Sistemi Operativi **(M. Cesati)**

Compito scritto del 17 febbraio 2014

Nome: _____ **Cognome:** _____

Matricola: _____ **Corso di laurea:** _____

Crediti da conseguire: 5 6 9

Scrivere i dati richiesti in stampatello. Al termine consegnare tutti i fogli. Tempo a disposizione: 2 ore.
Tenere presente che saranno valutati anche l'ordine e la chiarezza dell'esposizione.

Esercizio 1. Scrivere una applicazione C/POSIX costituita da 3 processi P_1, P_2 e P_3 collegati tra loro mediante 3 pipe A, B e C secondo lo schema:

$$\text{(stdin)} \longrightarrow P_1 \xrightarrow{A} P_2 \xrightarrow{B} P_3 \xrightarrow{C} P_1 \longrightarrow \text{(stdout)}$$

- Il processo P_1 continuamente legge linee di testo dallo standard input, ciascuna contenente un numero intero senza segno in base 10. Per ciascun numero letto:
  - converte il numero in formato binario a 32 bit e lo invia a P_2 tramite la pipe A;
  - attende una lista di coppie *(primo,esponente)* di numeri in formato binario a 32 bit dalla pipe C e la scrive in standard output in formato testuale
- Il processo P_2 continuamente legge numeri in formato binario a 32 bit dalla pipe A; per ciascuno di essi, fattorizza il numero inviando al processo P_3 i fattori primi elementari in formato binario a 32 bit per mezzo della pipe B.
- Il processo P_3 continuamente legge dalla pipe B una lista di primi (in ordine non decrescente) e ritrasmette al processo P_1 la equivalente lista nel formato *(primo,esponente)*.

Il formato binario è quello nativo del calcolatore. Le liste di numeri trasmesse sulle pipe B e C sono terminate dal valore zero. Ad esempio:

- P_1 legge in standard input una linea con il numero "`365904`"
- P_1 invia sulla pipe A: 365904 (in formato binario a 32 bit)
- P_2 legge da A il numero inviato da P_1 e scrive su B i numeri

  2 2 2 2 3 3 3 7 11 11 0 (in formato binario)
- P_3 legge da B la lista di numeri inviata da P_2 e scrive su C la lista di numeri

  2 4 3 3 7 1 11 2 0 (in formato binario)
- P_1 legge dalla pipe C la lista di numeri inviata da P_3 e scrive sullo standard output la stringa di testo "`2^4 * 3^3 * 7 * 11^2`"

Esercizio 2. Si descrivano in modo sintetico ed esauriente i vantaggi derivanti dall'implementazione del meccanismo della memoria virtuale nei moderni sistemi operativi.

Sistemi Operativi (M. Cesati)

Esempio del programma del compito scritto del 17 febbraio 2014

Esercizio 1

Svolgiamo l'esercizio secondo un approccio "top-down". Le strutture di dati essenziali del programma sono le tre pipe di comunicazione tra i tre processi. Per loro natura, queste pipe debbono essere create dal primo processo ed ereditate dai processi figli. Quindi l'ordine delle operazioni per il processo padre è: (1) creare le pipe, (2) creare i figli, (3) chiudere i descrittori dei file non utilizzati, e (4) eseguire il lavoro proprio di P_1.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

int pipeA[2]; /* data dir: from P1 to P2 */
int pipeB[2]; /* data dir: from P2 to P3 */
int pipeC[2]; /* data dir: from P3 to P1 */

int main(int argc, char *argv[])
{
    if (argc != 1) {
        fprintf(stderr, "Usage: %s (no arguments)\n",
                argv[0]);
        return EXIT_FAILURE;
    }
    make_pipes(pipeA, pipeB, pipeC);
    spawn_children();
    close_fds(pipeC[1], pipeA[0], pipeB[0], pipeB[1]);
    do_P1_work(pipeC[0], pipeA[1]);
    return EXIT_SUCCESS;
}
```

La funzione `make_pipes()` crea le tre pipe A, B e C:

```
void make_pipes(int pipeA[2], int pipeB[2], int pipeC[2])
{
    if (pipe(pipeA) || pipe(pipeB) || pipe(pipeC)) {
        perror("creating pipes");
        exit(EXIT_FAILURE);
    }
}
```

La funzione `spawn_children()` crea ed avvia l'esecuzione dei processi figli:

```
void spawn_children(void)
{
    pid_t pid = fork();
    if (pid == -1) {
        perror("fork(P2)");
        exit(EXIT_FAILURE);
    }
    if (pid == 0) {
        close_fds(pipeA[1], pipeB[0], pipeC[0], pipeC[1]);
        do_P2_work(pipeA[0], pipeB[1]);
    }
    pid = fork();
    if (pid == -1) {
        perror("fork(P3)");
        exit(EXIT_FAILURE);
    }
    if (pid == 0) {
        close_fds(pipeB[1], pipeC[0], pipeA[0], pipeA[1]);
        do_P3_work(pipeB[0], pipeC[1]);
    }
}
```

La funzione `close_fds()` chiude quattro descrittori di file passati in argomento:

```
void close_fds(int fd1, int fd2, int fd3, int fd4)
{
    if (close(fd1) || close(fd2) ||
        close(fd3) || close(fd4)) {
        perror("close");
        exit(EXIT_FAILURE);
    }
}
```

Le funzioni `do_P1_work()`, `do_P2_work()` e `do_P3_work()` sono specifiche per ciascuno dei tre processi.

`do_P1_work()` continuamente legge righe di testo dallo standard input. Per ciascuna linea, interpreta il suo contenuto come un numero senza segno in base 10, lo converte in formato nativo (binario) a 32 bit, e lo trasmette sulla pipe A al processo P_2. Infine legge una lista di numeri dalla pipe C, e li stampa nel formato opportuno in standard output:

```
void do_P1_work(int piper, int pipew)
{
    unsigned int v, w, flag;
    for (;;) {
```

```
        v = read_number_from_stdin();
        write_u32_to_pipe(v, pipew);
        if (v == 0) /* EOF in stdin */
            return;
        for (flag=0; ;flag=1) {
            v = read_u32_from_pipe(piper);
            if (v == 0)
                break; /* end of list */
            w = read_u32_from_pipe(piper);
            if (v == 0) {
                fprintf(stderr,
                    "Unexpected end of list from pipe\n");
                exit(EXIT_FAILURE);
            }
            if (flag)
                printf(" * ");
            if (w > 1)
                printf("%u^%u", v, w);
            else
                printf("%u", v);
        }
        fputc('\n', stdout);
    }
}
```

La funzione `read_number_from_stdin()` legge una linea di testo dallo standard input ed interpreta il contenuto come un numero in base 10 utilizzando la funzione `convert_number()`:

```
unsigned int read_number_from_stdin(void)
{
    const int max_line = 1024;
    char line[max_line];
    unsigned int v;
    if (fgets(line, max_line, stdin) == NULL) {
        if (feof(stdin))
            return 0;
        fprintf(stderr, "Error reading from stdin\n");
        exit(EXIT_FAILURE);
    }
    v = convert_number(line);
    return v;
}

unsigned int convert_number(char *str)
{
    unsigned int v;
    char *p;
    errno = 0;
    v = (unsigned int) strtoul(str, &p, 10);
```

```
    if (errno != 0 || (*p != '\0' && *p != '\n')) {
        fprintf(stderr,
                "Invalid number in line: %s (%d)\n",
                str, *p);
        exit(EXIT_FAILURE);
    }
    return v;
}
```

La funzione `write_u32_to_pipe()` scrive un numero binario su di una pipe, facendo attenzione ad eventuali scritture terminate prematuramente:

```
void write_u32_to_pipe(unsigned int v, int fd)
{
    int len = sizeof(v);
    char *b = (char *) &v;
    do {
        int rc = write(fd, b, len);
        if (rc == -1) {
            perror("write");
            exit(EXIT_FAILURE);
        }
        len -= rc;
        b += rc;
    } while (len > 0);
}
```

La funzione `read_u32_from_pipe()` è analoga: legge un numero in formato nativo da una pipe:

```
unsigned int read_u32_from_pipe(int fd)
{
    unsigned int v;
    int len = sizeof(v);
    char *b = (char *) &v;
    do {
        int rc = read(fd, b, len);
        if (rc == -1) {
            perror("read");
            exit(EXIT_FAILURE);
        }
        len -= rc;
        b += rc;
    } while (len > 0);
    return v;
}
```

La funzione `do_P2_work()` esegue il lavoro richiesto al processo P_2: legge un numero dalla pipe A e scrive sulla pipe B la lista dei suoi fattori primi elementari:

```
void do_P2_work(int piper, int pipew)
{
    unsigned int v, prime;
    for (;;) {
        v = read_u32_from_pipe(piper);
        if (v == 0) {
            write_u32_to_pipe(0, pipew);
            exit(EXIT_SUCCESS); /* end of work */
        }
        /* factorize v */
        prime = 2;
        while (v > 1) {
            if (v % prime == 0) { /* prime divides v */
                write_u32_to_pipe(prime, pipew);
                v /= prime;
                /* try again the same prime */
            } else
                prime += 1 + (prime & 1);
                    /* skip even numbers */
        }
        write_u32_to_pipe(0, pipew); /* end of list */
    }
}
```

Si noti che ricevere il numero 0 dalla pipe A viene interpretato come segnale di terminazione per il processo. Il primo fattore provato è 2, poi vengono provati tutti i numeri dispari (questo è ovviamente un algoritmo di fattorizzazione molto inefficiente, ma adeguato per gli scopi di questo esercizio).

La funzione `do_P3_work()` esegue il lavoro richiesto al processo P_3: legge una lista di fattori primi elementari dalla pipe B e scrive sulla pipe C una lista equivalente nel formato *primo, esponente, primo, esponente, ...* 0.

```
void do_P3_work(int piper, int pipew)
{
    unsigned int v, w, exp;
    for (;;) {
        v = read_u32_from_pipe(piper);
        if (v == 0)
            exit(EXIT_SUCCESS);
        exp = 1;
        do {
            w = read_u32_from_pipe(piper);
            if (v == w)
                exp++;
            if (v != w) {
```

```
                write_u32_to_pipe(v, pipew);
                write_u32_to_pipe(exp, pipew);
                exp = 1;
            }
            v = w;
        } while (w != 0);
        write_u32_to_pipe(0, pipew); /* end of list */
    }
}
```

Anche in questo caso ricevere come primo numero di una lista il valore 0 viene intepretato dal processo P_3 come segnale di terminazione.