Inter-Process Communication

Message Queues

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

License Information

This work is licensed under the license

CC BY-NC-ND 4.0

Attribution-NonCommercial-NoDerivatives 4.0 International

This license requires that reusers give credit to the creator. It allows reusers to copy and distribute the material in any medium or format in unadapted form and for noncommercial purposes only.

BY: Credit must be given to you, the creator.

NC: Only noncommercial use of your work is permitted.
Noncommercial means not primarily intended for or directed towards commercial advantage or monetary compensation.

ND: No derivatives or adaptations of your work are permitted.

To view a copy of the license, visit:
https://creativecommons.org/licenses/by-nc-nd/4.0/?ref=chooser-v1

Introduction

- FIFOs (and pipes) are used to pass streams of anonymous bytes
  - Applications using FIFOS have to manage their own data chunking
    - The have to agree on data delimiters, such as end-of-field, end-of-record, etc.

Data Chunking in FIFOs: When using FIFOs, the application must agree on a protocol for segmenting the data, which can add complexity'

.

Structured Data in Message Queues: Message queues simplify the handling of structured data by allowing messages to be sent and received as discrete units.

- To pass structured data chunks it is necessary to use message queues
  - A **message queue** is a linked list of **messages**
    - Messages are stored within the kernel
  - The queue is identified through an identification **key**

Message !

Key !

Messages

What is a message?

- Message queues manipulate messages
  - Each message is composed must to be defined by the user as a C data structure including
    - The message type **mtype**
    - The data field **mtext** of maximum size N_BYTES

Message Type (mtype): This field allows messages to be categorized and filtered based on their type, providing flexibility in message handling.
Message Text (mtext): This field contains the actual data being transmitted. The size of this field is defined by N_BYTES, which can be adjusted based on the application's needs.
Queue Behavior: Messages are enqueued and dequeued in a first-in, first-out (FIFO) manner, ensuring that messages are processed in the order they are sent.

```
struct mesg_buffer {
  long int mtype;
  char mtext[N_BYTES];
} message;
```

The message type, used to categorize messages.

The data field, which holds the actual message content, with a maximum size of N_BYTES.

  - Each new message is placed at the end of the queue

Each new message is placed at the end of the queue: Messages are enqueued in the order they are sent.

Keys

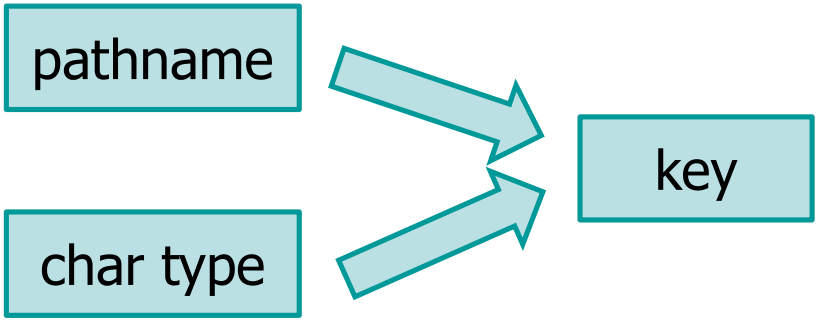
What is a key?

- Keys identify IPC structures
  - Have datatype **key_t**
  - Are defined as a long integer in header sys/types.h
- To generate an IPC key we can use function **ftok**
  - Function **ftok** converts a path and an integer value (in the range [0,255]) into a key
    - To share a key, the different processes (e.g., the clients and the server) must agree on the path and the value

Datatype key_t: Keys used for IPC structures have the datatype key_t.

Defined as a long integer: The key_t type is defined as a long integer in the header file sys/types.h.

Function ftok: This function converts a path and an integer value (in the range [0, 255]) into a key.
Usage:
Pathname and Char Type: The ftok function takes a pathname and a character type to generate a unique key.
Shared Key: For different processes (e.g., clients and servers) to share a key, they must agree on the same pathname and integer value.



Logic flow

- Logic flow to use a message queue
  - Generate an IPC key we can use function **ftok**

  - Create or open a message queue with **msgget**
  - Send new messages with **msgsnd**
  - Fetch messages from the queue with **msgrcv**

  - Possibly, control the queue with **msgctl**

Function ftok

```
#include <sys/msg.h>

key_t ftok (const char *path, int id);
```

Return value
Message key, on success
The value -1, on error

Purpose: Generates a key from a pathname and a project ID

Key Generation: The ftok function does not perform any communication. It simply generates a unique key based on the provided pathname and project ID. This key is then used during the communication phase to identify IPC structures like message queues.

- Parameters
  - A standard (file) pathname (that **must** exist)
  - A project ID (i.e., a char between 0 and 255)
- The system call ftok generates a key from a pathname and a project ID
  - Does not perform any sort of communication
  - The key is used during the communication phase

Operations

System Call	Meaning
int msgget (key_t key, int flag);	Creates or opens an existing message queue. The first parameter is a system key. The second one is a flag used to define the permission to a data structure associated to the message queue. The return value is the message queue identifier (**msqid**), on success, or the value −1, on error.
int msgctl (int msqid, int cmd, struct msqid_ds *buf);	Controls the queue, performs various operations on it. Parameter cmd (IPC_STAT, IPC_SET, IPC_RMID) specifies the command to be performed on the queue. The queue is specified by its identifier (**msqid**). The parameter msqid is the value returned by msgget.

Purpose: Creates or opens an existing message queue.

Purpose: Creates or opens an existing message queue.

key: The system key generated by ftok.
flag: Defines the permissions and behavior of the message queue.
Return Value:
Message queue identifier (msqid) on success.

Purpose: Controls the queue, performing various operations.

Parameters:
msqid: The message queue identifier.
cmd: The command to be performed (e.g., IPC_STAT, IPC_SET, IPC_RMID)
buf: A pointer to a structure used for the command.

msgget: This function is used to create a new message queue or access an existing one. The flag parameter can include permissions (e.g., read/write) and behavior flags (e.g., IPC_CREAT to create the queue if it doesn't exist).
msgctl: This function allows for various control operations on the message queue, such as querying its status, changing its permissions, or removing it.

Operations

System Call	Meaning

int msgsnd (
  int msqid,
  const void *ptr,
  size_t nbytes,
  int flag
);
 	
Purpose: Sends new messages to the queue.
	Sends new messages. A new message is added to the end of a queue. The identifier msqid specifies the queue on which to send a message. The ptr argument points to the specific defined message data structure **mymesg**. nbytes specify the size of the data array in mymsg. The flag value is 0 or IPC_NOWAIT (to deal with error on full queues).

Parameters:
msqid: The message queue identifier.
ptr: A pointer to the message to be sent.
nbytes: The size of the message.
flag: Flags to control the behavior (e.g., IPC_NOWAIT to avoid blocking).
Return Value:
0 on success.

msgsnd: This function adds a new message to the end of the message queue. The message is specified by the ptr parameter, which points to a user-defined structure containing the message type and data. The nbytes parameter specifies the size of the message data.

Operations

System Call	Meaning

ssize_t msgrcv (
  int msqid,
  void *ptr,
  size_t nbytes,
  long type,
  int flag
);
	Fetches messages from a queue. Messages do not have to be fetched in a first-in, first-out order but can be fetched based on their type field. Parameters are the same than for msgsnd. Type argument specifies which message we want. The first message in the queue (type=0), the first message in the queue whose message type equals type is returned (type>0), etc.

Purpose: Fetches messages from the queue.

Return value
The value 0, on success
The value -1, on error

Parameters:
msqid: The message queue identifier.
ptr: A pointer to the buffer where the message will be stored.
nbytes: The size of the buffer.
type: The type of message to receive (0 to receive the first message).
flag: Flags to control the behavior (e.g., IPC_NOWAIT to avoid blocking).
Return Value:
Number of bytes received on success.
-1 on error.
Additional Explanation:
msgrcv: This function retrieves a message from the message queue. The ptr parameter points to a buffer where the message will be stored. The type parameter allows for selective retrieval of messages based on their type. If type is 0, the first message in the queue is retrieved.

Example

- Use a message queue to transfer a structured message message

```
struct mesg_buffer {
  long int mtype;
  char mtext[N_BYTES];
} message;
```

- From a sender (producer) to a receiver (consumer)

(W) P1 -> (R) P2: Process P1 writes to the message queue, and Process P2 reads from it.

Structured Message: The message structure includes a message type (mtype) and a message text (mtext). This allows for organized and categorized data transfer between processes.

Example

1 Reader + 1 Writer

The Writer


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

#define L 512

struct mesg_buffer {
  long mesg_type;
  char mesg_text[L];
} message;
```

Message Structure: Defines the message structure with a type and text field.

Solution

The Writer

(W) P_1 → (R) P_2

```
int main() {
  key_t key;
  int msgid;

  key = ftok ("progfile", 65);
  msgid = msgget (key, 0666 | IPC_CREAT);
  message.mesg_type = 1;

  printf ("Read data: ");
  fgets (message.mesg_text, L, stdin);

  msgsnd (msgid, &message, L*sizeof(char), 0);
  printf ("Data send: %s\n", message.mesg_text);
  return 0;
}
```

Get key

Get the id

Read message from stdin

Send message

Key Generation: Uses ftok to generate a unique key based on the file "progfile" and project ID 65.
Message Queue Creation: Uses msgget to create or access a message queue with the generated key.
Message Type: Sets the message type to 1.
Read and Send Data:
Read Input: Reads input from the user using fgets.
Send Message: Sends the message to the message queue using msgsnd.
Print Confirmation: Prints the sent message to confirm the operation.

Solution

The Reader


```
struct mesg_buffer {
  long mesg_type;
  char mesg_text[L];
} message;

int main() {
  key_t key;
  int msgid;
  key = ftok ("progfile", 65);
  msgid = msgget (key, 0666 | IPC_CREAT);
  msgrcv (msgid, &message, L*sizeof(char), 1, 0);
  printf ("Data received: %s\n", message.mesg_text);
  msgctl (msgid, IPC_RMID, NULL);
  return 0;
}
```

Get key

Get the id

Receive message

Remove the queue from the system