Sistemi Operativi **(M. Cesati)**

Compito scritto del 28 giugno 2016 (Traccia A)

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

Scrivere una applicazione multiprocesso C/POSIX il cui scopo è determinare le approssimazioni del numero π basate sulla seguente formula (algoritmo di Gauss-Legendre):

$$a_0 = 1, \quad b_0 = \frac{1}{\sqrt{2}}, \quad t_0 = \frac{1}{4}, \quad p_0 = 1$$

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n \cdot b_n}$$

$$t_{n+1} = t_n - p_n \cdot (a_n - a_{n+1})^2, \quad p_{n+1} = 2 \cdot p_n$$

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4 \cdot t_{n+1}}$$

L'applicazione è costituita da tre processi P_1, P_2 e P_3. Il processo P_1 calcola i valori successivi a_i e b_i, e li invia tramite pipe al processo P_2. Il processo P_2 calcola i valori successivi di t_i e p_i, ed invia tutti i valori necessari a P_3 tramite pipe. Il processo P_3 calcola le approssimazioni successive di π e le scrive in standard output.
Tutte le variabili utilizzate debbono essere in virgola mobile con precisione doppia. Si trascurino i problemi legati alla precisione delle variabili, ma si presti attenzione alla sincronizzazione tra i processi. Per calcolare la radice quadrata si utilizzi la funzione `sqrt()`.

Esercizio 2

Si spieghi il significato del termine "memoria virtuale" nel contesto di un moderno sistema operativo, e si descrivano i principali algoritmi o meccanismi software utilizzati per implementare tale concetto.

Sistemi Operativi **(M. Cesati)**

Compito scritto del 28 giugno 2016 (Traccia B)

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

Scrivere una applicazione multiprocesso C/POSIX il cui scopo è determinare le approssimazioni del numero π basate sulla seguente formula (algoritmo di Gauss-Legendre):

$$a_0 = 1, \quad b_0 = \frac{1}{\sqrt{2}}, \quad t_0 = \frac{1}{4}, \quad p_0 = 1$$

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n \cdot b_n}$$

$$t_{n+1} = t_n - p_n \cdot (a_n - a_{n+1})^2, \quad p_{n+1} = 2 \cdot p_n$$

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4 \cdot t_{n+1}}$$

L'applicazione è costituita da tre processi P_1, P_2 e P_3 che comunicano tramite code di messaggi. Il processo P_1 calcola i valori successivi a_i e b_i, e li invia al processo P_2. Il processo P_2 calcola i valori successivi di t_i e p_i, ed invia tutti i valori necessari a P_3. Il processo P_3 calcola le approssimazioni successive di π e le scrive in standard output.
Tutte le variabili utilizzate debbono essere in virgola mobile con precisione doppia. Si trascurino i problemi legati alla precisione delle variabili, ma si presti attenzione alla sincronizzazione tra i processi. Per calcolare la radice quadrata si utilizzi la funzione `sqrt()`.

Esercizio 2

Si descrivano i principali algoritmi o meccanismi software utilizzati nei moderni sistemi operativi per la gestione della memoria fisica del calcolatore, spiegando esaurientemente le motivazioni alla base di ciascuno di essi.

Sistemi Operativi (M. Cesati)

Esempi dei programmi del compito scritto del 28 giugno 2016

Esercizio 1, traccia A

I tre processi debbono comunicare tramite pipe. La soluzione più semplice consiste nell'utilizzare una pipe per i dati inviati da P_1 a P_2, ed un'altra pipe per i dati inviati da P_2 a P_3.

Svolgiamo l'esercizio seguendo un approccio "top-down". La funzione `main()` inizializza i canali di comunicazione e crea i processi necessari:

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

int main()
{
    init_comm_channels();
    spawn_processes();
}
```

La funzione `init_comm_channels()` crea le due pipe, salvando i quattro descrittori in due vettori globali:

```
int pipe12[2];
int pipe23[2];

void init_comm_channels(void)
{
    int rc = pipe(pipe12);
    abort_on_error(rc == -1, "Error in 1st pipe()");
    rc = pipe(pipe23);
    abort_on_error(rc == -1, "Error in 2nd pipe()");
}
```

`abort_on_error()` è una macro che controlla la condizione indicata nel primo argomento; nel caso sia vera (condizione d'errore), viene stampato un messaggio d'errore e il processo viene terminato:

```
#define abort_on_error(cond, msg) do { \
    if (cond) { \
        fprintf(stderr, "%s (%d)\n", msg, errno); \
        exit(EXIT_FAILURE); \
    } \
} while (0)
```

La funzione `main()` continua invocando la funzione `spawn_processes()`, che ha il compito di creare gli altri processi dell'applicazione:

```
void spawn_processes(void)
{
    pid_t p = fork();
    abort_on_error(p == -1, "Error in 1st fork()");
    if (p == 0)
        do_P1_work();
    p = fork();
    abort_on_error(p == -1, "Error in 2nd fork()");
    if (p == 0)
        do_P2_work();
    do_P3_work();
}
```

Le funzioni `do_P1_work()`, `do_P2_work()` e `do_P3_work()` implementano gli algoritmi specifici di ciascun processo. Ciascuna di queste funzioni esegue un ciclo senza fine calcolando sempre nuove approssimazioni delle variabili utilizzate dall'algoritmo di Gauss-Legendre.

La funzione `do_P1_work()` esegue il calcolo delle variabili di tipo a_i e b_i:

```
void do_P1_work(void)
{
    double a = 1.0, b = 1.0/sqrt(2);
    int rc = close(pipe23[0]);
    abort_on_error(rc != 0, "Error closing pipe23 r-fd");
    rc = close(pipe23[1]);
    abort_on_error(rc != 0, "Error closing pipe23 w-fd");
    rc = close(pipe12[0]);
    abort_on_error(rc != 0, "Error closing pipe12 r-fd");
    for (;;) {
        double new_a = ( a + b )/2.0;
        double new_b = sqrt( a * b );
        send_value(pipe12[1], new_a);
        send_value(pipe12[1], new_b);
        a = new_a;
        b = new_b;
    }
```

```
}
```

Inizialmente la funzione chiude i descrittori delle pipe non utilizzati dal processo P_1. In ogni iterazione del ciclo senza fine vengono calcolati i nuovi valori delle variabili a_i e b_i, che vengono poi scritti sulla pipe `pipe12`.

La scrittura è realizzata dalla generica funzione `send_value()`, che si prende cura di scrivere tutti i byte del numero in virgola mobile gestendo anche eventuali terminazioni premature della chiamata di sistema `write()`:

```
void send_value(int fd, double v)
{
    int rc, size = sizeof(v);
    unsigned char *p = (unsigned char *) &v;
    while (size > 0) {
        rc = write(fd, p, size);
        abort_on_error(rc == -1, "Error writing to pipe");
        size -= rc;
        p += rc;
    }
}
```

La funzione `do_P2_work()` implementa in modo analogo l'algoritmo seguito dal processo P_2:

```
void do_P2_work(void)
{
    double prev_a = 1.0;
    double t = 0.25;
    double p = 1.0;
    double a, b;
    int rc = close(pipe23[0]);
    abort_on_error(rc != 0, "Error closing pipe23 r-fd");
    rc = close(pipe12[1]);
    abort_on_error(rc != 0, "Error closing pipe12 w-fd");
    for (;;) {
        double x;
        recv_value(pipe12[0], &a);
        recv_value(pipe12[0], &b);
        x = prev_a - a;
        t -= p * (x * x);
        p *= 2.0;
        send_value(pipe23[1], a);
        send_value(pipe23[1], b);
        send_value(pipe23[1], t);
        prev_a = a;
    }
```

```
}
```

La funzione `recv_value()` è naturalmente l'analoga di `send_value()`, e serve a leggere un valore di tipo `double` da una pipe:

```
void recv_value(int fd, double *v)
{
    int rc, size = sizeof(*v);
    unsigned char *p = (unsigned char *) v;
    while (size > 0) {
        rc = read(fd, p, size);
        abort_on_error(rc == -1, "Error reading pipe");
        size -= rc;
        p += rc;
    }
}
```

Infine la funzione `do_P3_work()` implementa l'algoritmo del processo P_3:

```
void do_P3_work(void)
{
    double a, b, t;
    int rc = close(pipe12[0]);
    abort_on_error(rc != 0, "Error closing pipe12 r-fd");
    rc = close(pipe12[1]);
    abort_on_error(rc != 0, "Error closing pipe12 w-fd");
    rc = close(pipe23[1]);
    abort_on_error(rc != 0, "Error closing pipe23 w-fd");
    for (;;) {
        double x, pi;
        recv_value(pipe23[0], &a);
        recv_value(pipe23[0], &b);
        recv_value(pipe23[0], &t);
        x = a + b;
        pi = ( x * x ) / ( 4.0 * t );
        printf("%.30f\n", pi);
    }
}
```

Esercizio 1, traccia B

L'esercizio è del tutto analogo all'esercizio dato nella traccia A, con l'unica differenza che i tre processi debbono comunicare tramite code di messaggi invece che pipe.

In questo caso la soluzione più semplice consiste nel creare una unica coda di messaggi privata dell'applicazione, ed identificare il mittente di ciascun messaggio utilizzando l'opportuno campo `mtype` definito nella struttura `struct msgbuf` che descrive il formato del messaggio.

La funzione `main()` è identica al programma della traccia A:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main()
{
    init_comm_channels();
    spawn_processes();
}
```

La funzione `init_comm_channels()` è invece differente:

```
int mq;

void init_comm_channels(void)
{
    mq = msgget(IPC_PRIVATE, 0600);
    abort_on_error(mq == -1,
                   "Error in message queue creation");
}
```

La variabile gloale `mq` memorizza l'identificatore della nuova coda di messaggi. Si noti che il valore in ottale `0600` indica che la coda viene creata con i soli permessi di lettura e scrittura per l'utente che esegue il programma. La macro `abort_on_error()` è del tutto identica a quella del programma della traccia A.

Anche la funzione `spawn_processes()` è identica a quella del programma della traccia A, dunque vengono creati altri due processi, e poi si eseguono le funzioni `do_P1_work()`, `do_P2_work()` e `do_P3_work()`. Cominciamo dalla funzione `do_P1_work()`:

```
void do_P1_work(void)
{
    double a = 1.0, b = 1.0/sqrt(2);
    for (;;) {
        double new_a = ( a + b )/2.0;
        double new_b = sqrt( a * b );
        send_values(TYPE_FROM_P1, new_a, new_b, 0.0);
        a = new_a;
        b = new_b;
    }
}
```

La funzione `send_values()` ha il compito di inviare un messaggio con tre valori di tipo `double` alla coda di messaggi. Nel caso specifico, P_1 ha necessità di spedire due soli valori, perciò il terzo valore è impostato a zero (un valore qualsiasi).

```
#define TYPE_FROM_P1 1
#define TYPE_FROM_P2 2

struct msgbuf {
    long mtype;
    double v[3];
};

void send_values(int type, double v1, double v2, double v3)
{
    int rc;
    struct msgbuf mb;
    mb.mtype = type;
    mb.v[0] = v1;
    mb.v[1] = v2;
    mb.v[2] = v3;
    rc = msgsnd(mq, &mb, sizeof(mb.v), 0);
    abort_on_error(rc == -1, "Error in msgsnd()");
}
```

L'argomento `type` di `send_values()` è il tipo del messaggio, che in questo caso coincide con numero del processo mittente: `TYPE_FROM_P1` per il processo P_1, e `TYPE_FROM_P2` per il processo P_2.

La funzione `do_P2_work()` è analoga a quanto già visto:

```
void do_P2_work(void)
{
    double prev_a = 1.0;
    double t = 0.25;
    double p = 1.0;
    double a, b;
    for (;;) {
        double x;
        recv_values(TYPE_FROM_P1, &a, &b, &x);
        x = prev_a - a;
        t -= p * (x * x);
        p *= 2.0;
        send_values(TYPE_FROM_P2, a, b, t);
        prev_a = a;
    }
}
```

La funzione `read_values()` legge dalla coda di messaggi il più vecchio messaggio del tipo indicato come primo argomento (in questo caso, spedito dal processo P_1), e copia i valori `double` in esso contenuti nelle tre variabili i cui indirizzi sono passati come argomenti. (In questo caso particolare, il processo P_1 spedisce due soli valori significativi; il terzo valore viene copiato nella variabile `x`, che però viene sovrascritta subito dopo.)

```
void recv_values(int type, double *v1, double *v2,
                 double *v3)
{
    struct msgbuf mb;
    int rc = msgrcv(mq, &mb, sizeof(mb.v), type, 0);
    abort_on_error(rc == -1, "Error in msgrcv()");
    *v1 = mb.v[0];
    *v2 = mb.v[1];
    *v3 = mb.v[2];
}
```

Infine, la funzione `do_P3_work()` è la seguente:

```
void do_P3_work(void)
{
    double a, b, t;
    for (;;) {
        double x, pi;
        recv_values(TYPE_FROM_P2, &a, &b, &t);
        x = a + b;
        pi = ( x * x ) / ( 4.0 * t );
        printf("%.30f\n", pi);
    }
}
```