Università degli studi di bologna
facoltà di ingegneria

Presentazione esercitazione 0

Lettura e scrittura file in Java e C

Corso di reti di calcolatori T

- Daniele Magagnoli (capitano)
- Giulia Martina Bal
- Antonio Cassanelli
- Gianmiriano Porrazzo

Anno Accademico 2021-2022

Produttore

Lo scopo del produttore è leggere le righe da input finché non si riceve un EOF e salvare il contenuto in un file specificato come argomento.

Il problema principale è comprendere quando l'utente immette l'EOF, che corrisponde a premere la combinazione di tasti ctrl+d, per unix, e ctrl+z, per windows da terminale, che identificheremo come una stringa nulla (null). Useremo tale stringa nulla per uscire dal ciclo di lettura.

```
do{
    printf("Inserisci la nuova riga\n");
    ch= gets (riga); /* la gets legge tutta la riga,
                    separatori inclusi,e trasforma il fine
                     linea in fine stringa */
    if(ch){
        riga[strlen(riga)+1]='\0';
        riga[strlen(riga)]='\n';
        written = write(fd, riga, strlen(riga));
    }
        // uso della primitiva
    if (written < 0){
        perror("P0: errore nella scrittura sul file");
        exit(3);
    }
}while(ch);
```

```
while((inputl=in.readLine())!=null){
    System.out.println("Inserisci la nuova riga");
    inputl = inputl+"\n";
    fout.write(inputl, 0, inputl.length());
}
```

Consumatore

Per quanto riguarda il consumatore si deve comportare come un programma filtro che prende come input un file passato o come argomento o mediante ridirezione e produce in output l'insieme delle righe a cui sono stati tolti i caratteri passati come ulteriore argomento.

Per gestire la possibilità di passare il file da aprire in lettura nelle due modalità sopra indicate abbiamo pensato di controllare il numero di argomenti passati al programma per distinguere le due casistiche.
Nel caso in cui prendiamo il file da ridirezione, e non da argomento, leggiamo il file aperto come standard input.
Per eliminare i caratteri specificati come argomento abbiamo pensato di usare un ciclo innestato nel ciclo di lettura che confronta carattere per carattere.

```
if (nread >= 0)
{
    for(int i = 0; argv[1][i]!='\0'; i++)
    {
        if(read_char == argv[1][i])
            match=1;
    }
}
```

```
while ((x = r.read()) >= 0) {
    b = false;
    ch = (char) x;
    for(int i = 0; i<args[0].length(); i++)
        if(ch==args[0].charAt(i))
            b=true;
    if(!b) System.out.print(ch);
    /*if(!args[0].contains(ch+""))
        System.out.print(ch);*/
}
r.close();
```

Soluzioni alternative

Le soluzione precedentemente presentate possono essere implementate tramite funzioni di libreria.

Nel caso di Java abbiamo pensato di usare la funzione contains, mentre nel caso di C la funzione strstr.

Dopo aver effettuato dei test in C con file grandi, usare la strstr o la soluzione col ciclo non comporta sostanziali differenze a tempo di esecuzione.
Mentre in Java abbiamo notato che usare la funzione contains comporta maggior tempo nella lettura di file grandi.

```java
while ((x = r.read()) >= 0) {
    ch = (char) x;
    if(!args[0].contains(ch+""))
        System.out.print(ch);
}
r.close();
```

```c
while(nread=read(fd, &read_char, sizeof(char))) {

    if (nread >= 0) {
        if(!(strstr(argv[1], read_char)))
            putchar(read_char[0]);
    }
    else{
        printf("(PID %d) impossibile leggere dal file %s", getpid(), file_in);
        perror("Errore!");
        close(fd);
        exit(3);
    }
}
```

GRAZIE PER L’ATTENZIONE