

## **LabSO2021 - Simulazione Esame parte I (prima prova parziale)**

### **PROBLEMI [punti 30]**

#### **Problema 1 [punti 3]**

Scrivere uno script bash denominato “*sub.sh*” che generi un numero di sottoprocessi pari a quello indicato al primo e unico argomento passato (fino a un massimo di 10).

Esempio: `./sub.sh 3` deve generare 3 sottoprocessi

#### **Problema 2 [punti 6]**

Utilizzando il comando “`read`”, che legge l’input dell’utente e lo può memorizzare in una variabile (ad esempio `read data` attende l’input dell’utente, immissione testo + INVIO, e lo memorizza nella variabile `data`) scrivere un alias denominato “*linecount*” bash che chiede in input il nome di un file (con percorso relativo, assoluto o senza) e se tale file esiste ne mostra il numero di righe, altrimenti fornisce un messaggio d’errore. Qualunque variabile utilizzata non deve essere visibile nella shell chiamante (o sovrascriverne un’eventuale omonima)

Esempio: `linecount` attende un input, ad esempio `/tmp/sub.sh`, poi deve mostrare il numero di righe del file `/tmp/sub.sh` se esiste o “`?File not found`” altrimenti. Se il comando “`read`” utilizza la variabile “`data`” (o comunque questa è utilizzata in qualche modo) e questa prima dell’invocazione dell’alias ha un certo valore, tale valore non deve essere modificato.

#### **Problema 3 [punti 21]**

Scrivere un’applicazione C in un singolo file “*firstchars.c*” che accetti in input una lista di file (fino a 10: con o senza path relativo o assoluto) e per ciascuno di essi:

- generi un sottoprocesso [punti 6]
- dentro ogni sottoprocesso si accede al primo byte del file e lo si visualizza [punti 2]
- l’invio del byte resta così com’è se è una lettera o una cifra, altrimenti si invia un carattere “-” (operatore “meno”) [punti 2]
- si invia il carattere come indicato se non ci sono problemi su *stdout*, altrimenti si invia un carattere “?” (punto interrogativo) su *stderr* ~~[punti 5]~~ [punti 3]
- Non si lasciano processi “zombie” e l’output sia su *stdout* che su *stderr* avviene senza nessun “a capo” (quindi l’output compone una sorta di “stringa” a video e non un carattere per linea) che viene aggiunto solo quando tutti i sottoprocessi hanno terminato (per cui il prompt della shell compare comunque poi su una nuova linea) ~~[punti 4]~~ [punti 6]

Creare un makefile con una regola che compili opportunamente il codice generando un binario denominato “firstchars” [punti 2]

Esempio:

Supponendo di avere nella cartelle “testi” i file “uno.txt” che contiene il testo “one”, “tre.txt” che contiene il testo “three” e niente altro in caso di soluzione completa si devono avere i seguenti risultati (a meno dell’ordine dei caratteri stampati):

```
./firstchars testi/uno.txt testi/due.txt testi/tre.txt
```

a video si deve vedere o?t e il prompt successivo è in una nuova riga

```
./firstchars testi/uno.txt testi/due.txt testi/tre.txt 2>/dev/null
```

a video si deve vedere ot e il prompt successivo è in una nuova riga

## **LabSO2021 - Simulazione Esame parte I (prima prova parziale)**

### **SOLUZIONI [punti 30]** *(sono possibili altre soluzioni)*

#### Soluzione 1

```
#!/usr/bin/env bash
# sub.sh

echo "BASHPID=$BASHPID"
if [[ $# -ne 1 ]]; then echo "?Usage: $0 <num>" 1>&2 ; exit 2 ; fi

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

#### Soluzione 2

```
alias linecount='echo $( read filename; [[ -f "${filename}" ]] && ( cat "${filename}" | wc -l
) || echo "?File not found" 1>&2 )'
```

#### Soluzione 3

```
# Makefile
firstchars: firstchars.c
    @echo "Compiling..."
    @gcc -o firstchars firstchars.c
    @echo "Done."

/* firstchars.c */

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

void printout(char msg) {
    fprintf(stdout, "%c", msg); fflush(stdout);
}

void printerr(char msg) {
    fprintf(stderr, "%c", msg); fflush(stderr);
}
```

```

int main(int argc, char **argv) {
    // Check syntax:
    if (argc<2 || argc>11) {
        fprintf(stderr, "?Usage: %s <file1> <file2> ... (upto 10)\n", argv[0]); exit(2);
    };

    int p;    // loop counter
    char *fn; // filename placeholder
    char ch;  // single character to read
    int cn;   // read's return value
    int f;    // fork's id
    int fd;   // file descriptor

    for (p=1; p<argc; p++) {
        fn=argv[p];
        f=fork();
        if (f<0) { fprintf(stderr, "?SYSEERRR. Fork error\n"); exit(3); };
        if (f==0) {
            fd = open(fn, O_RDONLY);
            if (fd<0) {
                printerr('?');
            } else {
                cn=read(fd, &ch, 1);
                if (cn!=1) {
                    printerr('?');
                } else {
                    if ( (ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || (ch>='0' && ch<='9') ) {
                        printout(ch);
                    } else {
                        printout('-');
                    };
                }
                close(fd);
            };
        };
        exit(0);
    };
};

while(wait(NULL)>0);
fprintf(stdout, "\n");
fprintf(stderr, "\n");
}

```

**Codici parziali *progressivi* per i sottopunti.**

Struttura base + un sottoprocesso per ogni argomento e verifica file:

```

/* firstchars-1.c */
#include <stdio.h> // (f)printf
#include <unistd.h> // fork
#include <stdlib.h> // exit
#include <fcntl.h> // open

int main(int argc, char **argv) {
    // Check syntax:
    if (argc<2 || argc>11) {
        fprintf(stderr, "?Usage: %s <file1> <file2> ... (upto 10)\n", argv[0]); exit(2);
    };
};

```

```

};
int p;    // loop counter
char *fn; // filename placeholder
int f;    // fork's id
int fd;   // file descriptor

for (p=1; p<argc; p++) {
    fn=argv[p];
    f=fork();
    if (f<0) { fprintf(stderr, "?SYSERRR. Fork error\n"); exit(3); };
    if (f==0) {
        fd = open(fn, O_RDONLY);
        if (fd<0) {
            fprintf(stderr, "?Error opening %s\n", fn);
        } else {
            fprintf(stdout, "[%d] Ok: %s\n", getpid(), fn);
        };
        exit(0);
    };
    fprintf(stdout, "[%d] Main process.\n", getpid());
};
}

```

+ accede al primo byte di ogni file e lo visualizza:

```

/* firstchars-2.c */
#include <stdio.h> // (f)printf
#include <unistd.h> // fork
#include <stdlib.h> // exit
#include <fcntl.h> // open

int main(int argc, char **argv) {
    // Check syntax:
    if (argc<2 || argc>11) {
        fprintf(stderr, "?Usage: %s <file1> <file2> ... (upto 10)\n", argv[0]); exit(2);
    };

    int p;    // loop counter
    char *fn; // filename placeholder
    char ch; // single character to read
    int cn;  // read's return value
    int f;    // fork's id
    int fd;   // file descriptor

    for (p=1; p<argc; p++) {
        fn=argv[p];
        f=fork();
        if (f<0) { fprintf(stderr, "?SYSERRR. Fork error\n"); exit(3); };
        if (f==0) {
            fd = open(fn, O_RDONLY);
            if (fd<0) {
                fprintf(stderr, "?Error opening %s\n", fn);
            } else {
                cn=read(fd, &ch, 1);
                if (cn==1) fprintf(stdout, "%c\n", ch);
            };
            exit(0);
        };
        fprintf(stdout, "[%d] Main process.\n", getpid());
    };
}

```

```
};
}
```

+ se non è “lettera o cifra” visualizza “-”:

```
/* firstchars-3.c */
#include <stdio.h> // (f)printf
#include <unistd.h> // fork
#include <stdlib.h> // exit
#include <fcntl.h> // open

int main(int argc, char **argv) {
    // Check syntax:
    if (argc<2 || argc>11) {
        fprintf(stderr, "?Usage: %s <file1> <file2> ... (upto 10)\n", argv[0]); exit(2);
    };

    int p; // loop counter
    char *fn; // filename placeholder
    char ch; // single character to read
    int cn; // read's return value
    int f; // fork's id
    int fd; // file descriptor

    for (p=1; p<argc; p++) {
        fn=argv[p];
        f=fork();
        if (f<0) { fprintf(stderr, "?SYSEERRR. Fork error\n"); exit(3); };
        if (f==0) {
            fd = open(fn, O_RDONLY);
            if (fd<0) {
                fprintf(stderr, "?Error opening %s\n", fn);
            } else {
                cn=read(fd, &ch, 1);
                if (cn==1) {
                    if ( !( (ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || (ch>='0' && ch<='9') ) ) ch='-';
                    fprintf(stdout, "%c\n", ch);
                };
            };
            exit(0);
        };
        fprintf(stdout, "[%d] Main process.\n", getpid());
    };
}
```

+ in caso d’errore mostra “?” su *stderr* e il resto su *stdout*:

```
/* firstchars-4.c */
#include <stdio.h> // (f)printf
#include <unistd.h> // fork
#include <stdlib.h> // exit
#include <fcntl.h> // open

int main(int argc, char **argv) {
    // Check syntax:
    if (argc<2 || argc>11) {
        fprintf(stderr, "?Usage: %s <file1> <file2> ... (upto 10)\n", argv[0]); exit(2);
    };
}
```

```

};

int p;    // loop counter
char *fn; // filename placeholder
char ch;  // single character to read
int cn;   // read's return value
int f;    // fork's id
int fd;   // file descriptor

for (p=1; p<argc; p++) {
    fn=argv[p];
    f=fork();
    if (f<0) { fprintf(stderr, "?SYSEERRR. Fork error\n"); exit(3); };
    if (f==0) {
        fd = open(fn, O_RDONLY);
        if (fd<0) {
            fprintf(stderr, "%c\n", '?');
        } else {
            cn=read(fd, &ch, 1);
            if (cn==1) {
                if ( !( (ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || (ch>='0' && ch<='9') ) )
ch='-';
                fprintf(stdout, "%c\n", ch);
            } else {
                fprintf(stderr, "%c\n", '?');
            };
        };
        exit(0);
    };
    fprintf(stdout, "[%d] Main process.\n", getpid());
};
}

```

+ altri dettagli: *vedi soluzione completa*