①
```
# include <stdio.h>
# include <stdlib.h>
main()
{
   Printf ("Hello---\n");
   System ("ls");
   Printf ("Hai---\n");
}
```

Output

Hello ----

a.out cal.c imp.c P1.c

Hai---

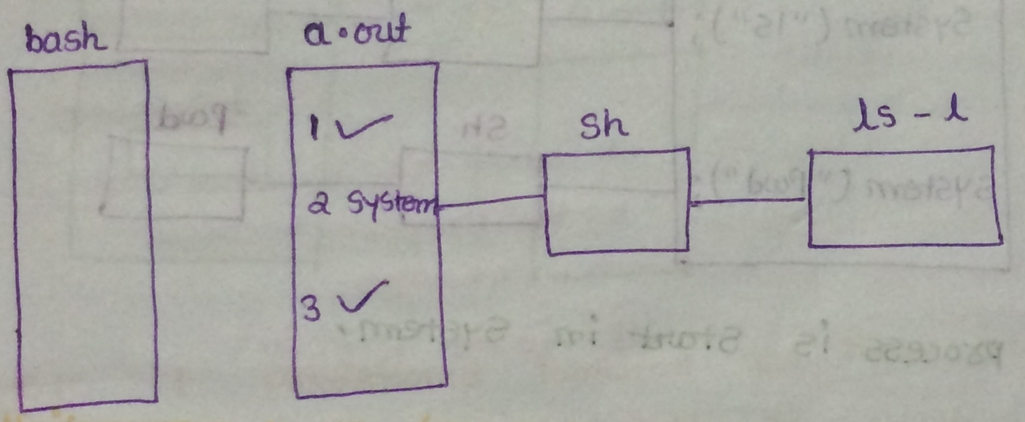
②
```
main()
{
   Printf ("Hello---\n");
   System ("cal");
   Printf ("Hai---\n");
}
```

Output :-

Hello ----

Calander for current month

Hai ----



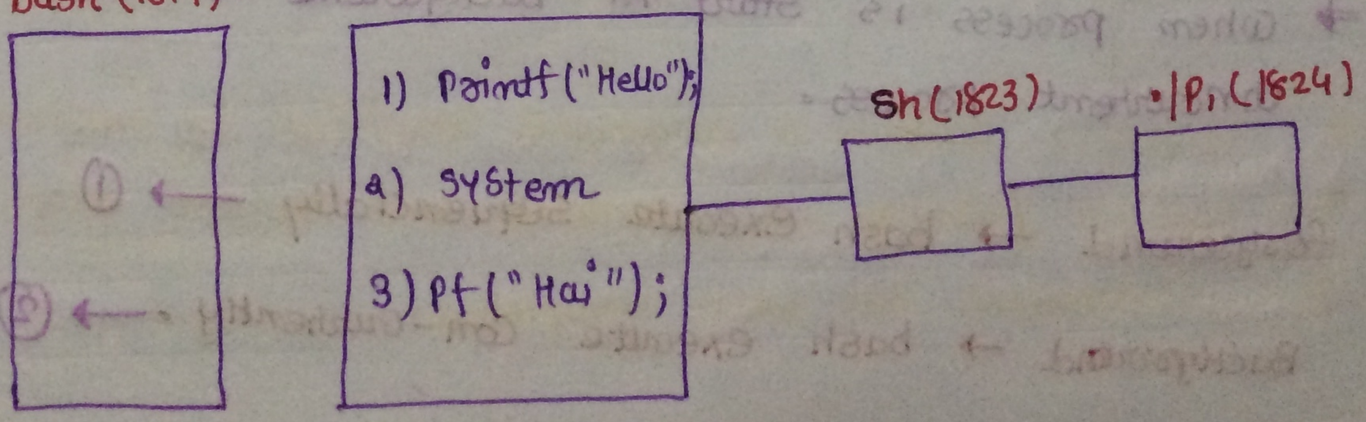
cc P1.c -o P1

③
```
main()
{
   Printf ("Hello---\n");
   system ("./P1");
   Printf ("Hai---\n");
}
```

Output

Hello ---

Pid = 1739



(*) check total process :-

$ ps -e | grep pts/1

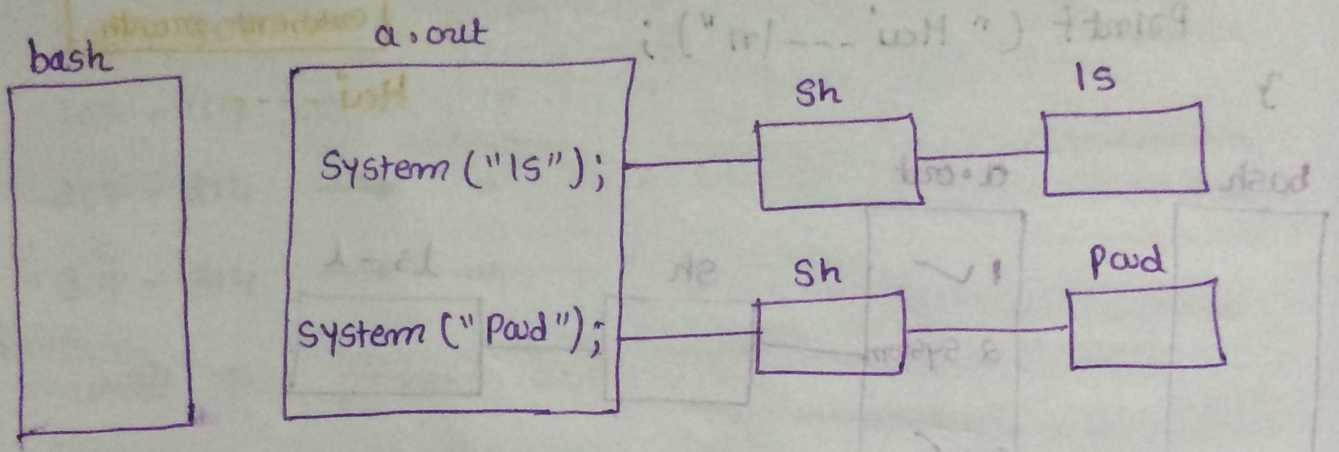
⇒ During execution of the command SIGCHLD will be blocked and SIGINT and SIGQUIT.

⇒ when we press ctrl+c then Hai is printed.

./P1 ↵

Then output Pid = 1621

⇒ so shell is waiting for the our program is completed (in forground). when we press ctrl+c then program is terminated.



Total 6 process is start in system.

⇒ System ("ls") and system ("Pwd") start sequentially because of "ls" is complited then "Pwd" is exeute.

(*) Sequencially process :-

⇒ when one process is completed after second process is execute is called sequentially process.

(*) Con-current process :-

⇒ When process is start in Background is called Concurrent process.

forground → bash execute sequencially ⟶ (1)

Background → bash execute Con-Currently. ⟶ (2)

① because bash is waiting for our program executed completely.

② bash is not waiting the program Run as it as background.

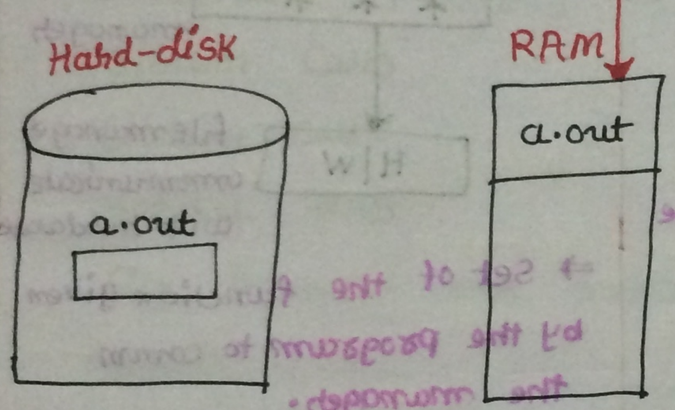
P1.C

```
main()
{
    ≡≡≡
system ("rm a.out");
    ≡≡≡
while(1);
}
```

⇒ because of a.out copy from the HDD so not delet from Hard-disk.

⇒ a.out is deleted from the RAM.

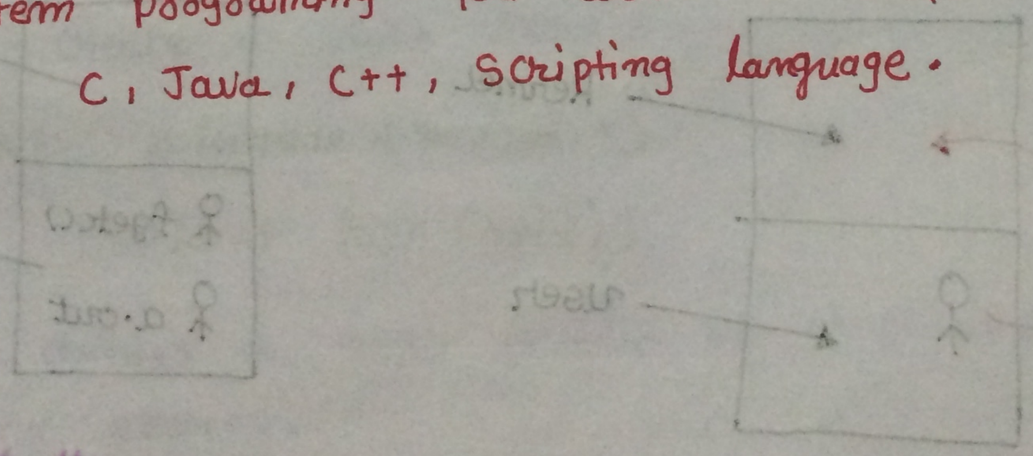
⇒ we can't achive the Job con-currently.

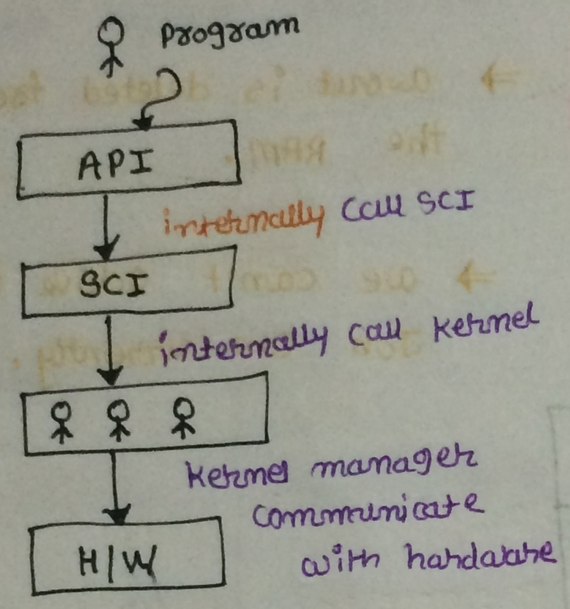
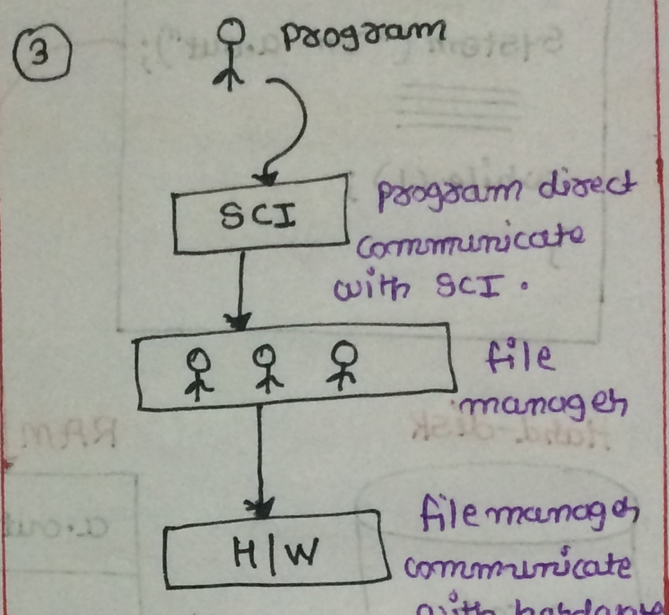


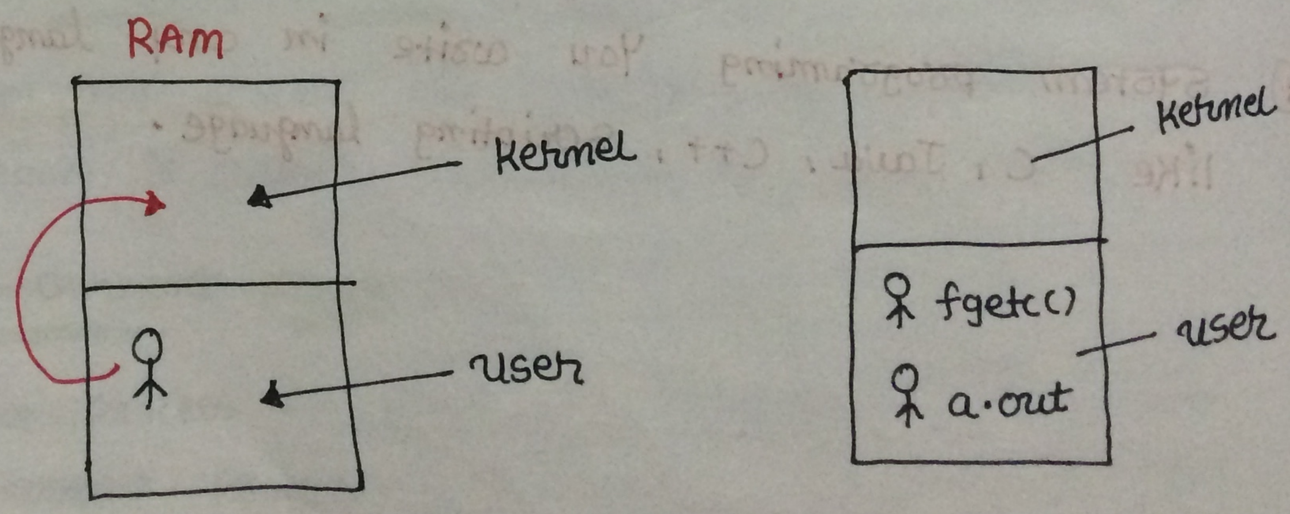
Que:- if predefined function are there so why we design our own function?

⇒ Compiler not supported all the function which user required.

⑧ System programming You write in any language like C, Java, C++, Scripting language.



library function	System Calls
① It is compiler supported function.	① It is operating system supported function.
② other name for library fun^{n} is API (Application program interface)	② other name for system calls is SCI (System call interface).
③ 	③ 
④ API is slower execution	⇒ Set of the function given by the program to comm the manager.
⑤ According to process API is faster.	④ faster execution.
	⑤ According to the process SCI is slower.



⇒ Kernel Space is restricated area because all the manager are there.

⇒ when process is goes to user space to Kernel space that time the background process use jmp.

⇒ API is not distrub to Kernel all the time, when its go to there that time its takes needed things after it gives to you (program).

Que :- Why system programing important ?

⇒ because directly communicate with operating system.

$ Man man :-

1) Executable program or shell command
2) System Calls
3) Library Calls
4) Special files
5) file formates and conventions
6) games
7) Miscellaneous
8) system administration Command
9) Kernel routine

$ man fork :-

fork - Create a child process.

Synopsis :- #include <unistd.h>

pid_t fork (void);

⇒ fork creates a new process by duplication the Calling process.

⇒ The New process, referred to as the child, is an exact dupliate of the calling process, referred to as the parent, except for the following point.

① The child has its own unique process ID and this PID does not match.

①
```
#include <stdio.h>
main()
{
    printf ("Hello ----\n");
    fork();
    printf ("Hai ---\n");
    while();
}
```

Output

```
Hello ----
Hai ---
Hai ---
```

```
ps -e | grep pts/3
1825  pts/3  00:00:00  bash
ps -e | grep pts/3
1825  pts/3  00:00:00  bash
2082  pts/3  00:00:07  a.out
2083  pts/3  00:00:07  a.out
```

⇒ after the fork() ; whatever the statement you written once executed by the parent and once executed by child.

```
#include <stdio.h>
main()
{
  printf (" Hello --- Pid = %d\n" Ppid = %d\n"
                        getpid(), getppid());
  fork();
  Print (" Hai --- Pid = %d\n  Ppid= %d\n",
                        getpid(), getppid());
}
```

Output :-

Hello --- Pid = 2102 Ppid = 1825

Hai --- Pid = 2102 Ppid = 1825

Hai --- Pid = 2103 Ppid = 1825

⊛ Program :-

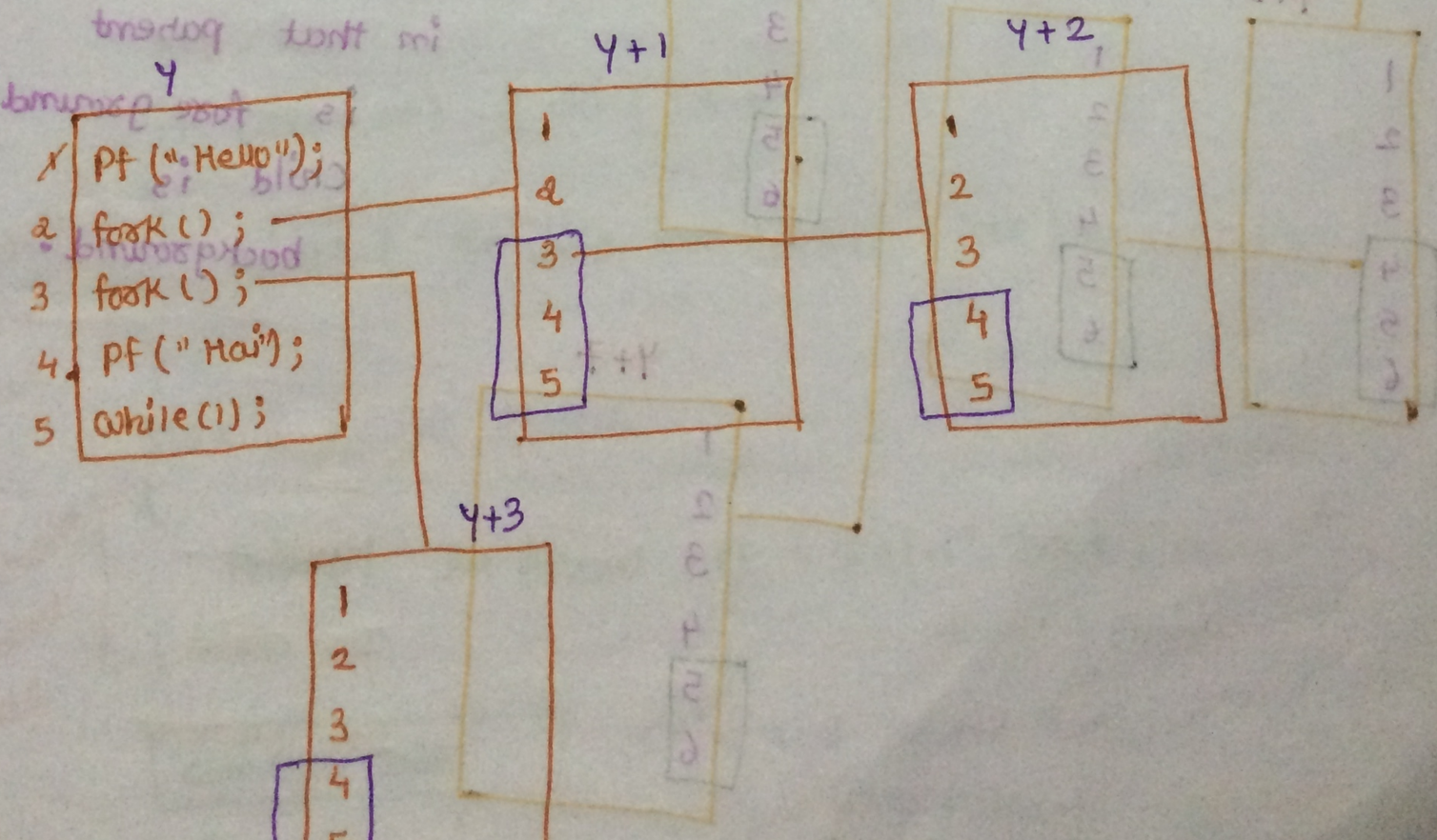
```
# include <stdio.h>
main()
{
    Printf("Hello---\n");
    fork();
    fork();
    Printf("Hai---\n");
    while(1);
}
```

Output :-

Hello --- (Y)
Hai --- (Y+2)
Hai --- (Y+1)
Hai --- (Y+3)
Hai --- (Y)

```
ps -e | grep pts/0
```

1825	pts/0	00:00:00	bash
2082	pts/0	00:00:05	a.out
2083	pts/0	00:00:05	a.out
2084	pts/0	00:00:05	a.out
2085	pts/0	00:00:05	a.out

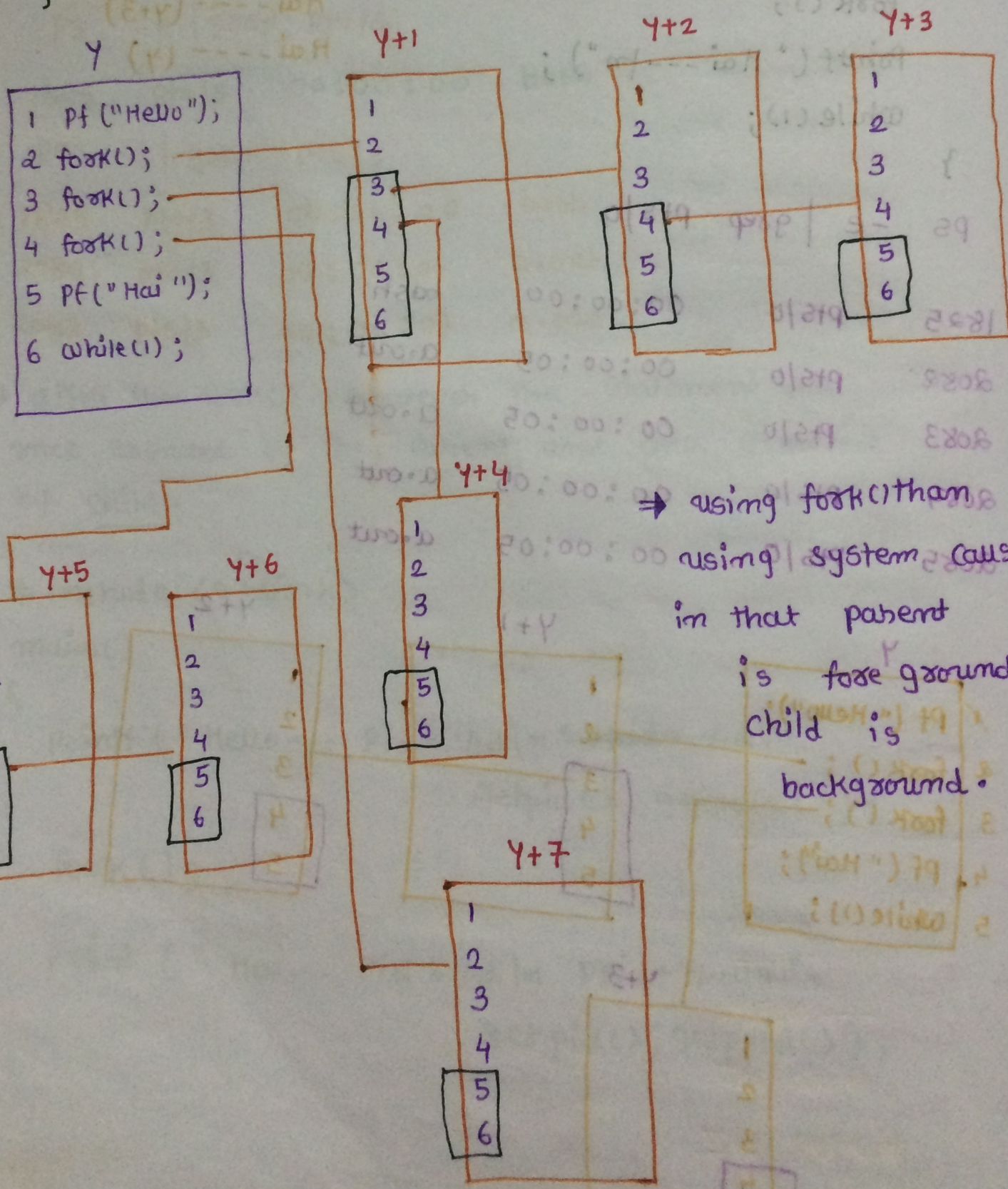


Program :-

```
#include <stdio.h>
main()
{
1  Printf ("Hello---\n");
2  fork();
3  fork();
4  fork();
5  Printf ("Hai----\n");
6  while(1);
}
```

Output :-

```
Hello ... Y
Hai ---   Y+3
Hai ---   Y+2
Hai ---   Y+4
Hai ---   Y+1
Hai ---   Y+6
Hai ---   Y+5
Hai ---   Y+7
Hai ---   Y
```



⇒ using fork() than using system calls in that parent is fore ground child is background.