LabSO2022

ESERCIZI SVOLTI - I

Nota: è possibile siano presenti dei refusi, per cui valgono le informazioni fornite a lezione, mentre in alcuni casi sono presenti delle semplificazioni a puro titolo di esempio.

BASH

PARTE I

Scrivere delle sequenze di comandi (singola riga da eseguire tutta in blocco) che utilizzano come "input" il valore della variabile DATA per:

1. Stampa "T" (per True) o "F" (per False) a seconda che il valor rappresenti un file o cartella esistente.

2. Stampa "file", "cartella" o "?" a seconda che il valore rappresenti un file (esistente), una cartella (esistente) o una voce non presente nel file-system

3. Stampa il risultato di una semplice operazione aritmetica (es: '1 < 2') contenuta nel file indicato dal valore di DATA, oppure "?" se il file non esiste.

SOLUZIONI:

1. `[[ -e "${DATA}" ]] && echo "T" || echo "F"`

2. `[[ -f "${DATA}" ]] && echo "file" || ( [[ -d "${DATA}" ]] && echo "cartella" || echo "?" )`

3. `[[ -f "${DATA}" ]] && echo $(( $(cat "${DATA}") )) || echo "?"`

PARTE II

1. scrivere uno script che dato un qualunque numero di argomenti li restituisca in output in ordine inverso:

2. scrivere uno script che mostri il contenuto della cartella corrente in ordine inverso rispetto all'output generato da "ls" (che si può usare ma senza opzioni)

SOLUZIONI:

1. `# revargs.sh`

```
nargs=$#
largs=()
while [[ $# -gt 0 ]]; do
        largs+=("$1")
        shift
done
for l in ${!largs[@]}; do
        p=$(( $nargs-$l-1 ))
        echo "${largs[$p]}”
done
```

2. `# revls.sh`

```
./rev.sh $(ls)
```

PARTE III

stringrev.c

```
        #include <stdio.h>

        #define MEMSIZE 32
        char mem[MEMSIZE];

        char *stringrev(char *str) {
                char *i=str;
                int p=MEMSIZE;
                mem[--p]=0;
                // printf("p=%d, str='%s'\n", p, str);
                while (*i != 0) {
                        mem[--p]=*i;
                        // printf("p=%d, i=%d:%c\n", p, *i, *i);
                        i++;
                };
                return mem+p;
        }

        void main() {
                char *txt="Hello world!";
                printf("txt='%s', reversed='%s', mem='%s'\n", txt, stringrev(txt));
        }
```

Generare N figli che restituiscono al padre un numero casuale (da 0 a 255 compresi), il quale mostra poi il minimo e il massimo con i “pid” corrispondenti

```
// forkrnd.c

#define N 10

#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>

int main() {
	int f; // forking
	int i; // loop
	int r; // random
	int w; // wait
	int s; // status
	int c; // code
	int minval=256, minpid, maxval=-1, maxpid; // min 'n' max
	i=0; while (i++<N) {
		f=fork();
		if (f<0) { printf("[parent] ?Error\n"); exit(2); }
		if (f==0) {
			srand(time(NULL)+getpid());
			r=rand()%256;
			printf("[child]  #%d: %d\n", getpid(), r);
			exit(r);
		};
	};
	printf("[parent] waiting...\n");
	while ((w=wait(&s))>0) {
		c=WEXITSTATUS(s);
		printf("[parent] child #%d: %d\n", w, c);
		if (c<minval) { minval=c; minpid=w; };
		if (c>maxval) { maxval=c; maxpid=w; };
	};
	printf("[parent] min=%d (#%d), max=%d (%d)\n", minval, minpid, maxval, maxpid);
	return 0;
}
```

Generare un figlio per ogni argomento ed eseguirlo come comando (es.: ./execvp pwd ls) con stdout “normale” o tutti su un unico file

```
// execvp.c I
#define MAXLEN 100
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
int main(int argc, char **argv) {
        int a=0, num=argc-1, f;
        char cmd[MAXLEN], *lst[2];
         printf("[#%d] %d command(s)\n", getpid(), num); fflush(stdout);
        while (a++<num) {
                printf("[#%d] Command %d: '%s'...\n", getpid(), a, argv[a]); fflush(stdout);
                f=fork();
                if (f==0) {
                        strcpy(cmd, argv[a]); lst[0]=cmd; lst[1]=NULL;
                        printf("[#%d] '%s'\n", getpid(), cmd); fflush(stdout);
                        execvp(cmd, lst); // end of code
                };
        };
        printf("Waiting for children... ");
        while (wait(NULL)>0);
        printf("done.\n");
}
```

```
// execvp.c II
#define MAXLEN 100
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
int main(int argc, char **argv) {
        int a=0, num=argc-1, f;
        char cmd[MAXLEN], *lst[2];
        int outfile = open("/tmp/out.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );
         printf("[#%d] %d command(s)\n", getpid(), num); fflush(stdout);
        while (a++<num) {
                printf("[#%d] Command %d: '%s'...\n", getpid(), a, argv[a]); fflush(stdout);
                f=fork();
                if (f==0) {
                        strcpy(cmd, argv[a]); lst[0]=cmd; lst[1]=NULL;
                        dup2(outfile, 1); // copy outfile to FD 1
                        execvp(cmd, lst);
                };
        };
        printf("Waiting for children... ");
        while (wait(NULL)>0);
        printf("done.\n");
}
```

Generare due figli per due argomenti ed eseguirli in modo da avere l'effetto del piping bash attraverso però un file su disco (es.: … ls wc come se fosse ls | wc)

```
#define MAXLEN 100

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {
        int a=0, num=argc-1, f;
        char cmd[MAXLEN], *lst[2];
        int passfile;
        if (num!=2) {
                printf("?Error. Usage: ... <cmd1> <cmd2>, to emulate <cmd1> | <cmd2>\n");
                exit(2);
        };

        printf("[%d] running '%s'...\n", getpid(), argv[1]);
        f=fork();
        if (f==0) {
                passfile = open("/tmp/pass.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );
                if (passfile<0) { printf("?Error. Cannot open output file\n"); exit(3); };
                if (dup2(passfile, 1)<0) { printf("?Error. Cannot dup2 output\n"); exit(4); };
                close(passfile); // copy passfile to FD 1
                strcpy(cmd, argv[1]); lst[0]=cmd; lst[1]=NULL;
                execvp(cmd, lst);
        };
        while (wait(NULL)>0); printf("[%d] child %d ok\n", getpid(), f);  sleep(2);

        printf("[%d] running '%s'...\n", getpid(), argv[2]);
        f=fork();
        if (f==0) {
                passfile = open("/tmp/pass.txt", O_RDONLY, S_IRUSR | S_IWUSR );
                if (passfile<0) { printf("?Error. Cannot open input file\n"); exit(3); };
                if (dup2(passfile, 0)<0) { printf("?Error. Cannot dup2 input\n"); exit(5);  };
                close(passfile); // copy passfile to FD 0
                strcpy(cmd, argv[2]); lst[0]=cmd; lst[1]=NULL;
                execvp(cmd, lst);
        };
        while (wait(NULL)>0); printf("[%d] child %d ok\n", getpid(), f);  sleep(2);

        printf("[%d] done.\n", getpid());
}
```