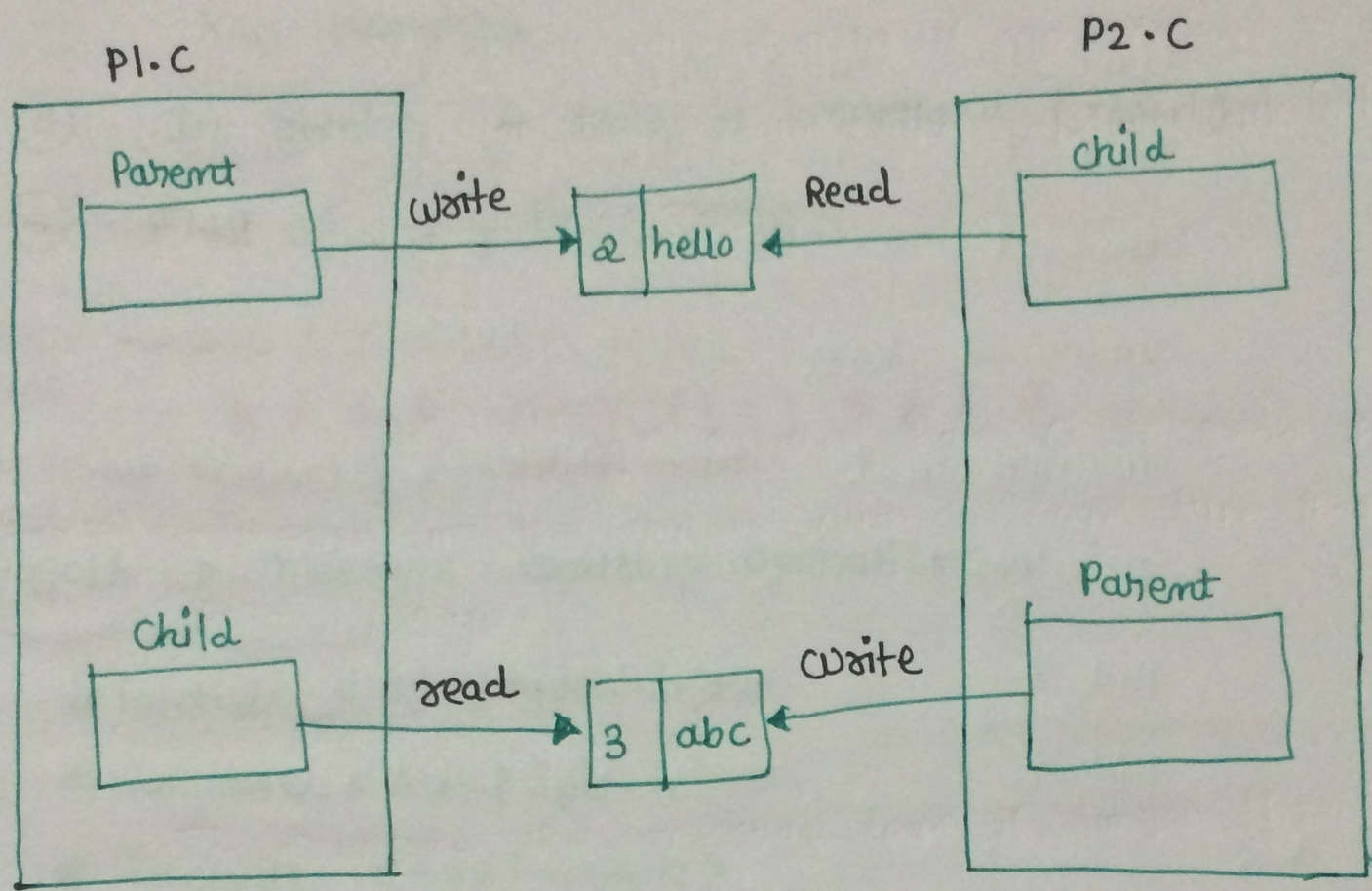
Program :- Full-duplex Communication using a Msgqueue() Function.



⊛ Msgsnd() and msgrcv() :-

1. design a Structure
2. Key required
3. In receive 4 place is important (-1,0,1)
4. Flag is Important role.

**** msgctl() ****

msgctl → message control operations.

```
#include <sys|types.h>
#include <sys|ipc.h>
#include <sys|msg.h>
```

① int msgctl (int msgid , int cmd , struct msgid_ds *buf);

⇒ msgctl() performs the control operation specified by cmd on the system V message queue with identifier msqid.

⊛ struct ipc_perm

```
{
    key_t   _key;   // Key supplied to msgget()
    uid_t   uid;    // UID owner
    gid_t   gid;    // GID owner
    uid_t   cuid;   // UID Creator
    gid_t   cgid;   // GID Creator
    unsigned short mode;   // permission
    unsigned short _seq;   // sequence Number.
};
```

```
struct msqid_ds
{
    struct ipc_perm   msg_perm ;  // ownership & permission
    time_t            msg_stime ;  // last msgsnd()
    time_t            msg_rtime ;  // last msgrcv()
    time_t            msg_ctime ;  // time of the last change.
    unsigned long     _msg_cbytes ;  // current total bytes in queue
    msgqnum_t         msg_qnum ;  // current no. of msg in queue
    msglen_t          msg_qbytes ;  // maximum no. of bytes allowed in queue.
    pid_t             msg_lspid ;  // PID of last msgsnd()
    pid_t             msg_lrpid ;  // PID of last msgrcv()
};
```

cmd

① IPC_STAT :- Copy information from the kernel data structure associated with msqid into the msqid_ds structure pointed to by buf.

② IPC_SET :- (id, IPC_SET, &V);

write the value of some member of the msqid_ds structure pointed by to a buf to the kernel data structure accoicated with this message queue, updating also its also msg_ctime member.

ie msg_perm.uid

msg_perm.gid

③ IPC_RMID (id, IPC_RMID, NULL);

⇒ remove the message queue by using a program.

④ IPC - INFO :-

Return information about system wide message queue limits and parameters in the structure pointed to by buf.

On success IPC_STAT, IPC_SET, IPC_RMID = $\boxed{0}$

on error returnend = $\boxed{-1}$.

✱ advantage :-

① both process does not to alive.

② transmitter can send data to perticular receiver.

✱ disadvantage :-

① Slowest IPC mechanism.

To overcome this disadvantages used to shared-memory IPC mechanism.

*** Shared Memory ****

1) shmget()
2) shmat()
3) shmdt()
4) shmctl()

① shmget() :-

It is useful for creating as well as opening the shared memory segment.

② shmat() :-

It is useful for attaching the shared memory segment to our process.

③ shmdt() :-

It is useful for detching the shared memory segment from our process.

④ shmctl() :-

It is useful for doing some control operation on shared memory.

$ man shmget

shmget → allocates a shared memory segment.

```
#include <sys/ipc.h>
# include <sys/shm.h>

int shmget (key_t key , size_t size , int shmflg);
```

- malloc create the dynamic memory for process-1 Then process-2 can't access it.
- shmget creat the shared memory for process-1 Then process-2 can't access it.

program :-

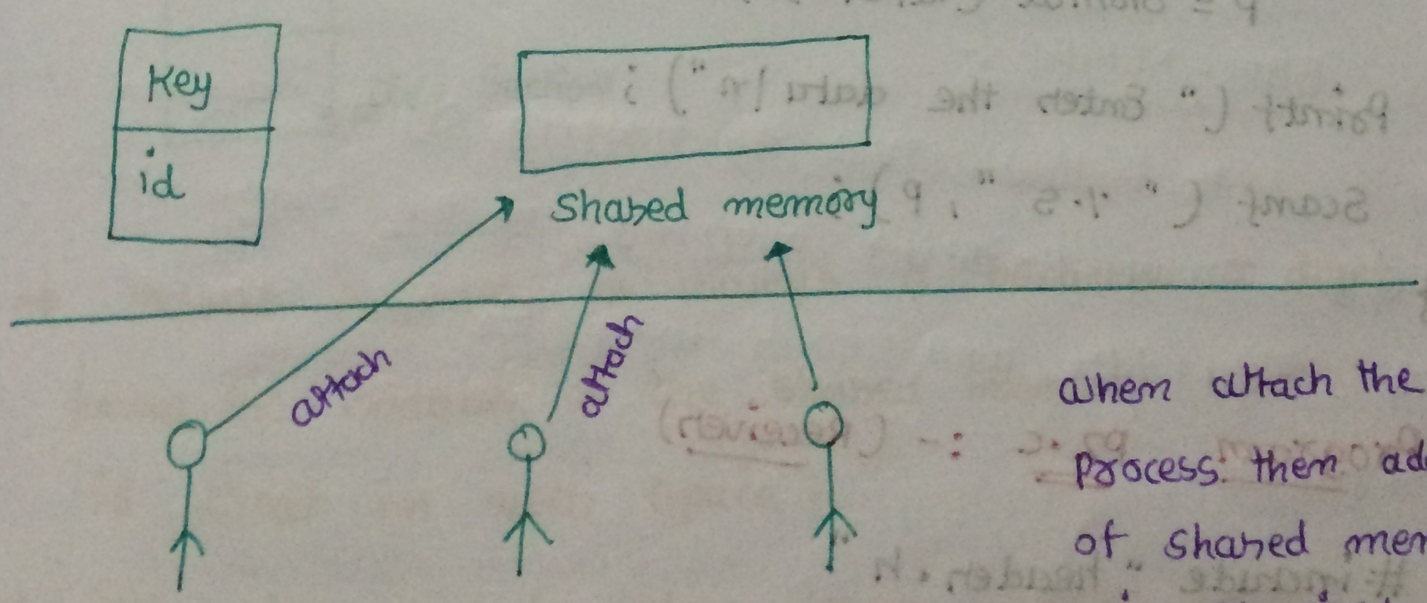
```
#include "header.h"
main()
{
    int id;
    id = shmget (5, 50, IPC_CREAT|0644);
    if (id<0)
    {
        perror ("shmget");
        return;
    }
    Printf(" id = %d\n", id);
}
```

ipcs -m :→ show the shared memory in system.

⇒ Some processes created already this processes maybe created by deamone.

⇒ Key = 0x000000 meas it is a private. Other processes can not used it.

⇒ Key_t Key = IPC_PRIVATE means it is private shared memory but can't used other processes.



when attach the process then address of shared memory is required.

(*) $ man shmop

```
#include <sys/types.h>
#include <sys/shm.h>
```

(1) Void * shmat (int shmid, const Void * shmaddr, int shmflg);

(2) int shmdt (const Void * shmaddr);

Program P1.C :- (Sender)

```
#include "header.h"
main()
{
    char *p;
    int fd;
    id = shmget (5, 50, IPC_CREAT | 0644);
    if (id<0)
    {
        Perror ("shmget");
        return;
    }
    Printf (" id = %d\n", id);
        P = Shmat (id, 0, 0);
    Printf (" Enter the data\n");
    Scanf (" %s ", p);
}
```

Program P2.C :- (Receiver)

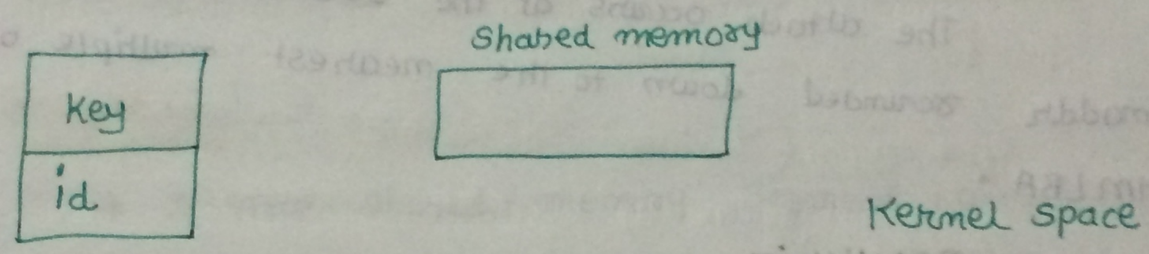
```
#include "header.h"
main()
{
```

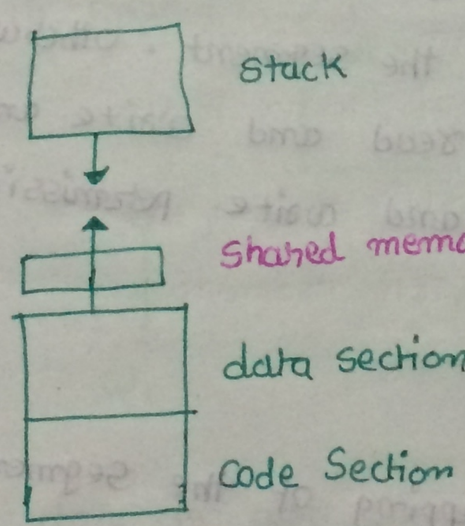
```
onst Void * shmaddr,
  int shmflg);

shmaddr);

);
```

```
char *p;
int id;
id = shmget (5,50, IPC_CREAT | 0644);
if (id <0)
{
    Perror ("shmget");
    return;
}
Printf (" id = %d\n", id);
    P = Shmat (id,0,0);
Printf (" DATA : %s\n", P);
}
```





Creat using shmat() function.

⇒ Second parameter is always 0 because I don't know the address of the shared memory because it is creat in user space.

⇒ If shmaddr is NULL, then system chooses a suitable address.

Shmat() return Value :-

Success → return the address of the attached shared memory segment.

Failure (error) → -1 is returned.

Shmdt() return Value :-

Success → return 0

on error → return -1 and errno.

Flags details :-

① **SHM_RND :-**

The attach occurs at the address equal to Shmaddr rounded down to the nearest multiple of SHMLBA.

② **SHM_RDONLY :-**

The segment is attached for reading and process must have read permission for the segment. otherwise the segment is attached for read and write and the process must have read and write permission for the segment.

③ **SHM_REMAP :-**

to indicate that the mapping of the segment should Replace may existing mapping in the range starting at Shmaddr and continuing for the size of segment.