Sistemi Operativi **(M. Cesati)**

Compito scritto del 3 febbraio 2015

Nome: _____ **Cognome:** _____

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

Crediti da conseguire: [5] [6] [9]

Scrivere i dati richiesti in stampatello. Al termine consegnare tutti i fogli. Per conseguire un voto sufficiente è necessario ottenere un voto non gravemente insufficiente in entrambi gli esercizi. Tempo a disposizione: 2,5 ore.

Esercizio 1. Una matrice rettangolare di numeri di tipo `int` è memorizzata in un file con il seguente formato binario:

- All'inizio del file sono memorizzate le dimensioni N (numero di righe) e M (numero di colonne) della matrice come sequenza di byte corrispondenti al formato nativo del calcolatore.
- Di seguito sono memorizzati riga per riga gli $N \times M$ elementi della matrice, ciascuno come sequenza di byte nel formato nativo del calcolatore.

Ad esempio, se il calcolatore è little-endian e con interi `int` di 32 bit, la matrice $\begin{pmatrix} 100 & 200 \\ 300 & 400 \\ 500 & 600 \end{pmatrix}$ è memorizzata con la sequenza di byte nel file:

3 0 0 0 2 0 0 0 100 0 0 0 200 0 0 0 44 1 0 0 144 1 0 0 244 1 0 0 88 2 0 0

Realizzare una applicazione C/POSIX multiprocesso che riceva come argomento tre percorsi di file. I primi due file sono di ingresso e contengono ciascuno una matrice rettangolare con la codifica appena descritta, con dimensioni (non prefissate) $N \times M$ e $M \times P$. Il terzo file è di uscita: alla fine dell'esecuzione dovrà memorizzare con la stessa codifica la matrice corrispondente al prodotto "riga-colonna" delle due matrici di ingresso:

$$c_{i,j} = \sum_{k=1}^{M} a_{i,k} \cdot b_{k,j}, \qquad i = 1, \ldots N, \quad j = 1, \ldots P$$

Se le dimensioni delle matrici rettangolari di ingresso non consentono di effettuare il prodotto (ossia il numero di colonne della prima matrice non coincide con il numero di righe della seconda matrice) l'applicazione deve terminare segnalando un errore. L'applicazione deve utilizzare $N \times P$ processi, ciascuno dei quali svolgerà il calcolo di un elemento della matrice d'uscita. Si tengano in considerazione gli eventuali problemi dovuti all'esecuzione concorrente dei vari processi.

Esercizio 2. Si descrivano i principali algoritmi di schedulazione dei processi utilizzati nei moderni Sistemi Operativi.

Sistemi Operativi (M. Cesati)

Esempio di programma del compito scritto del 3 febbraio 2015

Svolgiamo l'esercizio seguendo un approccio "top-down". Decidiamo preliminarmente di mappare i file in memoria per accedere alle matrici di ingresso e uscita. Pertanto possiamo considerare le operazioni principali del programma: (1) apertura e mappatura dei file di ingresso, (2) creazione e mappatura del file di uscita, (3) calcolo e scrittura degli elementi della matrice prodotto.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    int fdA, fdB;  /* file descriptors */
    int rA, rB, cA, cB; /* dimensions of input matrices */
    int *mA, *mB, *mC;

    if (argc != 4) {
        fprintf(stderr,"Usage: %s <filename> <filename> <filename>\n",
                argv[0]);
        return EXIT_FAILURE;
    }

    fdA = open_file(argv[1]);
    map_matrix(fdA, &mA, &rA, &cA);
    if (close(fdA) == -1)
        perror(argv[1]);  /* non fatal error */

    fdB = open_file(argv[2]);
    map_matrix(fdB, &mB, &rB, &cB);
    if (close(fdB) == -1)
        perror(argv[2]);  /* non fatal error */

    if (cA != rB) {
        fprintf(stderr,
                "Input matrices have wrong dimensions, aborting\n");
        exit(EXIT_FAILURE);
    }

    mC = create_output_matrix(argv[3], rA, cB);
    matrix_product(mC, mA, mB, rA, cA, cB);
    return EXIT_SUCCESS;
```

```
}
```

Per aprire il file in lettura controllando gli eventuali errori:

```
int open_file(const char *path)
{
    int fd = open(path, O_RDONLY);
    if (fd == -1) {
        perror(path);
        exit(EXIT_FAILURE);
    }
    return fd;
}
```

La funzione `map_matrix()` legge le dimensioni della matrice dal file e mappa la matrice in memoria:

```
void map_matrix(int fd, int **matr, int *row, int *col)
{
    int r, c;
    int *m;

    /* read dimensions of the matrix */

    r = read_int(fd);
    c = read_int(fd);

    if (r <= 0 || c <= 0) {
        fprintf(stderr,
                "Non positive dimension in input file, aborting\n");
        exit(EXIT_FAILURE);
    }

    *row = r;
    *col = c;

    /* create a memory mapping of the matrix */

    m = mmap(NULL, (2+r*c)*sizeof(int), PROT_READ,MAP_PRIVATE, fd,0);
    if (m == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    *matr = m+2; /* skip the matrix dimensions */
}
```

Per leggere un intero dal file utilizziamo la funzione `read_int()`. Poiché gli interi sono memorizzati nel file nel formato nativo dell'architettura, è sufficiente copiare i byte che costituiscono l'intero in una variabile di tipo `int`, esattamente nell'ordine in cui si trovano nel file.

```
int read_int(int fd)
{
    int v, c;

    c = read(fd, &v, sizeof(v));
    if (c == sizeof(v))
        return v;

    if (c == -1 && errno != EINTR) {
        perror("Read from input file");
        exit(EXIT_FAILURE);
    }

    if (c == 0) {
        fprintf(stderr, "Unexpected end of file\n");
        exit(EXIT_FAILURE);
    }

    /* a short read: either a signal, or a partial read from a FIFO...
        */

    fprintf(stderr, "Read operation interrupted, aborting\n");
    exit(EXIT_FAILURE);
}
```

Per creare il file di uscita e mapparlo in memoria utilizziamo la funzione `create_output_matrix()`. L'unica complicazione è che è necessario estendere il file fino alla sua dimensione finale per poterlo mappare in memoria. Ciò non costituisce un problema in quanto le dimensioni della matrice risultato sono già note.

```
int * create_output_matrix(const char *path, int row, int col)
{
    int *m;
    int fd, size = (row*col+2)*sizeof(int);

    /* create the output file */

    fd = open(path, O_RDWR|O_CREAT, 0644);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    /* write the dimensions */

    write_int(fd, row);
```

```
    write_int(fd, col);

    /* extend the file size */

    if (lseek(fd, size-sizeof(int), SEEK_SET) == -1) {
        perror("lseek");
        exit(EXIT_FAILURE);
    }

    write_int(fd, fd); /* actual data does not matter */

    m = mmap(NULL, size, PROT_WRITE, MAP_SHARED, fd, 0);
    if (m == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    if (close(fd) == -1)
        perror("close"); /* non fatal error */

    return m+2;
}
```

Per scrivere un intero nel file utilizziamo la funzione `write_int()`, analoga alla `read_int()` già definita:

```
void write_int(int fd, int v)
{
    int c;

    c = write(fd, &v, sizeof(v));
    if (c == sizeof(v))
        return;

    if (c == -1 && errno != EINTR) {
        perror("Write to output file");
        exit(EXIT_FAILURE);
    }

    fprintf(stderr, "Write operation interrupted, aborting\n");
    exit(EXIT_FAILURE);
}
```

Finalmente per calcolare la matrice prodotto definiamo la funzione `matrix_product()`. Per ciascun elemento della matrice risultato, la funzione crea un nuovo processo che effettua il calcolo riga-per-colonna.

```
void matrix_product(int *C, int *A, int *B, int rA, int cA, int cB)
{
    int i, j;
    pid_t pid;

    for (i=0; i<rA; ++i)
        for (j=0; j<cB; ++j) {
            pid = fork();
            if (pid == -1) {
                fprintf(stderr, "Cannot fork a process\n");
                exit(EXIT_FAILURE);
            }
            if (pid == 0) {
                prod_rowcol(i, j, C, A, B, cA, cB);
                return;
            }
        }
}
```

La funzione `prod_rowcol()` effettua il calcolo di un elemento della matrice risultato. Poiché il numero di colonne delle matrici non è costante (noto al momento della compilazione), non è possibile far riferimento ad un elemento della matrice `m` con la notazione naturale `m[i][j]`, ed è perciò necessario calcolare l'indirizzo dell'elemento in modo esplicito.

```
void prod_rowcol(int i, int j, int *C, int *A, int *B, int cA, int cB)
{
    int k, v;
    int *row, *col;

    row = A + (i*cA);
    col = B + j;
    for (k = v = 0; k < cA; ++k, ++row, col += cB)
        v += (*row) * (*col);
    *(C+(i*cB)+j) = v;
}
```