- Esercizi Script, Pipe, Fork, Exec-

Esercizio n° 1

- Scrivere uno script che concatena tutti i file regolari che contengono una parola che inizia con la prima lettera del file e finisce con la seconda lettera dell'ultima riga.

Soluzione

```
#!/bin/bash
l= $( ls *.txt )
for i in $l ; do
    a=cat $i | head -1 | cut -c -1
    b=cat $i | tail -1 | cut -c -2
    c=grep \<$a[a-z]*$b\>" $i
    if [ -z "$c" ] ; then
            echo no match
    else
            cat $i >> file
    fi
done
```

Esercizio n° 2

- Scrivere uno programma che crea due processi figlio, il padre legge i caratteri dal file Input.txt e li passa 2 alla volta al primo figlio. Questo passa il primo carattere al secondo figlio solo se primo e secondo carattere sono uguali. Il secondo figlio stamperà i caratteri ricevuti in Output.txt

Esercizio n° 3

- Sia S un processo server che effettua semplici operazioni aritmetiche e C un processo figlio di S (client) che prende in input i due operandi e l'operatore e li invia al server per ottenere il risultato. La comunicazione da Client a Server sarà effettuata tramite pipe mentre il risultato elaborato da S sarà mostrato su standard output

Soluzione

```
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h> #include <stdio.h>
#include <signal.h>
main() {
        int fd[2], flag=1; pid_t pid; float res; char opz; key_t my_key=1234;
struct richiesta { float op1; float op2; char op; char msg[5]; pid_t pid; } msg;

pipe(fd);
pid=fork();
if(pid==0) {
        close(fd[0]);
        while(1) {
                flag=1;
                while(flag) {
                        printf("\nDigitare:\nC per calcolare un'espressione\n");
                        printf("D per uscire e disconnettere il server\n\n"); scanf("%c",&opz);
                        opz=toupper(opz);
```

Soluzione

```
if(opz=='C' || opz=='D')  flag=0;
}
if(opz=='D')
{
        sprintf(msg.msg,"FINE");
        write(fd[1],&msg,sizeof(msg));
        close(fd[1]);
        exit(0);
}
flag=1;
    while(flag) {
        printf("Digitare l'espressione da calcolare\n\n");
        scanf("%f %c %f", &msg.op1, &msg.op, &msg.op2);
        if(msg.op!='+' && msg.op!='-' && msg.op!='*' && msg.op!='/')
                printf("Le operazioni possibili sono +, -, *, /\n\n");
        else (flag=0);
    }
    write(fd[1],&msg,sizeof(msg)); sleep(2);
    }
}
```

Soluzione

```
else{
        close(fd[1]);
        while(1) {
                read(fd[0],&msg,sizeof(msg));
                if(strcmp(msg.msg,"FINE")==0) {
                        close(fd[0]);
                        printf("Fine esecuzione\n");
                        exit(0);
                }
                switch (msg.op) {
                        case '+': res=msg.op1+msg.op2; break;
                        case '-': res=msg.op1-msg.op2; break;
                        case '*': res=msg.op1*msg.op2; break;
                        case '/': res=msg.op1/msg.op2;
                }
                printf("Risultato = %f \n",res);
        }
    }
}
```

Esercizio n° 4

- Scrivere un programma che crea due processi figli, il primo figlio stamperà il listing (ls -l) della directory corrente in un file passato come parametro; il secondo figlio aprirà quel file stampandone il contenuto in STDOUT.

Soluzione

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>#include <errno.h>
#include <fcntl.h>
#define MAXBUF 1000

int main(int argc, char **argv)      {
    pid_t val1, val2;
    int fd;
    char buff[MAXBUF];

    if ((val1=fork())==0) {
        fd = open(argv[1],O_WRONLY | O_CREAT, S_IWUSR | S_IRUSR);
        if(dup2(fd, STDOUT_FILENO)!=STDOUT_FILENO)
                perror("dup2"), exit(1);
        execl("/bin/ls", "ls", "-l", 0);
    }
    else {
        if ((val2=fork())==0) {
                printf("Child2 PID: %d\n", (int) getpid());
                fd = open("myfile",O_RDONLY);
                read(fd,buff,MAXBUF);
                printf("buffer %s",buff);
        }
        waitpid(val1,NULL,0); waitpid(val2,NULL,0);
    }
}
```