependentele necesare pentru a urmari regula; de obicei sunt fisiere necesare pentru obtinerea tintei.
- **<tab>** - reprezinta caracterul tab și trebuie neaparat folosit inaintea precizarii comenzii.
- **command** - o lista de comenzi (niciuna, una, oricate) rulate in momentul in care se trece la obtinerea tintei.

Pentru a lansa acest utilitar se introduce comanda:

$ make

Acesta cauta un fisier de configurare cu numele **makefile** sau **Makefile**. Daca nu gaseste nici un fisier cu unul din aceste nume atunci un mesaj de eroare o sa fie afisat.

Daca se doreste introducerea unui fisier de configurare explicit atunci se introduce comanda:

$ make -f nume_fisier

Implicit, se executa primul target gasit in fisierul makefile. Daca se doreste executia unui target explicit se introduce comanda:

$ make nume_target

sau

$ make -f nume_fisier nume_target

in cazul in care se doreste si un fisier de configurare explicit.

Argumente in linia de comanda

Pentru a returna un rezultat, un program are nevoie de o sursa de input pentru a isi lua datele ce trebuie computate. Acestea pot sa vina dintr-un fisier,de pe retea etc. In cele ce urmeaza vom studia cum preluam argumente din linia de comanda intr-un program C.

In runtime-ul de C al sistemului de operare exista functia **start** ce face apel la functia main, scrisa de utilizator. Aceasta are urmatorul prototip:

```
int main(int argc, char **argv, char **arge)
```

unde:

- argc este numarul de argumente
- argv este un vector de siruri de caractere reprezentand valorile celor argc argumente. argv[argc] = NULL. Iar primul element este numele programului executabil (argv[0])
- Parametrul arge este un vector de siruri de caractere ce reprezinta variabilele de mediu si valorile acestora.

Al treilea argument poate sa lipseasca astfel ca putem declara si definii functia:

```
int maint(int argc, char **argv)
{
      ….....
    /* corp functie */
 }
```

Exercitii

1. Scrieti un program **echoargs** ce afiseaza in consola argumentele transmise in linia de comanda.

2. Scrieti un program **sortargs** ce afiseaza in consola argumentele transmise in linia de comanda in

ordine lexicografica. **Hint** – pentru a compara 2 siruri se va folosi functia strcmp definita in string.h

3. Scrieti un program **anagrama** ce primeste ca argumente 2 cuvinte si verifica daca sunt anagrame. Adica formate din aceleasi litere dar scrise in ordine inversa. Se va verifica daca s-au introdus numarul de argumente.

4. Scrieti un progam **sum** ce primeste ca argumente in linia de comanda numere si afiseaza suma acestora. Se va verifica daca exista suficiente argumente. Daca un argument nu este numar nu se va aduna la suma finala.

5. Scrieti un program ce citeste de la tastatura un numar n. Apoi n numere naturale. Sa se verifice daca sirul introdus este egal cu inversul sau.

6. Scrieti un program ce citeste de la tastatura 2 numere n si m. Si apoi o matrice de n x m elemente. Sa se inlocuiasca fiecare element de pe linile impare cu cel mai mic numar palindrom mai mare decat el si de pe fiecare linie para cu cel mai mare numar prim mai mic decat el. Sa scrie proceduri pentru inlocuirea elementelor de pe fiecare linie si proceduri pentru verificare daca un numar este prim si un numar este palindrom.

7. Scrieti un program ce citeste dintr-un 2 numere n si m. Si apoi o matrice de n x m elemente si roteste matricea la dreapta cu 90 de grade.