Sistemi Operativi **(M. Cesati)**

Compito scritto del 9 luglio 2012

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.

Esercizio 1. Una matrice quadrata di numeri di tipo `float` è memorizzata in un file con il seguente formato:

- All'inizio del file è memorizzata la dimensione N (numero di righe e numero di colonne) della matrice come sequenza di cifre ASCII decimali
- Di seguito, separati da caratteri "spazio", sono memorizzati gli N^2 elementi della matrice riga per riga come sequenza di cifre ASCII decimali e punti decimali.

Ad esempio, la matrice $\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{pmatrix}$ è memorizzata con la sequenza di caratteri nel file:

```
2 1.0 2.0 3.0 4.0
```

a) Realizzare un programma C/POSIX che riceva come argomento il percorso del file e scriva in standard output il prodotto delle somme degli elementi delle colonne della matrice.

Ad esempio, per la matrice 2×2 precedente il risultato è $(1.0 + 3.0) \cdot (2.0 + 4.0) = 24.0$.

b) Modificare il programma precedente in modo che il calcolo delle somme degli elementi delle colonne sia effettuato in parallelo con thread POSIX.

Esercizio 2. La sincronizzazione dei flussi di esecuzione nel nucleo di un sistema operativo: descrivere in modo esauriente la problematica e le soluzioni hardware e software adottate nei moderni sistemi operativi.

Sistemi Operativi (M. Cesati)

Esempio di programma del compito scritto del 9 luglio 2012

Esercizio 1-a

Svolgiamo l'esercizio seguendo un approccio "top-down". Possiamo considerare le operazioni principali del programma come (1) apertura del file, (2) lettura della matrice, e (3) calcolo del valore richiesto:

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

int main(int argc, char *argv[])
{
    FILE *f;
    float *M, val;
    int N;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return EXIT_FAILURE;
    }
    f = open_file(argv[1]);
    read_matrix(f, &M, &N);
    val = prodsumcol0(M, N);
    printf("Result (sequent): %f\n", val);
    return EXIT_SUCCESS;
}
```

Per aprire il file in lettura controllando gli eventuali errori:

```
FILE *open_file(const char *path)
{
    FILE *f = fopen(path, "r");
    if (f == NULL) {
        perror(path);
        exit(EXIT_FAILURE);
    }
    return f;
}
```

La funzione `read_matrix()` legge la dimensione della matrice dal file, alloca spazio per la matrice in memoria, e poi legge ad uno ad uno gli elementi della matrice:

```
void read_matrix(FILE *f, float **M, int *N)
{
    int n;
    float *m;

    if (fscanf(f, "%d", &n) != 1) {
        perror("in fscanf");
        exit(EXIT_FAILURE);
    }
    *N = n;
    m = malloc(n*n*sizeof(float));
    if (m == NULL) {
        perror("in malloc");
        exit(EXIT_FAILURE);
    }
    *M = m;
    n *= n;
    for (; n > 0; --n) {
        float v;
        if (fscanf(f, "%f", &v) != 1) {
            perror("in fscanf");
            exit(EXIT_FAILURE);
        }
        *m++ = v;
    }
}
```

Il calcolo del valore richiesto è delegato alla funzione `prodsumcol0()`:

```
float prodsumcol0(float *m, int N)
{
    int c;
    float prod = 1.0;

    for (c=0; c<N; ++c)
        prod *= sumcol0(m+c, N);
    return prod;
}
```

La somma dei valori di una colonna è effettuata dalla funzione `sumcol0()`. Si noti che la funzione è invocata con l'indirizzo del primo elemento della colonna, ed ottiene gli indirizzi degli elementi successivi aggiungendo al puntatore il numero di elementi in una riga.

```
float sumcol0(float *m, int N)
{
    int r;
    float sum = 0.0;
```

```
    for (r=0; r<N; ++r, m+=N)
        sum += *m;
    return sum;
}
```

Esercizio 1-b

Il calcolo delle somme degli elementi della matrice deve essere svolto in parallelo, dunque si dovranno creare tanti thread quante sono le colonne della matrice.

Adattiamo il programma sviluppato per il punto precedente. È innanzi tutto necessario includere un nuovo header file e modificare la funzione `main()` in modo da invocare una nuova procedura di calcolo.

```
[...]
#include <pthread.h>

int main(int argc, char *argv[])
{
    [...]
    printf("Result (sequent): %f\n", val);
    val = prodsumcol(M, N);
    printf("Result (pthread): %f\n", val);
    return EXIT_SUCCESS;
}
```

Per tenere traccia del lavoro svolto da ciascun thread definiamo una struttura di dati apposita:

```
struct thread_job {
    pthread_t tid;      /* Thread ID */
    float *col;         /* Primo elemento della colonna */
    int N;              /* Dimensione della colonna */
    float sum;          /* Somma degli elementi della colonna */
};
```

La procedura `prodsumcol()` crea N thread ed aspetta che ciascuno di essi completi il proprio lavoro. Il risultato calcolato da ciascun thread, registrato entro la struttura di dati `thread_job`, è utilizzato per calcolare il prodotto finale.

```
float prodsumcol(float *m, int N)
{
    int c;
    float prod = 1.0;
    struct thread_job *tv;
```

```
    /* Allocazione del vettore di thread_job */

    tv = malloc(N*sizeof(struct thread_job));
    if (tv == NULL) {
        perror("malloc");
        exit(EXIT_FAILURE);
    }

    /* Creazione degli N thread */

    for (c=0; c<N; ++c) {
        tv[c].col = m+c;
        tv[c].N = N;
        if (pthread_create(&tv[c].tid, NULL, sumcol, tv+c) != 0) {
            fprintf(stderr, "Error in pthread_create()\n");
            exit(EXIT_FAILURE);
        }
    }

    /* Attesa del completamento di ciascun thread */

    for (c=0; c<N; ++c) {
        if (pthread_join(tv[c].tid, NULL) != 0) {
            fprintf(stderr, "Error in pthread_join()\n");
            exit(EXIT_FAILURE);
        }
        prod *= tv[c].sum;
    }
    return prod;
}
```

Infine la procedura `sumcol()` è eseguita da ciascuno degli N thread per calcolare la somma di una colonna. I parametri su cui operare vengono letti dal puntatore alla struttura di dati `thread_job` il cui valore è ricavato, con un opportuno *cast*, dall'argomento passato alla procedura.

```
void *sumcol(void *p)
{
    struct thread_job *j = (struct thread_job *) p;
    float *m = j->col;
    int r;

    j->sum = 0.0;
    for (r=0; r < j->N; ++r, m += j->N)
        j->sum += *m;
    pthread_exit(NULL); /* si puo' omettere */
}
```