Operating Systems(CSL3030)

Lab Assignment 7 - Report

Soumen Kumar
Roll No-B22ES006

1 Introduction

This report covers the design, implementation, and testing of a shared memory communication program in C. The program consists of two separate processes: a sender (Sender.c) and a receiver (Receiver.c). The inter-process communication (IPC) is facilitated via shared memory, where the sender transmits a string containing multiple words to the receiver. The receiver processes the string by identifying the largest word and sorting the words based on their lengths.

2 Objective

The primary goals of this project are:

- To demonstrate the use of shared memory for inter-process communication.
- To implement a sender program that writes a multi-word string into shared memory.
- To implement a receiver program that reads the string, identifies the largest word, and sorts the words by length.

3 System Design

The program is designed with two main components, each performing a distinct role:

3.1 Sender Process (Sender.c)

The sender program takes a multi-word string input from the user and writes it into a shared memory segment. The design steps are:

- **Key Generation**: A unique key is generated using the `ftok()` function, which both the sender and receiver will use to access the shared memory.
- **Shared Memory Creation**: The `shmget()` function is called to create a shared memory segment. The size of the segment is defined by the constant `SHM_SIZE`.
- **Memory Attachment**: `shmat()` attaches the shared memory segment to the sender's address space, allowing it to write the input string to shared memory.

3.2 Receiver Process (Receiver.c)

The receiver program reads the string from shared memory, identifies the largest word, sorts the words by length, and prints the results. The main steps are:

- **Memory Attachment**: The receiver attaches to the shared memory using `shmat()` and retrieves the string written by the sender.
- **Word Processing**: The string is tokenized into words. Each word is stored with its length and original position to preserve order for sorting.
- **Largest Word Identification**: The program finds the longest word by comparing the lengths of all words.
- **Sorting**: The words are sorted by length (ascending or descending) based on user input. Original order is preserved for words of equal length.
- **Output and Memory Detachment**: The sorted words and largest word are displayed, and `shmdt()` is called to detach from shared memory, followed by `shmctl()` to destroy the segment.

4 Code Explanation

This section provides a detailed explanation of the main functions and operations in the sender and receiver programs.

4.1 Sender Program (Sender.c)

- **Key Generation**: The `ftok()` function generates a key based on a filename and an integer to ensure that both processes can reference the same shared memory.
- **Shared Memory Creation and Attachment**: `shmget()` allocates the memory segment, while `shmat()` attaches it to the process's address space. The user is prompted to enter a string, which is stored in the shared memory.
- **Detachment**: After writing the data, `shmdt()` detaches the shared memory segment from the sender's address space, allowing the receiver to read it.

4.2 Receiver Program (Receiver.c)

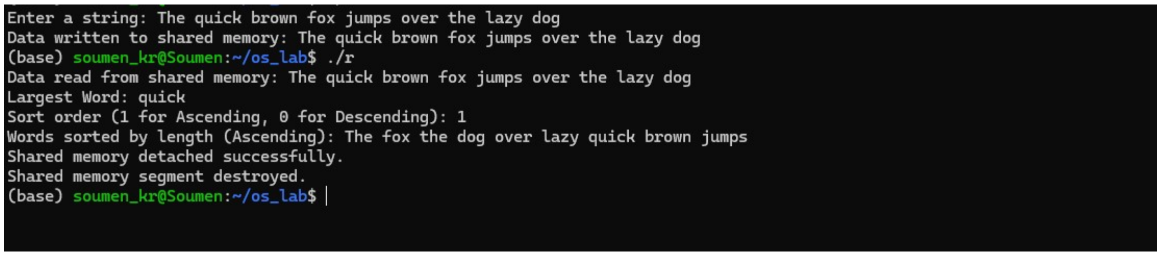
- **Reading from Shared Memory**: The receiver attaches to the shared memory segment created by the sender and reads the string.
- **Tokenizing the String**: `strtok()` splits the string into words, which are stored in an array with their original index for sorting purposes.
- **Finding the Largest Word**: A loop iterates through the array of words to find the one with the maximum length.
- **Sorting Words by Length**: A sorting function orders the words based on their lengths. If two words have equal length, they remain in the original order, as indexed in the array.
- **Output and Cleanup**: The sorted words and the largest word are printed to the console. `shmdt()` and `shmctl()` detach and destroy the shared memory, respectively.

5 Testing and Output

The program was tested with various input strings to confirm correct functionality. Screenshots for each test case output are provided below.

5.1 Test Case 1

- **Input (Sender)**: ``‘‘The quick brown fox jumps over the lazy dog’’``
- **Expected Output (Receiver)**:
  - Largest Word: `jumps`
  - Words sorted by length (Ascending): `The fox the dog lazy over quick brown jumps`

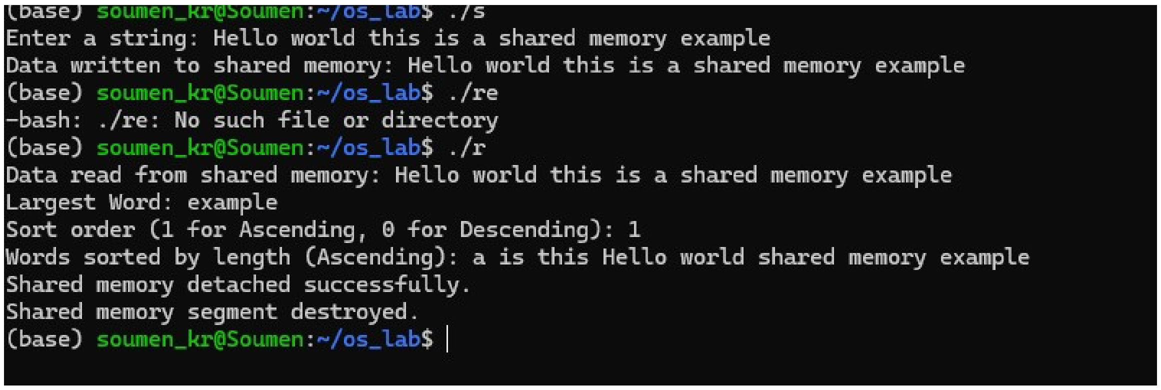


```
Enter a string: The quick brown fox jumps over the lazy dog
Data written to shared memory: The quick brown fox jumps over the lazy dog
(base) soumen_kr@Soumen:~/os_lab$ ./r
Data read from shared memory: The quick brown fox jumps over the lazy dog
Largest Word: quick
Sort order (1 for Ascending, 0 for Descending): 1
Words sorted by length (Ascending): The fox the dog over lazy quick brown jumps
Shared memory detached successfully.
Shared memory segment destroyed.
(base) soumen_kr@Soumen:~/os_lab$ |
```

Figure 1: Screenshot of Test Case 1 Output

5.2 Test Case 2

- **Input (Sender)**: ``‘‘Hello world this is a shared memory example’’``
- **Expected Output (Receiver)**:
  - Largest Word: `example`
  - Words sorted by length (Ascending): `a is this Hello world memory shared example`

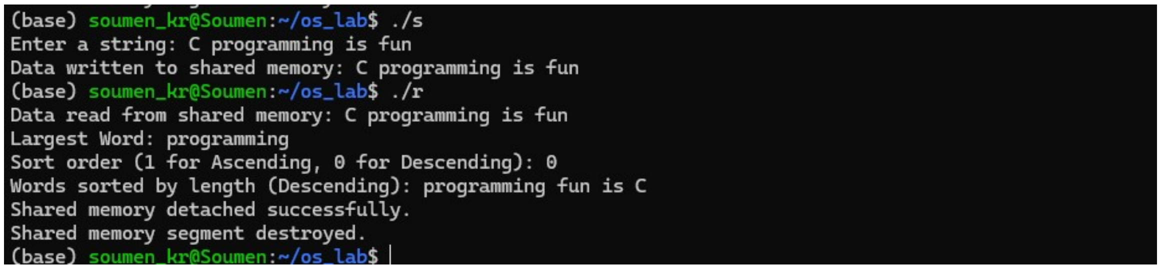


```
(base) soumen_kr@Soumen:~/os_lab$ ./s
Enter a string: Hello world this is a shared memory example
Data written to shared memory: Hello world this is a shared memory example
(base) soumen_kr@Soumen:~/os_lab$ ./re
-bash: ./re: No such file or directory
(base) soumen_kr@Soumen:~/os_lab$ ./r
Data read from shared memory: Hello world this is a shared memory example
Largest Word: example
Sort order (1 for Ascending, 0 for Descending): 1
Words sorted by length (Ascending): a is this Hello world shared memory example
Shared memory detached successfully.
Shared memory segment destroyed.
(base) soumen_kr@Soumen:~/os_lab$ |
```

Figure 2: Screenshot of Test Case 1 Output

5.3 Test Case 3

- **Input (Sender)**: ``C programming is fun''
- **Expected Output (Receiver)**:
  - Largest Word: `programming`
  - Words sorted by length (Descending): `programming fun C is`



```
(base) soumen_kr@Soumen:~/os_lab$ ./s
Enter a string: C programming is fun
Data written to shared memory: C programming is fun
(base) soumen_kr@Soumen:~/os_lab$ ./r
Data read from shared memory: C programming is fun
Largest Word: programming
Sort order (1 for Ascending, 0 for Descending): 0
Words sorted by length (Descending): programming fun is C
Shared memory detached successfully.
Shared memory segment destroyed.
(base) soumen_kr@Soumen:~/os_lab$ |
```

Figure 3: Screenshot of Test Case 1 Output

6 Conclusion

This project demonstrates the efficiency of using shared memory as an IPC mechanism. The sender and receiver processes successfully communicate through shared memory, allowing the receiver to process and sort the words based on length. This approach demonstrates the advantage of shared memory in IPC, offering efficient data sharing without complex setup.

7 References

- Advanced Programming in the UNIX Environment by W. Richard Stevens and Stephen A. Rago.
- C Programming Language by Brian W. Kernighan and Dennis M. Ritchie.
- Linux man pages for `shmget`, `shmat`, `shmdt`, and `ftok`.