# Assignment#3

# Task

Write a C program where the parent process sends several numbers to its two child processes using shared memory. The child processes receive these numbers, compute their sum, and then send the result back to the parent process through the same shared memory. Subsequently, the parent process displays the final sum. For reliable communication, you must implement the bounded buffer (i.e., given in the textbook) on shared memory. You need to use the semaphore library for synchronization. The numbers that are to be passed to the child processes must be passed to the parent process as command-line arguments. Example: $ ./a.out 1 2 3 4 5 6 7 8 9 0 Sum one is: 45

## Solution Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <sys/wait.h>

#include <string.h>

#define BUFFER\_SIZE 10

// Shared data

int sharedBuffer[BUFFER\_SIZE];

int inIndex = 0;

int outIndex = 0;

int totalSum = 0;

// Semaphore declarations

sem\_t emptySem, fullSem, mutexSem;

void produceItem(int num) {

sem\_wait(&emptySem);

sem\_wait(&mutexSem);

sharedBuffer[inIndex] = num;

inIndex = (inIndex + 1) % BUFFER\_SIZE;

sem\_post(&mutexSem);

sem\_post(&fullSem);

}

int consumeItem() {

sem\_wait(&fullSem);

sem\_wait(&mutexSem);

int num = sharedBuffer[outIndex];

outIndex = (outIndex + 1) % BUFFER\_SIZE;

sem\_post(&mutexSem);

sem\_post(&emptySem);

return num;

}

int main(int argc, char \*argv[]) {

if (argc < 2) {

fprintf(stderr, "Usage: %s num1 num2 ... numN\n", argv[0]);

exit(EXIT\_FAILURE);

}

// Create shared memory

key\_t key = ftok("sharedMemoryKey", 65);

int sharedMemoryID = shmget(key, sizeof(int) \* (BUFFER\_SIZE + 2), 0666 | IPC\_CREAT);

if (sharedMemoryID == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

int \*sharedMemoryData = (int \*)shmat(sharedMemoryID, NULL, 0);

if (sharedMemoryData == (int \*)-1) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Initialize shared data

memset(sharedMemoryData, 0, sizeof(int) \* (BUFFER\_SIZE + 2));

// Initialize semaphores

if (sem\_init(&emptySem, 1, BUFFER\_SIZE) == -1 || sem\_init(&fullSem, 1, 0) == -1 || sem\_init(&mutexSem, 1, 1) == -1) {

perror("sem\_init");

exit(EXIT\_FAILURE);

}

// Fork two child processes

pid\_t firstChild = fork();

if (firstChild == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (firstChild == 0) { // First child process

for (int i = 1; i < argc; ++i) {

int num = atoi(argv[i]);

produceItem(num);

}

exit(EXIT\_SUCCESS);

} else {

pid\_t secondChild = fork();

if (secondChild == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (secondChild == 0) { // Second child process

for (int i = 1; i < argc; ++i) {

int num = atoi(argv[i]);

produceItem(num);

}

exit(EXIT\_SUCCESS);

} else { // Parent process

wait(NULL);

wait(NULL);

// Consume data from the buffer and calculate the sum

for (int i = 1; i < argc; ++i) {

int num = consumeItem();

totalSum += num;

}

// Display the final sum

printf("Total Sum: %d\n", totalSum);

// Detach and remove shared memory

if (shmdt(sharedMemoryData) == -1) {

perror("shmdt");

exit(EXIT\_FAILURE);

}

if (shmctl(sharedMemoryID, IPC\_RMID, NULL) == -1) {

perror("shmctl");

exit(EXIT\_FAILURE);

}

// Close and unlink semaphores

if (sem\_destroy(&emptySem) == -1 || sem\_destroy(&fullSem) == -1 || sem\_destroy(&mutexSem) == -1) {

perror("sem\_destroy");

exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

}

}