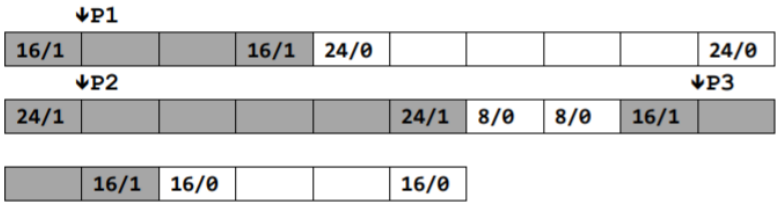
ICS EXE 12 Solution

May 27, 2020

1 Memory Allocation

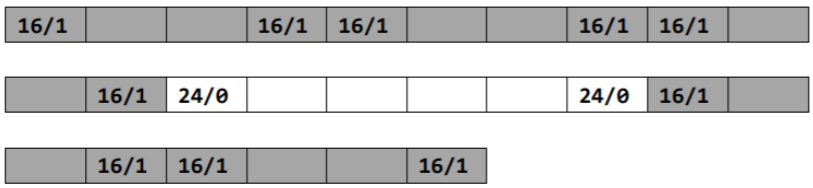
Now we organize the heap as a sequence of **contiguous** allocated and free blocks, as shown below. **Allocated** blocks are shaded, and **free** blocks are blank (each block represents 1 word = 4 bytes). **Headers** and **footers** are labeled with the number of bytes and allocated bit. The allocator maintains **double-word** alignment. You are given the execution sequence of memory allocation operations (`malloc()` or `free()`) from 1 to 6.




```
1. P4 = malloc(8)
2. free(P1)
3. P5 = malloc(6)
4. free(P2)
5. P6 = malloc(3)
6. P7 = malloc(2)
```

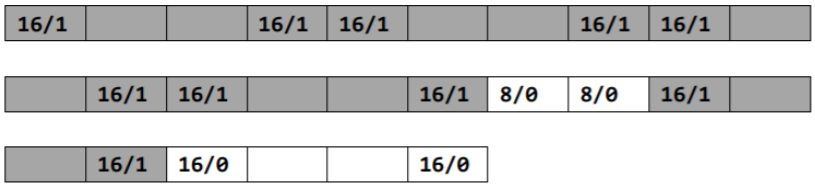
Please answer the questions below. Assume that **immediate coalescing** strategy and **splitting free blocks** are employed. (NOTE: **DON'T** need consider P1, P2 and P3 when calculating **internal** fragments)

1. Assume **best-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed. Please also calculate the total bytes of the **internal** fragments.
   ANS:

   Internal fragments: 45 bytes.

2. Assume **first-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed. Please also calculate the total bytes of the **internal** fragments.
ANS:



Internal fragments: 45 bytes.

3. Suppose that we use the entire free block instead of plitting when doing allocation and coalesce immediately after freezing. The total size of heap we can use is shown in the picture. Please identify that whether the operation sequence will succeed or not for both **first-fit** and **best-fit** respectively. If yes, please give the total bytes of the **internal** fragments. fragments. If no, please point out **which step** causes the failure (out of memory).
ANS:
For first-fit: Yes. Internal fragments: 69 bytes
For best-fit: No. Failure in step 6, P7 = malloc(2)

2 Dynamic Linking

ICSTA wrote two C programs as shown below: `subvec.c` and `dynamic_line.c`. We compile `subvec.c` as a shared library (`linux > gcc -shared -fpic -o libvector.so subvec.c`):

```c
/* subvec.c */
int delcnt;
void subvec(int *x, int *y, int *z, int n) {
        for (int i = 0; i < n; i++) {
                z[i] = x[i] - y[i];
        }
}
```

```c
/* dynamic_link.c */
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

int x[2] = {1,2};
```

```
7  int y[2] = {3,4};
8  int z[2];
9  int main(void) {
10         /* we can call subvec() like */
11         /* any other function */
12         subvec(x,y,z,2);
13         printf("z = [%d %d]", z[0], z[1]);
14
15         return 0;
16 }
```

1. Please give two ways that we can link the shared libraries `libvector.so` (NOTE: You can modify the `dynamic_link.c`)
   ANS:
   We can link it when the application is loaded. gcc -o prog dynamic_link.c ./libvector.so

   We can link it when the application executes through modifying the main function.

```
1  int main() {
2    void handle;
3    void (*subvec)(int*, int*, int*, int);
4    handle = dlopen("./libvector.so",
5           RTLD_LAZY);
6    subvec = dlsym(handle, "subvec");
7    subvec(x, y, z, 2);
8    printf("z = [%d %d]", z[0], z[1]);
9    dlclose(handle);
10 }
```

2. Why GOT is needed to relocate functions in `libvector.so`?
   ANS:
   Because with GOT, the code segment is position-independent(PIC). So a single copy of a shared module's code segment can be shared by an unlimited number of processes.

3. After the shared libraries **libvector.so** was loaded in memory, please fill in the address of **GOT entry of subvec** before and after first invocation. Suppose that the address of PLT[0] is 0x404360, the address of PLT entry of subvec is 0x404560, and the address of subvec() is 0x400128, the value of GOT[0] is 0x406670.
   ANS:

   Before: 0x404566
   After: 0x400128