```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
  int freq[MAXPAROLA] ; /* vettore di contatori
  delle frequenze delle lunghezze delle parole */
  char riga[MAXRIGA] ;
  int i, inizio, lunghezza ;
  FILE * f ;

  for(i=0; i<MAXPAROLA; i++)
    freq[i]=0 ;

  if(argc != 2)
  {
    fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
  }
  f = fopen(argv[1], "rt") ;
  if(f==NULL)
  {
    fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
  }

  while( fgets( riga, MAXRIGA, f ) != NULL )
```

Inter-Process Communication

Shared Memory

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

License Information

Introduction

- Shared memory allows two or more processes to share a given region of the main memory space
  - All processes can access the shared memory
  - Changes made by one process can be viewed by all other processes
- Shared memory is similar to file mapping
  - Shared memory is less restrictive but harder to use
    - Function **mmap** is a "recent" POSIX system call
    - Function **shmget** has the widest support and is the old System V model

Unit 07
Section 05

Shared Memory:
Allows two or more processes to share a given region of the main memory space:
All processes can access the shared memory: Multiple processes can read from and write to the shared memory region.
Changes made by one process can be viewed by all other processes: Any modification in the shared memory by one process is immediately visible to all other processes sharing that memory.
Similar to File Mapping:
Shared memory is less restrictive but harder to use:
Function mmap: A more recent POSIX system call for memory mapping.
Function shmget: Has the widest support and is part of the older System V model.
Additional Explanation:
Shared Memory: This IPC mechanism allows processes to communicate by directly accessing a common memory space, which can lead to faster data exchange compared to other IPC methods.
File Mapping: Similar to shared memory, file mapping allows a file to be mapped into the address space of a process, enabling direct access to the file's contents.

Introduction

- Shared memory and file mapping are **faster** than pipes or messages queues
  - Data does not need to be copied between the client and the server and does not have to pass through the kernel
- Shared memory requires synchronization accesses to a given region among multiple processes
  - If the server is placing data into a shared memory region, the client should not try to access the data until the server is done
  - Semaphores must be used to **synchronize** shared memory access

Performance Advantage: Shared memory and file mapping avoid the overhead of copying data between processes, making them faster than other IPC methods like pipes and message queues.
Synchronization: Proper synchronization mechanisms, such as semaphores, are essential to ensure that multiple processes can safely access and modify the shared memory without causing data inconsistencies.

Logic flow

- Logic flow to use the shared memory
  - Generate an IPC key we can use the function **ftok**
  - Get an identifier with **shmget**
  - Attach the user to the shared memory with **shmat**
  - Use pointers to manipulate the shared memory
  - Detach the process from the memory with **shmdt**
  - Remove the identifier, and possibly, control the shared memory with **shmctl**

!

Additional Explanation:
ftok: Generates a unique key based on a file path and a project identifier.
shmget: Creates or accesses a shared memory segment using the generated key.
shmat: Attaches the shared memory segment to the process's address space, allowing the process to read from and write to the shared memory.
shmdt: Detaches the shared memory segment from the process's address space, indicating that the process no longer needs to access the shared memory.
shmctl: Performs control operations on the shared memory, such as querying its status, changing its permissions, or removing it.

Operations

Return value
Shared memory id, on success
The value -1, on error

System call	Meaning
int shmget (key_t key, size_t size, int flag);	**Obtains** a shared memory identifier given the key of the IPC object. The parameter size is the size of the shared memory segment in bytes. The parameter flag set the mode field of the IPC structure. See the example for further details.
int shmctl (int shmid, int cmd, struct shmid_ds *buf); Return value The value 0, on success The value -1, on error	Performs **various operations** on a shared memory. The memory is specified by its identifier (shmid). Parameter cmd specifies the command to be performed on the segment. As with function msgctl, it is possible to specify: PC_STAT, IPC_SET, IPC_RMID, or, when the process runs in super-user mode SHM_LOCK and SHM_UNLOCK.

Additional Explanation:
shmget: This function is used to create a new shared memory segment or access an existing one. The size parameter specifies the size of the segment, and the flag parameter can include permissions and behavior flags (e.g., IPC_CREAT to create the segment if it doesn't exist).
shmctl: This function allows for various control operations on the shared memory segment, such as querying its status (IPC_STAT), changing its permissions (IPC_SET), or removing it (IPC_RMID). The buf parameter points to a structure that holds information about the shared memory segment.

Operations

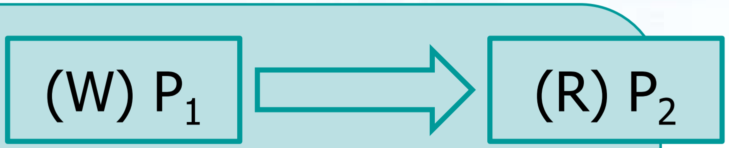
System call	Meaning
void *shmat (int shmid, const void *addr, int flag); Return value Pointer to the memory, on success The value -1, on error	**Attaches** a process to its address space. The address in the calling process at which the segment is attached depends on: The addr argument specify how to attach the segment (zero, to the first available address selected by the kernel; nonzero to the segment specified by addr or by (addr – (addr modulus SHMLBA))
int shmdt (const void *addr); Return value The value 0, on success The value -1, on error	When we are done with a shared memory segment, we call shmdt to **detach** it. Note that this does not remove the identifier and its associated data structure from the system. The identifier must be removed by calling shmctl with a command of IPC_RMID

Example

- Write two processes sharing 1KByte of memory and making some modification on it

Solution

1 Reader + 1 Writer



The same process works as
- a reader (no parameter)
- a writer (writes the parameter
on the shared memory)

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

#define SHM_SIZE 1024
```

Create a shared
memory segment of 1K

```
int main (int argc, char *argv[]) {
  key_t key;
  int shmid;
  char *data;
```

Solution

Make the key

Here the file must exist

```
if ((key = ftok ("hello.txt", 5)) == -1) {
  perror ("ftok");
  exit (1);
}
if ((shmid = shmget (key, SHM_SIZE,
             0644 | IPC_CREAT)) == -1) {
  perror ("shmget");
  exit (1);
}
data = shmat (shmid, NULL, 0);
if (data == (char *)(-1)) {
  perror ("shmat");
  exit (1);
}
```

Create the segment

Attach the segment to the local pointer **data**

Solution

```
  if (argc == 2) {
    printf (“Writing to segment: \"%s\"\n", argv[1]);
    strncpy (data, argv[1], SHM_SIZE);
  }
  else
  {
    printf("segment contains: \"%s\"\n", data);
  }
  if (shmdt(data) == -1) {
    perror ("shmdt");
    exit (1);
  }
  return 0;
}
```

Writer
Modify the segment, based on the command line

Reader
Read the segment

Detach from the segment