

Mar 11, 05 12:17

sol20040312_1.c

Page 1/1

```
/* Possibile soluzione della prova in itinere del 12 marzo 2004
* analizzata nelle lezioni di Laboratorio di Sistemi Operativi
* del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.1 2005/03/11 09:00:06 valealex Exp $ */
/* $Log: sol20040312.c,v $
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
* */
```

Mar 11, 05 12:17

sol20040312_2.c

Page 1/1

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.2 2005/03/11 09:15:04 valealex Exp $ */

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

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int main(int argc, char **argv, char **envp) {
    int i;

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    return(0);
}

/* $Log: sol20040312.c,v $
 * Revision 1.2 2005/03/11 09:15:04 valealex
 * Prelievo e controllo degli argomenti della linea di comando
 *
 * Revision 1.1 2005/03/11 09:00:06 valealex
 * Initial revision
 * */

```

Mar 11, 05 12:17

sol20040312_3.c

Page 1/2

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.3 2005/03/11 09:24:57 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int main(int argc, char **argv, char **envp) {
    int i;

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione
     * del figlio di indice 'k' si ottiene attendendo un
     * carattere dalla pipe che lo collega al figlio di
     * indice 'k-1'. */
    Npipe = N; /* una pipe in uscita da ogni figlio */
    Pfds = malloc(Npipe*sizeof(pipe_t));
    if (Pfds==NULL) {
        fprintf(stderr, "Impossibile allocare Pfds\n");
        return(1);
    }
    for (i=0; i<Npipe; i++) {
        if (pipe(Pfds[i])!=0) {
            fprintf(stderr, "Impossibile creare la pipe %d\n", i);
            return(1);
        }
    }
}

```

Friday March 11, 2005

sol20040312_3.c

Mar 11, 05 12:17

sol20040312_3.c

Page 2/2

```

    }
    return(0);
}

/* $Log: sol20040312.c,v $
 * Revision 1.3 2005/03/11 09:24:57 valealex
 * Creazione pipe per sincronizzazione
 *
 * Revision 1.2 2005/03/11 09:15:04 valealex
 * Prelievo e controllo degli argomenti della linea di comando
 *
 * Revision 1.1 2005/03/11 09:00:06 valealex
 * Initial revision
 * */

```

3/29

Mar 11, 05 12:17

sol20040312_4.c

Page 1/2

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.4 2005/03/11 09:38:56 valealex Exp $ */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

typedef int pipe_t[2]; /* tipo per pipe */

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
indice 'i' */

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

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
 * fra i processi figli: si utilizza una pipe per
 * collegare ogni figlio al successivo; la sincronizzazione
 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
    Npipe = N; /* una pipe in uscita da ogni figlio */
    Pfds = malloc(Npipe*sizeof(pipe_t));
    if (Pfds==NULL) {
        fprintf(stderr, "Impossibile allocare Pfds\n");

```

Mar 11, 05 12:17

sol20040312_4.c

Page 2/2

```

        return(1);
    }
    for (i=0; i<Npipe; i++) {
        if (pipe(Pfds[i])!=0) {
            fprintf(stderr, "Impossibile creare la pipe %d\n", i);
            return(1);
        }
    }
    /* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
    Nfigli = N;
    Pids=malloc(Nfigli*sizeof(int));
    if (Pids==NULL) {
        fprintf(stderr, "Impossibile allocare Pids\n");
        return(1);
    }
    for (i=0; i<Nfigli; i++) {
        Pids[i]=fork();
        switch (Pids[i]) {
            case 0: /* figlio di indice i */
                return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo */
            case -1:
                fprintf(stderr, "Impossibile forkare il processo %d\n", i);
                return(1);
                break;
        }
    }
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
    return(0);
}

/* Funzione figlio, per ora vuota */
int figlio(int indice) {
    return (0);
}

/* $Log: sol20040312.c,v $
 * Revision 1.4 2005/03/11 09:38:56 valealex
 * Creati i processi figli
 * Eliminato un errore di digitazione nella malloc del vettore di pipe
 *
 * Revision 1.3 2005/03/11 09:24:57 valealex
 * Creazione pipe per sincronizzazione
 *
 * Revision 1.2 2005/03/11 09:15:04 valealex
 * Prelievo e controllo degli argomenti della linea di comando
 *
 * Revision 1.1 2005/03/11 09:00:06 valealex
 * Initial revision
 * */

```

Mar 11, 05 12:17

sol20040312_5.c

Page 1/3

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.5 2005/03/11 09:47:55 valealex Exp $ */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

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

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione
     * del figlio di indice 'k' si ottiene attendendo un
     * carattere dalla pipe che lo collega al figlio di
     * indice 'k-1'. */

```

Mar 11, 05 12:17

sol20040312_5.c

Page 2/3

```

Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if (write(Pfds[0][SCRIT], &ok, 1) != 1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
return(0);
}

/* Funzione figlio, per ora vuota */
int figlio(int indice) {

```

Mar 11, 05 12:17

sol20040312_5.c

Page 3/3

```
    return (0);
}
/* $Log: sol20040312.c,v $
* Revision 1.5  2005/03/11 09:47:55  valealex
* Inserito primo start del processo padre
*
* Revision 1.4  2005/03/11 09:38:56  valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3  2005/03/11 09:24:57  valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2  2005/03/11 09:15:04  valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1  2005/03/11 09:00:06  valealex
* Initial revision
* */
```

Mar 11, 05 12:17

sol20040312_6.c

Page 1/3

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.6 2005/03/11 09:55:02 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione
     * del figlio di indice 'k' si ottiene attendendo un
     * carattere dalla pipe che lo collega al figlio di

```

Friday March 11, 2005

Mar 11, 05 12:17

sol20040312_6.c

Page 2/3

```

    * indice 'k-1'. */
    Npipe = N; /* una pipe in uscita da ogni figlio */
    Pfds = malloc(Npipe*sizeof(pipe_t));
    if (Pfds==NULL) {
        fprintf(stderr, "Impossibile allocare Pfds\n");
        return(1);
    }
    for (i=0; i<Npipe; i++) {
        if (pipe(Pfds[i])!=0) {
            fprintf(stderr, "Impossibile creare la pipe %d\n", i);
            return(1);
        }
    }
    /* Creazione processi figli:
     * dal testo, bisogna creare un processo figlio per ogni carattere
     * quindi Nfigli=N
     * Visto che il padre deve stampare i valori di uscita dei processi
     * figli, si devono memorizzare i pid in ordine di indice per poi
     * abbinare i risultati delle wait. L'array di pid deve essere
     * allocato dinamicamente */
    Nfigli = N;
    Pids=malloc(Nfigli*sizeof(int));
    if (Pids==NULL) {
        fprintf(stderr, "Impossibile allocare Pids\n");
        return(1);
    }
    for (i=0; i<Nfigli; i++) {
        Pids[i]=fork();
        switch (Pids[i]) {
            case 0: /* figlio di indice i */
                return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
            case -1:
                fprintf(stderr, "Impossibile forkare il processo %d\n", i);
                return(1);
                break;
        }
    }
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {
    int pid_x;
    int status;

```

sol20040312_6.c

7/29

Mar 11, 05 12:17

sol20040312_6.c

Page 3/3

```
        pid_x = wait(&status);
        /* abbinamento delle informazioni all'indice del figlio */
        for (j=0; j<Nfigli; j++) {
            if (pid_x==Pids[j]) {
                printf("Il figlio che cerca il carattere %c ha trovato %d linee\n",
                    C[i],WEXITSTATUS(status));
            }
        }
    }
    return(0);
}

/* Funzione figlio, per ora vuota */
int figlio(int indice) {
    return (0);
}

/* $Log: sol20040312.c,v $
 * Revision 1.6  2005/03/11 09:55:02  valealex
 * Ciclo di recupero exit status e abbinamento ai pid
 *
 * Revision 1.5  2005/03/11 09:47:55  valealex
 * Inserito primo start del processo padre
 *
 * Revision 1.4  2005/03/11 09:38:56  valealex
 * Creati i processi figli
 * Eliminato un errore di digitazione nella malloc del vettore di pipe
 *
 * Revision 1.3  2005/03/11 09:24:57  valealex
 * Creazione pipe per sincronizzazione
 *
 * Revision 1.2  2005/03/11 09:15:04  valealex
 * Prelievo e controllo degli argomenti della linea di comando
 *
 * Revision 1.1  2005/03/11 09:00:06  valealex
 * Initial revision
 */
```

Mar 11, 05 12:17

sol20040312_7.c

Page 1/4

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.7 2005/03/11 10:01:39 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_7.c

Page 2/4

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_7.c

Page 3/4

```

        int pid_x;
        int status;

        pid_x = wait(&status);
        /* abbinamento delle informazioni all'indice del figlio */
        for (j=0; j<Nfigli; j++) {
            if (pid_x==Pids[j]) {
                if (WEXITSTATUS(status)==255) {
                    /* convenzione: segnalazione d'errore */
                    printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                                C[i]);
                } else {
                    printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                                C[i],WEXITSTATUS(status));
                }
            }
        }
        return(0);
    }

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
        return(-1);
    }
    return (0);
}

/* $Log: sol20040312.c,v $
* Revision 1.7  2005/03/11 10:01:39  valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6  2005/03/11 09:55:02  valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5  2005/03/11 09:47:55  valealex
* Inserito primo start del processo padre
*
* Revision 1.4  2005/03/11 09:38:56  valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3  2005/03/11 09:24:57  valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2  2005/03/11 09:15:04  valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1  2005/03/11 09:00:06  valealex
* Initial revision

```

Friday March 11, 2005

sol20040312_7.c

Mar 11, 05 12:17

sol20040312_7.c

Page 4/4

* */

10/29

Mar 11, 05 12:17

sol20040312_8.c

Page 1/4

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.8 2005/03/11 10:07:23 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_8.c

Page 2/4

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
                                e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_8.c

Page 3/4

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[i]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[i],WEXITSTATUS(status));
        }
    }
}
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
        return(-1);
    }
    /* il file deve essere letto per linee */
    linenum=1; /* prima linea */
    for (;;) {
        nr=read(fd,&ch,1);
        if (nr==0) {
            return(retval);
        }
        if (nr<0) {
            /* errore */
#ifdef DEBUG
            fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
            return(-1);
        }
        /* carattere letto, controllo se fine linea */
        if (ch=='\n') {
            linenum++;
        }
    }
    /* mai qui! */
    return (-1);
}

```

Mar 11, 05 12:17

sol20040312_8.c

Page 4/4

```

}
/* $Log: sol20040312.c,v $
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7 2005/03/11 10:01:39 valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5 2005/03/11 09:47:55 valealex
* Inserito primo start del processo padre
*
* Revision 1.4 2005/03/11 09:38:56 valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3 2005/03/11 09:24:57 valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2 2005/03/11 09:15:04 valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
*/

```

Mar 11, 05 12:17

sol20040312_9.c

Page 1/4

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.9 2005/03/11 10:13:25 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_9.c

Page 2/4

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_9.c

Page 3/4

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[i]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[i],WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
        return(-1);
    }
    /* il file deve essere letto per linee */
    linenum=1; /* prima linea */
    flag=0;
    for (;;) {
        nr=read(fd,&ch,1);
        if (nr==0) {
            return(retval);
        }
        if (nr<0) {
            /* errore */
#ifdef DEBUG
            fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
            return(-1);
        }
        /* carattere letto, controllo se fine linea */
        if (ch=='\n') {
            linenum++;
            flag=0;
        }
    }
}

```

Mar 11, 05 12:17

sol20040312_9.c

Page 4/4

```

/* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
    * il numero di linea su stdout. Se il carattere si trova piu'
    * volte nella stessa linea devo riportare il numero solo
    * una volta, quindi uso un flag che controllo a fine linea */
if ((ch==C[indice])&&flag==0) {
    flag=1;
    printf("Linea %d contiene %c\n", linenum, C[indice]);
}
/* mai qui! */
return (-1);
}

/* $Log: sol20040312.c,v $
* Revision 1.9 2005/03/11 10:13:25 valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7 2005/03/11 10:01:39 valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5 2005/03/11 09:47:55 valealex
* Inserito primo start del processo padre
*
* Revision 1.4 2005/03/11 09:38:56 valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3 2005/03/11 09:24:57 valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2 2005/03/11 09:15:04 valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
* */

```

Mar 11, 05 12:17

sol20040312_10.c

Page 1/5

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.10 2005/03/11 10:47:53 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_10.c

Page 2/5

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_10.c

Page 3/5

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[j]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[j], WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */
    int nrp; /* numero di car letti da read su pipe */
    char dummy; /* carattere di appoggio per lett/scritt pipe */

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
        return(-1);
    }
    /* il file deve essere letto per linee */
    linenum=1; /* prima linea */
    flag=0;
    for (;;) {
        nr=read(fd,&ch,1);
        if (nr==0) {
            return(retval);
        }
        if (nr<0) {
            /* errore */
#ifdef DEBUG
            fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
            return(-1);
        }
        /* carattere letto, controllo se fine linea */
        if (ch=='\n') {
            linenum++;

```

Mar 11, 05 12:17

sol20040312_10.c

Page 4/5

```

        flag=0;
    }
    /* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
    * il numero di linea su stdout. Se il carattere si trova piu'
    * volte nella stessa linea devo riportare il numero solo
    * una volta, quindi uso un flag che controllo a fine linea */
    if ((ch==C[indice])&&flag==0) {
        flag=1;
        /* aggiungo sincronizzazione, attendo carattere da pipe
        * con indice i */
        nrp=read(Pfds[indice][LETT],&dummy,1);
        if (nrp==0) {
            /* pipe chiusa! cosa fare? */
        }
        if (nrp==-1) {
            /* errore, segnalare */
#ifdef DEBUG
            fprintf(stderr, "Errore pipe di sincronizzazione\n");
#endif
            return(-1);
        }
        /* se sono qui posso scrivere */
        printf("Linea %d contiene %c\n", linenum, C[indice]);
        /* ora do il via al figlio successivo: l'indice della pi
pe
        * in cui scrivere e' indice+1 a meno che no si tratti d
ell'ultimo
        * figlio (quello con indice=(Nfigli-1)) che deve scrive
re sulla
        * pipe di indice 0 */
        if (indice!=Nfigli-1) {
            write(Pfds[indice+1][SCRIT],&dummy,1);
        } else {
            write(Pfds[0][SCRIT],&dummy,1);
        }
    }
}
/* mai qui! */
return (-1);
}

/* $Log: sol20040312.c,v $
* Revision 1.10 2005/03/11 10:47:53 valealex
* Abbozzo di sincronizzazione con le pipe.
* Provoca un deadlock quando un figlio termina prima degli altri in quanto
* la pipe sulla quale dovrebbe inviare il carattere di start non e'
* stata chiusa dagli altri fratelli
* Corretto errore indice i->j nella stampa del padre
*
* Revision 1.9 2005/03/11 10:13:25 valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7 2005/03/11 10:01:39 valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid

```

Mar 11, 05 12:17

sol20040312_10.c

Page 5/5

```
*  
* Revision 1.5 2005/03/11 09:47:55 valealex  
* Inserito primo start del processo padre  
*  
* Revision 1.4 2005/03/11 09:38:56 valealex  
* Creati i processi figli  
* Eliminato un errore di digitazione nella malloc del vettore di pipe  
*  
* Revision 1.3 2005/03/11 09:24:57 valealex  
* Creazione pipe per sincronizzazione  
*  
* Revision 1.2 2005/03/11 09:15:04 valealex  
* Prelievo e controllo degli argomenti della linea di comando  
*  
* Revision 1.1 2005/03/11 09:00:06 valealex  
* Initial revision  
* /
```

Mar 11, 05 12:17

sol20040312_11.c

Page 1/5

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.11 2005/03/11 10:53:52 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_11.c

Page 2/5

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
                               e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_11.c

Page 3/5

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[j]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[j], WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */
    int nrp; /* numero di car letti da read su pipe */
    char dummy; /* carattere di appoggio per lett/scritt pipe */

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
        return(-1);
    }
    /* il file deve essere letto per linee */
    linenum=1; /* prima linea */
    retval=0; /* linee che soddisfano=0 */
    flag=0;
    for (;;) {
        nr=read(fd,&ch,1);
        if (nr==0) {
            return(retval);
        }
        if (nr<0) {
            /* errore */
#ifdef DEBUG
            fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
            return(-1);
        }
        /* carattere letto, controllo se fine linea */
        if (ch=='\n') {

```

Mar 11, 05 12:17

sol20040312_11.c

Page 4/5

```

        linenum++;
        flag=0;
    }
    /* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
    * il numero di linea su stdout. Se il carattere si trova piu'
    * volte nella stessa linea devo riportare il numero solo
    * una volta, quindi uso un flag che controllo a fine linea */
    if ((ch==C[indice])&&flag==0) {
        flag=1;
        retval++; /* incremento contatore */
        /* aggiungo sincronizzazione, attendo carattere da pipe
        * con indice i */
        nrp=read(Pfds[indice][LETT],&dummy,1);
        if (nrp==0) {
            /* pipe chiusa! cosa fare? */
        }
        if (nrp==-1) {
            /* errore, segnalare */
#ifdef DEBUG
            fprintf(stderr, "Errore pipe di sincronizzazione\n");
#endif
            return(-1);
        }
        /* se sono qui posso scrivere */
        printf("Linea %d contiene %c\n", linenum, C[indice]);
        /* ora do il via al figlio successivo: l'indice della pi
pe
        * in cui scrivere e' indice+1 a meno che no si tratti d
ell'ultimo
        * figlio (quello con indice=(Nfigli-1)) che deve scrive
re sulla
        * pipe di indice 0 */
        if (indice!=Nfigli-1) {
            write(Pfds[indice+1][SCRIT],&dummy,1);
        } else {
            write(Pfds[0][SCRIT],&dummy,1);
        }
    }
}
/* mai qui! */
return (-1);
}

/* $Log: sol20040312.c,v $
* Revision 1.11 2005/03/11 10:53:52 valealex
* Calcolo di retval come specificato nel testo
*
* Revision 1.10 2005/03/11 10:47:53 valealex
* Abbozzo di sincronizzazione con le pipe.
* Provoca un deadlock quando un figlio termina prima degli altri in quanto
* la pipe sulla quale dovrebbe inviare il carattere di start non e'
* stata chiusa dagli altri fratelli
* Corretto errore indice i->j nella stampa del padre
*
* Revision 1.9 2005/03/11 10:13:25 valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7 2005/03/11 10:01:39 valealex

```

Mar 11, 05 12:17

sol20040312_11.c

Page 5/5

```
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5 2005/03/11 09:47:55 valealex
* Inserito primo start del processo padre
*
* Revision 1.4 2005/03/11 09:38:56 valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3 2005/03/11 09:24:57 valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2 2005/03/11 09:15:04 valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
* */
```

Mar 11, 05 12:17

sol20040312_12.c

Page 1/5

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.12 2005/03/11 10:59:49 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_12.c

Page 2/5

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_12.c

Page 3/5

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[j]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[j], WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */
    int nrp; /* numero di car letti da read su pipe */
    char dummy; /* carattere di appoggio per lett/scritt pipe */
    int i; /* un contatore */

    /* Chiusura dei lati delle pipe per evitare il deadlock quando
    * un figlio termina prima degli altri e non puo' proseguire
    * nell'attivita' di sincronizzazione. Se almeno i lati
    * di scrittura sono posseduti ognuno da un solo processo
    * la terminazione di questo provoca un comportamento non
    * bloccante per la read */
    for (i=0; i<Npipe; i++) {
        /* chiudo i lati di scrittura se non e' la pipe
        * che usa questo figlio */
        if (indice!=Nfigli-1) {
            if (i!=(indice+1)) {
                close(Pfds[i][SCRIT]);
            }
        } else {
            if (i!=0) {
                close(Pfds[i][SCRIT]);
            }
        }
    }

    fd=open(F,O_RDONLY);
    if (fd<0) {
#ifdef DEBUG
        fprintf(stderr, "Impossibile aprire il file %s\n", F);

```

Mar 11, 05 12:17

sol20040312_12.c

Page 4/5

```

#endif
        return(-1);
    }
    /* il file deve essere letto per linee */
    linenum=1; /* prima linea */
    retval=0; /* linee che soddisfano=0 */
    flag=0;
    for (;;) {
        nr=read(fd,&ch,1);
        if (nr==0) {
            return(retval);
        }
        if (nr<0) {
            /* errore */
#ifdef DEBUG
                fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
            return(-1);
        }
        /* carattere letto, controllo se fine linea */
        if (ch=='\n') {
            linenum++;
            flag=0;
        }
        /* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
        * il numero di linea su stdout. Se il carattere si trova piu'
        * volte nella stessa linea devo riportare il numero solo
        * una volta, quindi uso un flag che controllo a fine linea */
        if ((ch==C[indice])&&flag==0) {
            flag=1;
            retval++; /* incremento contatore */
            /* aggiungo sincronizzazione, attendo carattere da pipe
            * con indice i */
            nrp=read(Pfds[indice][LETT],&dummy,1);
            if (nrp==0) {
                /* pipe chiusa! cosa fare? */
            }
            if (nrp==-1) {
                /* errore, segnalare */
#ifdef DEBUG
                    fprintf(stderr, "Errore pipe di sincronizzazione\n");
#endif
                return(-1);
            }
            /* se sono qui posso scrivere */
            printf("Linea %d contiene %c\n", linenum, C[indice]);
            /* ora do il via al figlio successivo: l'indice della pi
pe
            * in cui scrivere e' indice+1 a meno che no si tratti d
ell'ultimo
            * figlio (quello con indice=(Nfigli-1)) che deve scrive
re sulla
            * pipe di indice 0 */
            if (indice!=Nfigli-1) {
                write(Pfds[indice+1][SCRIT], &dummy, 1);
            } else {
                write(Pfds[0][SCRIT], &dummy, 1);
            }
        }
    }
}
/* mai qui! */

```

Mar 11, 05 12:17

sol20040312_12.c

Page 5/5

```
    return (-1);
}
/* $Log: sol20040312.c,v $
* Revision 1.12  2005/03/11 10:59:49  valealex
* Eliminato il problema del deadlock. Ora se termina un processo
* figlio il figlio successivo ottiene sempre il 'via' (read torna 0)
* e si perde la sincronizzazione ma il testo indica che si puo'
* ipotizzare la terminazione in cascata. In tal caso la
* mancanza di sincronizzazione dopo la prima terminazione
* non si nota in quanto tutti gli altri processi terminano
* contestualmente
*
* Revision 1.11  2005/03/11 10:53:52  valealex
* Calcolo di retval come specificato nel testo
*
* Revision 1.10  2005/03/11 10:47:53  valealex
* Abbozzo di sincronizzazione con le pipe.
* Provoca un deadlock quando un figlio termina prima degli altri in quanto
* la pipe sulla quale dovrebbe inviare il carattere di start non e'
* stata chiusa dagli altri fratelli
* Corretto errore indice i->j nella stampa del padre
*
* Revision 1.9  2005/03/11 10:13:25  valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8  2005/03/11 10:07:23  valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7  2005/03/11 10:01:39  valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6  2005/03/11 09:55:02  valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5  2005/03/11 09:47:55  valealex
* Inserito primo start del processo padre
*
* Revision 1.4  2005/03/11 09:38:56  valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3  2005/03/11 09:24:57  valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2  2005/03/11 09:15:04  valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1  2005/03/11 09:00:06  valealex
* Initial revision
* */
```

Mar 11, 05 12:17

sol20040312_13.c

Page 1/6

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.13 2005/03/11 11:13:20 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_13.c

Page 2/6

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane!) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_13.c

Page 3/6

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[j]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[j],WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */
    int nrp; /* numero di car letti da read su pipe */
    char dummy; /* carattere di appoggio per lett/scritt pipe */
    int i; /* un contatore */
    int pipe_di_sinc; /* indice della pipe di sincronizzazione da
    cui fare la read */

    /* Chiusura dei lati delle pipe per evitare il deadlock quando
    * un figlio termina prima degli altri e non puo' proseguire
    * nell'attivita' di sincronizzazione. Se almeno i lati
    * di scrittura sono posseduti ognuno da un solo processo
    * la terminazione di questo provoca un comportamento non
    * bloccante per la read */
    for (i=0; i<Npipe; i++) {
        /* chiudo i lati di scrittura se non e' la pipe
        * che usa questo figlio */
        if (indice!=Nfigli-1) {
            if (i!=(indice+1)) {
                close(Pfds[i][SCRIT]);
            }
        } else {
            if (i!=0) {
                close(Pfds[i][SCRIT]);
            }
        }
    }

    fd=open(F,O_RDONLY);
    if (fd<0) {

```

Mar 11, 05 12:17

sol20040312_13.c

Page 4/6

```

#ifdef DEBUG
    fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
return(-1);
}
/* il file deve essere letto per linee */
linenum=1; /* prima linea */
retval=0; /* linee che soddisfano=0 */
flag=0;
/* sincronizzazione 'estesa' in grado di assicurarci l'alternanza delle
* informazioni anche in seguito alla terminazione di uno o piu' figli
* si usa una variabile per memorizzare l'indice della pipe da cui atten
dere
* l'ok. Se la read da tale indice restituisce 0 -> allora il figlio rel
ativo
* e' terminato e bisogna 'agganciarsi' al precedente. Nel caso estremo
* in cui un solo processo figlio rimanga in essere, la pipe di lettura
* e quella di scrittura vanno a coincidere sbloccando continuamente
* l'output di tale processo */
pipe_di_sinc = indice; /* iniziamo con quella 'canonica' */
for (;;) {
    nr=read(fd,&ch,1);
    if (nr==0) {
        return(retval);
    }
    if (nr<0) {
        /* errore */
#ifdef DEBUG
        fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
        return(-1);
    }
    /* carattere letto, controllo se fine linea */
    if (ch=='\n') {
        linenum++;
        flag=0;
    }
    /* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
    * il numero di linea su stdout. Se il carattere si trova piu'
    * volte nella stessa linea devo riportare il numero solo
    * una volta, quindi uso un flag che controllo a fine linea */
    if ((ch==C[indice])&&flag==0) {
        flag=1;
        retval++; /* incremento contatore */
        /* aggiungo sincronizzazione, attendo carattere da pipe
        * con indice pipe_di_sinc e gestisco l'eventuale
        * terminazione del figlio associato */
        for (;;) {
            nrp=read(Pfds[pipe_di_sinc][LETT],&dummy,1);
            if (nrp==0) {
                /* pipe chiusa, agganciamoci al figlio p
recedente */
                if (pipe_di_sinc>0) {
                    pipe_di_sinc--;
                } else {
                    pipe_di_sinc=Npipe-1;
                }
                continue;
            }
            if (nrp==-1) {
                /* errore, segnalare */

```

Mar 11, 05 12:17 **sol20040312_13.c** Page 5/6

```
#ifndef DEBUG
    fprintf(stderr, "Errore pipe di sincronizzazione\n"
);
#endif

    return(-1);
}
break;
}
/* se sono qui posso scrivere */
printf("Linea %d contiene %c\n",linenum,C[indice]);
/* ora do il via al figlio successivo: l'indice della pi
pe
ell'ultimo
re sulla
* in cui scrivere e' indice+1 a meno che no si tratti d
* figlio (quello con indice=(Nfigli-1)) che deve scrive
* pipe di indice 0 */
if (indice!=Nfigli-1) {
    write(Pfds[indice+1][SCRIT],&dummy,1);
} else {
    write(Pfds[0][SCRIT],&dummy,1);
}
}
}
/* mai qui! */
return (-1);
}
/* $Log: sol20040312.c,v $
* Revision 1.13 2005/03/11 11:13:20 valealex
* Inserita la gestione della terminazione anticipata di uno
* o piu' figli (non richiesta dal testo)
*
* Revision 1.12 2005/03/11 10:59:49 valealex
* Eliminato il problema del deadlock. Ora se termina un processo
* figlio il figlio successivo ottiene sempre il 'via' (read torna 0)
* e si perde la sincronizzazione ma il testo indica che si puo'
* ipotizzare la terminazione in cascata. In tal caso la
* mancanza di sincronizzazione dopo la prima terminazione
* non si nota in quanto tutti gli altri processi terminano
* contestualmente
*
* Revision 1.11 2005/03/11 10:53:52 valealex
* Calcolo di retval come specificato nel testo
*
* Revision 1.10 2005/03/11 10:47:53 valealex
* Abbozzo di sincronizzazione con le pipe.
* Provoca un deadlock quando un figlio termina prima degli altri in quanto
* la pipe sulla quale dovrebbe inviare il carattere di start non e'
* stata chiusa dagli altri fratelli
* Corretto errore indice i->j nella stampa del padre
*
* Revision 1.9 2005/03/11 10:13:25 valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee
*
* Revision 1.7 2005/03/11 10:01:39 valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
```

Mar 11, 05 12:17 **sol20040312_13.c** Page 6/6

```
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5 2005/03/11 09:47:55 valealex
* Inserito primo start del processo padre
*
* Revision 1.4 2005/03/11 09:38:56 valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3 2005/03/11 09:24:57 valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2 2005/03/11 09:15:04 valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
* */
```

Mar 11, 05 12:17

sol20040312_14.c

Page 1/6

```

/* Possibile soluzione della prova in itinere del 12 marzo 2004
 * analizzata nelle lezioni di Laboratorio di Sistemi Operativi
 * del 14 e 15 marzo 2005 */
/* $Id: sol20040312.c,v 1.14 2005/03/11 11:14:45 valealex Exp $ */

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

typedef int pipe_t[2]; /* tipo per pipe */
#define SCRIT 1
#define LETT 0

char *F; /* Nome del file, variabile globale */
int N; /* Numero di caratteri */
char *C; /* Elenco dei caratteri, allocazione dinamica */
int Npipe; /* Numero di pipe da creare */
pipe_t *Pfds; /* Array di pipe, allocazione dinamica */
int Nfigli; /* Numero di figli da forkare */
int *Pids; /* Array di pid, allocazione dinamica */

int figlio(int); /* prototipo della funzione figlio con
                 indice 'i' */
char ok='Q'; /* carattere (casuale) che inviamo sulla pipe
             per la sincronizzazione */

int main(int argc, char **argv, char **envp) {
    int i,j; /* qualche contatore */

    /* Determinazione dei dati dalla command line */
    switch (argc) {
        case 0:
        case 1:
        case 2:
            fprintf(stderr, "Inserire il nome di file ed almeno un carattere\n");
            return(1);
    }
    F = argv[1];
    N = argc-2; /* argc vale 1 se non ci sono argomenti */
    C = malloc(N*sizeof(char));
    if (C==NULL) {
        fprintf(stderr, "Impossibile allocare C\n");
        return(1);
    }
    for (i=0; i<N; i++) {
        C[i]=argv[i+2][0]; /* prendo il primo carattere della stringa ar
gv[i+2] */
    }
#ifdef DEBUG
    printf("Nome del file: %s\n", F);
    printf("Numero di caratteri %d\n", N);
    for (i=0; i<N; i++) {
        printf("Carattere di indice %d = %c\n", i, C[i]);
    }
#endif
    /* Predisposizione del sistema di sincronizzazione
     * fra i processi figli: si utilizza una pipe per
     * collegare ogni figlio al successivo; la sincronizzazione

```

Mar 11, 05 12:17

sol20040312_14.c

Page 2/6

```

 * del figlio di indice 'k' si ottiene attendendo un
 * carattere dalla pipe che lo collega al figlio di
 * indice 'k-1'. */
Npipe = N; /* una pipe in uscita da ogni figlio */
Pfds = malloc(Npipe*sizeof(pipe_t));
if (Pfds==NULL) {
    fprintf(stderr, "Impossibile allocare Pfds\n");
    return(1);
}
for (i=0; i<Npipe; i++) {
    if (pipe(Pfds[i])!=0) {
        fprintf(stderr, "Impossibile creare la pipe %d\n", i);
        return(1);
    }
}
/* Creazione processi figli:
 * dal testo, bisogna creare un processo figlio per ogni carattere
 * quindi Nfigli=N
 * Visto che il padre deve stampare i valori di uscita dei processi
 * figli, si devono memorizzare i pid in ordine di indice per poi
 * abbinare i risultati delle wait. L'array di pid deve essere
 * allocato dinamicamente */
Nfigli = N;
Pids=malloc(Nfigli*sizeof(int));
if (Pids==NULL) {
    fprintf(stderr, "Impossibile allocare Pids\n");
    return(1);
}
for (i=0; i<Nfigli; i++) {
    Pids[i]=fork();
    switch (Pids[i]) {
        case 0: /* figlio di indice i */
            return(figlio(i)); /* chiamo funzione figlio
e uso valore di ritorno come exit del processo*/
        case -1:
            fprintf(stderr, "Impossibile forkare il processo %d\n", i);
            return(1);
            break;
    }
}
#ifdef DEBUG
    /* solo il processo padre esegue queste linee */
    printf("Forkato processo %d per indice %d\n", Pids[i], i);
#endif
}
/* Il padre non utilizza nessuna pipe in quando la sincronizzazione
 * viene fatta totalmente fra i figli. Eventualmente (ma ci sono
 * anche altri modi) il padre puo' dare il via al primo figlio.
 * Utilizziamo questa possibilita' ed utilizziamo un singolo
 * carattere memorizzato in ok per l'invio */
if(write(Pfds[0][SCRIT], &ok, 1)!=1) {
    /* controllo di scrittura avvenuta */
    fprintf(stderr, "Impossibile inviare su %d\n", Pfds[0][SCRIT]);
    return(1);
}
/* ora chiudiamo tutte le pipe inutilizzate dal padre, compresa
 * quella in cui abbiamo appena scritto (il carattere rimane) */
for (i=0; i<Npipe; i++) {
    close(Pfds[i][SCRIT]);
    close(Pfds[i][LETT]);
}
/* Recupero delle informazioni dai processi figli tramite la wait */
for (i=0; i<Nfigli; i++) {

```

Mar 11, 05 12:17

sol20040312_14.c

Page 3/6

```

int pid_x;
int status;

pid_x = wait(&status);
/* abbinamento delle informazioni all'indice del figlio */
for (j=0; j<Nfigli; j++) {
    if (pid_x==Pids[j]) {
        if (WEXITSTATUS(status)==255) {
            /* convenzione: segnalazione d'errore */
            printf("Il figlio che cerca il carattere %c ha incontrato
un errore\n",
                C[j]);
        } else {
            printf("Il figlio che cerca il carattere %c ha trovato %d
linee\n",
                C[j],WEXITSTATUS(status));
        }
    }
}
return(0);
}

/* Funzione figlio */
int figlio(int indice) {
    /* Ogni figlio deve cercare le occorrenze di C[i] nel file F,
    * quindi il file NON deve essere condiviso ma aperto indipendentemente
    * da ogni figlio */
    int fd;
    int linenum; /* contatore di linea corrente */
    char ch; /* carattere di appoggio */
    int retval; /* valore di ritorno */
    int nr; /* numero di caratteri letti dalla read */
    int flag; /* se 0 carattere non trovato, se 1 trovato */
    int nrp; /* numero di car letti da read su pipe */
    char dummy; /* carattere di appoggio per lett/scritt pipe */
    int i; /* un contatore */
    int pipe_di_sinc; /* indice della pipe di sincronizzazione da
    cui fare la read */

    /* Chiusura dei lati delle pipe per evitare il deadlock quando
    * un figlio termina prima degli altri e non puo' proseguire
    * nell'attivita' di sincronizzazione. Se almeno i lati
    * di scrittura sono posseduti ognuno da un solo processo
    * la terminazione di questo provoca un comportamento non
    * bloccante per la read */
    for (i=0; i<Npipe; i++) {
        /* chiudo i lati di scrittura se non e' la pipe
        * che usa questo figlio */
        if (indice!=Nfigli-1) {
            if (i!=(indice+1)) {
                close(Pfds[i][SCRIT]);
            }
        } else {
            if (i!=0) {
                close(Pfds[i][SCRIT]);
            }
        }
    }

    fd=open(F,O_RDONLY);
    if (fd<0) {

```

Mar 11, 05 12:17

sol20040312_14.c

Page 4/6

```

#ifdef DEBUG
    fprintf(stderr, "Impossibile aprire il file %s\n", F);
#endif
return(-1);
}
/* il file deve essere letto per linee */
linenum=1; /* prima linea */
retval=0; /* linee che soddisfano=0 */
flag=0;
/* sincronizzazione 'estesa' in grado di assicurarci l'alternanza delle
* informazioni anche in seguito alla terminazione di uno o piu' figli
* si usa una variabile per memorizzare l'indice della pipe da cui atten
dere
* l'ok. Se la read da tale indice restituisce 0 -> allora il figlio rel
ativo
* e' terminato e bisogna 'agganciarsi' al precedente. Nel caso estremo
* in cui un solo processo figlio rimanga in essere, la pipe di lettura
* e quella di scrittura vanno a coincidere sbloccando continuamente
* l'output di tale processo */
pipe_di_sinc = indice; /* iniziamo con quella 'canonica' */
for (;;) {
    nr=read(fd,&ch,1);
    if (nr==0) {
        return(retval);
    }
    if (nr<0) {
        /* errore */
#ifdef DEBUG
        fprintf(stderr, "Impossibile leggere dal file %s\n", F);
#endif
        return(-1);
    }
    /* carattere letto, controllo se fine linea */
    if (ch=='\n') {
        linenum++;
        flag=0;
    }
    /* dal testo: se trovo il carattere assegnato C[indice], devo ri
portare
    * il numero di linea su stdout. Se il carattere si trova piu'
    * volte nella stessa linea devo riportare il numero solo
    * una volta, quindi uso un flag che controllo a fine linea */
    if ((ch==C[indice])&&flag==0) {
        flag=1;
        retval++; /* incremento contatore */
        /* aggiungo sincronizzazione, attendo carattere da pipe
        * con indice pipe_di_sinc e gestisco l'eventuale
        * terminazione del figlio associato */
        for (;;) {
            nrp=read(Pfds[pipe_di_sinc][LETT],&dummy,1);
            if (nrp==0) {
                /* pipe chiusa, agganciamoci al figlio p
recedente */
                if (pipe_di_sinc>0) {
                    pipe_di_sinc--;
                } else {
                    pipe_di_sinc=Npipe-1;
                }
                continue;
            }
            if (nrp==-1) {
                /* errore, segnalare */

```

```

Mar 11, 05 12:17                sol20040312_14.c                Page 5/6
#ifdef DEBUG
                                fprintf(stderr, "Errore pipe di sincronizzazione\n"
);
#endif
                                return(-1);
                                }
                                break;
                                }
/* se sono qui posso scrivere */
printf("Linea %d contiene %c\n",linenum,C[indice]);
fflush(stdout);
/* ora do il via al figlio successivo: l'indice della pi
pe
ell'ultimo
re sulla
                                * in cui scrivere e' indice+1 a meno che no si tratti d
                                * figlio (quello con indice=(Nfigli-1)) che deve scrive
                                * pipe di indice 0 */
                                if (indice!=Nfigli-1) {
                                    write(Pfds[indice+1][SCRIT],&dummy,1);
                                } else {
                                    write(Pfds[0][SCRIT],&dummy,1);
                                }
                                }
                                }
/* mai qui! */
return (-1);
}
/* $Log: sol20040312.c,v $
* Revision 1.14 2005/03/11 11:14:45 valealex
* Aggiunta la chiamata a fflush per poter visualizzare le linee
* prodotte anche in pipe con grep o pager
*
* Revision 1.13 2005/03/11 11:13:20 valealex
* Inserita la gestione della terminazione anticipata di uno
* o piu' figli (non richiesta dal testo)
*
* Revision 1.12 2005/03/11 10:59:49 valealex
* Eliminato il problema del deadlock. Ora se termina un processo
* figlio il figlio successivo ottiene sempre il 'via' (read torna 0)
* e si perde la sincronizzazione ma il testo indica che si puo'
* ipotizzare la terminazione in cascata. In tal caso la
* mancanza di sincronizzazione dopo la prima terminazione
* non si nota in quanto tutti gli altri processi terminano
* contestualmente
*
* Revision 1.11 2005/03/11 10:53:52 valealex
* Calcolo di retval come specificato nel testo
*
* Revision 1.10 2005/03/11 10:47:53 valealex
* Abbozzo di sincronizzazione con le pipe.
* Provoca un deadlock quando un figlio termina prima degli altri in quanto
* la pipe sulla quale dovrebbe inviare il carattere di start non e'
* stata chiusa dagli altri fratelli
* Corretto errore indice i->j nella stampa del padre
*
* Revision 1.9 2005/03/11 10:13:25 valealex
* Nel figlio, aggiunta ricerca carattere e scrittura
* su stdout (per ora non sincronizzata)
*
* Revision 1.8 2005/03/11 10:07:23 valealex
* Nei figli, aggiunta la gestione per linee

```

```

Mar 11, 05 12:17                sol20040312_14.c                Page 6/6
*
* Revision 1.7 2005/03/11 10:01:39 valealex
* Aggiunta open nei figli
* Controllo exit status=255 per errore
*
* Revision 1.6 2005/03/11 09:55:02 valealex
* Ciclo di recupero exit status e abbinamento ai pid
*
* Revision 1.5 2005/03/11 09:47:55 valealex
* Inserito primo start del processo padre
*
* Revision 1.4 2005/03/11 09:38:56 valealex
* Creati i processi figli
* Eliminato un errore di digitazione nella malloc del vettore di pipe
*
* Revision 1.3 2005/03/11 09:24:57 valealex
* Creazione pipe per sincronizzazione
*
* Revision 1.2 2005/03/11 09:15:04 valealex
* Prelievo e controllo degli argomenti della linea di comando
*
* Revision 1.1 2005/03/11 09:00:06 valealex
* Initial revision
* */

```