FILE HANDLING

cat >file1 overwrites the existing file

---

1)writing into a file

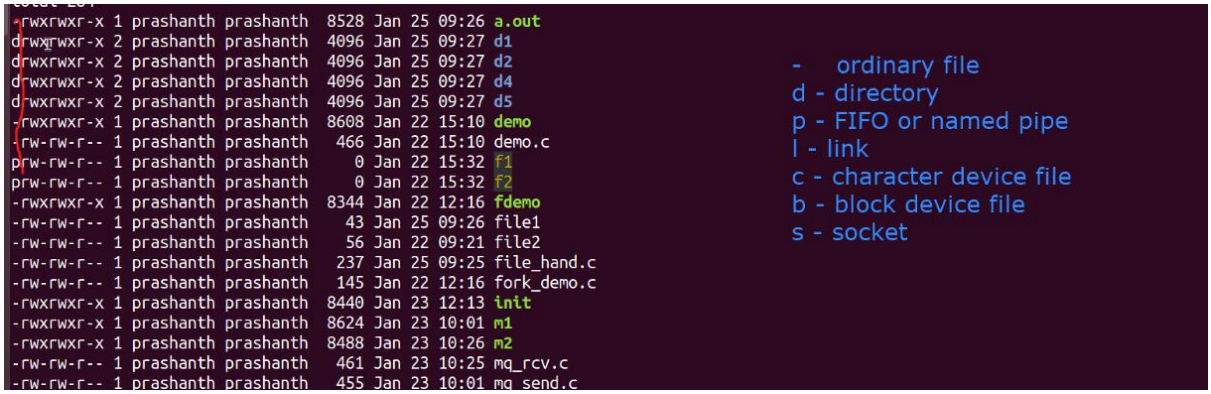
2)read from a file if no. of chars are known

3)read from a file if no. of chars are not known

FILE STATS

ls -l=>long list

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ls -l
total 284
-rwxrwxr-x 1 prashanth prashanth  8528 Jan 25 09:26 a.out
drwxrwxr-x 2 prashanth prashanth  4096 Jan 25 09:27 d1
drwxrwxr-x 2 prashanth prashanth  4096 Jan 25 09:27 d2
drwxrwxr-x 2 prashanth prashanth  4096 Jan 25 09:27 d4
drwxrwxr-x 2 prashanth prashanth  4096 Jan 25 09:27 d5
-rwxrwxr-x 1 prashanth prashanth  8608 Jan 22 15:10 demo
-rw-rw-r-- 1 prashanth prashanth   466 Jan 22 15:10 demo.c
prw-rw-r-- 1 prashanth prashanth     0 Jan 22 15:32 f1
prw-rw-r-- 1 prashanth prashanth     0 Jan 22 15:32 f2
-rwxrwxr-x 1 prashanth prashanth  8344 Jan 22 12:16 fdemo
-rw-rw-r-- 1 prashanth prashanth    43 Jan 25 09:26 file1
-rw-rw-r-- 1 prashanth prashanth    56 Jan 22 09:21 file2
-rw-rw-r-- 1 prashanth prashanth   237 Jan 25 09:25 file_hand.c
-rw-rw-r-- 1 prashanth prashanth   145 Jan 22 12:16 fork_demo.c
-rwxrwxr-x 1 prashanth prashanth  8440 Jan 23 12:13 init
-rwxrwxr-x 1 prashanth prashanth  8624 Jan 23 10:01 m1
-rwxrwxr-x 1 prashanth prashanth  8488 Jan 23 10:26 m2
-rw-rw-r-- 1 prashanth prashanth   461 Jan 23 10:25 mq_rcv.c
-rw-rw-r-- 1 prashanth prashanth   455 Jan 23 10:01 mq_send.c
-rwxrwxr-x 1 prashanth prashanth  8472 Jan 22 10:11 myapp
-rwxrwxr-x 1 prashanth prashanth  8560 Jan 22 15:31 p1
-rwxrwxr-x 1 prashanth prashanth  8560 Jan 22 15:31 p2
-rwxrwxr-x 1 prashanth prashanth  8384 Jan 22 10:04 proc
-rw-rw-r-- 1 prashanth prashanth   301 Jan 22 15:31 proc1.c
```



- =>ordinary file

d =>directory

p=>fifo or named pipe

l=>link file/ shortcuts

c=>character device file

b=>block device file

s=>socket file

Next 9 characters are UGO permisions

Next character =>links or no of ways to access it

Next character=>User name

Next=>Group Name

Next=>File size

Next=>Created time

Next=>Name of file

Note: Every file determined by its unique inode number

ls -i

---

struct stat{

};

Contains all the file info

Note:fstat takes fd as arg

stat takes file name as arg

GDB => DEBUGGING c program

1)gdb -g file1.c -o f

2)gdb ./f

3)note: if we want to use normal cmd to execute inside (gdb)

shell ls

shell clear

4)To view source file from program ...It will show first 10 lines

list 1

list

5)If we want to see other than 10 lines

set listsize 15

list1

list .....

6)run

7)breakpoints can be set on 2 conditions :a)line num b)fn name

break 65

break isPalindrome

info break

delete 2

quit

//exits from gdb mode

break main

info break

run

//halts at main

next

print choice

next

next//skips the fn

step // goes inside the fn

---

watch point vs display point difference

watch only for a variable

It will show the var only when its value changes

display always show value of variable irrespective its value changes or not

---

frame 0 //main

info locals

step

frame 0 //isPalindrome

frame1// main() will become frame1

---

mkfile & INCREMENTAL BUILD

To just create an obj file

gcc -c display.c

//it will create display.o

Need for mkfile ?

Compiles the modified files and tracks them

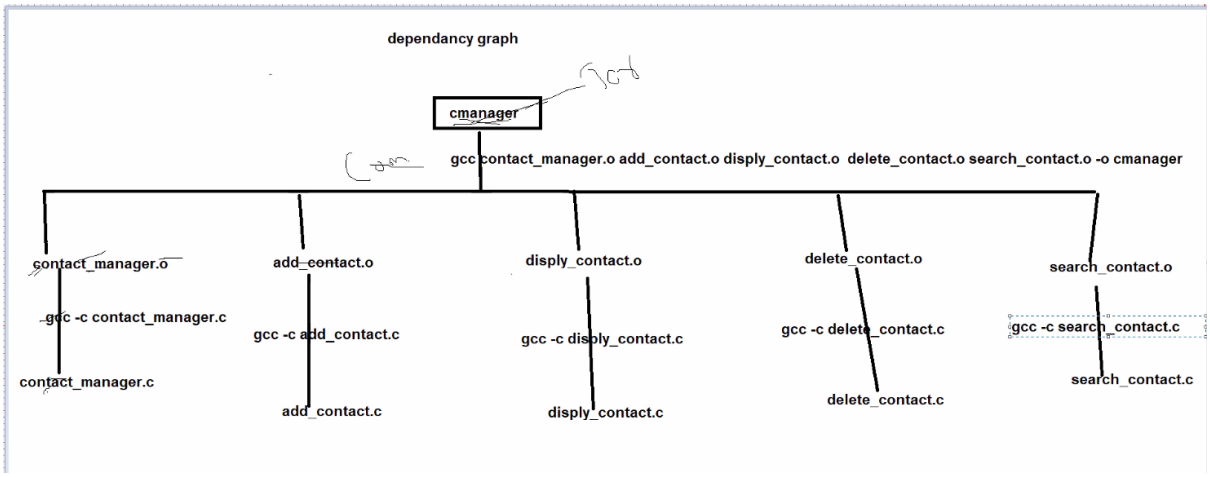
who modified which file

If number of files increase say suppose 100 files , we cannot manually write names in the gcc ….. -c

Command

```
add_contact.c  contact.h  contact_manager.c  delete_contact.c  disply_contact.c  search_contact.c
prashanth@DESKTOP-EEREG1R:~/testproj$ gcc contact_manager.o add_contact.o disply_contact.o search_contac
t.o delete_contact.o -o cmanager
```

DEPENDENCY GRAPH



---

it will have

a)target

b)dependency

c)commands

Dependency graph needs to be added in mkfile

```
cmanager : contact_manager.o add_contact.o disply_contact.o search_contact.o delete_contact.o
	gcc contact_manager.o add_contact.o disply_contact.o search_contact.o delete_contact.o -o cmnager

contact_manager.o : contact_manager.c
	gcc -c contact_manager.c

add_contact.o : add_contact.c
	gcc -c add_contact.c

disply_contact.o : disply_contact.c
	gcc -c disply_contact.c

search_contact.o : search_contact.c
	gcc -c search_contact.c

delete_contact.o : delete_contact.c
	gcc -c delete_contact.c
```

```
add_contact.c  contact.h  contact_manager.c  delete_contact.c  disply_contact.c  makefile  search_contact.c
prashanth@DESKTOP-EEREG1R:~/testproj$ vi makefile
prashanth@DESKTOP-EEREG1R:~/testproj$ vi makefile
prashanth@DESKTOP-EEREG1R:~/testproj$ ls
add_contact.c  contact.h  contact_manager.c  delete_contact.c  disply_contact.c  makefile  search_contact.c
prashanth@DESKTOP-EEREG1R:~/testproj$ vi makefile
prashanth@DESKTOP-EEREG1R:~/testproj$ make
gcc -c contact_manager.c
gcc -c add_contact.c
gcc -c disply_contact.c
gcc -c search_contact.c
gcc -c delete_contact.c
gcc contact_manager.o add_contact.o disply_contact.o search_contact.o delete_contact.o -o cmnager
prashanth@DESKTOP-EEREG1R:~/testproj$ ls
add_contact.c  cmnager    contact_manager.c  delete_contact.c  disply_contact.c  makefile          search_contact.o
add_contact.o  contact.h  contact_manager.o  delete_contact.o  disply_contact.o  search_contact.c
prashanth@DESKTOP-EEREG1R:~/testproj$ ./cmanager
-bash: ./cmanager: No such file or directory
prashanth@DESKTOP-EEREG1R:~/testproj$ ./cmnager
1 :Add Contact
2 :Display Contact
3 :Search contact
4: delete contact
0 :Exit
1
Adding a new contact to phonebook
Enter name of the Contact to add :: fghf
Enter phone number :: 345
Enter Email-ID :: 345
Enter alternate phone number :: rg
Contact added successfully to phonebook
1 :Add Contact
2 :Display Contact
3 :Search contact
4: delete contact
```

Note:if target is not present , and dependency is present

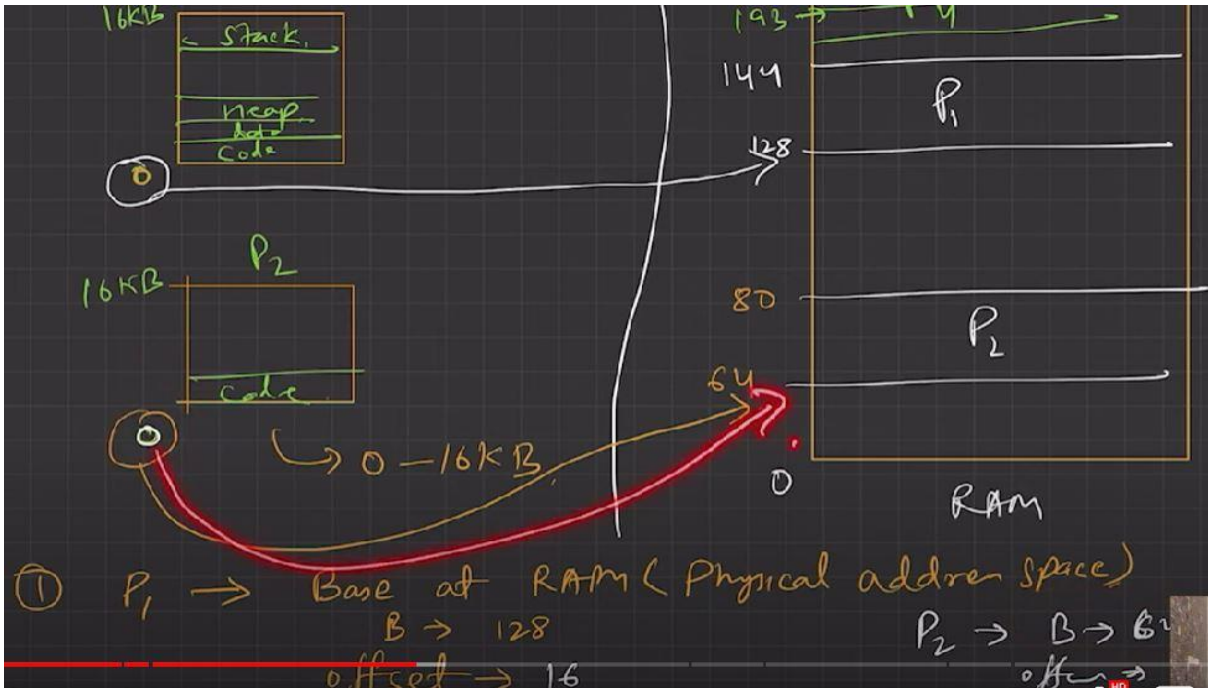
simply execute

---

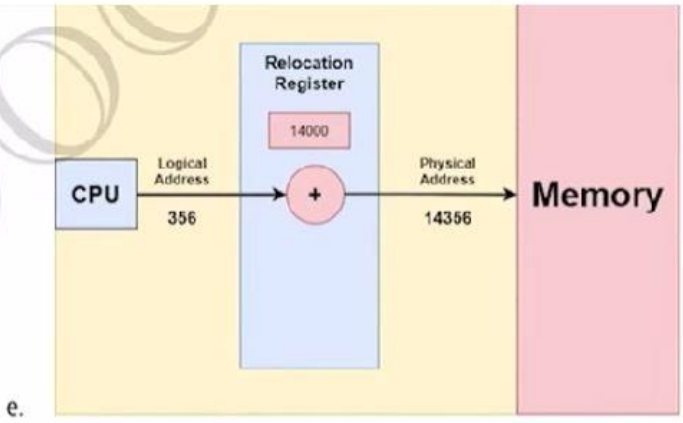
top half interrupts (MORE PRIORITY) vs bottom half interrupts

ISR

MMU

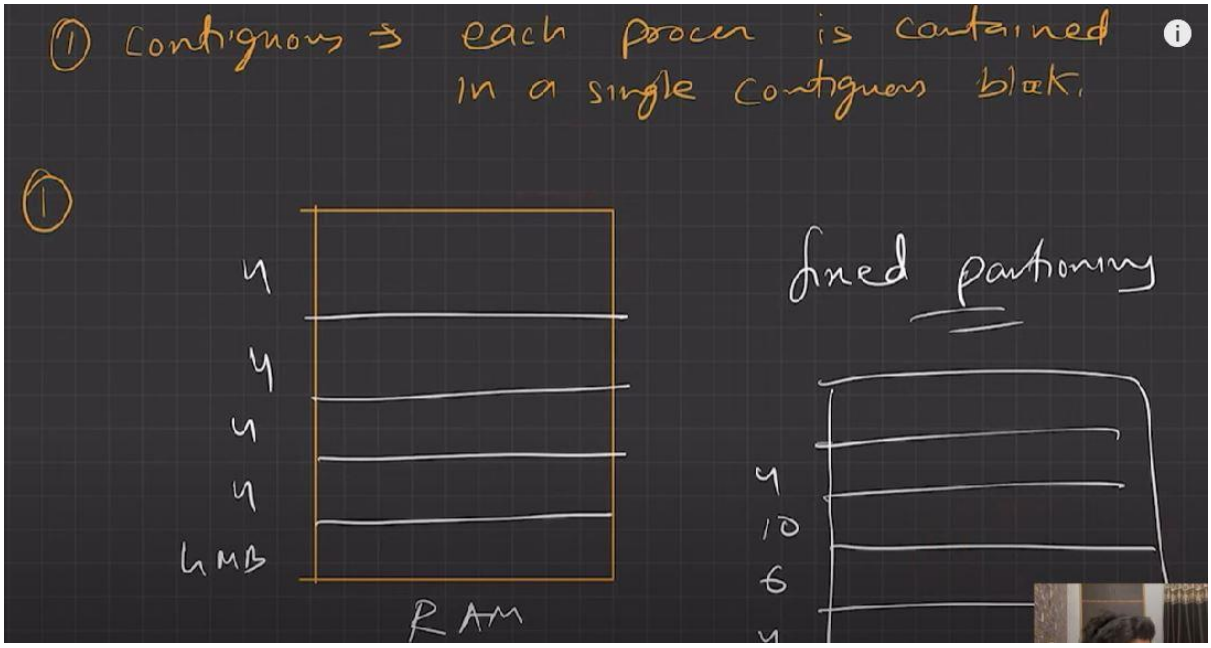


1. In Multi-programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to the users.
2. To realize this increase in performance, however, we must keep several processes in the memory; that is, we must **share** the main memory. As a result, we must **manage** main memory for all the different processes.
3. **Logical versus Physical Address Space**
    a. Logical Address
        i. An address generated by the **CPU**.
        ii. The logical address is basically the address of an instruction or data used by a process.
        iii. User can access logical address of the process.
        iv. User has indirect access to the physical address through logical address.
        v. Logical address does not exist physically. Hence, aka, **Virtual address.**
        vi. The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
        vii. **Range: 0 to *max*.**
    b. Physical Address
        i. An address loaded into the memory-address register of the physical memory.
        ii. User can never access the physical address of the Program.
        iii. The physical address is in the memory unit. It's a location in the main memory physically.
        iv. A physical address can be accessed by a user indirectly but not directly.
        v. The set of all physical addresses corresponding to the Logical addresses is commonly known as Physical Address Space.
        vi. It is computed by the **Memory Management Unit (MMU).**
        vii. **Range: (R + 0) to (R + max), for a base value R.**

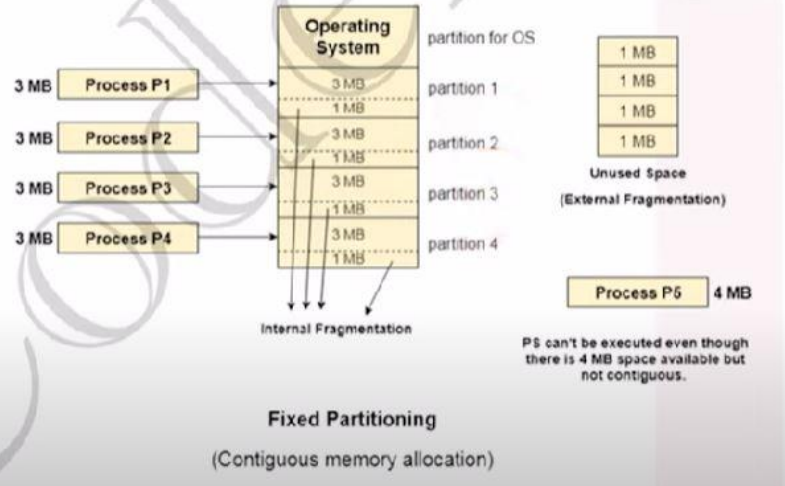


e.

4. How OS manages the isolation and protect? (**Memory Mapping and Protection**)
    a. OS provides this Virtual Address Space (VAS) concept.
    b. To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
    c. The relocation register contains value of smallest physical address (Base address [R]); the limit register contains the range of logical addresses (e.g., relocation = 100040 & limit = 74600).
    d. Each logical address must be less than the limit register.



6. **Contiguous Memory Allocation**
   a. In this scheme, each process is contained in a single contiguous block of memory.
   b. **Fixed Partitioning**
      i. The main memory is divided into partitions of equal or different sizes.



Fixed Partitioning

(Contiguous memory allocation)

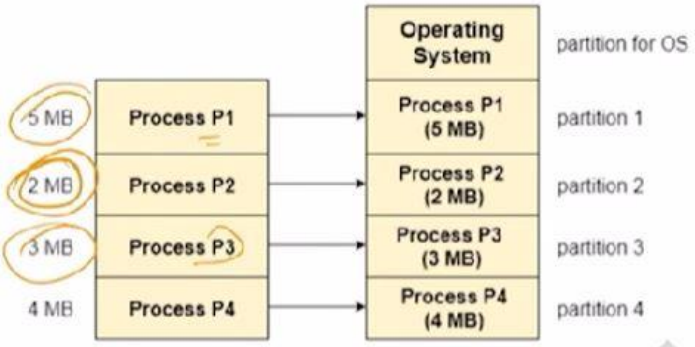
      ii.
      iii. Limitations:

iii. Limitations:

1. **Internal Fragmentation**: if the size of the process is lesser then the total size of the partition then some size of the partition gets wasted and remain unused. This is wastage of the memory and called internal fragmentation.
2. **External Fragmentation:** The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.
3. Limitation on process size: If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

c. **Dynamic Partitioning**

i. In this technique, the partition size is not declared initially. It is declared at the time of process loading.



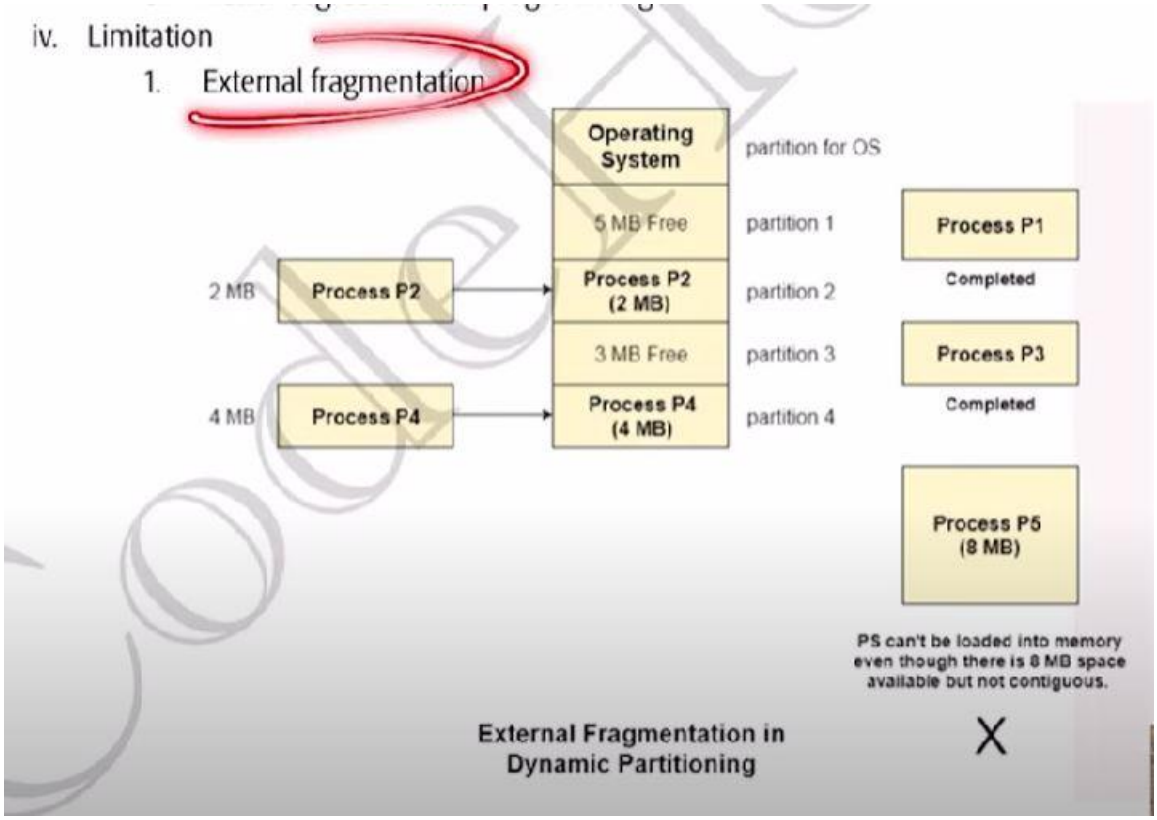
Dynamic Partitioning

(Process Size = Partition Size)

ii.

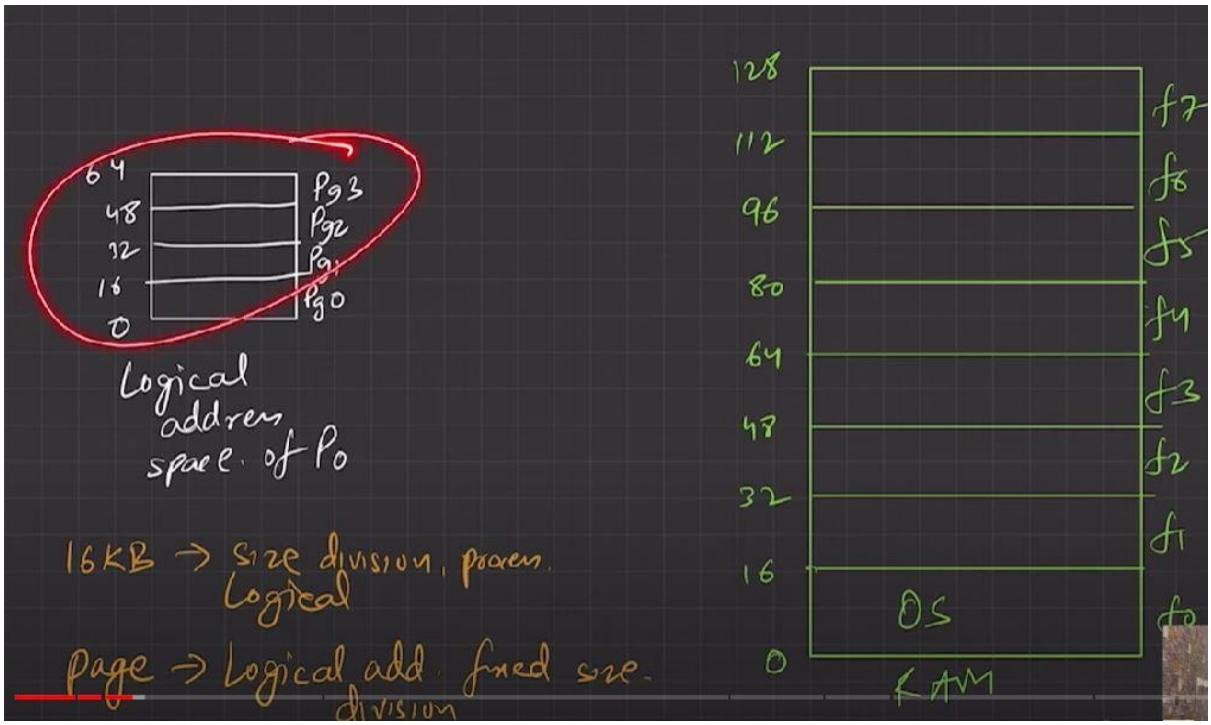
iii. Advantages over fixed partitioning

1. No internal fragmentation
2. No limit on size of process
3. Better degree of multi-programming



To avoid external fragmentation , Compaction is used .

PAGING

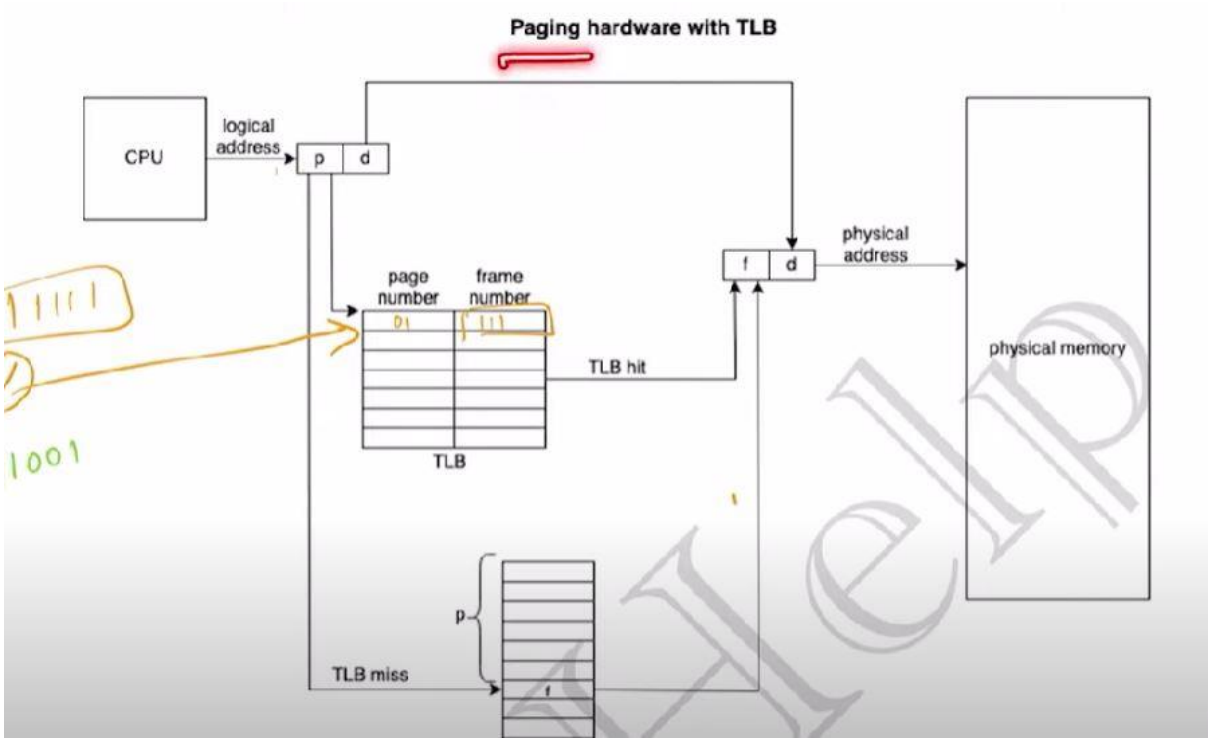


* Page table

Logical page no.	physical frame no.
Pg 0	frame 3
1	7
2	5
3	2

Changing page tables requires only this one register, at the time of context-switching.

4. **How Paging avoids external fragmentation?**
   a. Non-contiguous allocation of the pages of the process is allowed in the random free frames of the physical memory.
5. **Why paging is slow and how do we make it fast?**
   a. There are too many memory references to access the desired location in physical memory.
6. **Translation Look-aside buffer (TLB)**
   a. A Hardware support to speed-up paging process.
   b. It's a hardware cache, high speed memory.
   c. TBL has key and value.



g. **Address space identifier (ASIDs)** is stored in each entry of TLB. ASID uniquely identifies each process and is used to **provide address space protection and allow to TLB to contain entries for several different processes.** When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match, the attempt is treated as TLB miss.