Université de Perpignan Année 2025
Licence Info 3 C. Negre

TD 2 - Système d'exploitation

Pointeurs et espace mémoire des procéssus

Exercice 1. Dans les codes suivants identifiez le bug présent et proposez une correction:

1. Code 1:

```
int main()
{
  char *nom="";
  printf("Entrez votre nom:\n");
  scanf("%s",nom);
  printf("Bonjour à vous %s\n",nom);
  return(0);
}
```

2. Code 2:

```
int main()
{
  char c;
  char z='z';
  printf("Entrez un caractère:\n");
  scanf("%s",&c);
  printf("c vaut %c et z vaut %c\n",c,z);
  return(0);
}
```

3. Code 3:

```
int main()
{
  char capitale[100];
  printf("Quelle est la capitale du Zimbabwe ?\n");
  scanf("%s",capitale);
  if( capitale == "Harare" )
     printf("Gagné!\n");
  else
     printf("Perdu\n");
  return(0);
}
```

4. Code 4:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void concat(char *s1,char *s2)
{
  char *result;
  int n1,n2,i;

  n2=strlen(s2);
  for(i=0;i<n2;i++)
     s1[n1+i]=s2[i];

  printf("%s\n",s1);
  return;
}

int main()
{
  concat("aaaaaaaaaaa","bbbbbbbbbbb");
  return(0);
}
```

Exercice 2. On considère un tableau de `int` de taille 12.

```
int t[12]={0,1,2,4,8,16,32,64,128,256,512,1024};
```

- En utilisant l'arithmétique des pointeurs et le déréférencement `*` de pointeur afficher la valeur de chaque élément du tableau.
- Même question mais affichant chaque `int` comme quatre `char` non signé.

Exercice 3. 1. Dire ce qu'affiche le programme suivant

```
int main()
 {
   int a=2;
   int b=5;
   int *p,*q;

   p=&a;
   q=&b;

   printf("%d %d\n",*(p+1),(*p)+1);
 }
```

2. Dire ce qu'affiche le programme suivant

```
int main()
 {
   int a=2;
   int b=5;
   int *p,*q;

   p=&a;
```

```
    (p++);
   (*p)++;
   printf("a=%d b=%d\n",a,b);
 }
```

3. Déclarer un tableau de `int` en variable locale du `main` et un second tableau de `int` aloué dynamiquement avec `malloc` et un troisième tableau de `int` déclaré en variable globale tous trois de taille 8. Puis répondez aux questions suivantes:
   - Affichez l'adresse du premier élément des trois tableaux.
   - Déterminer l'écart entre les trois adresses
   - Prédire l'adresse du 5-ieme élément du tableau dans les trois cas. Vérifiez votre réponse avec votre programme.

Exercice 4. On considère le code suivant

```
#include <stdlib.h>
#include <stdio.h>

struct une_structure{
int a;
char b;
};


void main()
{
 char c='c';

 struct une_structure b;
int a[4];

 printf("%p,%p,%p\n",a,&b,&c);

}
```

1. Changez l'ordre des variables et analysez les adresses correspondantes.
2. En utilisant uniquement la variable `a` (i.e. ajouter une instruction du type `a[?]=?;`), faire en sorte d'affecter à `c` la valeur `'a'`

Exercice 5 (Pointeur de type `void *`).

1. Faire une fonction de signature suivante:

   ```
   void aff(void *pt, int size);
   ```

   cette fonction doit afficher octet par octet en hexadécimal l'élément pointé par `pt` qui est de taille `size` octets.

2. Faire une fonction `main` qui appelle la fonction `aff` pour afficher :
   - Un `char`
   - Un `int`
   - Une strucuture `S` de type

     ```
     struct S{
     int a;
     char c;
     };
     ```

Exercice 6. L'objectif est d'analyser la différence entre un tableau à deux dimensions alloué statiquement et dynamiquement.

```
#include <stdio.h>
#include <stdlib.h>
#define N 3

void aff1(int **t)
{
  int i,j;
  for(i=0;i<N;i++)
  {
    for(j=0;j<N;j++)
      printf("%d ",t[i][j]);

    printf("\n");
  }
}

void aff2(int t[N][N])
{
  int i,j;
  for(i=0;i<N;i++)
   {
    for(j=0;j<N;j++)
      printf("%d ",t[i][j]);

    printf("\n");
  }
}

int main()
{
  int i,j;
  int t1[N][N]={{1,2,3},{4,5,6},{7,8,9}};
  int **t2;
  t2=(int **) malloc(N*sizeof(int *));
  for(i=0;i<N;i++)
    t2[i]=(int *) malloc(N*sizeof(int));
  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
       t2[i][j]=t1[i][j];

 aff1(t1);
 aff2(t2);

}
```

1. Corrigez l'erreur de compilation afin d'afficher les deux tableaux.
2. Analysez l'organisation mémoire des deux tableaux, et expliquez l'erreur de compilation de la première question.

Exercice 7. Dans cet exercice, l'objectif est de faire une fonction qui détermine si un tableau `t1` de taille `n1` apparait comme sous tableau d'un autre tableau `t2` de taille `n2`:

- Par exemple le tableau d'entiers

```
t1={3,10,2}
```

est le sous-tableau souligné dans le tableau suivant

t2={0,19,3,10,2,27,19}

- Autre exemple, pour un tableau de `char`

```
t1={'a','a','b'}
```

est le sous-tableau souligné dans le tableau de char suivant

t2={'c','c','a','a','a','b','d'}.

Les tableaux passés en paramètre sont génériques:

```
int is_sub_tab(void *t1, int n1,void *t2, int n2, int size_element)
```

où

- n1 est le nombre d'éléments de t1,
- n2 le nombre d'éléments de t2
- `size_element` est la taille en octet d'un élément des tableaux (c'est un élément de même type mais inconnu).
- la valeur de retour cette fonction vaut 0 si t1 n'est pas sous-tableau de t2 et la valeur de retour vaut 1 si t1 est sous-tableau de t2.

Exercice 8 (Pile d'appel)**.** On considère le code suivant

```
unsigned char *p;

  void f()
{
int a=10;
long b=5;
char c='a';

}

int main( )
{
int a=5;
char u='b';

f();
}
```

- Déterminez des trois variables de la fonction `f` laquelle a l'adresse la plus petite.
- Affectez cette adresse à `p` et avec `p` affichez octet par octet en hexadécimal toutes les variables locales de la fonction `f`.
- Poursuivez l'affichage des octets jusqu'à arriver aux variables locales de la fonction `main`.

Exercice 9. On considère le code suivant qui est incomplet

```c
#include <stdio.h>


int main(void)
{
   void (*foncAffiche)(int);
   int (*foncSaisie)(void);
   int entier;

   // affectation manquante
   // affectation manquante

   entier = foncSaisie();
   foncAffiche(entier);

   return 0;
}
```

Complétez le code en ajoutant:

- le code pour une fonction `int SaisieEntier(void)` qui récupère un entier rentré au clavier et le retourne.
- le code pour une fonction `void AfficheEntier(int)` qui affiche l'entier `n` donné en argument.
- Complétez les affectations manquantes dans le `main` afin que le `main` récupère un entier au clavier et ensuite l'affiche.

Exercice 10 ((Facultatif) mymalloc)**.** L'objectif est d'implanter une version de malloc que l'on appellera mymalloc qui alloue la mémoire comme suit:

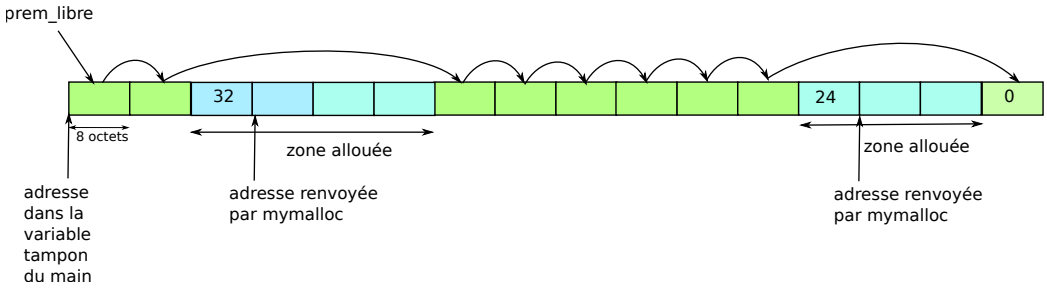
- La mémoire est allouée dans un tampon de $N = 128$ octets défini au début du programme (variable `tampon` dans le main).
- L'allocation se fait par mots de 8 octets. Lors d'un appel à mymalloc pour une allocation de k octets, soit $k' = 8 \times \lceil k/8 \rceil$ le plus petit multiple de 8 supérieur à k. On cherche une zone libre de $k' + 8$ octets consécutifs commençant à l'adresse a. Sur les 8 octets commençant à l'adresse a on écrit la valeur de $k' + 8$ (un unsigned long) et l'adresse renvoyée est $a + 8$.
- Les octets libres sont organisés en une chaîne de mots de 8 octets. Chaque octet contient l'adresse du prochain mot de 8 octets libre. Un pointeur dénommé `prem_libre` défini en variable globale contient l'adresse du premier mot de la chaîne. Le dernier mot libre de la chaîne contient la valeur 0.

Dans la figure 1 vous avez une illustration de cette organisation. En vert ce sont les blocs libres en bleu les blocs alloués. Il y a deux zones allouées.

Vous avez un squelette de code C `mymalloc.c` qui contient une fonction d'initialisation (qui fabrique la chaîne de bloc libre initiale) `void init_libre(void *tampon)` appelée au début du programme. Et les fonctions à compléter suivante:

1. `affiche_octets(void *tampon)`: la fonction affiche la valeur de l'adresse contenue dans `prem_libre`, puis affiche octet par octet en hexadécimal les $N = 124$ octets de la zone mémoire (8 octets par ligne).
   (a) Voici une sortie que j'ai obtenu après avoir fait quelques allocations/désallocations et écritures mémoire. Donnez la chaîne d'octet les libres et les zones mémoire allouées (la zone mémoire commence à l'adresse 0x5627b25292a0).

Figure 1: Organisation mémoire du tas




```
adresse dans prem_libre =0x5627b25292a0
a8 92 52 b2 27 56 00 00
c0 92 52 b2 27 56 00 00
10 00 00 00 00 00 00 00
61 61 61 61 61 61 61 00
c8 92 52 b2 27 56 00 00
d0 92 52 b2 27 56 00 00
f0 92 52 b2 27 56 00 00
18 00 00 00 00 00 00 00
00 00 00 00 01 00 00 00
02 00 00 00 27 56 00 00
f8 92 52 b2 27 56 00 00
00 93 52 b2 27 56 00 00
08 93 52 b2 27 56 00 00
10 93 52 b2 27 56 00 00
18 93 52 b2 27 56 00 00
00 00 00 00 00 00 00 00
```

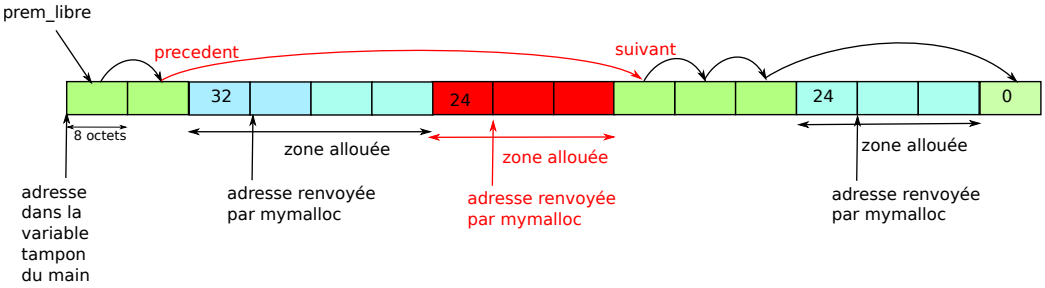
(b) Complétez le code la fonction `affiche`.

2. Faire une fonction `int next_multiple_8(int k)` qui détermine le plus petit multiple de 8 supérieur à k.

3. Faire une fonction `void *trouve_n_contigus(int n)` qui parcourt la chaîne de blocs libres, jusqu'à trouver n blocs libres contigu. Elle renvoie `NULL` si elle n'en trouve pas et l'adresse du début de la zone des n blocs contigus si elle en trouve.

   Indication : Pour savoir si deux mots de la chaîne sont contigus, si `pt` *pointe sur un élément de la chaîne (on suppose textttpt de type* `unsigned long *`*. Il faut s'assurer que *pt (qui contient l'adresse du mot suivant suivant) et bien égale à* `pt+1`*.*

4. Faire la fonction `void *mymalloc(int k)` elle cherche une séquence de mot de 8 octets dans la chaîne de bloc libres contigus pour contenir les `k` octets et supprime ces blocs de la chaine de blocs libres. Par exemple en partant de la situation dans la figure 1 et si on alloue 15 octets (qui sera arrondi à deux mots de 8 octets), on obtient cette nouvelle organisation (la zone allouée est en rouge):



Notez que la fonction devra faire pointer precedent vers suivant.