

CORSO DI SISTEMI OPERATIVI A - ESERCITAZIONE 1

1 Script

Riprendiamo con questo esercizio gli script Bash. Usare uno degli editor di testo a scelta (kate, kwrite, gedit) e scrivere il seguente script.

```
#!/bin/sh

if (test $# -ne 1)
then
    echo "Uso: $0 num";
    exit -1
fi

num=$1;
res=1;
for i in `seq 1 $num`
do
    res=$(( $res + $i ))
    echo "$i: $res";
done
echo "risultato $res"
```

Dopo aver impostato i permessi, eseguire lo script con diversi input. Infine, modificarlo in modo da fargli calcolare il fattoriale.

2 Compilazione dei programmi

Esercizio 1 (scrivere il seguente programma C e salvarlo come *hello.c*):

```
#include <stdio.h>

int main()
{
    printf("hello world!\n");

    return 0;
}
```

Dallo Shell, utilizzare *gcc* nel seguente modo: *gcc -o hello hello.c*

L'opzione *-o* seguita da *hello* comunica a gcc che deve creare un file eseguibile chiamato *hello*.

Verificare che il programma funziona, scrivendo nello Shell: *./hello*

In generale, la sintassi del comando gcc è: `gcc <opzioni> <filename>`

Oltre a `-o`, le opzioni più comuni sono:

- `-c` per creare il file oggetto anziché l'eseguibile: `gcc -c hello.c`
(verificare che il comando precedente genera il file oggetto `hello.o`)
- `-Wall` genera tutti i messaggi di warning che gcc può fornire
- `-pedantic` mostra tutti gli errori e i warning richiesti dallo standard ANSI C
- `-O -O1 -O2 -O3` servono per definire il livello di ottimizzazione (dal più basso al più alto)
- `-O0` per non avere nessuna ottimizzazione
- `-g` per un successivo debugging

Per creare l'eseguibile a partire dal file oggetto: `gcc hello.o -o hello`

Quest'ultima operazione prende il nome di linking, che in generale consiste nella risoluzione dei simboli tra programmi e l'inclusione di eventuali librerie.

Esercizio 2 (scrivere il seguente programma C e salvarlo come `squareroot.c`):

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

int main()
{
    double m, n;

    m = 12345;
    n = sqrt(m);
    printf("The square root of %f is %f.\n", m, n);

    return 0;
}
```

Verificare il fallimento del comando: `gcc -o squareroot squareroot.c`

Verificare il successo del comando: `gcc -o squareroot squareroot.c -lm`

Infatti non basta includere nel sorgente il file header `math.h`, serve l'opzione `-lm` per indicare al gcc di effettuare il linking della libreria matematica.

3 Debugging

Una applicazione per il debugging è `gdb`, che può essere messa in esecuzione dallo Shell.

Esercizio 3:

```

#include <stdio.h>

int main()
{
    double m, n;

    m = 1234.5678;
    n = 9.999;

    m = n;
    printf("m = %f, n = %f\n", m, n);

    return 0;
}

```

Il programma (per comodità chiamato *prova.c*) deve essere compilato con l'opzione *-g*:
gcc -g -o prova prova.c

Esercitazione con i comandi più comuni:

1. Invocare dalla stessa directory che contiene *prova* il programma: *gdb* (se si è in un'altra directory occorre specificare il path relativo). All'invocazione del programma ci si trova nello "shell" di *gdb*.
2. Caricamento del programma target: *file prova*
3. Esecuzione del programma: *run*
4. Le 10 righe attorno alla stringa *main*: *list main*
5. Inserire un breakpoint in corrispondenza di *main*: *break main* o *b main*
6. Informazioni sui breakpoint creati: *info breakpoints*
7. Se ora si utilizza il comando *run*, il programma viene eseguito fino al breakpoint
8. Esecuzione passo-passo: *step* (per ripetere basta premere invio)
9. Continuare l'esecuzione interrotta: *cont*
10. Ripetere i punti 2..6
11. Inserire un watchpoint per monitorare una variabile (*m*, ad esempio): *watch m*
12. Continuare l'esecuzione interrotta: *cont*
13. Invovare il comando *quit* (o *q*) per uscire dall'ambiente *gdb*.
 Si noti che questa volta l'esecuzione non arriva fino in fondo, ma si ferma appena il valore di *m* cambia; per continuare nuovamente, fino al successivo cambiamento del valore della variabile, ripetere il comando *cont* (basta premere invio).

4 Argomenti da riga di comando

Per passare informazioni alla funzione `main()` si utilizzano in genere gli argomenti della riga di comando, scritti nello Shell dopo il nome del programma da eseguire. Il C prevede l'uso di due speciali parametri: *argc* e *argv*, utilizzati per ricevere gli argomenti della riga di comando. Il parametro *argc* è un intero che contiene il numero di argomenti che si trovano nella riga di comando. Il suo valore è sempre almeno pari a 1 in quanto il nome del programma viene considerato come primo argomento. Il parametro *argv* è un puntatore a un array di puntatori a caratteri. Ogni elemento di questo array punta a un argomento della riga di comando.

Esercizio 4:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;

    printf("This program is %s and his number of arguments is %d.\n", argv[0], argc);
    for (i = 1; i < argc; i++)
    {
        printf("The argument number %d is %s.\n", i, argv[i]);
    }
    return 0;
}
```

Esiste anche un terzo parametro che consente di fornire al programma tutte le variabili d'ambiente: *envp*.

Esercizio 5:

```
#include <stdio.h>

int main(int argc, char **argv, char **envp)
{
    char *p;

    while (p = *envp)
    {
        printf("%s.\n", p);
        envp += 1;
    }

    return 0;
}
```

5 Funzioni per operare sulle variabili d'ambiente

Dallo Shell, creare la variabile d'ambiente `INCLUDE` assegnandole il valore “~” (oppure la propria home directory):

export INCLUDE=~/

Ciascun programma messo in esecuzione nello Shell è un processo figlio dello stesso processo Shell, da cui eredita le variabili d'ambiente. Un programma C può operare sulle proprie variabili d'ambiente grazie ad alcune funzioni fornite dalla Standard Library:

```
#include <stdlib.h>
char *getenv(const char *name);
int putenv(char *string);
```

Esercizio 6:

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

int main()
{
    char *path;

    /* ottieni e mostra il valore della variabile d'ambiente INCLUDE */
    path = getenv("INCLUDE");
    if (path != NULL)
    {
        printf("INCLUDE=%s\n", path);
    }

    /* imposta un nuovo valore per INCLUDE */
    if (putenv("INCLUDE=/src/include") != 0)
    {
        printf("putenv() failed setting INCLUDE\n");
        return -1;
    }

    /* ottieni e mostra il valore della variabile d'ambiente INCLUDE */
    path = getenv("INCLUDE");
    if (path != NULL)
    {
        printf("INCLUDE=%s\n", path);
    }

    return 0;
}
```

6 Generazione di Librerie Statiche

Il comando *ar* è usato per la generazione di librerie statiche. Ha la seguente sintassi:

```
ar rcs libmiallibreria.a file1.c file2.c ...
```

Per il significato delle varie opzioni (*r*, *c* e *s* sono quelle comunemente usate) cercare su *man ar*.

Copiare la directory *multifile* allegata al materiale dell'esercitazione nel proprio spazio di lavoro. Il suo contenuto è costituito dai file:

```
altra_operazione.h
altra_operazione.c
operazione.h
operazione.c
main.c
```

Compilare e generare i file oggetto da ciascun sorgente. Invocare poi il comando:

```
ar rcs liboperazioni.a operazione.c altra_operazione.c
```

Generare il file finale effettuando il linking della libreria *liboperazioni.a* invece che i corrispondenti file oggetto.

```
gcc -Wall -o operazioni main.o -L. -loperazioni -lm
```

L'opzione *-Ldirectorydellalibreria* specifica la directory in cui andare a cercare la libreria. Provare a togliere l'opzione *-L.* e verificare il risultato.

7 Makefile

Riprendere la directory *multifile* usata nel precedente esercizio. Dividere i file header (*.h*) ed i sorgenti (*.c*) rispettivamente in due sottodirectory *include* e *src*. Scrivere un *Makefile* che generi una libreria statica ed un file eseguibile come nell'esercizio sopra.

8 Compressione dei file

Per comprimere un file, utilizzare il comando: *gzip <nomefile>*

In tal modo viene creato il file compresso *<nomefile>.gz*

Per decomprimere un file *.gz*, utilizzare: *gzip -d <nomefile>.gz*

Per comprimere N file in un unico *.gz*, bisogna anzitutto creare un archivio tar:

```
tar -cvf <nomearchivio>.tar <nomefile1> <nomefile2> .. <nomefileN>
```

In tal modo viene creato il file *<nomearchivio>.tar*, che è una concatenazione dei file originari.

A questo punto si può utilizzare: *gzip <nomearchivio>.tar*

per ottenere il file compresso *<nomearchivio>.tar.gz*

Per comprimere un direttorio si opera come nel caso di N file, ma invece di specificare al comando *tar* N nomi di file si specifica il nome del direttorio target.

Per ricavare i file o il direttorio originari da un *.tar*, si utilizza la seguente sintassi:

```
tar -xvf <nomearchivio>.tar
```

Per spiegazioni esaustive, fare riferimento ai manuali di *gzip* e *tar*.