```
sigaction (2, &V, 0);
while(1);

}
```

Sa_mask :-

→ Sa-mask specifies a mask of signal which should be block during execution of signal handler.

```
#include <stdio.h>
#include <signal.h>
Void my_isr (int n)
{
    printf ("In isr----\n");
    sleep(10);
    printf (" In isr after sleep---\n");
}
main()
{
    Struct sigaction V;
    V.sa_handler = my_isr;
    sigfillset (&V.sa_mask);
    V.sa_flags = 0;
    sigaction (2, &V, 0);
    while (1);
}
```

→ sigemptyset → allow the new signal

sigfillset → Block the New signal

$ man sigemptyset :-

1) int sigemptyset (sigset_t *set);

2) int sigfillset (sigset_t *set);

3) int sigaddset (sigset_t *set, int signum);

4) int sigdelset (sigset_t *set, int signum);

5) int sig ismember (const sigset_t *set, int signum);

1) sigemptyset (&V.sa_mask);
2) sigaddset (&V.sa_mask, num) (3);

} → allow all other signal where isr is executing.

1) sigfillset (&V.sa_mask);
2) sigdelset (&V.sa_mask, num) (3);

} → allow all other except signum when isr is executing.

Program :-

```
#include <stdio.h>
# include <signal.h>
Void my_isr (int n)
{
    Printf ("In isr----\n");
    Sleep (20);
    Printf (" In isr after sleep---\n");
}
main()
{
    Struct sigaction V;
    V.sa_handler = my_isr;
    sigemptyset (&V.sa_mask);
    sigaddset (&V.sa_mask, 3);
    Va.sa_flags = 0;
    sigaction (2, &V, 0);
    Printf (" Hello ---- Pid = %d\n", getpid());
    while(1);
}
```

** Resource Managment **

1) setrlimit()
2) getrlimit()

success return = 0 ⟷ otherwise return = -1.

$ ulimit -a

⇒ all resource name
⇒ all resource limit } Show by this Command

1) core file size = 0
2) data seg size = unlimited
3) scheduling priority = 20
4) file size = unlimited
5) pending signals = 16852
6) max locked memory = ~~1024~~ 64
7) max memory size = unlimited
8) open files = 1024
9) pipe size = 8
10) CPU time = unlimited
11) stack size = 8192
12) file locks = unlimited
13) Virtual memory = unlimited.

There are two types of limit

- Soft limit
- hard limit.

→ "Soft limit" is limit when ever process is created default resource manager will assign it.

⇒ "Hard limit" is a limit, process can request to be the resource manager upto that limit.

$ man getrlimit :-

```
# include <sys/time.h>
# include <sys/resource.h>

int getrlimit (int resource, struct rlimit * rlim);
int setrlimit (int resource, const struct rlimit *rlim);
```

⇒ getrlimit and setrlimit() get and set resource limits respectively. Each resourced has an associated soft and hard limit, as defined by

```
struct rlimit
{
    rlim_t  rlim_cur ;  // soft limit
    rlim_t  rlim_max ;  // hard limit
}
```

Program :-

```
# include <stdio.h>
# include <sys/time.h>
# include <sys/resources.h>
main()
{
    struct rlimit V;

        getlimit (RLIMIT_STACK, &V);
    Printf (" Soft limit = %u\n  Hardlimit = %u\n",
                                V.rlim_cur, V.rlim_max);
  V.rlim_cur = 10000;
  setrlimit (RLIMIT_STACK, &V);
```

```
getrlimit ( RLIMIT_STACK , &V ) ;
Printf (" Soft Limit = %u \n  Hardlimit = %u\n",
                    V.rlim_cur , V.rlim_max ) ;
}
```

output

Soft limit = 8388608

hard limit = 4294967295

② RLIMIT_CORE :-

Maximum size of core file. when 0 no core dump files are created. when nonzero larger dumps are truncated to this size.

$ man core

core - core dump file

→ The default action of certain signals is to cause a process to terminate and produce a coredump file a disk file containing an image of the processes memory at the time of termination.

→ This image can be used in a debugger (eg. gdb tool).

$ man 7 signal

→ Each signal has a current dispation, which determines how the process behave when it is delivered the signal.

Term → terminate the process

Ign → Ignore the signal

Core → terminate the process and dump core.

Stop → stop the process.

Cont → continue the process.

Program :-

```
# include <stdio.h>
# include <sys/time.h>
# include <sys/resources.h>
main()
{
    struct rlimit V;
    getrlimit (RLIMIT_CORE, &V);
    Printf (" soft limit = %u\n  Hard limit = %u\n",
                    V.rlim_cur, V.rlim_max);
    V.rlim_cur = 10000;
    setrlimit(& RLIMIT_CORE, &V);
            getrlimit (RLIMIT_CORE, &V);
Printf (" soft limit = %u\n  Hard limit = %u\n"
                    V.rlim_cur, V.rlim_max);
    while (1);
}
```

output :- Soft limit = 0
Hard limit = 4294967295
Soft limit = 10000
Hard limit = 4294967295
Core dumped

③ RLIMIT_CPU :-

⇒ CPU time limit in seconds. when the process reches the soft limit, it is sent SIGXCPU signal. The default action for this signal is to terminate the process.

⇒ It will sent be sent SIGXCPU once per second until the hard limit is reached, at which time it is sent SIGKILL.

```
# include <stdio.h>
# include <sys|time.h>
# include <sys|resources.h>
main()
{
    struct rlimit V;
        getrlimit (RLIMIT_CPU , &V);
  Printf (" Soft limit = %u\n   Hard limit = %u\n ",
                V.rlim_cur , V.rlim_max);

  V.rlim_cur = 10;
  setrlimit (RLIMIT_CPU, &V);
        getrlimit (RLIMIT_CPU, &V);
Printf (" SoftLimit = %u\n    Hardlimit = %u\n ",
                V.rlim_cur, V.rlim_max);
  while(1);
}
```

④ **RLIMIT_DATA :-**

The maximum size of the process's data segment. This limit affects calls to brk() and sbrk(), which fail with the error ENOMEM upon encountering the soft limit of this resources.

⑤ **RLIMIT_FSIZE :-**

The maximum size of files that the process may create. Attempts to extend a file beyond this limit result in delivery of a SIGXFSZ signal.

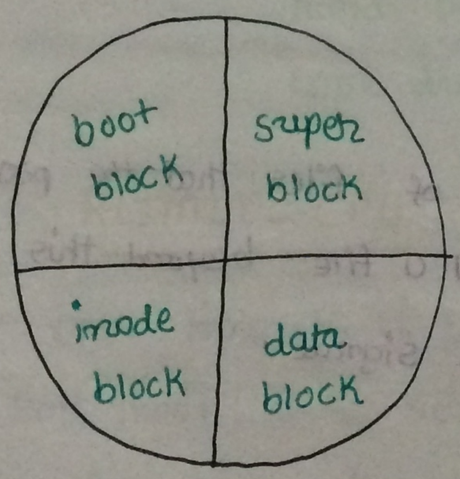
```
# include <stdio.h>
# include <sys|time.h>
# include <sys|resource.h>
main()
{   FILE *fp;
    struct rlimit V;
```

```
getrlimit (RLIMIT_FSIZE, &V);
Printf(" soft limit = %u\n  Hard limit = %u\n"
                    V.rlim_cur, V.rlim_max);

V.rlim_cur = 5 ;
setrlimit (RLIMIT_FSIZE, &V);
                getrlimit (RLIMIT_FSIZE, &V);
Printf (" soft limit = %u\n  Hard limit = %u\n",
                    V.rlim_cur , V.rlim_max);

fopen = ("data","W");
fwrite ("abcdefghi", 9, 1, fp);
}
```

Output :-

```
Soft Limit  = 4294967295
Hard limit  = 4294967295
Soft limit  = 5
Hard limit  = 4294967295
File size limit exceeded.
```

** FILE Managment **



→ where when we install the OS it is divided into the different block

1) boot block
2) Super block
3) inode block
4) data block

① **boot Block** :- consits of booting related information (boot loader)

→ OS is in this block.