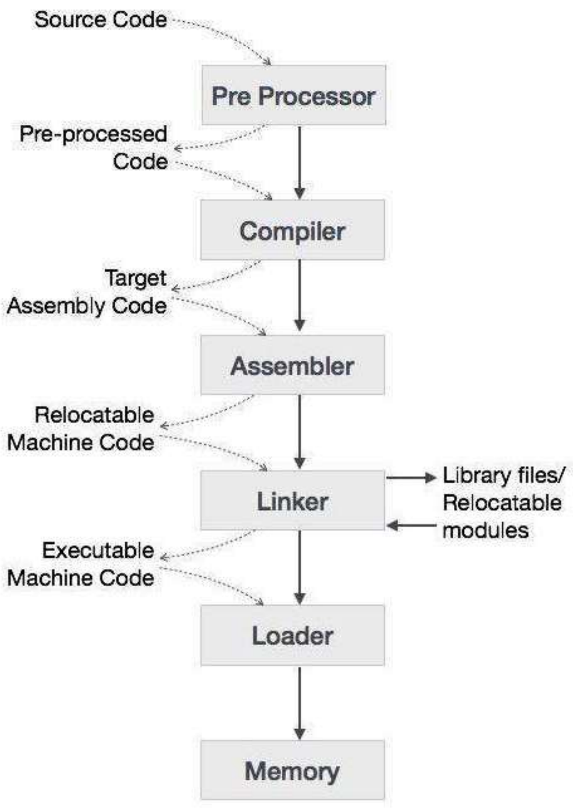
Inter Process Communication

```
int b=50;
int main()
 {
int pid; int a=5;
pid=fork();
if(pid>0)
{
a++;b++;
printf(“Values of  a and b %d %d,a,b”);
}
if(pid==0) {
printf(“Values of a and b from Child");
printf(“Values of  a and b %d %d,a,b”);
}
return 0; }
```

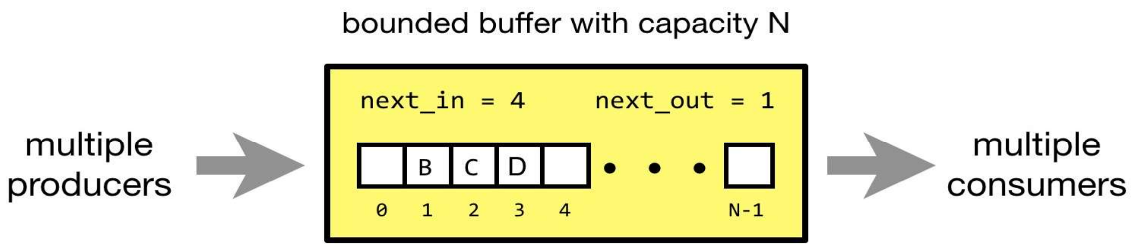
- You will be surprised to note that both local and global variables are actually not shared in a fork setup (amongst parent and child processes I mean
- In fact each process have their own address space and hence even the global variables
- Even manipulation via pointers and malloc does’nt help!
- address of memory returned by malloc is same but in actual they are pointing to or mapped to different physical address
- More in depth discussions will come in Memory Management
- **Essence there is a need for processes to exchange data. This is the justification for IPC!**

- ✓ Linux Processes can be either Independent or Cooperating
- ✓ **Independent** - cannot affect or be affected by other processes executing in the system
- ✓ - does not share data / address space with any other process
- ✓ **Cooperating** – reverse of independent processes.
- ✓ **Need for cooperation –**
- ✓ **Information Sharing** (many interested processes are there) ;
- ✓ **Computation Speedup** (tasks – sub tasks / parallelized execution)
- ✓ **Modularity ; Convenience**
- ✓ 2 models of IPC supported in Linux -
- ✓ **Shared Memory and Message Passing (SHM : Pipes)**
- ✓ Memory region is shared by cooperating processes
- ✓ Processes read / write to the shared regions of memory
- ✓ MPI is better prefered due to cache coherence issues

- ✓ The classical Producer Consumer problem is a case for cooperating processes that needs to exchange data!
- ✓ Below example is a case for P – C problem in CS



- P/C can be in essence viewed as a Shared Memory which is filled by Producer and Emptied by Consumer
- Producer produces 1 item that is consumed by a consumer process
- Issues impacting such setup – Synchronization / lack of it between P & C ; Buffer size – Bounded / Unbounded
- In an Unbounded setup – no limit on no of items that can be produced while consumer will have to wait for new items.
- In Bounded Scenario – Producer waits when full and Consumer waits when empty
- One more good instance (from net!)


bounded buffer with capacity N
multiple producers
next_in = 4
next_out = 1
B
C
D
0
1
2
3
4
N-1
multiple consumers