TP – Programmation `malloc`, `free`, ...

Christophe Morvan

Université de Marne-la-Vallée

17 novembre 2015

Résumé : gérer la mémoire

Objectif

Créer les fonctions suivantes :

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

Résumé : gérer la mémoire

Objectif

Créer les fonctions suivantes :

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

Pour pouvoir coder `free` il est nécessaire de mémoriser la quantité de mémoire allouée.

Résumé : gérer la mémoire

Objectif

Créer les fonctions suivantes :

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

Pour pouvoir coder `free` il est nécessaire de mémoriser la quantité de mémoire allouée.

Structure pour mémoriser les informations

```
struct meta_info {
    size_t size;
    struct meta_info *next;
    int free;
};
```

Fonctions auxiliaires

Programme exemple

```
struct meta_info *find_free_block(struct meta_info **last,
                                  size_t size){
  // Trouver un bloc libre.
  // ie: parcours des blocs à partir de global_head
}

struct meta_info *request_space(struct meta_info* last,
                                  size_t size){
  // Effectue le déplacement de "breakpoint".
}

struct meta_info *get_block_ptr(void *ptr){
  // L'adresse des infos à partir de l'adresse allouée
}
```

Programme exemple

```
void *malloc(size_t size) {
  struct meta_info *block;
  if (size <= 0) {return NULL;}

  if (!global_head) { // Premier appel
    block = request_space(NULL, size);
    if (!block) {return NULL;}
    global_head = block;
  } else { // Appel ultérieur
    struct meta_info *last = global_head;
    block = find_free_block(&last, size);
    if (!block) { // Pas de bloc libre
      block = request_space(last, size);
      if (!block) { return NULL; } // Echec
    } else { block->free = 0;}// Bloc  libre
  }
  return (void*)(block+1);
}
```

free

Programme exemple

```
struct meta_info *get_block_ptr(void *ptr){
  return ((struct meta_info*)ptr)-1; // Simple
}

void free(void *ptr) {
  if (!ptr) {return;}// pointeur NULL, rien à faire
  struct meta_info* block_ptr = get_block_ptr(ptr);
  block_ptr->free = 1;
}
```

Autres fonctions

```
void *realloc(void *ptr, size_t size);
```

1) Si `ptr` est le pointeur null : appel de malloc ;
2) Si l'espace disponible est suffisant, on renvoie `ptr` ;
3) Sinon, on utilise `memcpy` pour recopier le contenu dans un nouvel espace obtenu par un appel à `malloc` ;
4) On libère l'espace initial par un appel à `free`.

Autres fonctions

`void *realloc(void *ptr, size_t size);`

1) Si `ptr` est le pointeur null : appel de malloc ;
2) Si l'espace disponible est suffisant, on renvoie `ptr` ;
3) Sinon, on utilise `memcpy` pour recopier le contenu dans un nouvel espace obtenu par un appel à `malloc` ;
4) On libère l'espace initial par un appel à `free`.

`void *calloc(size_t nmemb, size_t size);`

1) `malloc` de `nmemb*size` ;
2) Initialisation de l'ensemble avec `memset`.