POLITECNICO DI MILANO





Gestione parametri, librerie e standard I/O

Laboratorio Software 2008-2009

C. Brandolese M. Grotto

Sommario

1. **Gestione degli argomenti**
   - La lista degli argomenti
   - Opzioni da riga di comando
   - Uso degli argomenti
2. **Standard I/O**
   - Descrizione
   - Proprietà
3. **Variabili di ambiente**
   - Descrizione
   - Utilizzo
4. **File temporanei**
   - Descrizione
   - Utilizzo
5. **Supporto al debugging**
   - Le asserzioni
   - Utilizzo
6. **Gestione degli errori**
   - Descrizione
   - Utilizzo
7. **Librerie**
   - Descrizione
   - Confronto
   - Librerie statiche
   - Librerie dinamiche

Gestione degli argomenti

La lista degli argomenti

- Modo per interagire con l'ambiente di esecuzione
  - Consente il passaggio dei parametri dall'esterno

- `argc` e `argv` come parametri della funzione `main()`
  - `int argc` è un intero che rappresenta il numero di elementi presenti nella lista degli argomenti
  - `char **argv` è un array di puntatori a carattere (`char*`)
    - La sua dimensione è `argc`
    - Gli elementi dell'array puntano alla lista di argomenti
    - Il primo elemento è il nome del programma

Gestione degli argomenti

Opzioni da riga di comando

- Due tipi di opzioni
  - Short options
    - Trattino singolo più un solo carattere
    - Più veloci da usare
  - Long options
    - Due trattini, seguiti da un nome
    - Più facili da leggere (script) e ricordare

- Solitamente un programma accetta entrambi i tipi di opzioni
  - Alcune sono le medesime opzioni (`-h` e `-help`)

Gestione degli argomenti

Uso degli argomenti

- `int getopt_long(…)`
  - Il suo utilizzo richiede 2 strutture dati
    - Una stringa di caratteri contenente le short options valide

      Es. La stringa "`ho:v`" indica che le opzioni valide sono "`-h`", "`-o`", "`-v`" e l'opzione "`-o`" richiede un argomento.

```
const char* const short_options = "ho:v";
```

    - Un array di `struct option` che caratterizza le long options valide

```
const struct option long_options[] = {
{ "help", 0, NULL, 'h' },
{ "output", 1, NULL, 'o' },
{ "verbose", 0, NULL, 'v' },
{ NULL, 0, NULL, 0 }
};
```

Gestione degli argomenti

Uso degli argomenti

- Ciclo di scansione della lista delle opzioni
  - `getopt_long` parsa una opzione alla volta e restituisce
    - Il carattere corrispondente alla short options
    - `-1` se la lista è terminata
    - `?` se incontra una options non valida
  - Viene effettuata la specifica azione in base alla opzione
  - Quando una opzione possiede un argomento, la variabile globale `optarg` punta al testo dell'argomento
  - Quando `getopt_long` ha finito il parsing delle options, la variabile globale `optind` contiene l'indirizzo nell'array `argv` del primo argomento che non è una options

Esempio
`getopt_long.c`

Standard I/O

Descrizione

- Streams standard per input e output usati da `scanf()`, `printf()` e dalle altre funzioni di libreria
  - `stdin` e `stdout`
  - Possibilità di concatenare i programmi sfruttando la redirezione ed il pipelining

- Standard error: `stderr`
  - I messaggi d'errore e i warning dovrebbero essere stampati su questo stream

```
fprintf(stderr,"Error: ..");
```

- Stream accessibili anche tramite le chiamate di basso livello
  - `read, write, …`

Standard I/O

Proprietà

- `stdout` è bufferizzato a linea
  - I dati non vengono scritti sulla console fino a quando il buffer non è pieno
  - Possibilità di forzare il flush

```
fflush(stdout);
```

- `stderr` non è bufferizzato (è bufferizzato a carattere)
  - I dati vengono direttamente scritti sulla console

Non stampa ogni secondo ma solo quando il buffer è pieno

```
while (1) {
  printf (".");
  sleep (1);
}
```

Stampa ogni secondo

```
while (1) {
  fprintf (stderr, ".");
  sleep (1);
}
```

Variabili di ambiente

Descrizione

- Le varibili di ambiente
  - Contengono informazioni relative alla sessione corrente
  - Determinano alcuni aspetti della configurazione dell'ambiente

- Varibili di uso comune:
  - `HOME:` Path della home directory dell'utente
  - `USER:` Username dell'utente

- `int main ( int argc, char **argv, char **env )`

- `environ`: variabile globale speciale definita nella GNU C Library
  - Di tipo `char**`
    - Array di puntatori a stringhe, ognuna nella forma `NAME=valore`
  - Sconsigliata la modifica di `environ`

```
extern char** environ;
```

Esempio
`print-env.c`

Variabili di ambiente

Utilizzo

- `char* getenv(const char* name);`
  - Restituisce il valore della variabile specificata
    - `NULL` se la variabile non è stata definita

```
getenv("VARIABLE_NAME");
```

- `int setenv(const char *name, const char *value, int overwrite);`
  - Aggiunge la variabile di ambiente `name` con il valore `value` se `name` non esiste
  - Se `name` esiste il suo valore
    - Verrà modificato se `overwrite` è non-zero
    - Non verrà modificato se `overwrite` è 0

- `int unsetenv(const char* name);`
  - Cancella la variaile `name` dall'ambiente

Esempio
`client.c`

File Temporanei

Descrizione

- Utili per memorizzare grandi quantità di dati o passare dati ad altri programmi
  - In GNU/Linux vengono memorizzati in `/tmp`
- Possibili problemi:
  - Più istanze del programma in esecuzione
    - nomi univoci per evitare perdita di dati
  - Inserimento di dati “maliziosi”
    - impostare I corretti permessi d’accesso
  - Possibili attacchi se I nomi dei file sono noti
    - impredicibilità
- GNU/Linux mette a disposizione le funzioni `mkstemp` e `tmpfile` che si prendono cura di questi problemi

File Temporanei

Utilizzo

- `int mkstemp(char *template);`
  - Crea un file temporaneo con un nome derivato da un template
    - Nome di file che termina con XXXXXX
    - I permessi di lettura/scrittura sono solo per l'utente corrente
  - Apre il file e restituisce un descrittore di file da usare con la famiglia di funzioni `write()`
  - I file non vengono cancellati automaticamente
  - Se il file è ad uso interno è opportuno invocare `unlink()`
    - Rimuove la directory entry corrispondente al file
    - I file sono reference-counted
    - Non viene rimosso fino alla chiusura con `fclose()` o alla terminazione del programma

Esempio

`temp_file.c`

File Temporanei

Utilizzo

- `FILE *tmpfile(void);`
  - Usata se il file non serve per il dialogo con altri programmi
  - Crea ed apre un file temporaneo e ritorna un file pointer
  - Già invocata la `unlink()`
    - Dopo una chiamata a `fclose()` o alla terminazione del programma il file è rimosso

Supporto al debugging

Le asserzioni

- `assert (test);`
  - È una macro. Valuta l'espressione condizonale `test`
    - Se l'espressione è vera, non ha alcun effetto
    - Se l'espressione è falsa, stampa un messaggio di errore e termina mediante la funzione `abort()`
  - L'argomento deve essere una espressione booleana
  - Serve anche come documentazione del codice sorgente

- `NDEBUG`
  - `assert()` comporta una penalità in termini di prestazioni
  - Utile in fase di debug ma da evitare nella versione dell'applicazione da rilasciare
  - Passare l'opzione `-DNDEBUG` a `gcc` in fase di compilazione
    - Le invocazioni ad `assert()` vengono eliminate in fase di preprocessing

Supporto al debugging

Utilizzo

- Come si deve usare `assert()`

```
for (i = 0; i < 100; ++i)
{
  int status = do_something();
  assert(status == 0);
}
```

- Come NON va usata `assert()`

```
for (i = 0; i < 100; ++i)
  assert(do_something() == 0);
```

Gestione degli errori

Descrizione

- La maggioranza delle funzioni di sistema restituisce 0 in caso di successo, ed un valore diverso da zero in caso di errore
  - A parte per lo 0, non esistono convenzioni circa il valore da restituire in caso di errore
  - Consultare sempre le man page

- `errno`
  - Variabile speciale che memorizza informazioni in caso di errori
    - A fronte di un fallimento il sistema imposta in questa variabile un valore che descrive il tipo di errore
    - Tutte le chiamate di sistema usano la stessa variabile
    - Copiare subito il valore in un'altra variabile

Gestione degli errori

Utilizzo

- Codici di errore sono valori interi
- Valori possibili definiti da macro
  - Per convenzione iniziano tutte con "`E`"
  - Comode per leggibilità
  - Includere `<errno.h>`
- `char *strerror(int errnum);`
  - restituisce una stringa di descrizione dell'errore
  - Includere `<string.h>`
- `void perror(const char *s);`
  - stampa il messaggio direttamente su `stderr`
  - Includere `<stdio.h>`

Librerie

Descrizione

- Solitamente tutti i programmi devono avere un link a più librerie
  - Se un programma utilizza la `printf()` deve avere un link alla libreria C
- Il linking viene effettuato quando il linker passa dal file oggetto al file eseguibile risolvendo tutti i simboli esterni
- In ognuno di questi casi si deve decidere se effettuare un link *statico* o *dinamico* delle librerie
  - Link statico: Tutte le librerie necessarie al programma vengono copiate all'interno del file eseguibile
  - Link dinamico: Tutte le librerie necessarie al programma rimangono in file separati ed il linker si occupa di generare un link tra il programma e le librerie utilizzate

Librerie

Confronto

- Le librerie dinamiche
  - Permettono di risparmiare spazio su disco
  - In GNU/Linux risiedono in `/lib` o in `/usr/lib`
  - Quando vengono aggiornate, vengono aggiornati tutti i programmi che le utilizzano rendendo il processo di aggiornamento del sistema più snello

- Le librerie statiche
  - Sono più sicure per un insieme di programmi mission-critical. Infatti, il contro di una libreria dinamica è che l'aggiornamento ad una libreria con bug provoca un malfunzionamento su tutti i programmi.
  - Sono preferibili quando non si hanno i privilegi di installare librerie dinamiche in `/lib` o `/usr/lib`

Librerie

Librerie statiche

- Consistono in una semplice collezione di file oggetto in un singolo file
- Quando il linker trova un archivio sulla riga di comando
  - Cerca nell'archivio tutte le definizioni di simboli (funzioni o variabili) che sono referenziati dai file oggetto in elaborazione ma non ancora definiti
  - I file oggetto che definiscono questi simboli sono estratti dall'archivio ed inclusi nel file eseguibile finale
- Per creare una libreria statica (archivio) si utilizza il comando `ar`
  - `ar cr libtest.a test1.o test2.o`
  - la flag `cr` dice ad `ar` di creare un archivio
- Per effettuare il link ad una libreria statica
  - `gcc -o app app.o -L. -ltest`

Librerie

Librerie dinamiche

- Al pari delle librerie statiche raggruppano file oggetto

- Ci sono però molte importanti differenze
  - Nel file eseguibile di un programma con un link ad una libreria dinamica non vi è presente codice della libreria dinamica
  - Il file eseguibile contiene semplicemente un riferimento alla libreria dinamica
  - Una libreria dinamica non è solo una collezione di file oggetto ma è un file oggetto unico
  - Un programma con un link ad una libreria dinamica include tutto il codice della libreria quando viene lanciato

Librerie

Librerie dinamiche

- Per creare una libreria dinamica è necessario compilare i file oggetto con l'opzione `-fPIC`
  - `gcc -c -fPIC test1.c`
  - `-fPIC` indica al compilatore di creare un file oggetto indipendente dalla posizione del codice
- Per combinare tutti i file oggetto in una libreria dinamica sono necessarie le opzioni `-shared` e `-fPIC`
  - `gcc -shared -fPIC -o libtest.so test1.o test2.o`
  - `-shared` indica al linker di produrre una libreria dinamica anzichè un file eseguibile
- Per effettuare il link di una libreria dinamica si procede allo stesso modo che per la libreria statica
  - `gcc -o app app.o -L. -ltest`