

ECS 408/608: Operating System

Assignment: Interprocess Communication

February 16, 2025

Rohan Mehra
(Roll No: 21224)

1. Write a C program using IPC-Shared Memory that performs the following two tasks using the producer-consumer paradigm:
 - (a) Producer (producer.c): Ask the user for an integer and string and place them in shared memory.
 - (b) Consumer (consumer.c): Print the string input in the producer and check if the input number is even or not.

Answer:

System Used

- **Operating System:** Windows
- **Compiler:** MinGW (GCC for Windows)

Solution

The problem involves implementing a producer-consumer paradigm using shared memory. Since Windows does not support POSIX shared memory, the solution uses the **Windows API**. Below are the adapted programs for Windows.

Producer Program (producer.c)

```
#include <stdio.h>
#include <windows.h>
#define SHM_SIZE 1024
typedef struct { int number; char text[100]; } SharedData;
```

```

int main() {
    HANDLE hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
        PAGE_READWRITE, 0, SHM_SIZE, "Local\\MySharedMemory");
    if (hMapFile == NULL) {
        printf("Could not create file mapping object (%lu).\n", GetLastError());
        return 1;
    }
    SharedData *data = (SharedData *)MapViewOfFile(hMapFile,
        FILE_MAP_ALL_ACCESS, 0, 0, SHM_SIZE);
    if (data == NULL) {
        printf("Could not map view of file (%lu).\n", GetLastError());
        CloseHandle(hMapFile); return 1;
    }
    printf("Enter an integer: "); scanf("%d", &data->number);
    printf("Enter a string: "); scanf("%s", data->text);
    printf("Data written to shared memory: Number = %d, Text = %s\n",
        data->number, data->text);
    printf("Press Enter to exit the producer...\n"); getchar(); getchar();
    UnmapViewOfFile(data); CloseHandle(hMapFile); return 0;
}

```

Consumer Program (consumer.c)

```

#include <stdio.h>
#include <windows.h>
#define SHM_SIZE 1024
typedef struct { int number; char text[100]; } SharedData;
int main() {
    HANDLE hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE,
        "Local\\MySharedMemory");
    if (hMapFile == NULL) {
        printf("Could not open file mapping object (%lu).\n", GetLastError());
        return 1;
    }
    SharedData *data = (SharedData *)MapViewOfFile(hMapFile,
        FILE_MAP_ALL_ACCESS, 0, 0, SHM_SIZE);
    if (data == NULL) {
        printf("Could not map view of file (%lu).\n", GetLastError());
        CloseHandle(hMapFile); return 1;
    }
    printf("Data read from shared memory: Number = %d, Text = %s\n",
        data->number, data->text);
    if (data->number % 2 == 0) printf("The number %d is even.\n", data->number);
    else printf("The number %d is odd.\n", data->number);
    UnmapViewOfFile(data); CloseHandle(hMapFile); return 0;
}

```

Explanation

- The producer creates shared memory using `CreateFileMapping` and writes user input to it.
- The consumer accesses the shared memory using `OpenFileMapping`, reads the data, and checks if the number is even or odd.

Screenshots

The top screenshot shows the producer program (producer.c) running in a Command Prompt. The user enters the integer 45 and the string 'Rohan'. The program outputs 'Data written to shared memory: Number = 45, Text = Rohan' and 'Press Enter to exit the producer...'. The bottom screenshot shows the consumer program (consumer.c) running in a Command Prompt. The program reads the data from shared memory and outputs 'Data read from shared memory: Number = 45, Text = Rohan' and 'The number 45 is odd.'.

2. Write a C program using IPC-Shared Memory that performs the following task using the producer-consumer paradigm:

(a) Producer (producer.c): Generate 100 random numbers and place them in shared

memory.

- (b) Consumer (consumer.c): Read all the 100 numbers and generate the cumulative sum and exit.

Answer:

System Used

- **Operating System:** Windows
- **Compiler:** MinGW (GCC for Windows)

Solution

The problem involves implementing a producer-consumer paradigm using shared memory. The producer generates 100 random numbers and stores them in shared memory, while the consumer reads the numbers, computes their cumulative sum, and prints the result.

Producer Program (producer.c)

```
#include <stdio.h>
#include <windows.h>
#include <time.h>

#define SHM_SIZE 1024 // Size of shared memory
#define NUM_COUNT 100 // Number of random numbers to generate

int main() {
    // Create a shared memory file mapping
    HANDLE hMapFile = CreateFileMapping(
        INVALID_HANDLE_VALUE, // Use the paging file
        NULL,                 // Default security
        PAGE_READWRITE,       // Read/write access
        0,                    // Maximum object size (high-order
        red↵ DWORD)
        SHM_SIZE,             // Maximum object size (low-order
        red↵ DWORD)
        "Local\\MySharedMemory"); // Name of mapping object

    if (hMapFile == NULL) {
```

```

        printf("Could not create file mapping object (%lu).\n",
               red↵ GetLastError());
        return 1;
    }

    // Map the shared memory into the process's address space
    int *data = (int *)MapViewOfFile(
        hMapFile,           // Handle to map object
        FILE_MAP_ALL_ACCESS, // Read/write permission
        0,
        0,
        SHM_SIZE);

    if (data == NULL) {
        printf("Could not map view of file (%lu).\n",
               red↵ GetLastError());
        CloseHandle(hMapFile);
        return 1;
    }

    // Generate 100 random numbers and store them in shared
    red↵ memory
    srand(time(NULL)); // Seed the random number generator
    for (int i = 0; i < NUM_COUNT; i++) {
        data[i] = rand() % 100; // Random number between 0 and
        red↵ 99
    }

    printf("Producer: Generated 100 random numbers and stored
           red↵ them in shared memory.\n");

    // Wait for user input before exiting
    printf("Press Enter to exit the producer...\n");
    getchar(); // Wait for Enter key

    // Unmap the shared memory and close the handle
    UnmapViewOfFile(data);
    CloseHandle(hMapFile);

    return 0;
}

```

Consumer Program (consumer.c)

```

#include <stdio.h>
#include <windows.h>

```

```
#define SHM_SIZE 1024 // Size of shared memory
#define NUM_COUNT 100 // Number of random numbers to read

int main() {
    // Open the shared memory file mapping
    HANDLE hMapFile = OpenFileMapping(
        FILE_MAP_ALL_ACCESS, // Read/write access
        FALSE,               // Do not inherit the name
        "Local\\MySharedMemory"); // Name of mapping object

    if (hMapFile == NULL) {
        printf("Could not open file mapping object (%lu).\n",
            red↵ GetLastError());
        return 1;
    }

    // Map the shared memory into the process's address space
    int *data = (int *)MapViewOfFile(
        hMapFile, // Handle to map object
        FILE_MAP_ALL_ACCESS, // Read/write permission
        0,
        0,
        SHM_SIZE);

    if (data == NULL) {
        printf("Could not map view of file (%lu).\n",
            red↵ GetLastError());
        CloseHandle(hMapFile);
        return 1;
    }

    // Compute the cumulative sum of the 100 numbers
    int sum = 0;
    for (int i = 0; i < NUM_COUNT; i++) {
        sum += data[i];
    }

    printf("Consumer: The cumulative sum of the 100 numbers is %d
        red↵ .\n", sum);

    // Unmap the shared memory and close the handle
    UnmapViewOfFile(data);
    CloseHandle(hMapFile);

    return 0;
}
```

- The producer creates shared memory using `CreateFileMapping` and writes 100 random numbers to it.
- The consumer accesses the shared memory using `OpenFileMapping`, reads the numbers, and computes their cumulative sum.

The screenshot shows a Windows 10 desktop with two Visual Studio Code windows open. The top window, titled 'Command Prompt - producer', shows the execution of 'producer.c'. The output indicates that 100 random numbers were generated and stored in shared memory. The bottom window, titled 'Command Prompt - consumer', shows the execution of 'consumer.c'. The output indicates that the cumulative sum of the 100 numbers is 4890. The Windows taskbar at the bottom shows the date and time as 17:02:20 on 03/04/2020.

Visual Studio Code - producer.c

```

1 // Producer
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define SIZE 100
5 #define NAME "producer"
6
7 int main()
8 {
9     // Open shared memory
10    FILE *f;
11    f = fopen("shared.dat", "w");
12    if (f == NULL)
13    {
14        printf("Error opening file\n");
15        return 1;
16    }
17
18    // Map shared memory
19    int *data;
20    data = (int *)mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, 0, 0);
21    if (data == MAP_FAILED)
22    {
23        printf("Error mapping shared memory\n");
24        return 1;
25    }
26
27    if (data)
28    {
29        printf("Data is mapped\n");
30        return 0;
31    }
32
33    // Comp
34    int sum = 0;
35    for (int i = 0; i < NAME_COUNT; i++) {
36        sum += data[i];
37    }
38    printf("Sum is %d\n", sum);
39    return 0;
40 }

```

Visual Studio Code - consumer.c

```

1 // Consumer
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define SIZE 100
5 #define NAME "consumer"
6
7 int main()
8 {
9     // Open shared memory
10    FILE *f;
11    f = fopen("shared.dat", "r");
12    if (f == NULL)
13    {
14        printf("Error opening file\n");
15        return 1;
16    }
17
18    // Map shared memory
19    int *data;
20    data = (int *)mmap(NULL, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, 0, 0);
21    if (data == MAP_FAILED)
22    {
23        printf("Error mapping shared memory\n");
24        return 1;
25    }
26
27    if (data)
28    {
29        printf("Data is mapped\n");
30        return 0;
31    }
32
33    // Comp
34    int sum = 0;
35    for (int i = 0; i < NAME_COUNT; i++) {
36        sum += data[i];
37    }
38    printf("Sum is %d\n", sum);
39    return 0;
40 }

```

- Page 7

- (c) Create 3 consumer processes where:
- (i) Consumer 1 computes the factorial of the first four numbers.
 - (ii) Consumer 2 computes the factorial of the next two numbers.
 - (iii) Consumer 3 computes the factorial of the remaining numbers.

Answer:**System Used**

- **Operating System:** Windows
- **Compiler:** MinGW (GCC for Windows)

Solution

The problem involves using shared memory to store an array of numbers. The parent process writes 10 numbers into shared memory, and 3 consumer processes compute the factorial of their assigned numbers. The parent process then collects and prints the results.

Parent Program (parent.c)

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define SHARED_MEMORY_SIZE 1024
#define ARRAY_SIZE 10
#define SHARED_MEMORY_NAME "Global\\SharedMemoryExample"

// Function to create and run a consumer process
void createConsumerProcess(int start, int count) {
    char command[100];
    snprintf(command, sizeof(command), "consumer.exe %d %d",
             start, count);

    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
```



```

    if (!CreateProcess(NULL, command, NULL, NULL, FALSE, 0, NULL,
        red↵ NULL, &si, &pi)) {
        printf("Parent: Failed to create consumer process for
            red↵ range [%d to %d].\n", start, start + count - 1);
        return;
    }

    // Wait for the process to complete
    WaitForSingleObject(pi.hProcess, INFINITE);

    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}

int main() {
    // Create shared memory
    HANDLE hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE,
        red↵ NULL, PAGE_READWRITE, 0, SHARED_MEMORY_SIZE,
        red↵ SHARED_MEMORY_NAME);
    if (hMapFile == NULL) {
        printf("Parent: Could not create file mapping. Error: %d\
            red↵ n", GetLastError());
        return 1;
    }

    int *data = (int *)MapViewOfFile(hMapFile,
        red↵ FILE_MAP_ALL_ACCESS, 0, 0, SHARED_MEMORY_SIZE);
    if (data == NULL) {
        printf("Parent: Could not map view of file. Error: %d\n",
            red↵ GetLastError());
        CloseHandle(hMapFile);
        return 1;
    }

    // Store 10 numbers in shared memory
    int numbers[ARRAY_SIZE] = {5, 4, 3, 2, 7, 6, 8, 9, 10, 12};
    for (int i = 0; i < ARRAY_SIZE; i++)
        data[i] = numbers[i];

    printf("Parent: Stored numbers in shared memory.\n");

    // Create 3 consumer processes
    createConsumerProcess(0, 4); // Consumer 1: First 4 numbers
    createConsumerProcess(4, 2); // Consumer 2: Next 2 numbers
    createConsumerProcess(6, 4); // Consumer 3: Last 4 numbers

    // Cleanup
    UnmapViewOfFile(data);
}

```

```

        CloseHandle(hMapFile);

        printf("Parent: All consumer processes finished execution.\n
               red↵ ");
        return 0;
    }

```

Consumer Program (consumer.c)

```

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define SHARED_MEMORY_SIZE 1024
#define SHARED_MEMORY_NAME "Global\\SharedMemoryExample"

// Function to compute factorial
unsigned long long factorial(int num) {
    if (num == 0 || num == 1) return 1;
    unsigned long long result = 1;
    for (int i = 2; i <= num; i++)
        result *= i;
    return result;
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: consumer.exe <start_index> <count>\n");
        return 1;
    }

    int start = atoi(argv[1]);
    int count = atoi(argv[2]);

    // Open shared memory
    HANDLE hMapFile = OpenFileMapping(FILE_MAP_READ, FALSE,
        red↵ SHARED_MEMORY_NAME);
    if (hMapFile == NULL) {
        printf("Consumer: Could not open file mapping. Error: %d\
               red↵ n", GetLastError());
        return 1;
    }

    int *data = (int *)MapViewOfFile(hMapFile, FILE_MAP_READ, 0,
        red↵ 0, SHARED_MEMORY_SIZE);
    if (data == NULL) {

```

```

        printf("Consumer: Could not map view of file. Error: %d\n",
               GetLastError());
        CloseHandle(hMapFile);
        return 1;
    }

    // Compute factorials for assigned numbers
    printf("Consumer handling indexes [%d to %d]:\n", start,
           red↵ start + count - 1);
    for (int i = start; i < start + count; i++) {
        printf("Factorial of %d is %llu\n", data[i], factorial(
            red↵ data[i]));
    }

    // Cleanup
    UnmapViewOfFile(data);
    CloseHandle(hMapFile);

    return 0;
}

```

Screenshots

