Sistemi Operativi **(M. Cesati)**

Compito scritto del 12 febbraio 2018

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

Scrivere una applicazione C/POSIX multiprocesso che riceve sulla riga comando un elenco di nomi di file presenti nel filesystem. L'elenco è di lunghezza variabile e non prefissata. L'applicazione crea un processo per ciascun file nell'elenco; tutti i processi lavorano in modo concorrente. Ciascun processo ricerca nel proprio file la parola di testo di lunghezza massima; a parità di lunghezza, la parola selezionata dovrà essere quella di ordine lessicografico minore. Una parola è definita come una sequenza di caratteri alfabetici minuscoli o maiuscoli. Al termine l'applicazione scrive in standard output la parola di lunghezza massima tra tutte quelle trovate all'interno dei file; a parità di lunghezza, vince la parola di ordine lessicografico minore. Curare gli aspetti di sincronizzazione tra i processi, ove necessario.

Sistemi Operativi **(M. Cesati)**

Esempio di programma del compito scritto del 12 febbraio 2018

Esercizio 1

Descriviamo il programma utilizzando un approccio "top-down". La funzione `main()` crea una coda di messaggi per la comunicazione con gli altri processi dell'applicazione, crea i processi figli, riceve le stringhe trovate dai figli, e stampa la stringa massimale risultante. Infine, `main()` termina distruggendo la coda di messaggi.

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

int main(int argc, char *argv[])
{
    int v, msq;
    char *maxstr;
    abort_on_error(argc <= 1,
                   "Specify some file names as argument");
    msq = msgget(IPC_PRIVATE, S_IRUSR|S_IWUSR);
    abort_on_error(msq == -1,
                   "Message queue creation failure");
    fork_children(argc-1, argv+1, msq);
    maxstr = get_strings(argc-1, msq);
    printf("Maximal string: %s (length %ld)\n", maxstr,
           strlen(maxstr));
    v = msgctl(msq, IPC_RMID, NULL);
    abort_on_error(v == -1,
                   "Message queue destruction failure");
    return EXIT_SUCCESS;
}
```

La macro `abort_on_error()` viene utilizzata per controllare le condizioni d'errore e, se necessario, interrompere prematuramente l'esecuzione del programma.

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

Per creare i processi figli `main()` invoca la funzione `fork_children()`:

```
void fork_children(int n, char *names[], int msq)
{
    for (int i=0; i<n; ++i) {
        pid_t p = fork();
        abort_on_error(p == -1, "Error in fork()");
        if (p == 0)
            child_work(names[i], msq);
    }
}
```

Ciascun figlio esegue la funzione `child_work()`, che sarà esaminata in seguito.

La funzione `main()` continua invocando `get_strings()` per ricevere le stringhe trovate dai processi figli:

```
char *get_strings(int n, int msq)
{
    char *max = NULL;
    int i, maxlen = 0;
    for (i=0; i<n; ++i) {
        char *s = get_1_string(msq);
        int sl = strlen(s);
        if (sl < maxlen ||
            (sl == maxlen && strcmp(s, max) >= 0)) {
            free(s);
        } else {
            free(max);
            max = s;
            maxlen = sl;
        }
    }
    return max;
}
```

La funzione riceve ciascuna stringa, ne calcola la lunghezza, e decide se essa rappresenta il nuovo elemento massimale. Per determinare l'ordine lessicografico delle stringhe è sufficiente utilizzare la funzione di libreria `strcmp()`. Si osservi che ad ogni iterazione (tranne la prima) viene rilasciata la memoria occupata da

una stringa (in alternativa, quella appena ricevuta oppure il vecchio elemento massimale). Infatti la funzione `get_1_string()` alloca dinamicamente lo spazio necessario per ciascuna stringa:

```
struct msg_buf {
    long mtype;
    char mtext[1];
};

char *get_1_string(int msq)
{
    int str_len = 128;
    struct msg_buf *m;
    ssize_t rc;
    char *str;
    for (;;) {
        m = malloc(str_len + sizeof(long));
        abort_on_error(!m, "Memory allocation error");
        rc = msgrcv(msq, m, str_len, 1, 0);
        abort_on_error(rc == -1 && errno != E2BIG,
                       "Error in msgrcv");
        if (rc == -1) { /* error E2BIG */
            str_len += str_len;
            free(m);
            continue;
        }
        str = strdup(m->mtext);
        abort_on_error(!str, "Error in strdup");
        free(m);
        return str;
    }
}
```

Questa funzione gestisce messaggi (e quindi stringhe) di lunghezza arbitraria (senza una dimensione massima prefissata). Inizialmente viene fissata una dimensione per le stringa pari a 128 byte. Nel caso in cui questa dimensione sia insufficiente, `msgrcv()` restituisce il codice d'errore `E2BIG`, e quindi la funzione raddoppia lo spazio a disposizione per memorizzare la stringa. Una volta che l'intero messaggio è stato ricevuto, la funzione alloca lo spazio necessario per la sola stringa (tramite `strdup()`) e rilascia lo spazio utilizzato per il messaggio.

Descriviamo ora il codice eseguito dai processi figli. Ciascuno di essi comincia ad eseguire la funzione `child_work()` avendo come argomenti il percorso del file da analizzare e il descrittore di file della coda di messaggi:

```
void child_work(const char *filename, int msq)
{
    FILE *f = open_file(filename);
    char *maxstr = search_max_string(f);
    send_string(msq, maxstr);
```

```
    abort_on_error(fclose(f) == EOF, "Error closing file");
    exit(EXIT_SUCCESS);
}
```

La funzione `open_file()` si limita a cercare di aprire il file assegnato:

```
FILE * open_file(const char *name)
{
    FILE *f = fopen(name, "r");
    abort_on_error(!f, "Cannot open input file");
    return f;
}
```

La ricerca della stringa massimale nel file è affidata a `search_max_string()`:

```
char *search_max_string(FILE *f)
{
    char *str, *max = NULL;
    int sl, rc, maxlen = 0;
    for (;;) {
        rc = fscanf(f, "%m[a-zA-Z]", &str);
        if (rc == EOF) {
            abort_on_error(!feof(f),
                           "Error while reading file");
            break;
        }
        if (rc == 0) {
            (void) fgetc(f);
            continue;
        }
        sl = strlen(str);
        if (sl > maxlen ||
            (sl == maxlen && strcmp(str, max) < 0)) {
            free(max);
            max = str;
            maxlen = sl;
        } else
            free(str);
    }
    return max;
}
```

Per la lettura delle parole costituite da caratteri alfabetici si è scelto di utilizzare la funzione di libreria `fscanf()`, in particolare utilizzando il formato "`%m[a-zA-Z]`". Questo formato legge dal file specificato la sequenza di caratteri compresi tra le parentesi quadre (ossia in questo caso i caratteri dell'alfabeto maiuscoli e minuscoli). La lettura termina non appena si incontra un carattere

non appartente all'insieme dato. La lettera "`m`" nel formato indica invece che la sequenza di caratteri letta deve essere copiata in una stringa allocata dinamicamente dalla libreria stessa. La funzione `fscanf()` al solito restituisce il numero di assegnazioni effettuate. Se dunque il valore restituito è zero, il primo carattere del file ancora da leggere non è alfabetico, e pertanto la funzione lo scarta invocando `fgetc()`. Dopo aver ottenuto una parola di caratteri alfabetici, la funzione la confronta con il miglior candidato già trovato. Per ciascuna parola trovata viene rilasciata una stringa (in alternativa, la nuova parola non massimale od il candidato precedente). Infine la funzione restituisce l'elemento massimale risultante.

L'ultimo fase del lavoro di ciascun processo figlio è l'invio della stringa massimale sulla coda di messaggi, utilizzando la funzione `send_string()`:

```
void send_string(int msq, char *str)
{
    int rc;
    int size_of_msg = sizeof(long) + strlen(str) + 1;
    struct msg_buf *m = malloc(size_of_msg);
    abort_on_error(!m, "Memory allocation error");
    m->mtype = 1; /* anything != 0 */
    strcpy(m->mtext, str);
    rc = msgsnd(msq, m, size_of_msg-sizeof(long), 0);
    abort_on_error(rc == -1,
                   "Error writing to message queue");
    free(m);
}
```

Poiché la stringa non ha una lunghezza massima predeterminata, lo spazio necessario per contenere il messaggio deve essere allocato dinamicamente, e la stringa copiata nel campo dati tramite la funzione di libreria `strcpy()`.