Process Memory Map Assignments

1. What are the various segments of memory created for a process?

Sol:

1. **Text Segment**: Stores the executable code.
2. **Data Segment**: Stores initialized global and static variables.
3. **BSS Segment**: Stores uninitialized global and static variables.
4. **Heap**: Stores dynamically allocated memory.
5. **Stack**: Stores function calls and local variables.
6. **Memory-Mapped Segment**: Stores files or devices mapped into memory.
7. **Shared Libraries Segment**: Stores shared libraries loaded by the process.
8. **Guard Pages**: Protects against stack overflow.
9. **Environment Variables**: Stores environment variables.

Each segment has its own purpose and properties, and the operating system uses these segments to manage memory and ensure the proper functioning of the process.

2. Refer the code and identify which variable goes in which segment? (Analyse line by line)

```
#include <stdio.h>
int glob_array[2000];
void func()
{
static int count = 0;
int func_local=10;
char *ptr; /* Assume pointer size is 4 or 8 bytes. Which segment do these belong
to? */
char array[1000];
ptr = malloc(100); /* From which segment, do these 100 bytes come? */
count++;
printf("func called %d times\n", count);
}
int main() /* Which segment do the compiled machine instructions of source code go
into? */
{
int main_local;
func();
return 0;
}
```

Sol:

Global Variable glob_array[2000]:

- Segment: Data Segment
- Reason: This is a globally declared variable, so it will be stored in the Data Segment. The Data Segment contains global and static variables that are initialized before the program starts.

Static Variable count (in func):

- Segment: Data Segment (BSS)
- Reason: count is a static variable, which means it retains its value across function calls. It will be stored in the Data Segment, specifically in the BSS Segment (since it is initialized to 0 and does not have an explicit initial value).

Local Variable func_local (in func):

- Segment: Stack Segment
- Reason: func_local is a regular local variable that is allocated on the Stack. The Stack stores local variables and keeps track of function calls and their return addresses.

Pointer Variable ptr (in func):

- Segment: Stack Segment
- Reason: ptr is a local pointer variable, which will be stored in the Stack. The pointer itself (i.e., the memory address it holds) will reside in the Stack.

Local Array array[1000] (in func):

- Segment: Stack Segment
- Reason: This is a local array. It will be allocated on the Stack because it is local to the func function.

Dynamically Allocated Memory (malloc(100)):

- Segment: Heap Segment
- Reason: malloc(100) allocates 100 bytes of memory from the Heap. The Heap is used for dynamic memory allocation, which persists until explicitly freed (using free()).

Function func() (in main):

- Segment: Text Segment (Code Segment)
- Reason: The compiled machine instructions for the func function (and the rest of the program) will be stored in the Text Segment (also known as the Code Segment). This segment contains the compiled code of the program, which is typically read-only to prevent modification during execution.

Local Variable main_local (in main):

- Segment: Stack Segment
- Reason: main_local is a local variable in main(), so it will be allocated in the Stack.

3. How many functions would be in stack when func() is being executed

Sol:

At the time func() is called from main(), the Stack will contain the following:

- main() function: The main() function is still in the Stack because it has not yet returned. Its state and local variables are stored in the stack.
- func() function: The func() function call is added to the Stack when it is invoked, along with its local variables (like func_local, array, ptr).

So, there are two functions in the Stack: main() and func().

4. Create an executable of above code, run size command and view the size.

The size command reports the size of various sections of an executable, such as:

- **Text Segment**: The size of the compiled code (the executable instructions).
- **Data Segment**: The size of initialized global variables.
- **BSS Segment**: The size of uninitialized global variables and static variables.
- **Heap**: Dynamic memory allocation (which is only visible when the program is running, so the size won't be displayed directly in the output of the size command).
- **Stack**: The stack size will depend on the system and how deep the function calls go