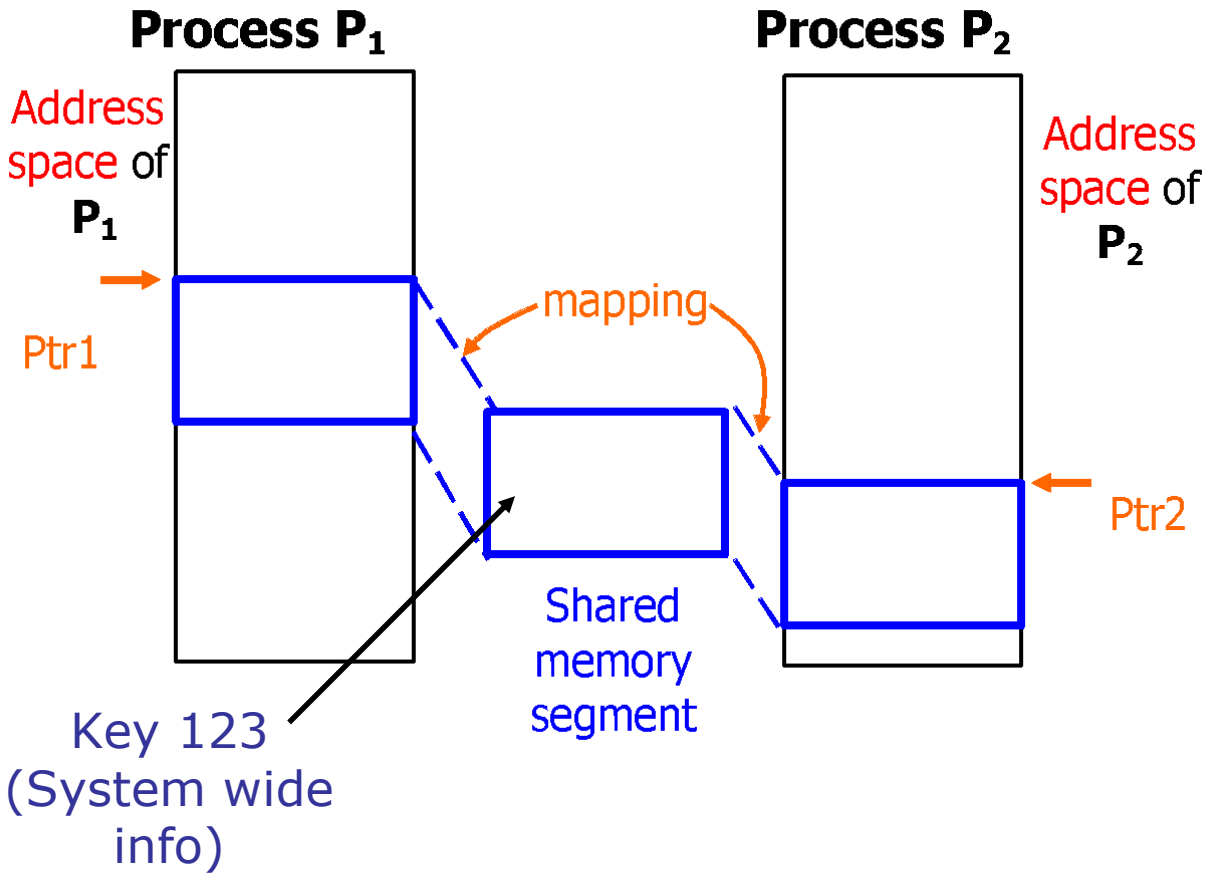
Interprocess Communication
IPC
Shared Memory

Shared Memory Concept



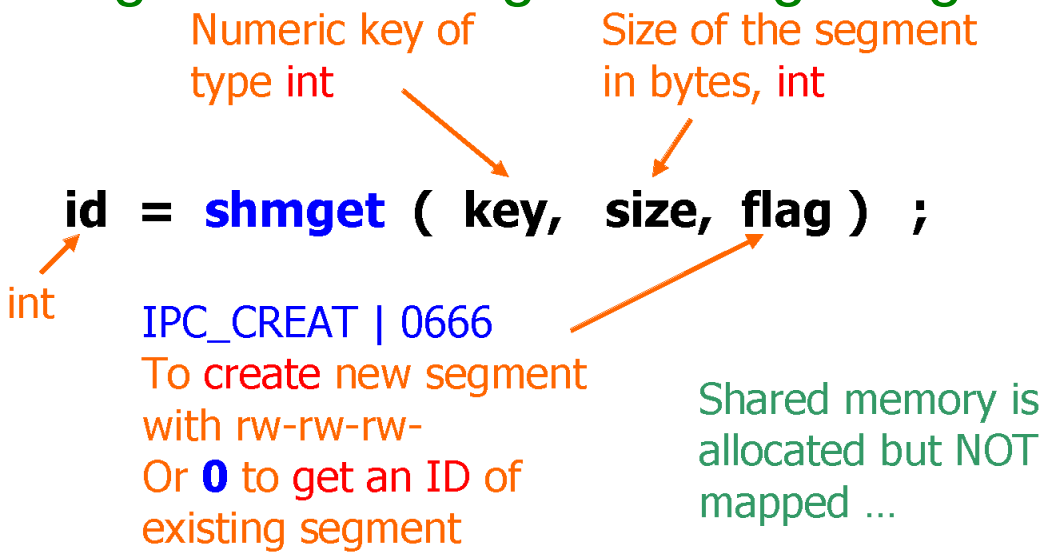
How to Create Shared Memory

- One of the processes creates a shared segment. Other processes get ID of the created segment
- Each process, using the same key, attaches to itself the created shared segment
- Now each process may write and read data into the shared segment
- Each process detaches the segment (after finishing)

System Calls for Shared Memory

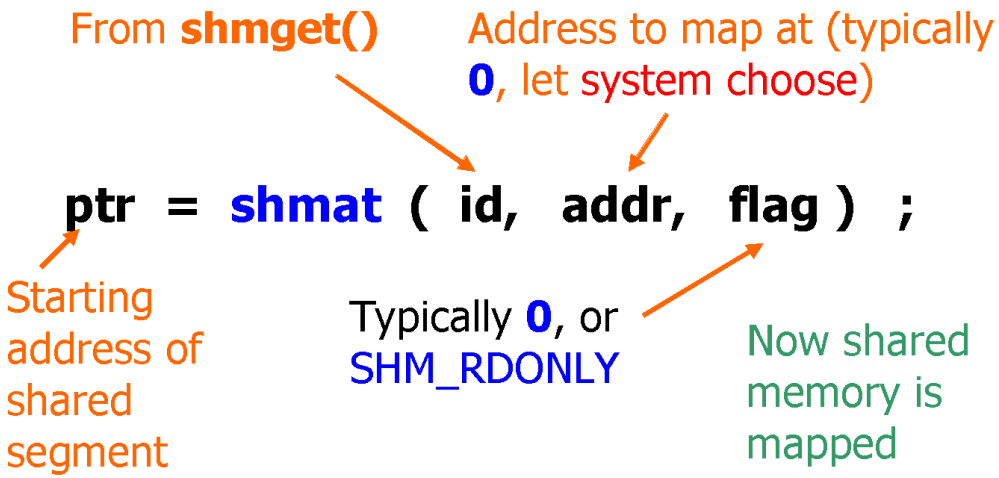
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

1. Creating a shared segment or getting its ID



System Calls for Shared Memory

2. Attaching (mapping) the existing memory segment (getting its pointer).



```
Typedef struct shmid_ds{
struct ipc_perm shm_perm; /* operation permission
                             structure */
size_t shm_segsz;            /* size of segment in
  bytes */
pid_t shm_lpid;        /* process ID of last operation
  */
pid_t shm_cpid;        /* process ID of creator */
shmatt_t shm_nattch;    /* number of current attaches
  */
time_t shm_atime;            /* time of last shmat */
time_t shm_dtime;            /* time of last shmdt */
time_t shm_ctime;     /* time of last shctl */
} shmid_ds
```

Example

```
/* Header file line.h */
struct info {
            char c;
            int length;
};
#define key ((key_t)(key = 1243));
#define  MSIZ  sizeof(struct info)
```

Example

Process A

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include "line.h"

main(void)
{
        int  i, id  ;
        struct info *ctrl;

        if ( (id = shmget( key,MSIZ,IPC_CREAT | 0666) ) <
0 )
        {
                printf("error 1\n");
                exit(1) ;
        }

        if ( (ctrl = (struct info *) shmat( id, 0, 0)) <= (struct
info *) (0) )
        {

                printf("error 2\n");
                exit(1) ;
        }
```

Process B

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "line.h"

main(int argc, char *argv[])
{
        int  id  ;
        struct info *ctrl;

        if (argc < 3)
        {
                exit(3);
        }
        if ( (id = shmget( key,MSIZ, 0 )) < 0 )
        {
                exit(1) ;
        }

        ctrl = (struct info *) shmat( id, 0, 0);
        if ( ctrl <= (struct info *) (0) )
        {
                exit(1) ;
        }
```

Example

Process A

```
    ctrl->c = 'a';
    ctrl->length = 10;

    // print line every 4 seconds
    for (i = 0 ; i <ctrl->length ; i++ )
    {
                        putchar (ctrl->c);
                        putchar ('\n');
                        sleep(4);
    }

    //now detach the pointer from the shared memory
    shmdt(ctrl);
    //let us delete the shared memory
    shmctl(id,IPC_RMID,NULL);

    //job done
}
```

Process B

```
        ctrl->c = argv[1][0] ;
                ctrl->length = atoi(argv[2]);


                //detach the pointer from the shared memory
                shmdt(ctrl);
                exit(0);
}
```

Example 2

Header

```
struct info {
        char c;
        int length;
                char flag;
};
key_t  key = 1243 ;
#define  MSIZ  sizeof(struct info)
```

Example 2

Process C

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "header.h"

main(void)
{
        int  i, id  ;
        struct info *ctrl;

        if ( (id = shmget( key,MSIZ,IPC_CREAT | 0666) ) <
0 )
        {

        printf("error 1\n");
        exit(1) ;
        }

        if ( (ctrl = (struct info *) shmat( id, 0, 0)) <= (struct
info *) (0) )   {
                        printf("error 2\n");
                       exit(1) ;
    }

        ctrl->c = 'a';
        ctrl->length = 10;
        ctrl->flag = 0;
```

Process D

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "header.h"

main(int argc, char *argv[])
{
        int  id  ;
        struct info *ctrl;
        if (argc < 3)
        {
                exit(3);
        }

        if ( (id = shmget( key,MSIZ, 0 )) < 0 )
        {
                exit(1) ;
        }

        ctrl = (struct info *) shmat( id, 0, 0);
        if ( ctrl <= (struct info *) (0) )
        {
                exit(1) ;
        }
  // wait until Process C is ready to receive from process D
        while(ctrl->flag==0);
```

Example 2

Process C

```
                for (i = 0 ; i <ctrl->length ; i++ )
                 {
                                putchar (ctrl->c);
                                putchar (' ');
                 }
                putchar('\n');
                fflush(stdout);
                ctrl->flag = 2;

                while(ctrl->flag!=1);

  // print the character ctrl->c for ctrl->length times in a single line
                 for (i = 0 ; i <ctrl->length ; i++ )
                 {
                                putchar (ctrl->c);
                                putchar (' ');
                 }
                putchar ('\n');
                // flushing stdout just in case
                fflush(stdout);

                //now detach the pointer from the shared memory
                shmdt(ctrl);
                //let us delete the shared memory
                shmctl(id,IPC_RMID,NULL);

                //job done
}
```

Process D

```
        /* copy command line data to shared memory  */
                ctrl->c = argv[1][0] ;
                ctrl->length = atoi(argv[2]);
                ctrl->flag = 1;

  //detach the pointer from the shared memory
// .. we do not need to delete here, can be done in either of the process
                shmdt(ctrl);
                exit(0);
}
```

Shared Memory Features

- Efficiency (no intermediate copying of data as in pipes)
- Random access (a sequential byte stream in pipes)
- Many-to-many mechanism of IPC: many processes may attach the same segment (pipes provide one-to-one mechanism of IPC)
- No synchronization is provided (pipes provide synchronization)
- Limitations on the number of shared memory segments:
  - Some of these can be changed by a system administrator, by configuring a new kernel. These limits are: maximum and minimum size in bytes; max no. of shared memory segments, system wide and per process.