Gestione della memoria in programmi scritti in C

Tipi di allocazione

- La memoria può essere allocata *staticamente* (all'atto del caricamento del programma) se la dimensione dei dati è conosciuta a compile-time, è questo il caso di variabili globali e variabili statiche che persistono per tutta la durata del programma.

- ...oppure *automaticamente* sullo Stack e *dinamicamente* sull'Heap

  stack allocation: quando dichiariamo una variabile/struttura, quando passiamo parametri ad una funzione, i valori di ritorno di una funzione, etc... La deallocazione dello stack è automatica all'uscita dello scope delle variabili o al ritorno delle funzioni.

  heap allocation: servono esplicite chiamate di libreria sia per l'allocazione che per la deallocazione. Le funzioni di libreria si appoggiano a chiamate di sistema quali *brk* ed *mmap*.

  - In C non esiste un sistema di **garbage collection** automatico
  - c'e' il rischio di produrre **memory leak** se la memoria non viene correttamente liberata.

Chiamate di libreria

- Ad ogni processo viene assegnato uno spazio chiamato **heap** per l'allocazione dinamica di strutture dati
- L'allocazione dinamica avviene con le seguenti chiamate di libreria (vedere man 3 malloc):
  - *malloc, calloc, realloc*
  - Altre funzioni **NON C89/C99 STANDARD** sono: posix_memalign, memalign, aligned_alloc, valloc, palloc, alloca, etc....
- La deallocazione avviene con le seguenti chiamate di libreria (vedere man 3 free) :
  - *free*
- Le funzioni di libreria per l'allocazione restituiscono un *puntatore* allo spazio allocato se eseguite con successo oppure *NULL*. *free* non ritorna alcun valore.

Allocazione

- **#include <stdlib.h>**
- **void *malloc(size_t size)**
  - alloca un blocco contiguo di memoria di dimensione 'size' **bytes** restituendo il puntatore all'area allocata. Se il SO non riesce ad allocare memoria ritorna NULL.
- Lo spazio realmente allocato dal SO puo' essere maggiore di 'size' bytes per motivi di allineamento dei dati.
- In Linux la memoria non viene allocata all'atto della chiamata ma al momento del primo accesso (**optimistic memory allocation, first touch policy**)
- **void *calloc(size_t N,size_t size)**
  - alloca un array di 'N' elementi ognuno di dimensione 'size' bytes
  - ogni elemento è inizializzato a zero
  - La memoria viene realmente allocata dal SO all'atto della chiamata
- **size_t sizeof(<datatype>)** è un *operatore* che restituisce la *size in bytes* di un tipo primitivo o composto

Deallocazione

- **void free(void *ptr)**

  libera un segmento di memoria precedentemente allocato con *malloc,calloc,realloc*

- Se free viene invocata su un puntatore non allocato sull'heap o su un puntatore già dellocato, si possono avere errori e comportamenti imprevedibili a run-time

Esempi di utilizzo

- char *p = malloc(10); // alloca 10 bytes
- struct S { long a; char b[10]; }

  struct S *p = malloc(sizeof(struct S)); // alloca 16 bytes su sistemi a 32 bits e 24 bytes su sistemi a 64 bits
- long *p = malloc(10*sizeof(long)); // alloca un vettore di 10 long int (40 o 80 bytes)
- long * p = calloc(10, 20*sizeof(struct S)); // alloca ed inizializza 10*20*sizeof(struct S) bytes
- free(p); // dealloca la memoria puntata da p