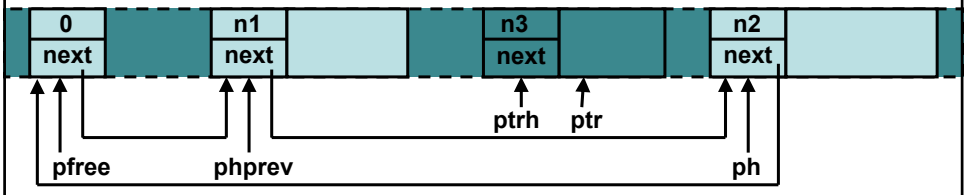
Réalisation d'un allocateur de mémoire

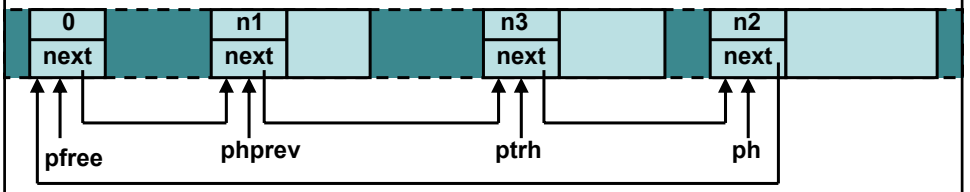
Algorithmes de free et malloc

Algorithme de void myfree(void *ptr)

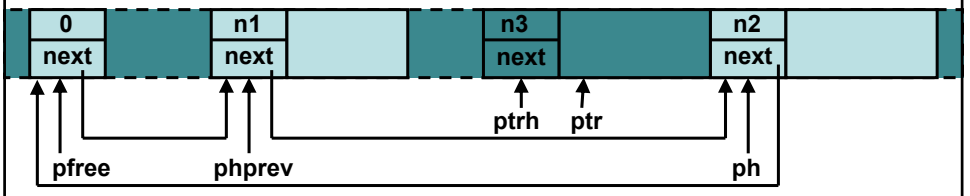
ptrh = (struct Header *) ptr - 1



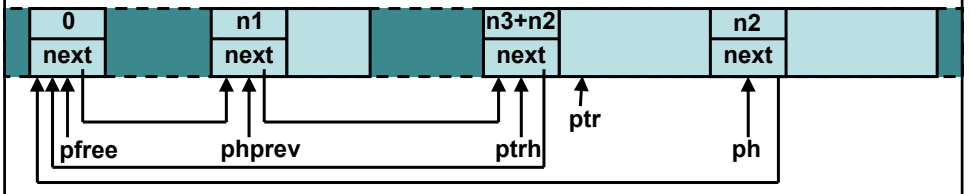
- Insertion dans la liste des blocs libres du bloc pointé par ptrh : phprev < ptrh < ph



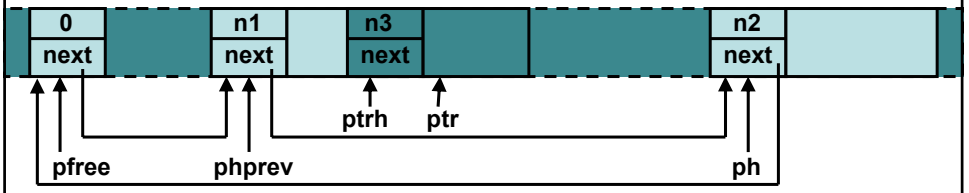
ptrh + n3 = ph



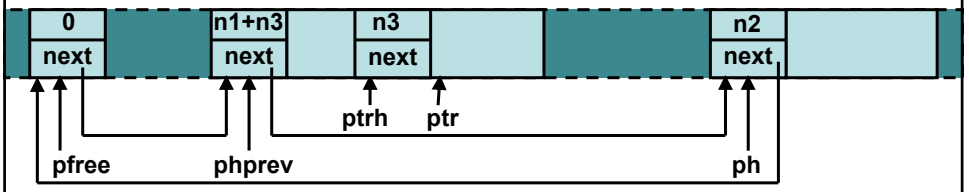
- Fusion de 2 blocs adjacents



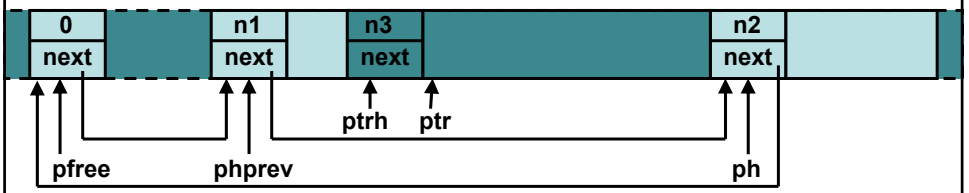
phprev + n1 = ptrh



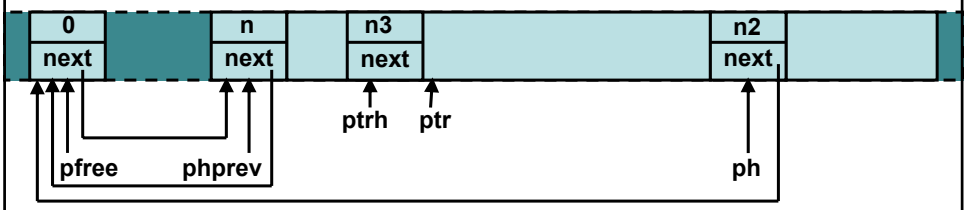
- Fusion de 2 blocs adjacents



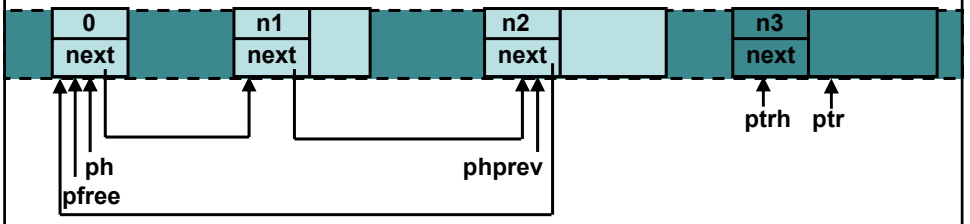
ptrh + n3 = ph et phprev + n1 = ptrh



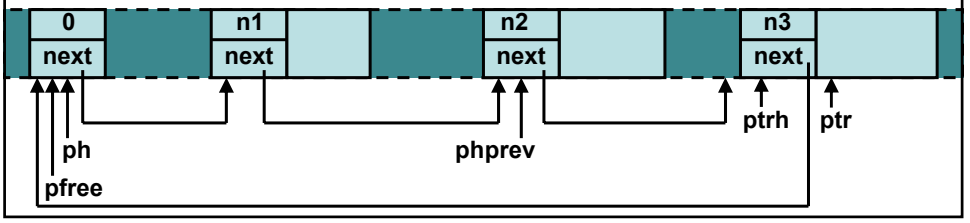
- Fusion de 3 blocs adjacents : n=n1+n2+n3



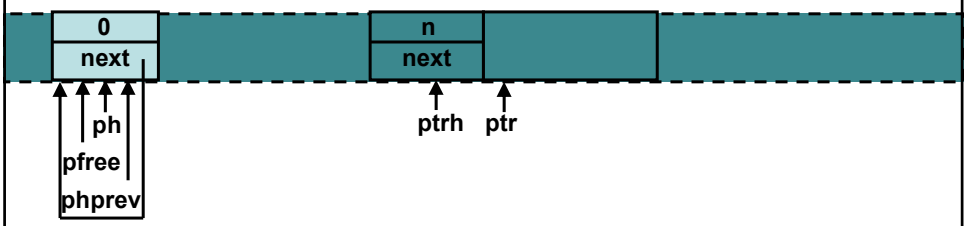
ph == pfree



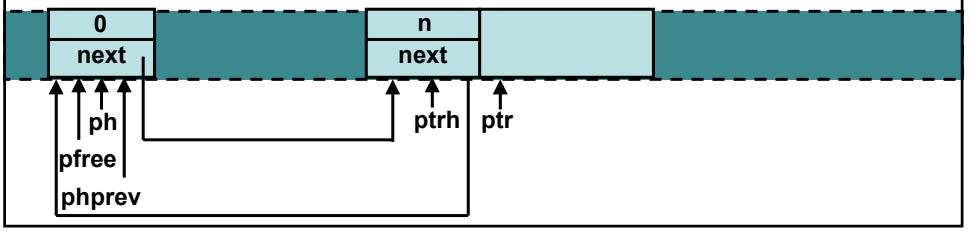
- Ajout du bloc pointé par ptrh à la fin la liste des blocs libres



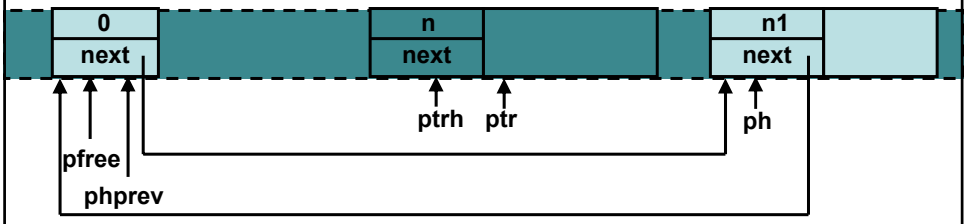
phprev == pfree et ph == pfree



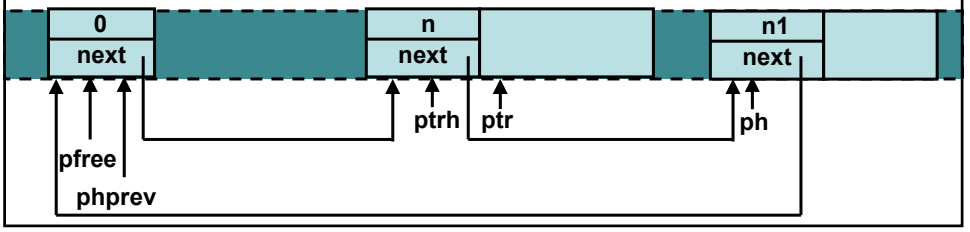
- Le bloc pointé par ptrh est le premier à être ajouté à la liste des blocs libres



phprev == pfree et ph > ptrh



- Ajout du bloc pointé par ptrh en tête de la liste des blocs libres (on est dans le cas général)

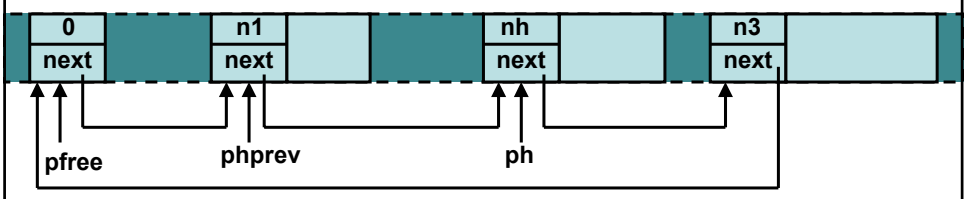


Algorithme de
void *mymalloc(unsigned int sz)

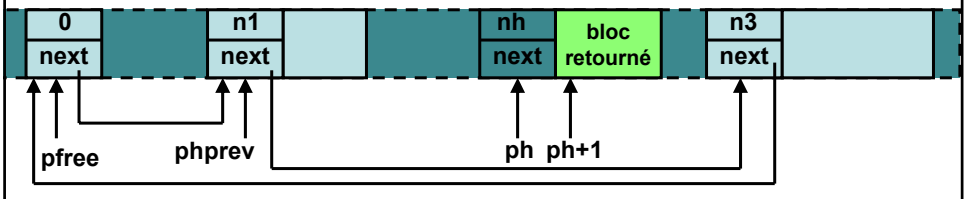
Principe général

- sz = nombre de caractères (d'octets)
- nh = nombre de headers (y compris de gestion)
- HEADER_SZ = sizeof(struct Header)
- nh= (sz-1)/HEADER_SZ +2
- Rechercher le premier bloc dans la liste des blocs libres de taille suffisante: nheaders ≥ nh
- En cas d'échec, appeler sbrk et ajouter un nouveau bloc libre à la liste

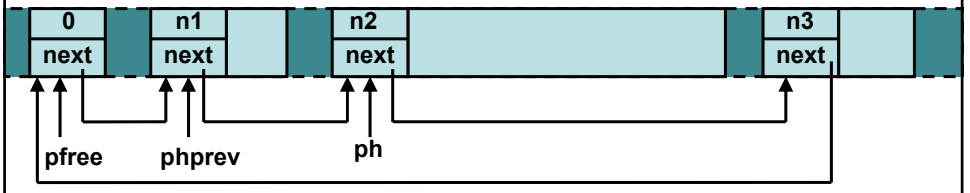
ph->nheaders == nh



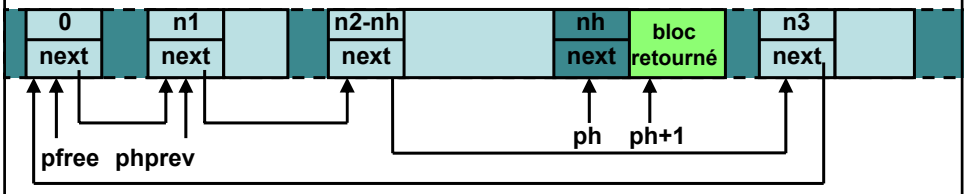
- Suppression du bloc pointé par ph dans la liste des blocs libres



ph->nheaders > nh

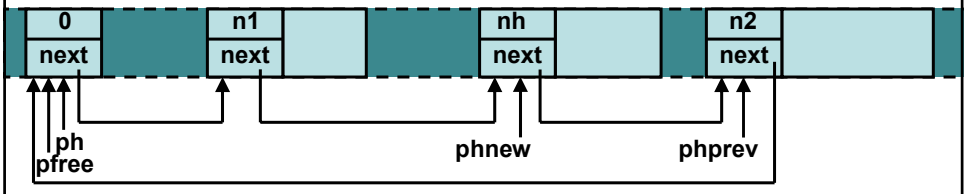


- Mise à jour du bloc pointé par ph: ph->nheaders=n2-nh
- Mise à jour de ph: ph et ph->nheaders



ph == pfree

- Insertion dans la liste des blocs libres de *phnew
  phnew=(struct Header *)sbrk(nh*HEADER_SZ)



Parcours de la liste des blocs libres

- Initialisation:

  phprev = pfree

  ph = pfree -> next

- Incrémentation:

  phprev = ph

  ph = ph -> next