a[i] = rand() %.100 ; betn (0 to 99)

a[i] = rand () %.10 ; betn (0to9)

a[i] = rand () %. 10 +10 ; betn (0 to 19)

① WAp to generate 5 random number im betn -50 to + 50 .

⇒ X = (rand () %. 101) - 50

0 to 100
-50 to +50
-50 to 50

50
+50
100
+ 1
101

② wap to generate 5 Random number in between -50 to 0..

⇒ X = -rand () %. 51

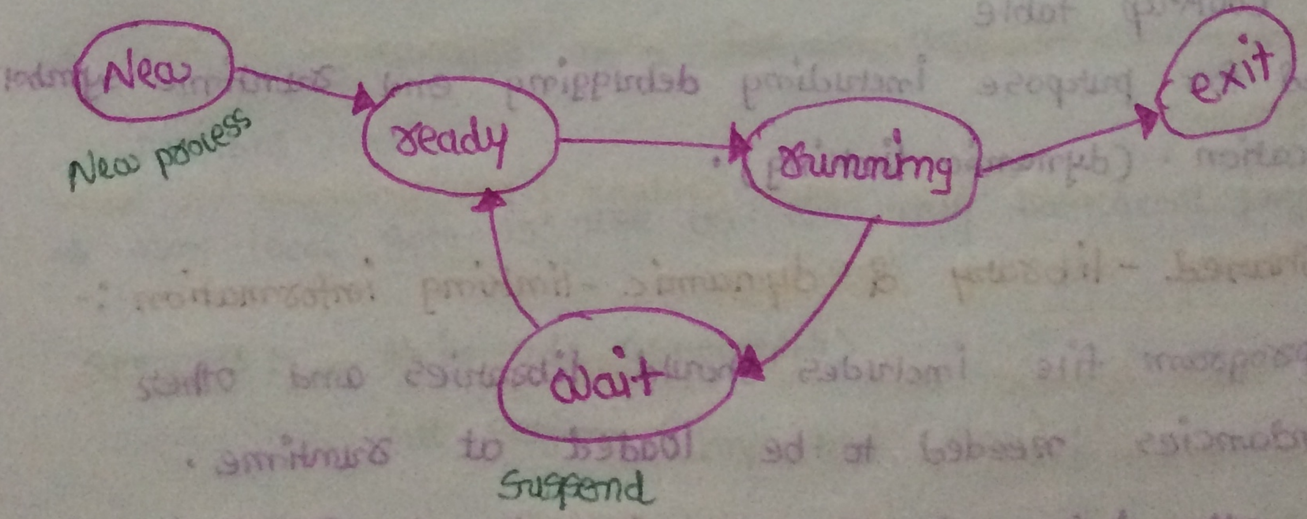
printf ("%d \n", x);

③ wap to generate 5 Random number in between +0.25 to 0.75.

0 to 50
25 to 75

X = ((rand () %. 76) * 0.01) + 0.25) ;

Printf ("%f \n", x);



1. Program :-

→ A program is a file that contains information that describes how to construct a process of runtime.

Information :-

① Binary formate identification.

Contains meta info → formate of file

Prior UNIX implementation → a.out formate COFF

Present UNIX implemention → a.out formate ELF

COFF → Common object File Formate.

ELF → Executable and Linking Formate..

② Machine Language instructions :-

→ These encode the algorithm of the program.

③ program entry-point address :-

Location of the instruction at which execution of program should start.

④ Data :-

contains Values used to initialize Variables and also constant used by the program.

⑤ Symbol and relocation table :-

Lookup table

used for purpose including debugging and runtime symbol relocation. (dynamic linking).

⑥ Shared-library & dynamic-linking information :-

→ program file includes should libraries and other dependancies needed to be loaded at runtime.

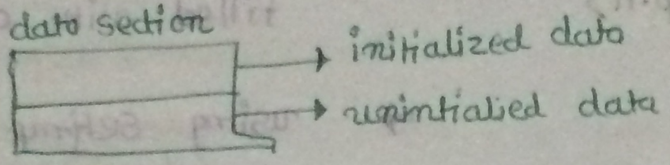
→ So, other information that describes how to construct a process from a program.

In short,

Process is an abstract entity, defined by Kernel to which system resource are allocated in order to execute a program.

Process Control Block :-

1) Program code
2) Variable used by that code



3) PID
4) Virtual memory table
5) Signal delivery and handling
6) process Vesource usage and limits ; Paid information.

⇒ If a child process becomes orphaned becaruse its "birth" Parent terminate then it is adopted by init process.

⇒ Now getpid() returns process id as 1 as init is Parent Now.

Non Local goto :-

1) Setjmp ( ) ≅ setting lable
2) longjmp ( ) ≅ calling lable

⇒ Local goto is use as a forward and backward jmp possible.

⇒ non-local goto is use as an only backward jmp.

Synopsis :-
```
#include <setjmp.h>
int setjmp (jmp_buf env);
int sigsetjmp (sigjmp_buf env, int savesigs);
```

Synopsis :- #include <setjmp.h>

Void longjmp (jmp_buf env, int val);

Void siglongjmp (sigjmp_buf env, int val);

Ⓐ Program :-

```
# include <stdio.h>
void print (Void);
#include <setjmp.h>
jmp_buf V;
main()
{
    Printf ("Hello----\n");
    setjmp (V);
    Printf ("Hai ----\n");
    print();
}
Void print (Void)
{
    Printf ("In print---\n");
    longjmp (V,1);
    Printf (" in print after long jmp\n");
}
```

→ stack hold fun return address so 8MB stack is filled with address then

→ using setjmp and longjmp stack is fully filled

Output :- not segmentation fault.

Hello
Hai---
In print
Hai----
In print
Hai----
Inprint

→ setjmp() is creat a stack upto longjmp().

→ longjmp() roll the stack upto setjmp().

→ setjmp() and longjmp() useful for dealing with errors and interrupt encounted in a low-level subroutine of a program.

→ setjmp() saves the stack context / environment in env for later use by longjmp().

→ The stack context will be invalidated if the function which called setjmp return.

→ setjmp() return a 0.

→ jmp_buf is internally as a one structure and same member consits in structure.

→ program Counter hold the next instruction.

→ Stack pointer hold the top of the stack address.

→ if setjmp() executed first time it returns 0.

→ if setjmp() is executed because of long jmp() → setjmp() return longjmp() second parameter.

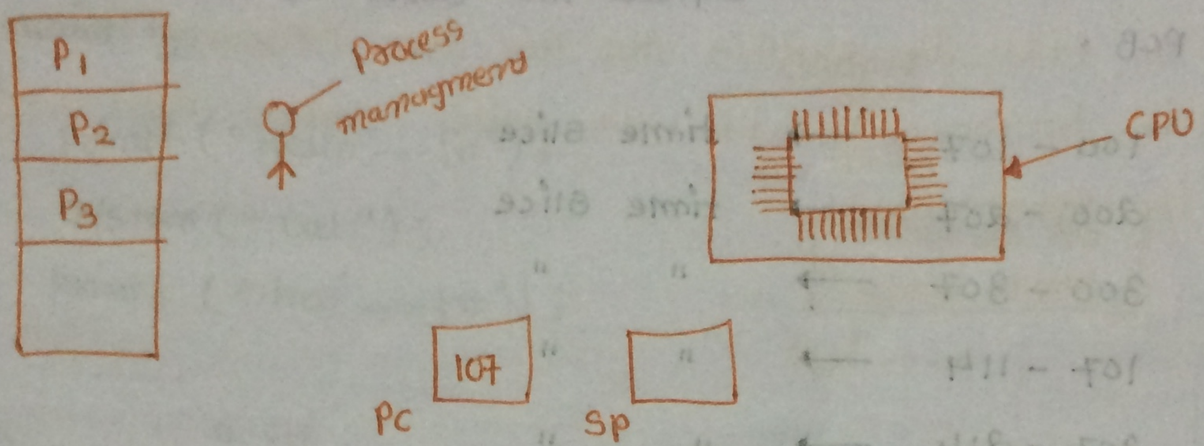
(*) Program :-

```
#include <stdio.h>
#include <setjmp.h>
int sum (int, int) ;
int sub (int, int) ;
int mul (int, int) ;
int div (int, int) ;
main()
{
    int n1, n2, n3, op, r ;
    Printf ("Enter the numbers-----\n");
    Scanf ("%d %d", &n1, &n2) ;
    Printf ("Enter the op \n 1) Sum \n 2) Sub \n 3) mul \n
                4) div \n ") ;
    Scanf ("%d", &op) ;
Printf (" Hello World \n");
 r = setjmp(v);
 if (r==3)
 printf (" Error in multiplication \n");
 elseif (r==4)
     printf (" Error in division \n") ;
```

```
if (op==1)
{
    n3 = sum (n1, n2);
    Printf ("n3 = %d\n", n3);
}
else if (op==2)
{
    n3 = sub (n1, n2);
    Printf ("n3 = %d\n", n3);
}
else if (op==3)
{
    n3 = mul (n1, n2);
    Printf ("n3 = %d\n", n3);
}
else if (op==4)
{
    n3 = div (n1, n2);
    Printf ("n3 = %d\n", n3);
}
else
    Printf ("unknown option\n");
    longjmp (v,1);
}

int sum (int i, int j)
{
    return (i+j);
}

int sub (int i, int j)
{
    return (i-j);
}

int mul (int i, int j)
{
    if (i==0 || j==0)
    longjmp (v,3);
    return (i*j);
}
```

```
int div (int i, int j)
{
    if (j == 0)
    longjmp (v, 4);
    return (i/j);
}
```

⇒ RAM does not capacity to executed the instruction because of RAM is temporary memory.



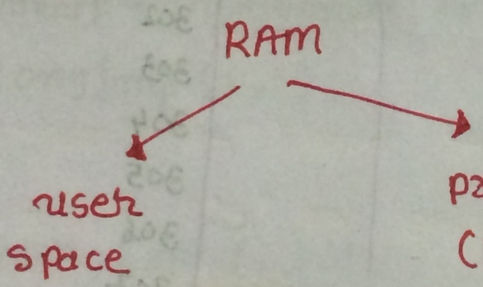
⇒ The information about the running process is loaded or saved into respected PCB and the information about next process which to be executed is loaded into CPU. This is called context switch.

⇒ The PC hold 107 and expired the spicing time the above the 107 data copy in recently PCB.

⇒ The PC context the 200 address, the 207 hold the PC the spicing time expired the above the 207 data copy in PCB.

100 - 107 ⟶ time slice
200 - 207 ⟶ time slice
300 - 307 ⟶ " "
107 - 114 ⟶ " "
207 - 214 ⟶ " "
307 - 314 ⟶ " "

until the process are over



System function :-

System → Execute a shell command.

synopsis :- #include <stdlib.h>

int system (const char *command)

⇒ system() execute a command specified in command by calling /bin/sh -c command and return after the command has been completed.