

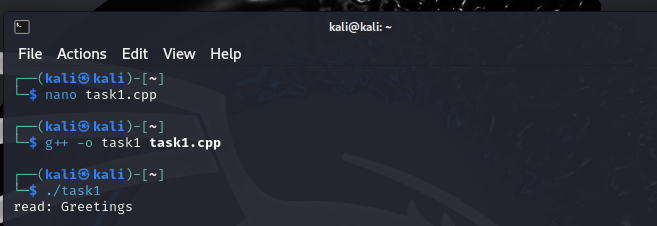
Operating

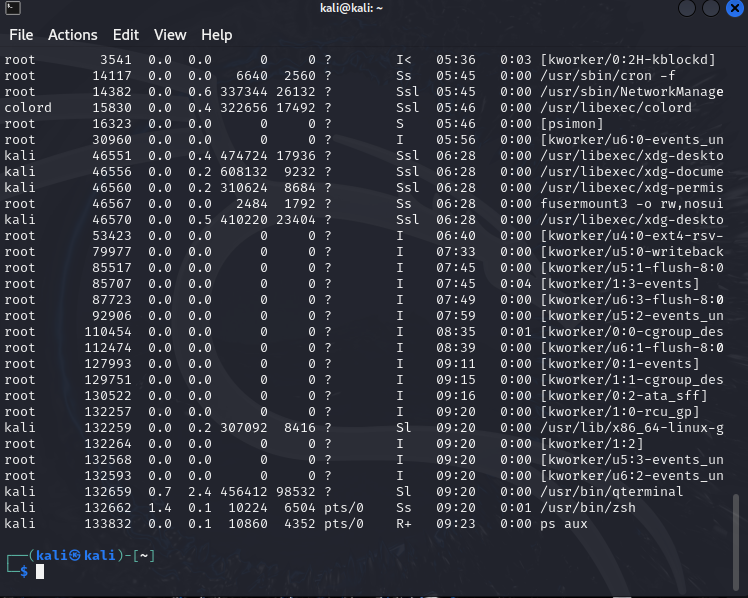
System

LAB - 06

* Faculty name: Sir Usama bin Umar
* Student ID: k224779
* Student name: Muhammad Bilal
* Section: BCYS-4A

**TASK # 1**





Modified Code:

#include <sys/types.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#define BUFFER\_SIZE 25

#define READ\_END 0

#define WRITE\_END 1

int main(void) {

char write\_msg[BUFFER\_SIZE];

char read\_msg[BUFFER\_SIZE];

int fd[2];

pid\_t pid;

printf("Enter a message to send: ");

fgets(write\_msg, BUFFER\_SIZE, stdin); // Read message from user

if (pipe(fd) == -1) { // create a pipe

fprintf(stderr, "Pipe failed");

return 1;

}

pid = fork();

if (pid < 0) { /\* error occurred \*/

fprintf(stderr, "Fork Failed");

return 1;

}

if (pid > 0) { /\* parent process \*/

close(fd[READ\_END]); // close the unused end of the pipe

write(fd[WRITE\_END], write\_msg, strlen(write\_msg) + 1);

close(fd[WRITE\_END]);

} else { /\* child process \*/

close(fd[WRITE\_END]); // close the unused end of the pipe

read(fd[READ\_END], read\_msg, BUFFER\_SIZE); // read from the pipe

printf("read: %s\n", read\_msg);

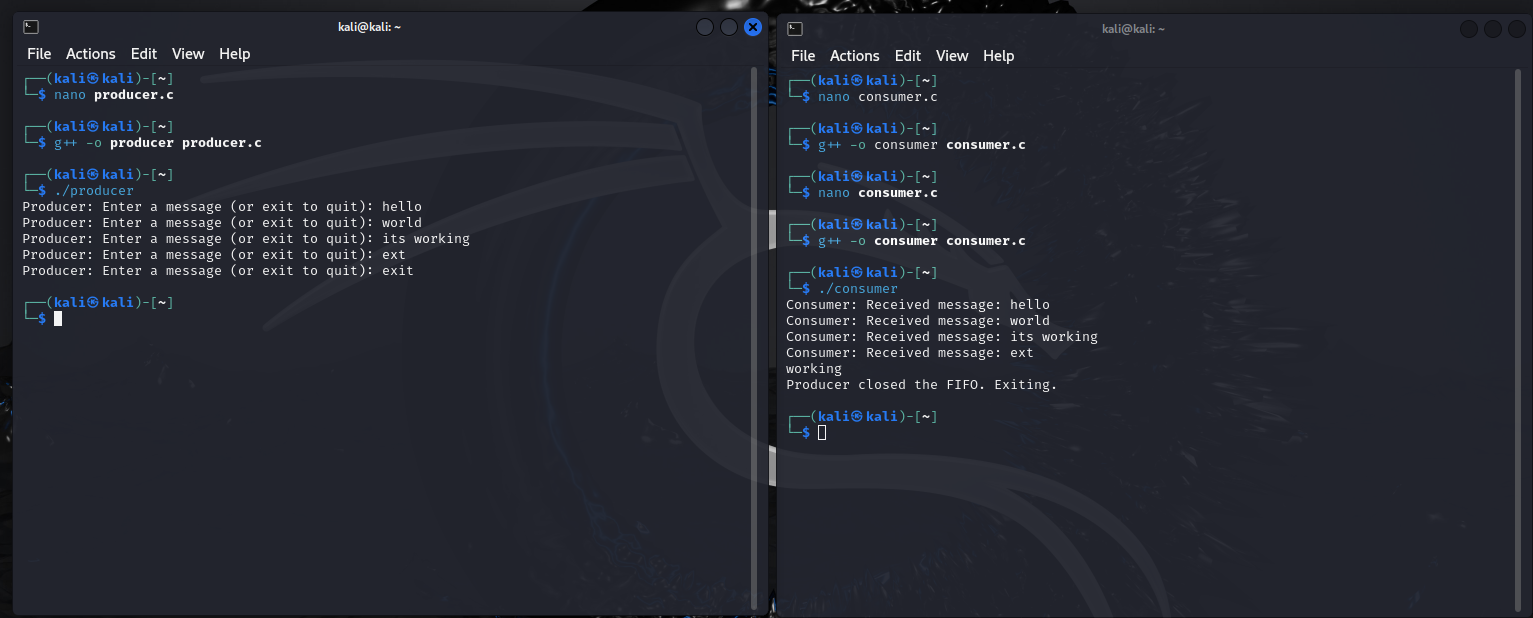
close(fd[READ\_END]); // close the read end of the pipe

}

return 0;

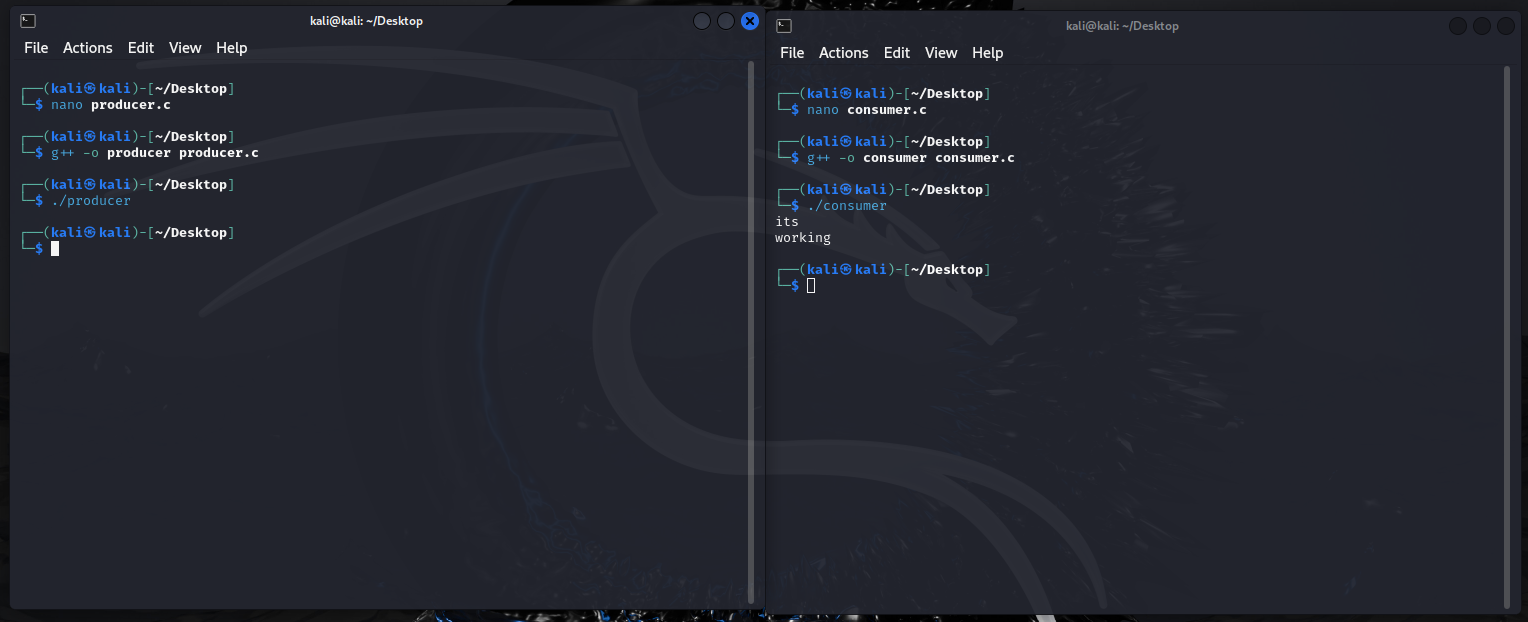
}

**TASK # 2**



|  |  |
| --- | --- |
| Producer code:  #include <sys/types.h>  #include <stdlib.h>  #include <sys/stat.h>  #include <stdio.h>  #include <unistd.h>  #include <fcntl.h>  #include <string.h>  #define FIFO\_FILE "/home/kali/fifofile.txt"  #define BUFSIZ 8192  int main() {  int fd;  char buffer[BUFSIZ];  ssize\_t num\_bytes;  mkfifo(FIFO\_FILE, 0666); // Corrected mode parameter  fd = open(FIFO\_FILE, O\_WRONLY);  if (fd == -1) {  perror("open");  exit(EXIT\_FAILURE);  }  while (1) {  printf("Producer: Enter a message (or exit to quit): ");  if (fgets(buffer, BUFSIZ, stdin) == NULL) {  perror("fgets");  exit(EXIT\_FAILURE);  }  // Check if the entered string includes a newline character  if (strncmp(buffer, "exit", 4) == 0) {  break;  }  num\_bytes = write(fd, buffer, strlen(buffer));  if (num\_bytes == -1) {  perror("write");  exit(EXIT\_FAILURE);  }  }  close(fd);  return 0;  } | Consumer code:  #include <sys/types.h>  #include <stdlib.h>  #include <sys/stat.h>  #include <stdio.h>  #include <unistd.h>  #include <fcntl.h>  #include <string.h>  #define FIFO\_FILE "/home/kali/Desktop/k224792/fifofile.txt"  #define BUFSIZ 8192  int main() {  int fd;  char buffer[BUFSIZ];  ssize\_t num\_bytes;  fd = open(FIFO\_FILE, O\_RDONLY);  if (fd == -1) {  perror("open");  exit(EXIT\_FAILURE);  }  while (1) {  num\_bytes = read(fd, buffer, BUFSIZ);  if (num\_bytes == -1) {  perror("read");  exit(EXIT\_FAILURE);  } else if (num\_bytes == 0) {  printf("Producer closed the FIFO. Exiting.\n");  break;  }  printf("Consumer: Received message: %s", buffer);  }  close(fd);  return 0;  } |

**TASK # 3**

****

|  |  |
| --- | --- |
| Producer code:  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <fcntl.h>  #include <sys/shm.h>  #include <sys/stat.h>  #include <sys/mman.h>  #include <unistd.h>  int main()  {  const int SIZE = 4096;// the size (in bytes) of shared memory objec  const char \*name = "OS"; // name of the shared memory object  const char \*message\_0 = "its "; // strings written to shared memory  const char \*message\_1 = "\nworking"; // shared memory file descriptor  int fd;  char \*ptr; // pointer to shared memory obect  fd = shm\_open(name, O\_CREAT | O\_RDWR, 0666); // create the shared memory objec  ftruncate(fd, SIZE); // configure the size of the shared memory object  // memory map the shared memory object  ptr = (char\*)mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  // write to the shared memory object  sprintf(ptr, "%s", message\_0);  ptr += strlen(message\_0);  sprintf(ptr, "%s", message\_1);  ptr += strlen(message\_1);  return 0;  } | Consumer code:  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/shm.h>  #include <sys/stat.h>  #include <sys/mman.h>  #include <unistd.h> // Include unistd.h for close function  #include <sys/types.h> // Include sys/types.h for data types  int main() {  const int SIZE = 4096;  const char \*name = "OS";  int fd;  char \*ptr;    // Open shared memory with read-only access  fd = shm\_open(name, O\_RDWR, 0666);  if (fd == -1) {  perror("shm\_open");  exit(EXIT\_FAILURE);  }    // Map the shared memory  ptr = (char \*)mmap(0, SIZE, PROT\_READ, MAP\_SHARED, fd, 0);  if (ptr == MAP\_FAILED) {  perror("mmap");  exit(EXIT\_FAILURE);  }    // Read from shared memory  printf("%s", ptr);  // Unmap the shared memory  if (munmap(ptr, SIZE) == -1) {  perror("munmap");  }    // Close the shared memory file descriptor  if (close(fd) == -1) {  perror("close");  }  return 0;  } |

**Runtime error:**

The issue was caused because the code did not unmap the memory before termination.