**TD 4**

**Exercise 1:**

Write a program that facilitates communication between a parent and child process using pipes. The parent process should use fork to create the child process. After the child is created, the parent process should wait for the user to enter an integer value. The parent then sends this integer value to the child process. Upon receiving the value, the child process should calculate the square of the integer and return the result to the parent process. The parent process should then print the squared value.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/wait.h>  int main() {  int pipe1[2]; // Parent to child  int pipe2[2]; // Child to parent  // Create the pipes  if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {  perror("pipe");  exit(EXIT\_FAILURE);  }  // Fork the process  pid\_t pid = fork();  if (pid < 0) {  // Fork failed  perror("fork");  exit(EXIT\_FAILURE);  } else if (pid > 0) {  // This is the parent process  close(pipe1[0]); // Close the read end of pipe1  close(pipe2[1]); // Close the write end of pipe2  int value;  printf("Enter an integer: ");  scanf("%d", &value);  // Write the value to the pipe1  write(pipe1[1], &value, sizeof(int));  close(pipe1[1]); // Close the write end of pipe1 after writing  // Wait for the child process to finish  wait(NULL);  int result;  // Read the squared result from pipe2  read(pipe2[0], &result, sizeof(int));  close(pipe2[0]); // Close the read end of pipe2 after reading  printf("Squared value received from child: %d\n", result);  } else {  // This is the child process  close(pipe1[1]); // Close the write end of pipe1  close(pipe2[0]); // Close the read end of pipe2  int value;  // Read the value from pipe1  read(pipe1[0], &value, sizeof(int));  close(pipe1[0]); // Close the read end of pipe1 after reading  // Calculate the square of the value  int result = value \* value;  // Write the result to pipe2  write(pipe2[1], &result, sizeof(int));  close(pipe2[1]); // Close the write end of pipe2 after writing  // Exit the child process  exit(EXIT\_SUCCESS);  }  return 0;  } |

**Exercise 2:**

Write two programs that communicate between two processes using named pipes (FIFOs). Process 1 should wait for the user to enter an integer value. Process 1 then sends this integer value to Process 2 using a named pipe. Upon receiving the value, Process 2 should calculate the square of the integer and return the result to Process 1 through another named pipe. Process 1 then prints the squared value.

Process 1

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/stat.h>  #include <unistd.h>  #define FIFO1 "/tmp/fifo1"  #define FIFO2 "/tmp/fifo2"  int main() {  int num, squared;  // Create the FIFOs  mkfifo(FIFO1, 0666);  mkfifo(FIFO2, 0666);  // Get an integer from the user  printf("Enter an integer: ");  scanf("%d", &num);  // Write the integer to FIFO1  int fd1 = open(FIFO1, O\_WRONLY);  write(fd1, &num, sizeof(num));  close(fd1);  // Read the squared result from FIFO2  int fd2 = open(FIFO2, O\_RDONLY);  read(fd2, &squared, sizeof(squared));  close(fd2);  // Print the squared value  printf("Squared value received: %d\n", squared);  // Remove the FIFOs  unlink(FIFO1);  unlink(FIFO2);  return 0;  } |

Process 2:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/stat.h>  #include <unistd.h>  #define FIFO1 "/tmp/fifo1"  #define FIFO2 "/tmp/fifo2"  int main() {  int num, squared;  // Read the integer from FIFO1  int fd1 = open(FIFO1, O\_RDONLY);  read(fd1, &num, sizeof(num));  close(fd1);  // Calculate the square of the integer  squared = num \* num;  // Write the squared result to FIFO2  int fd2 = open(FIFO2, O\_WRONLY);  write(fd2, &squared, sizeof(squared));  close(fd2);  return 0;  } |

**Exercise 3:**

Write a C project (ie two C programs *using named pipes FIFO* to implement the following operations).

The first *named "simulator.c" offers the following service to a second program “Entry.c"*:

The first reads two integers, and a character “C” (C= “S” or “R”) from the second; (S designates a square form and R designates a rectangle form). It calculates the surface and sends back the result to the second.

An “invalid form” message will be sent as a result in case of any other value of C.

The second receives the result and displays it.

**Program 1: simulator.c**

|  |
| --- |
| // simulator.c  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/stat.h>  #include <unistd.h>  #include <string.h>  #define FIFO1 "/tmp/fifo1"  #define FIFO2 "/tmp/fifo2"  int main() {  int a, b;  char C;  char result[256];  // Create the FIFOs  mkfifo(FIFO1, 0666);  mkfifo(FIFO2, 0666);  // Read the inputs from FIFO1  int fd1 = open(FIFO1, O\_RDONLY);  read(fd1, &a, sizeof(a));  read(fd1, &b, sizeof(b));  read(fd1, &C, sizeof(C));  close(fd1);  // Calculate the surface based on the form type  if (C == 'S') {  int surface = a \* a;  snprintf(result, sizeof(result), "Surface of square: %d", surface);  } else if (C == 'R') {  int surface = a \* b;  snprintf(result, sizeof(result), "Surface of rectangle: %d", surface);  } else {  snprintf(result, sizeof(result), "Invalid form");  }  // Write the result to FIFO2  int fd2 = open(FIFO2, O\_WRONLY);  write(fd2, result, strlen(result) + 1);  close(fd2);  // Remove the FIFOs  unlink(FIFO1);  unlink(FIFO2);  return 0;  } |

**Program 2: Entry.c**

|  |
| --- |
| // Entry.c  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/stat.h>  #include <unistd.h>  #define FIFO1 "/tmp/fifo1"  #define FIFO2 "/tmp/fifo2"  int main() {  int a, b;  char C;  char result[256];  // Create the FIFOs  mkfifo(FIFO1, 0666);  mkfifo(FIFO2, 0666);  // Get inputs from the user  printf("Enter two integers: ");  scanf("%d %d", &a);  scanf("%d %d", &b);  printf("Enter the form type (S for square, R for rectangle): ");  scanf(" %c", &C);  // Write the inputs to FIFO1  int fd1 = open(FIFO1, O\_WRONLY);  write(fd1, &a, sizeof(a));  write(fd1, &b, sizeof(b));  write(fd1, &C, sizeof(C));  close(fd1);  // Read the result from FIFO2  int fd2 = open(FIFO2, O\_RDONLY);  read(fd2, result, sizeof(result));  close(fd2);  // Print the result  printf("Received result: %s\n", result);  // Remove the FIFOs  unlink(FIFO1);  unlink(FIFO2);  return 0;  } |

**Exercise 5: Shared Memory**

Write two C programs that allow communication between two processes by using shared memories as follows:

* The first called “writer” does the following tasks:
* Create the memory area
* Attach the area to the process
* Read a message from the keyboard
* Write this message in the shared memory area
* Ask to re-enter a new message
* When it enters “end” it asks to release the memory content
* The second called “reader” reads the message in the memory area every 5 seconds and displays its content on the screen. When it reads “end”, it closes the session.

**writer.c**

|  |
| --- |
| // writer.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <unistd.h>  #define SHM\_KEY 12345  #define SHM\_SIZE 1024  int main() {  int shmid;  char \*shmaddr;  char message[SHM\_SIZE];  // Create the shared memory segment  shmid = shmget(SHM\_KEY, SHM\_SIZE, IPC\_CREAT | 0666);  if (shmid < 0) {  perror("shmget");  exit(1);  }  // Attach the shared memory segment to the process's address space  shmaddr = (char \*)shmat(shmid, NULL, 0);  if (shmaddr == (char \*)-1) {  perror("shmat");  exit(1);  }  // Loop to read messages from the keyboard and write them to shared memory  while (1) {  printf("Enter a message: ");  fgets(message, SHM\_SIZE, stdin);  message[strcspn(message, "\n")] = '\0'; // Remove newline character  strcpy(shmaddr, message);  if (strcmp(message, "end") == 0) {  break;  }  }  // Detach from the shared memory segment  shmdt(shmaddr);  // Mark the shared memory segment for deletion  shmctl(shmid, IPC\_RMID, NULL);  return 0;  } |

Reader.c

|  |
| --- |
| // reader.c  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <unistd.h>  #define SHM\_KEY 12345  #define SHM\_SIZE 1024  int main() {  int shmid;  char \*shmaddr;  // Get the shared memory segment  shmid = shmget(SHM\_KEY, SHM\_SIZE, 0666);  if (shmid < 0) {  perror("shmget");  exit(1);  }  // Attach the shared memory segment to the process's address space  shmaddr = (char \*)shmat(shmid, NULL, 0);  if (shmaddr == (char \*)-1) {  perror("shmat");  exit(1);  }  // Loop to read and display the message from shared memory every 5 seconds  while (1) {  printf("Message from shared memory: %s\n", shmaddr);  if (strcmp(shmaddr, "end") == 0) {  break;  }  sleep(5);  }  // Detach from the shared memory segment  shmdt(shmaddr);  return 0;  } |