

Lab. di Sistemi Operativi

Esercitazioni proposte per le lezioni del 6-7/05/10

Utilizzando il compilatore gcc in Linux e disponendosi in gruppi di due persone per ogni PC del laboratorio.

1. Utilizzando la funzione `fork()` creare un nuovo processo (`es71.c`)
2. Presentare su `stdout` il valore ritornato dalla `fork()`. Si utilizzi la coppia `snprintf()/write()` al posto della semplice `printf()`
3. Per quale motivo troviamo su `stdout` due messaggi (con valori differenti)?
4. Verificare il ruolo di padre e figlio visualizzandolo su standard output.
5. La funzione `fork()` restituisce -1 in caso di errore (e, ovviamente, non genera il nuovo processo). Aggiungere questa verifica associandola ad un opportuno messaggio d'errore.
6. Sia il processo padre che il processo figlio hanno un proprio `exit value`. Utilizzare la funzione `wait()` per recuperare l'`exit value` del processo figlio.
7. La `wait` svolge anche un ruolo di sincronizzazione fra processo padre e processo figlio?
8. Condivisione di file. Si modifichi `es71.c` in modo che il file rappresentato dal primo argomento sia aperto prima della `fork()` ed utilizzato in lettura in modo condiviso da padre e figlio, ad esempio per contare i caratteri contenuti leggendoli ad uno ad uno.
9. Creare un `Makefile` per la compilazione di `es71`.
10. Comportamento della `printf` in applicazioni concorrenti. Creare un sorgente (`es72.c`) che utilizzi la `printf` per emettere una stringa priva di fine linea prima della `fork()`.
11. L'esecuzione di `es72` riporta la stringa iniziale per due volte su `stdout`. Proviamo a modificare il sorgente introducendo la `write` al posto della `printf`.
12. Creazione di più processi. Creare un sorgente (`es73.c`) che generi un numero di figli uguale al numero passato come primo argomento.
13. Aggiornare il `Makefile` in modo che contenga le regole per tutti i sorgenti di oggi. Si aggiunga un target "all" per la creazione di tutti gli eseguibili
14. Modificare `es73.c` in modo che ogni figlio stampi su `stdout` il proprio indice.
15. Recupero degli `exit value` dei processi figli: modificare `es73.c` in modo che il processo padre attenda la terminazione di tutti i processi figli riportando su `stdout` gli `exit value` di ogni processo accanto al `pid` del processo terminato.
16. Un po' di ordine: modifichiamo `es73.c` in modo che le operazioni delegate ai processi figli siano incluse in una funzione.
17. Utilizzare un file a parte (`es73b.c`) in cui spostare la funzione figlio e modificare di conseguenza il `Makefile`

Soluzione

Esercitazioni proposte per le lezioni del 6-7/05/10

Utilizzando il compilatore gcc in Linux e disponendosi in gruppi di due persone per ogni PC del laboratorio.

1. Utilizzando la funzione `fork()` creare un nuovo processo (`es71.c`)

Soluzione:

```
/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char **argv) {
    fork();
    return(0);
}
```

2. Presentare su `stdout` il valore ritornato dalla `fork()`. Si utilizzi la coppia `sprintf()/write()` al posto della semplice `printf()`

Soluzione:

```
/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
char msg[256];
int main(int argc, char **argv) {
    pid_t fr; /* fork return value */
    fr=fork();
    sprintf(msg, sizeof(msg),
            "fork: %d\n", fr);
    write(1, msg, strlen(msg));
    return(0);
}
```

3. Per quale motivo troviamo su `stdout` due messaggi (con valori differenti)?

Soluzione:

La funzione `fork()` invoca la `system call` omonima che provoca la creazione di un nuovo processo 'clonando' quello corrente. Il processo 'originale', detto padre, ottiene dalla `fork` un numero intero strettamente positivo che rappresenta il pid del nuovo processo creato. Il nuovo processo, detto 'figlio', si trova con una copia esatta del processo padre e, quindi, inizierà la sua esecuzione dall'istruzione (macchina) appena successiva

alla chiamata a `fork()`. Il valore di ritorno recuperato dal figlio, tuttavia, e' diverso da quello che ha ottenuto il processo padre ed e' sempre uguale a 0. Utilizzando questa differenza, sara' possibile determinare a programma se il codice viene eseguito dal padre o dal figlio operando in modo diverso.

4. Verificare il ruolo di padre e figlio visualizzandolo su standard output.

Soluzione:

```
/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
char msg[256];
int main(int argc, char **argv) {
    pid_t fr; /* fork return value */
    fr=fork();
    if (fr==0) { /* figlio */
        snprintf(msg, sizeof(msg), "Figlio\n");
        write(1, msg, strlen(msg));
    } else { /* padre */
        snprintf(msg, sizeof(msg),
            "Padre_\del_\figlio_%d\n", fr);
        write(1, msg, strlen(msg));
    }
    return(0);
}
```

5. La funzione `fork()` restituisce -1 in caso di errore (e, ovviamente, non genera il nuovo processo). Aggiungere questa verifica associandola ad un opportuno messaggio d'errore.

Soluzione:

```
/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
char msg[256];
int main(int argc, char **argv) {
    pid_t fr; /* fork return value */
    fr=fork();
    switch (fr) {
        case 0: /* figlio */
            snprintf(msg, sizeof(msg), "Figlio\n");
            write(1, msg, strlen(msg));
            break;
    }
```

```

        case -1: /* errore, nessun nuovo processo */
            snprintf(msg, sizeof(msg), "Errore_in_fork\n");
            write(1, msg, strlen(msg));
            break;
        default: /* padre */
            snprintf(msg, sizeof(msg),
                "Padre_del_figlio_%d\n", fr);
            write(1, msg, strlen(msg));
    }
    return(0);
}

```

6. Sia il processo padre che il processo figlio hanno un proprio exit value. Utilizzare la funzione wait() per recuperare l'exit value del processo figlio.

Soluzione:

```

/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
char msg[256];
int main(int argc, char **argv) {
    pid_t fr; /* fork return value */
    int st; /* spazio per l'exit value */
    fr=fork();
    switch (fr) {
        case 0: /* figlio */
            snprintf(msg, sizeof(msg), "Figlio\n");
            write(1, msg, strlen(msg));
            return(4); /* exit value del figlio */
            break;
        case -1: /* errore, nessun nuovo processo */
            snprintf(msg, sizeof(msg), "Errore_in_fork\n");
            write(1, msg, strlen(msg));
            break;
        default: /* padre */
            snprintf(msg, sizeof(msg),
                "Padre_del_figlio_%d\n", fr);
            write(1, msg, strlen(msg));
            /* uso la wait per attendere il processo
             * figlio e recuperare il suo exit value */
            wait(&st);
            snprintf(msg, sizeof(msg),
                "Exit_value:%d\n", WEXITSTATUS(st));
            write(1, msg, strlen(msg));
    }
    return(0);
}

```

7. La wait svolge anche un ruolo di sincronizzazione fra processo padre e processo figlio?

Soluzione:

Se proviamo ad eseguire diverse volte il programma es71, potremmo notare che la sequenza con la quale vediamo il messaggio "Figlio..." e quello "Padre..." non e' ripetitiva. La stampa di "Exit...", al contrario, sara' sempre successiva a "Figlio..." anche se operate da processi differenti.

Questo e' dovuto alla presenza della chiamata a wait() che, per recuperare l'exit value del figlio, deve forzare il processo corrente (padre) ad attendere (wait, appunto) la terminazione del processo figlio.

8. Condivisione di file. Si modifichi es71.c in modo che il file rappresentato dal primo argomento sia aperto prima della fork() ed utilizzato in lettura in modo condiviso da padre e figlio, ad esempio per contare i caratteri contenuti leggendoli ad uno ad uno.

Soluzione:

```
/* file: es71.c
 * job: uso della fork
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
char msg[256];
int fd;
int main(int argc, char **argv) {
    pid_t fr; /* fork return value */
    int st; /* spazio per l'exit value */
    int count; /* contatore di read */
    char ch; /* spazio per la read */
    fd=open(argv[1],ORDONLY);
    if (fd<0) return(-1);
    fr=fork();
    switch (fr) {
        case 0: /* figlio */
            for (count=0; read(fd,&ch,1)==1; count++);
            snprintf(msg, sizeof(msg), "Figlio_conta_%d\n", count);
            write(1, msg, strlen(msg));
            return(4); /* exit value del figlio */
            break;
        case -1: /* errore, nessun nuovo processo */
            snprintf(msg, sizeof(msg), "Errore_in_fork\n");
            write(1, msg, strlen(msg));
            break;
        default: /* padre */
            for (count=0; read(fd,&ch,1)==1; count++);
            snprintf(msg, sizeof(msg),
                "Padre_del_figlio_%d, conta:_%d\n", fr,
```

```

        count);
    write(1, msg, strlen(msg));
    /* uso la wait per attendere il processo
     * figlio e recuperare il suo exit value */
    wait(&st);
    snprintf(msg, sizeof(msg),
             "Exit_value: %d\n", WEXITSTATUS(st));
    write(1, msg, strlen(msg));
}
return(0);
}

```

9. Creare un Makefile per la compilazione di es71.

Soluzione:

```

#!/bin/make -f
# file: Makefile
# formato della regola = obiettivo: dipendenze
es71: es71.c
    gcc -Wall -o es71 es71.c
# Ricordarsi del tab all'inizio di ogni comando

```

10. Comportamento della printf in applicazioni concorrenti. Creare un sorgente (es72.c) che utilizzi la printf per emettere una stringa priva di fine linea prima della fork().

Soluzione:

```

/* file: es72.c
 * job: fork e printf, una verifica
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

char stringa[256];
int main(int argc, char **argv) {
    /* subito una printf senza \n */
    printf("Prima della fork(): _");
    switch (fork()) {
        case 0: /* figlio */
            printf("figlio\n");
            return(0);
        case -1:
            printf("errore\n");
            return(1);
        default:
            printf("padre\n");
            return(0);
    }
}

```

11. L'esecuzione di es72 riporta la stringa iniziale per due volte su stdout. Proviamo a modificare il sorgente introducendo la write al posto della printf.

Soluzione:

```

/* file: es72.c
 * job: fork e printf, una verifica
 */
#include <sys/types.h>
#include <unistd.h>

char stringa[256];
int main(int argc, char **argv) {
    /* subito una printf senza \n */
    write(1,"Prima_della_fork():_ ",20);
    switch (fork()) {
        case 0: /* figlio */
            write(1,"figlio\n",7);
            return(0);
        case -1:
            write(1,"errore\n",7);
            return(1);
        default:
            write(1,"padre\n",6);
            return(0);
    }
}

```

12. Creazione di più processi. Creare un sorgente (es73.c) che generi un numero di figli uguale al numero passato come primo argomento.

Soluzione:

```

/* file: es73.c
 * job: fork() multipla
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char stringa[256];
int main(int argc, char **argv) {
    int nfigli; /* N */
    int indice; /* indice di creazione */
    int pid; /* process identifier */
    if (argc<2) {
        sprintf(stringa,256,"Uso:_%s_N\n",argv[0]);
        write(2,stringa,strlen(stringa));
        return(1);
    } else {
        nfigli=atoi(argv[1]);
    }
    for (indice=0; indice<nfigli; indice++) {
        pid=fork();
        switch (pid) {
            case 0: /* figlio */
                return(0); /* terminazione */
            case -1: /* errore */

```

```

        snprintf(stringa,256,"Errore alla %d-ma fork\n",indice);
        write(2,stringa,strlen(stringa));
        return(1); /* terminazione padre */
    default: /* padre, continua */
        snprintf(stringa,256,"Creato figlio %d con pid %d\n",
            indice,pid);
        write(1,stringa,strlen(stringa));
    }
}
/* il padre arriva qui dopo le N creazioni */
return(0);
}

```

13. Aggiornare il Makefile in modo che contenga le regole per tutti i sorgenti di oggi. Si aggiunga un target "all" per la creazione di tutti gli eseguibili

Soluzione:

```

#!/bin/make -f
# file: Makefile
all: es71 es72 es73

# Ricordarsi la linea vuota come separatore
# formato della regola = obiettivo: dipendenze
es71: es71.c
    gcc -Wall -o es71 es71.c
# Ricordarsi del tab all'inizio di ogni comando

es72: es72.c
    gcc -Wall -o es72 es72.c

es73: es73.c
    gcc -Wall -o es73 es73.c

# fine file

```

14. Modificare es73.c in modo che ogni figlio stampi su stdout il proprio indice.

Soluzione:

```

/* file: es73.c
 * job: fork() multipla
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char stringa[256];
int main(int argc, char **argv) {
    int nfigli; /* N */
    int indice; /* indice di creazione */
    int pid; /* process identifier */
    if (argc < 2) {
        snprintf(stringa,256,"Uso: %s N\n",argv[0]);
    }
}

```

```

        write(2, stringa , strlen( stringa ));
        return(1);
    } else {
        nfigli=atoi( argv [1]);
    }
    for (indice=0; indice<nfigli; indice++) {
        pid=fork ();
        switch (pid) {
            case 0: /* figlio */
                snprintf( stringa ,256," Figlio _con _indice _%d\n", indice );
                write(1, stringa , strlen( stringa ));
                return(0); /* terminazione */
            case -1: /* errore */
                snprintf( stringa ,256," Errore _alla _%d_ma_fork\n", indice );
                write(2, stringa , strlen( stringa ));
                return(1); /* terminazione padre */
            default: /* padre , continua */
                snprintf( stringa ,256," Creato _figlio _%d _con _pid _%d\n",
                    indice ,pid);
                write(1, stringa , strlen( stringa ));
        }
    }
    /* il padre arriva qui dopo le N creazioni */
    return(0);
}

```

15. Recupero degli exit value dei processi figli: modificare es73.c in modo che il processo padre attenda la terminazione di tutti i processi figli riportando su stdout gli exit value di ogni processo accanto al pid del processo terminato.

Soluzione:

```

/* file: es73.c
 * job: fork() multipla
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>

char stringa [256];
int main(int argc, char **argv) {
    int nfigli; /* N */
    int indice; /* indice di creazione */
    int pid; /* process identifier */
    int st; /* exit status */
    if (argc<2) {
        snprintf( stringa ,256," Uso: _%s _N\n", argv [0]);
        write(2, stringa , strlen( stringa ));
        return(1);
    } else {
        nfigli=atoi( argv [1]);
    }
}

```

```

for (indice=0; indice<nfigli; indice++) {
    pid=fork();
    switch (pid) {
        case 0: /* figlio */
            snprintf(stringa,256,"Figlio_con_indice_%d\n",indice);
            write(1,stringa,strlen(stringa));
            return(0); /* terminazione */
        case -1: /* errore */
            snprintf(stringa,256,"Errore_alla_%d-ma_fork\n",indice);
            write(2,stringa,strlen(stringa));
            return(1); /* terminazione padre */
        default: /* padre, continua */
            snprintf(stringa,256,"Creato_figlio_%d_con_pid_%d\n",
                indice,pid);
            write(1,stringa,strlen(stringa));
    }
}
/* il padre arriva qui dopo le N creazioni */
for (indice=0; indice<nfigli; indice++) {
    /* l'ordine in cui i figli terminano NON e' prevedibile,
     * quindi il valore recuperato alla i-esima iterazione
     * non necessariamente si riferisce all' i-esimo
     * figlio. */
    pid=wait(&st);
    snprintf(stringa,256,"Il_figlio_con_pid_%d_ritorna_%d\n",
        pid,WEXITSTATUS(st));
    write(1,stringa,strlen(stringa));
}
return(0);
}

```

16. Un po' di ordine: modifichiamo es73.c in modo che le operazioni delegate ai processi figli siano incluse in una funzione.

Soluzione:

```

/* file: es73.c
 * job: fork() multipla
 */
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>

char stringa[256];
/* Prototipo della funzione figlio */
int figlio(int argomento);
int main(int argc, char **argv) {
    int nfigli; /* N */
    int indice; /* indice di creazione */
    int pid; /* process identifier */
    int st; /* exit status */
    if (argc<2) {

```

```

        snprintf(stringa,256,"Usò: %s \n",argv[0]);
        write(2,stringa,strlen(stringa));
        return(1);
    } else {
        nfigli=atoi(argv[1]);
    }
    for (indice=0; indice<nfigli; indice++) {
        pid=fork();
        switch (pid) {
            case 0: /* figlio */
                /* chiamata alla funzione figlio
                 * e recupero del valore di
                 * ritorno */
                return( figlio(indice) );
            case -1: /* errore */
                snprintf(stringa,256,"Errore alla %d-ma fork\n",indice);
                write(2,stringa,strlen(stringa));
                return(1); /* terminazione padre */
            default: /* padre, continua */
                snprintf(stringa,256,"Creato figlio %d con pid %d\n",
                    indice,pid);
                write(1,stringa,strlen(stringa));
        }
    }
    /* il padre arriva qui dopo le N creazioni */
    for (indice=0; indice<nfigli; indice++) {
        /* l'ordine in cui i figli terminano NON e' prevedibile,
         * quindi il valore recuperato alla i-esima iterazione
         * non necessariamente si riferisce all' i-esimo
         * figlio. */
        pid=wait(&st);
        snprintf(stringa,256,"Il figlio con pid %d ritorna %d\n",
            pid,WEXITSTATUS(st));
        write(1,stringa,strlen(stringa));
    }
    return(0);
}
/* Corpo della funzione figlio */
int figlio(int argomento) {
    /* usa stringa perche' e' una variabile globale */
    snprintf(stringa,256,"Figlio con indice %d\n",argomento);
    write(1,stringa,strlen(stringa));
    return(0);
}

```

17. Utilizzare un file a parte (es73b.c) in cui spostare la funzione figlio e modificare di conseguenza il Makefile

Soluzione:

```

/* file: es73.c
 * job: fork() multipla
 */
#include <sys/types.h>
#include <unistd.h>

```

```

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

char stringa[256];
/* Prototipo della funzione figlio , ancora in es73.c */
int figlio(int argomento);
int main(int argc, char **argv) {
    int nfigli; /* N */
    int indice; /* indice di creazione */
    int pid; /* process identifier */
    int st; /* exit status */
    if (argc<2) {
        sprintf(stringa,256,"Uso: %s N\n",argv[0]);
        write(2,stringa,strlen(stringa));
        return(1);
    } else {
        nfigli=atoi(argv[1]);
    }
    for (indice=0; indice<nfigli; indice++) {
        pid=fork();
        switch (pid) {
            case 0: /* figlio */
                /* chiamata alla funzione figlio
                 * e recupero del valore di
                 * ritorno */
                return( figlio(indice) );
            case -1: /* errore */
                sprintf(stringa,256,"Errore alla %d-ma fork\n",indice);
                write(2,stringa,strlen(stringa));
                return(1); /* terminazione padre */
            default: /* padre, continua */
                sprintf(stringa,256,"Creato figlio %d con pid %d\n",
                    indice,pid);
                write(1,stringa,strlen(stringa));
        }
    }
    /* il padre arriva qui dopo le N creazioni */
    for (indice=0; indice<nfigli; indice++) {
        /* l'ordine in cui i figli terminano NON e' prevedibile,
         * quindi il valore recuperato alla i-esima iterazione
         * non necessariamente si riferisce all' i-esimo
         * figlio. */
        pid=wait(&st);
        sprintf(stringa,256,"Il figlio con pid %d ritorna %d\n",
            pid,WEXITSTATUS(st));
        write(1,stringa,strlen(stringa));
    }
    return(0);
}

```

```

/* file: es73b.c
 * job: funzione figlio
 */
#include <unistd.h>
#include <string.h>
#include <stdio.h>
/* stringa a' ancora accessibile (globale) ma il
 * compilatore non sa cosa sia, istruiamolo in merito */
extern char stringa [];
/* Attenzione: la definizione precedente e' diversa da
 * extern char *stringa in quanto il simbolo "stringa"
 * si riferisce all'indirizzo del primo carattere
 * dell'array e non all'indirizzo del puntatore ad
 * esso. In altre parole, il compilatore deve sapere
 * che stringa e' un array e non un puntatore (stesso
 * tipo ma storage class differente) */

/* Corpo della funzione figlio in es73b.c */
int figlio(int argomento) {
    snprintf(stringa, 256, "Figlio con indice %d\n", argomento);
    write(1, stringa, strlen(stringa));
    return(0);
}

#!/bin/make -f
# file: Makefile
all: es71 es72 es73

# Ricordarsi la linea vuota come separatore
# formato della regola = obiettivo: dipendenze
es71: es71.c
    gcc -Wall -o es71 es71.c
# Ricordarsi del tab all'inizio di ogni comando

es72: es72.c
    gcc -Wall -o es72 es72.c

es73: es73.c es73b.c
    gcc -Wall -o es73 es73.c es73b.c

# fine file

```