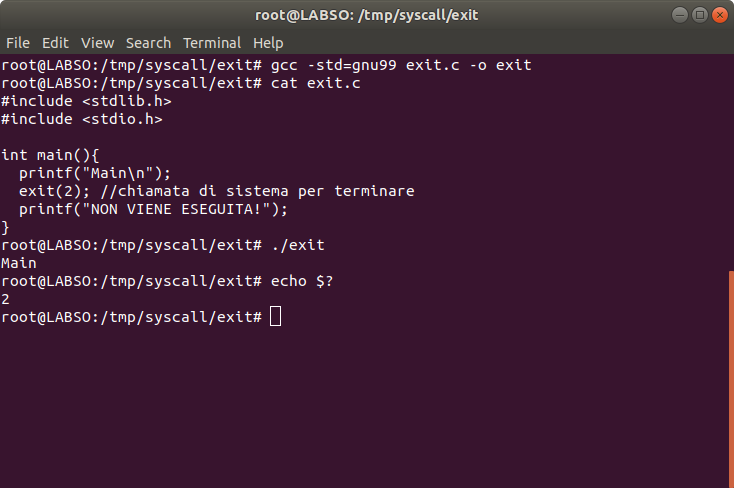
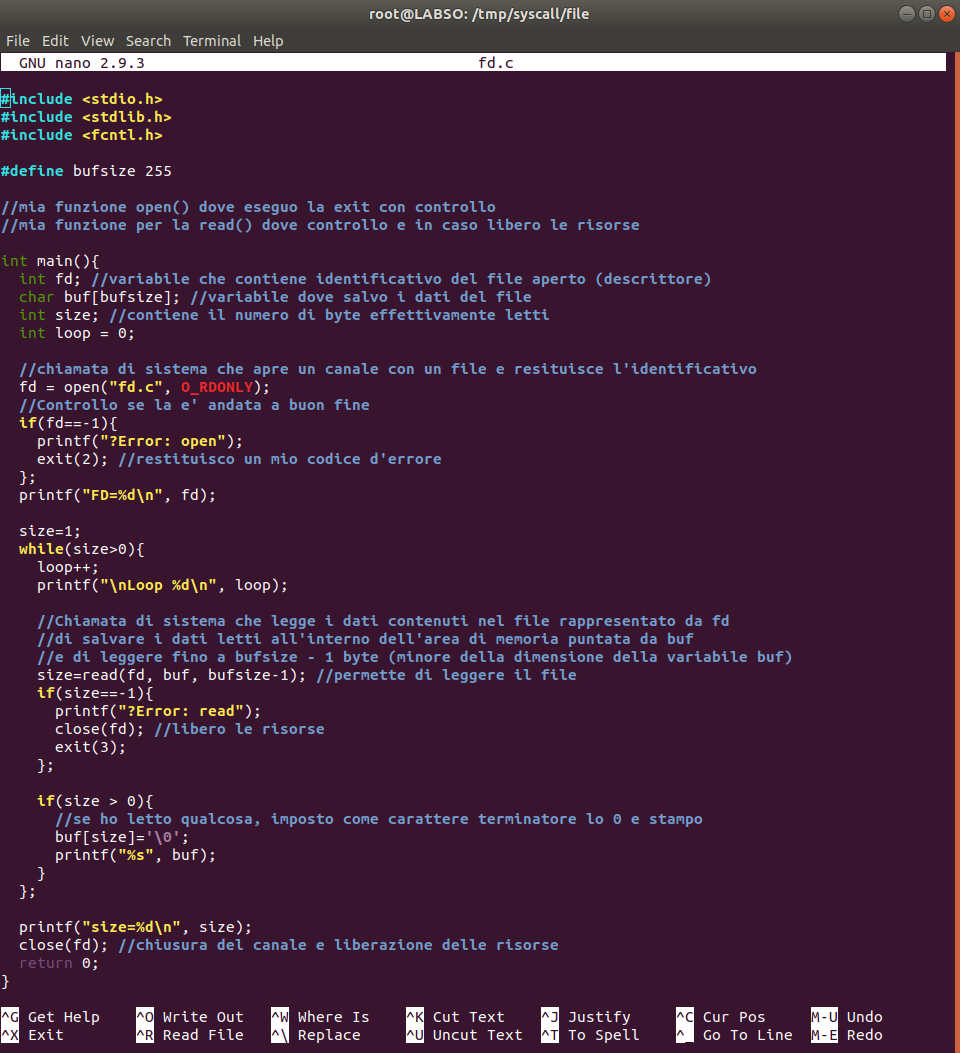
**Laboratorio Sistemi Operativi: System Calls**

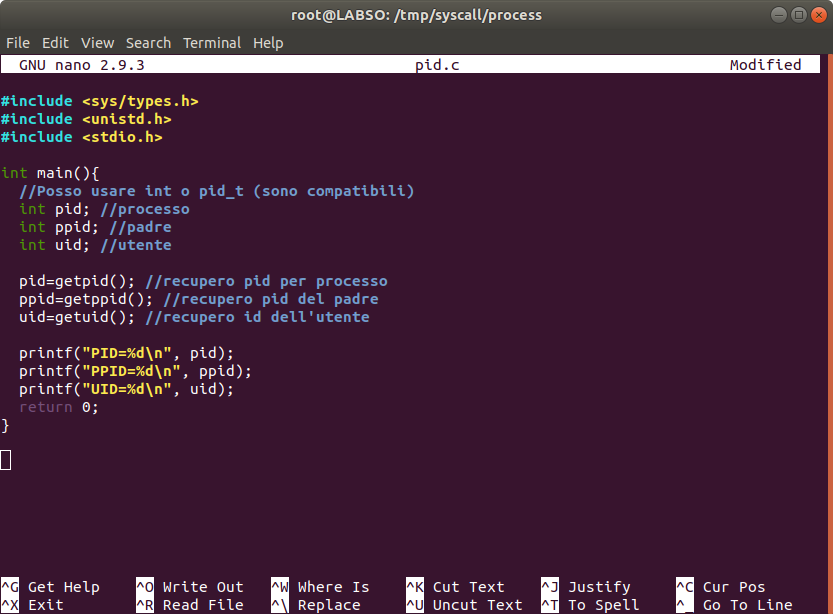
* **Sistema e Architettura**:
  + L’architettura in cui andiamo ad operare in generale è costituita da un **Sistema Operativo** (strutturato a strati) dove è presente uno **strato fondamentale** chiamato **Kernel**.
  + Il **Kernel** è il **nucleo operativo** che contiene una **serie di funzionalità** **di base** per gestire:
    - CPU
    - Memoria
    - Dispositivi
    - Accesso alle risorse a basso livello (hardware).
  + All’avvio del sistema, il **Kernel predispone tutto l’ambiente operativo** e **dà l’avvio** al sistema stesso, **eseguendo un primo processo di base** chiamato **init()**, processo che **rimane attivo per tutto il ciclo** d’esecuzione del sistema stesso (gestendolo).
  + Fa iniziare una serie di processi che a loro volta eseguiranno altri processi (effetto a cascata).
    - Init fa da root process a tutti i processi che vengono generati (**init** ha PID=1).
* **Kernel**:
  + Insieme di **funzioni di basso livello** che accedono all’hardware.
  + I **processi Kernel** sono **eseguiti** in **modalità privilegiata**, hanno un **accesso completo all’hardware** sottostante (tutti gli **altri processi** sono **eseguiti in modalità protetta**, una **modalità limitata**).
* **User Programs**:
  + Per accedere ai dispositivi, i **processi utente chiedono al Kernel l’accesso** (viene usato come intermediario).
  + Il Kernel **controlla**, **accetta** ed **esegue** la richiesta.
* **Spaces Enviroments**:
  + Tutti i programmi sono eseguiti all’interno di uno spazio di lavoro:
    - **Kernel Space**:
      * **Ambiente in cui vengono eseguiti i processi del Kernel**, quando le sue funzionalità vengono richiamate.
    - **User Space**:
      * **Ambiente dove si trovano ad operare i singoli programmi** che vengono eseguiti (programmi che non fanno parte del kernel).
      * Ogni programma utente vede sé stesso come se fosse l’unico programma in esecuzione non curandosi della gestione delle risorse, della CPU, ecc. (sono gestiti da altri processi).
      * I **processi utente possono scambiarsi dati** e **interagire con altri processi** (usando delle chiamate di sistema specifiche che fanno affidamento al kernel).
* **System Call**:
  + Il **Kernel** è composto da un **insieme di funzionalità** per **accedere alle risorse del sistema** e **offre agli altri processi un insieme di funzioni** (definite in librerie standard) per accedere a queste funzionalità.
  + Queste **funzioni** (disponibili per tutti) sono chiamate **System Calls** e possono avere argomenti (definiti prima della chiamata).
  + Sono **eseguite nel Kernel Space** (con privilegi alti) e una volta eseguite, i **dati vengono restituiti al chiamante** (nel User Space).
  + Quando un processo utente esegue una **syscall**, **notifica al Kernel dove salvare i dati generati**, così dopo può recuperarli.
  + Le System Calls hanno sempre un codice d’uscita (valore intero):
    - **0**: **successo**
    - **1-255**: **errore** (o significati particolari)
    - **-1**: **fallimento**
      * La **chiamata non è stata eseguita** (problemi con permessi, se la risorsa esiste o no, ecc) fa riferimento a singole chiamate a basso livello che non sono state eseguite.
      * Bisogna sempre verificare che il codice di ritorno non sia -1 (è comodo crearsi la propria funzione che esegue la Syscall e controlla che sia andato tutto bene).
      * Oltre a restituire il codice d’uscita -1 di fallimento, le **chiamate di sistema che falliscono settano anche la variabile errno** (variabile globale), specificando l’errore che si è verificato.
* **Categorie di Syscall**:
  + **Processi e controllo memoria**: richiesta di memoria, info su processi, ecc.
  + **Accesso al file system**: funzionalità che ci permettono di accedere a file o di crearli, leggerli, ecc.
  + **Gestione dispostivi**: accessi a dispositivi particolari e funzionalità relative ai driver.
  + **Informazioni** (generiche).
  + **IPC**: consentono la comunicazione tra processi, creano canali dedicati per la comunicazione.
* **Syscalls**:
  + **Exit**:
    - **void exit(int status):**
      * Chiamata di sistema a cui posso passare una variabile intera (status) che **viene usata per uscire da un programma** (termina il processo).
      * Tutti i **descrittori vengono chiusi automaticamente** (**rilascia** in modo automatico tutte le **risorse** e file).
      * **L’argomento d’uscita è restituito al chiamante** (0 in caso di successo o 1-255 per un errore).



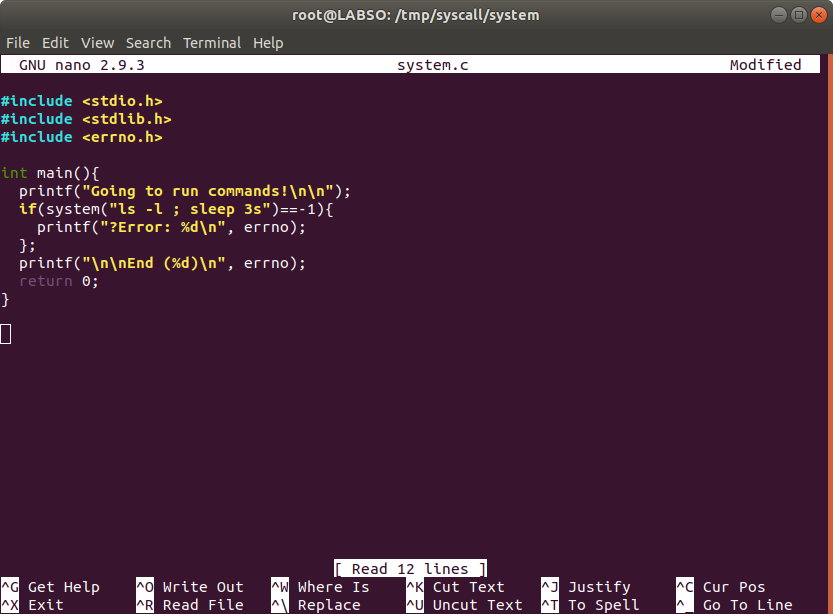
* + **Desctiptors**:
    - I descrittori sono degli indici che ci permettono di accedere al file system e di leggere/scrivere su un file:
      * **int open(…)**
      * **int read(…)**
      * **int write(…)**
      * **int close(…)**
    - Ci sono diverse modalità d’accesso ai file e i permessi a loro associati.
  + **Permissions**:
    - **int chmod(…)**: modifica i permessi del file (lettura/scrittura/esecuzione)
    - **int chwn(…)**: modifica l’autore del file
    - Se abbiamo i permessi di farlo, possiamo modificare i permessi associati ad un file.
  + Esempio:



* + **Processes**:
    - Forniscono indicazioni ed informazioni riguardo ai processi.
    - Funzioni:
      * **pit\_t getpid()**: identificativo del processo.
      * **pit\_t getppid()**: identificativo del padre del processo.
      * **int getuid()**: identificativo dell’utente.



* + **External call**:
    - **int system(char \*cmd)**:
      * Il processo che la esegue **crea un nuovo processo figlio**.
      * Questo **processo figlio esegue**, **con la shell di sistema**, **il comando che viene passato** come parametro alla funzione.
      * Una volta **eseguito il comando**, il **processo figlio termina** e il **processo padre** **continua la sua esecuzione**.



**Esercizio**:

