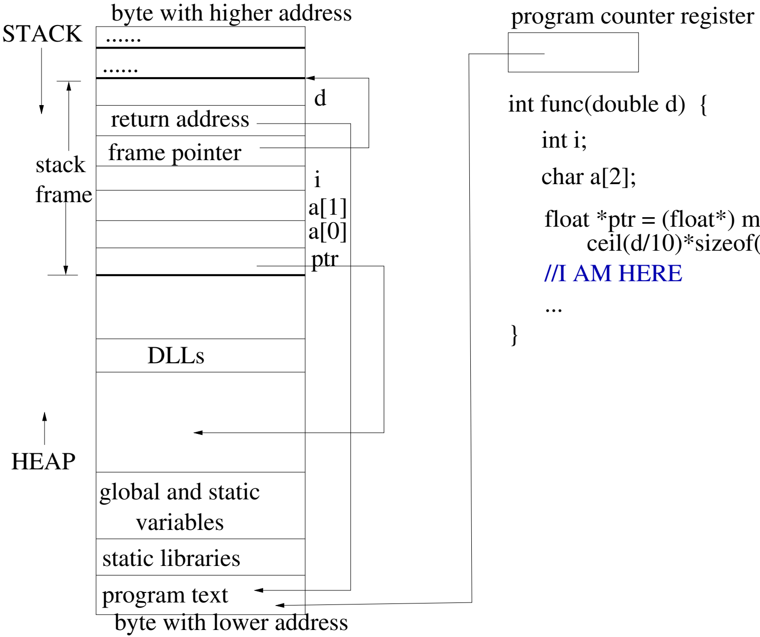
Organization of process memory

R. Inkulu
http://www.iitg.ac.in/rinkulu/



```
int func(double d)  {
    int i;
    char a[2];
    float *ptr = (float*) malloc(
          ceil(d/10)*sizeof(float));
    //I AM HERE
    ...
}
```

* every byte is addressable

Process memory segments/regions

- *text/code segment* and the segment storing static libraries are not mutable once the program is loaded into memory; all the other regions are mutable
- *data segment* comprises of
  - * *bss segment* comprises of global and static variables that are intialized to zero or do not have explicit initialization in source code
  - * explicitly initialized global variables, static variables, mutable constant-lengthed strings, and
  - * the heap[1]
- *stack region* comprises of a sequence of *stack frames*, each correspond to a function on the call stack

[1] dynamic linked libraries (DLLs) are loaded as and when needed i.e., while the program is in execution

More on the stack frame

- suppose main calls func$_1$, func$_1$ calls func$_2$, . . ., func$_{i-1}$ calls func$_i$; and let the func$_i$ is being currently executed: then the stack frames are organized from the top of the stack region, one corresponding to each of main, func$_1$, func$_2$, . . ., func$_i$, respectively

- *return address* of a stack frame points to the address of the instruction that needs to be executed when this function returns

- *frame pointer* of a stack frame points to the beginning address of the current stack frame — useful to remove the stack frame when the function scope ceases

Program counter

- *program counter register* points to the address of the instruction that is being executed

Virtual memory vs physical memory

- RAM is the *physical memory*

  hard disk etc., are said to be *secondary storage* devices

- memory assigned to each process (typically 4GB) is from *virtual memory*, which comprises of pages; each page is a contiguous 4KB block (typically): any page may reside either in physical memory or on the secondary storage

* virtual memory mainly helps in using secondary memory as if it is part of the main memory

* when the physical memory is full, a page is stored on a secondary device; when a page located on secondary device is needed, the operating system copies it to the main memory; however, the changes in addresses due to these moves are hidden from the user

* the space allotted on a secondary device for virtual memory scheme utilization is called the *swap space*

Benefits of having virtual memory

- permits using more memory than what is available in physical memory
- gives each process a private memory space
- hides the programmer from the fragmentation of physical memory
- helps in nicely managing memory shared (*shared memory*) between processes

Page tables

- each process is associated with a *page table*
- each entry in the page table holds a flag indicating whether the corresponding page is in physical memory or not, and the corresponding address
- when a page that is not currently in physical memory is referred, the hardware raises a *page fault* exception, which causes the OS to
  - find a page of memory in physical memory and bring the page located in secondary storage to physical memory, and
  - accordingly update the appropriate page table entry