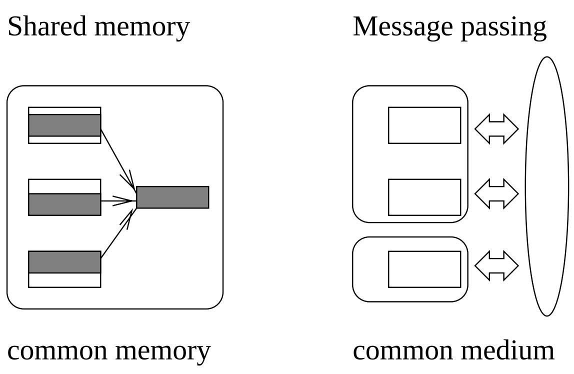
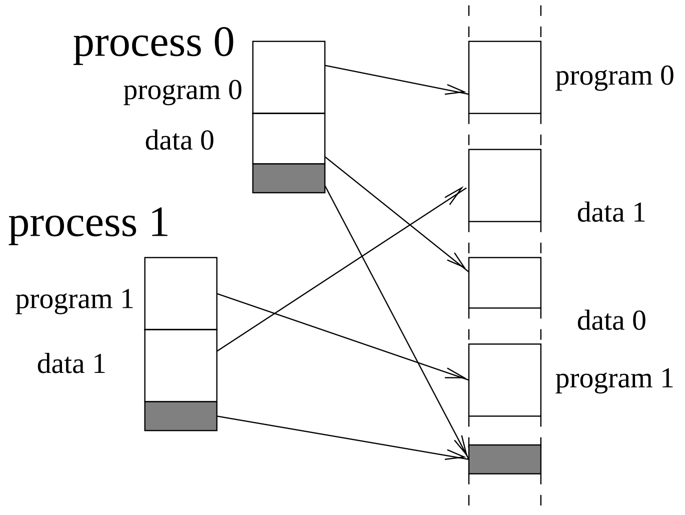
Shared memory and messages

- Shared memory vs message passing
- Shared memory - C functions
- Shared memory - example program
- Message queues - C functions
- Message queues - example program





Functions

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

Get shared memory segment.

```
void *shmat(int shmid, void *shmaddr, int shmflg);
```

Attach shared memory segment.

```
int shmdt(void *shmaddr);
```

Detach shared memory segment.

Functions

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Shared memory control.

IPC_STAT – get status
IPC_SET – set owner, group and mode
IPC_RMID – destroy memory segment
SHM_LOCK – lock memory segment in RAM memory
SHM_UNLOCK – unlock memory segment from RAM memory

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

int mid;
typedef struct
  {
  int ver;
  char buffer[200];
  } my_buff;

void delay( void )
{
int i, j;
```

```
for (i=0; i<100000; i++)
  j=i+1;
}

void fatal( char *msg )
{
perror( msg );
exit( 1 );
}

void mem_init( void )
{
int res;
my_buff *mem;
```

```
mem = shmat( mid, NULL, 0 );
if (mem == (void *)-1)
  fatal( "Buffer not attached!\n" );
mem->ver=0;
strcpy( mem->buffer, "" );
res=shmdt( mem );
if (res == -1)
  fatal( "Can not detach memory\n" );
}
```

```
void mem_create( void )
{
mid=shmget(333, sizeof(my_buff), IPC_CREAT | 0666 );
if (mid == -1)
  fatal( "Error getting memory!\n" );
printf( "Memory ID=%d\n", mid );
}

void mem_destroy( void )
{
int res;

res=shmctl( mid, IPC_RMID, NULL );
if (res == -1)
  fatal( "Can not destroy memory\n" );
```

```
}
```

```
my_buff *mem_attach( void )
{
my_buff *mem;

mem = shmat( mid, NULL, 0 );
if (mem == (void *)-1)
  fatal( "Child Buffer not attached!\n" );
return mem;
}

void mem_detach( my_buff *mem )
{
int res;

res=shmdt( mem );
```

```
if (res == -1)
  fatal( "Can not detach memory\n" );
}
```

```
void child( void )
{
int res;
int ver;
my_buff *mem;

mem = mem_attach();
while (mem->ver == 0)
  delay();
printf( "[%s]\n", mem->buffer );
mem->ver++;
strcpy( mem->buffer, "OK" );
mem_detach( mem );
exit(0);
}
```

```
void parent( void )
{
int res;
int ver;
my_buff *mem;

mem = mem_attach();
strcpy( mem->buffer, "Hurrah!" );
mem->ver++;
while (mem->ver == 1)
  delay();
printf( "[%s]\n", mem->buffer );
mem_detach( mem );
mem_destroy();
exit(0);
```

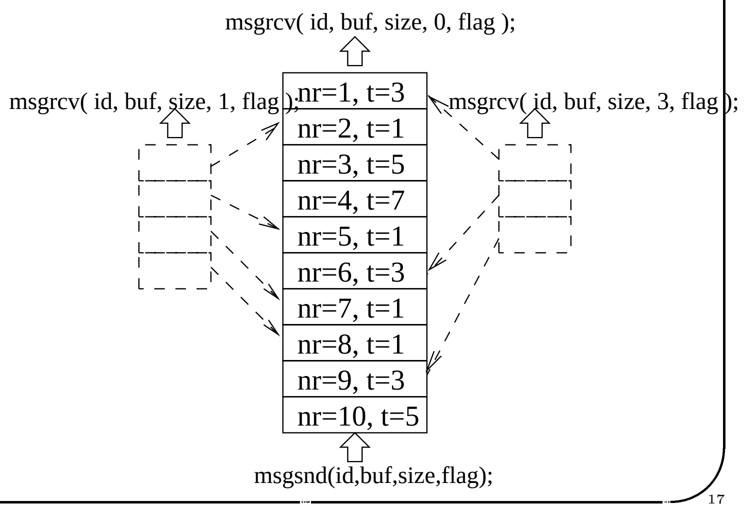
```
}
```

```
int main( int narg, char *argv[] )
{
int pid;

mem_create();
mem_init();

pid=fork();
if (pid == 0)
  child();
else if (pid > 0)
  parent();
else
  printf( "Error\n" );
}
```

Messages


msgrcv(id, buf, size, 0, flag);
msgrcv(id, buf, size, 1, flag);
msgrcv(id, buf, size, 3, flag);
nr=1, t=3
nr=2, t=1
nr=3, t=5
nr=4, t=7
nr=5, t=1
nr=6, t=3
nr=7, t=1
nr=8, t=1
nr=9, t=3
nr=10, t=5
msgsnd(id,buf,size,flag);

Functions

```
int msgget(key_t key, int msgflg);
```

Get message queue.

```
int msgsnd(int msqid, const void *msgp, size_t msgsz,
                                        int msgflg);
```

Send a message.

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
                                        int msgflg);
```

Receive a message.

Functions

```
int msgctl(int msqid, int cmd,
           /* struct  msqid_ds  *buf  */ ...);
```

Controll message queue.

IPC_STAT – get information about message queue
IPC_SET – set owner, group, mode and size of the queue
IPC_RMID – remove queue

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int qid;
typedef struct
 {
 long mtype;
 char mtxt[200];
 } msg_buff;

void fatal( char *msg )
{
perror( msg );
exit( 1 );
}
```

```
void msg_create( void )
{
qid=msgget( 123, IPC_CREAT | 0666 );
if (qid == -1)
  fatal( "Can not open queue\n" );
}

void msg_send( char *txt, int mt )
{
int res;
msg_buff msg;

msg.mtype = mt;
strcpy( msg.mtxt, txt );
res=msgsnd( qid, &msg, sizeof(msg), 0 );
```

```
if (res == -1)
  fatal( "Error sending message" );
}
```

```
void msg_destroy( void )
{
int res;

res=msgctl( qid, IPC_RMID );
if (res == -1)
  fatal( "Can not destroy queue!\n" );
}
char *msg_receive( int mt )
{
int res;
static msg_buff msg;

res=msgrcv( qid, &msg, sizeof(msg), mt, 0 );
if (res == -1)
```

```
  fatal( "Error sending message" );
return msg.mtxt;
}
```

```
void child( void )
{
msg_send( "Hello", 1 );
printf( "Child got: [%s]\n", msg_receive(2) );
msg_destroy();
exit(0);
}

void parent( void )
{
printf( "Parent got: [%s]\n", msg_receive(1) );
msg_send( "Hi", 2 );
exit(0);
}
```

```
int main( int narg, char *argv[] )
{
int pid;

msg_create();

pid=fork();
if (pid == 0)
  child();
else if (pid > 0)
  parent();
else
  printf( "Error\n" );
}
```