Esercitazione 3 - 28-30/04/2021

Esercizio 1

Rivedere l'oggetto utilizzato nelle due esercitazioni precedenti per tenere traccia di allocazioni e de-allocazioni

1. Quale problema presenta questa implementazione? E' possibile tracciare in modo distinto gli oggetti tracked divisi per tipo?

```
class Tracker {
  static u_int  count;
public:
  Tracker(){count++;}
  ~Tracker(){count--};
};

u_int Tracker::count = 0;

class Tracked1: Tracker {
….
}
class Tracked2: Tracker {
….com
}
```

2. Implementare un tracker in grado di tracciare in modo distinto classi arbitrarie. Suggerimento: una classe template base che usi il pattern CRTP, a compile time per ogni specializzazione genera un tipo differente e quindi si hanno diversi attributi "count" static, uno per nuovo tipo generato, e non solo uno condiviso fra tutti.

Esercizio 2

Implementare una versione semplificata di shared_ptr (senza gestire i weak ptr) che permetta di tenere traccia delle referenze a puntatori nativi e de-allocarli automaticamente quando le referenze sono zero.
Ecco una possible interfaccia. TipoGenerico *counter è il riferimento ad un oggetto che deve essere condiviso tra gli shared pointer che incapsulano lo stesso puntatore e mantiene in numero di referenze.

```
template <class T>
class my_shared {
   T *ref;
   TipoGenerico* counter;
public:
   my_shared(T *p);
```

```
    my_shared(const my_shared<T> &sp);
    ~my_shared();
    T* operator->();
    T& operator*();
    my_shared<T> &operator=(const my_shared<T> &other);
    my_shared<T> &operator=(my_shared<T> &&other);
    uint use_count();
}
```

Testare l'implementazione allocando più copie degli oggetti all'interno di un vector e verificare allocazioni / de-allocazioni con la classe implementata nell'esercizio 1.

Esercizio 3

Linux espone informazioni sui processi attivi sotto la cartella **/proc** leggibile come un insieme di directory e file regolari.
Per una dettagliata descrizione delle informazioni rintracciabili fare riferimento a questo link:

https://man7.org/linux/man-pages/man5/proc.5.html

Tutte le informazioni relative a ciascuno processo sono leggibili in una cartella separata, che ha lo stesso nome del process id

/proc/12345 contiene le informazioni del processo 12345

Analizziamo alcune informazioni interessanti:

- /proc/nnn/status è un file che contiene informazioni generali sul processo, nella forma "chiave: valore". Esempi
  - Name: come programma
  - State: S (sleeping) (stato)
  - Pid: process id
  - PPid: process id del padre
  - VmSize: dimensione della memoria virtuale
  - VmRSS: memoria residente allocata
  - Threads: numero di thread
- /proc/nnn/stat contiene gran parte delle misure di status più alcune info sul tempo di esecuzione del processo
- /proc/nnn/fd: è una directory che contiene un link simbolico a tutti i file aperti dal processo; il nome del link è il file descriptor (0,1,2 sono collegati a standard input, output e error)
- /proc/nnn/map_file contiene un link ai file mappati in memoria dal processo (in genere sono le librerie dinamiche condivise), il nome del link è l'indirizzo in memoria su cui la porzione di file è mappata

Per riferimento è fornito uno zip con le info selezionate di un pc linux (se avete linux potete usare la vostra /proc).
Il formato del file è il seguente (per ogni processo):

```
----
/proc/nnn
----
```

/proc/nnn/status
out di cat /proc/nnn/status

/proc/nnn/stat
out di cat/proc/nnn/stat

/proc/nnn/fd
out di ls -la /proc/nnn/fd (la stringa dopo “->” è il file)

/proc/nnn/map_files
out di ls -la /proc/nnn/map_files (la stringa dopo “->” è il file)

Realizzare:

1. una classe che memorizzi:
    a. le informazioni descritte in status
    b. il tempo di esecuzione preso da stat
    c. i nomi dei file aperti
    d. i nomi dei file mappati in memoria
2. Memorizzare questi oggetti in una struttura che permetta l’aggiornamento quando si ri-processa il file (alcuni processi possono essere terminati, altri possono essere stati avviati) senza ricreare da zero la struttura tutte le volte.
3. Costruire l’albero del processi guardando la relazione padre/figlio presente in status (ogni processo può avere n figli e un solo padre, 0 è l’antenato comune di tutti i processi)
   NB: l’albero deve coesistere con la struttura al punto 2 senza replicare le informazioni dei processi, quindi è necessario utilizzare in modo corretto dei puntatori!
4. Implementare le seguenti funzionalità sulla struttura al punto 2:
    a. listare i processi aperti, ordinandoli per memoria allocata o numero di file aperti
    b. listare tutti i file aperti con il numero di processi che li usano, ordinati in ordine alfabetico o per utilizzo
    c. listare tutti i file mappati in memoria con il numero di processi che li usano, ordinati in ordine alfabetico e per uso
    d. listare tutti i processi in un particolare stato
5. implementare le stesse funzionalità al punto 4 sulla struttura albero, permettendo di partire da un pid selezionato dall’utente e mostrando solo le informazioni relative a tutti i suoi figli e discendenti, divise per processo o aggregate