122CS0067

Assignment-IV

Name: **Amiya Chowdhury** Roll: **122CS0067** Date: **02/09/2024**

1. You have an existing program, “input.c,” which prints a welcome message for the currently logged-in user and prints the date and time. The compiled file of this program is “input”. Write another program that calls the compiled file “input” and prints the data entered by the user.

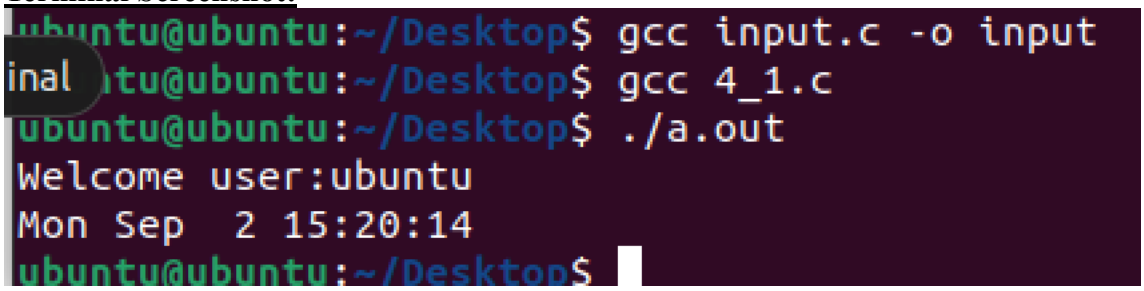
4_1.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
int main()
{
	FILE *rd;
	char buffer[50];
	rd=popen("./input","r"); //pipe opened in reading mode
	int i=fread(buffer, 1, 40, rd); //will read only 50 characters
	buffer[i]='\n';
	write(1,buffer,i+1);
	pclose(rd);
}
```

input.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
int main()
{
	FILE *rd,*rd1;
	char buffer[50],buffer1[20];
	int i,j;
	rd=popen("date","r"); //pipe opened in reading mode
	rd1=popen("whoami","r"); //pipe opened in reading mode
	i=fread(buffer, 1, 40, rd); //will read only 50 characters
	j=fread(buffer1, 1, 20, rd1);
	buffer1[j]='\n';
	write(1,"Welcome user:",13);
	write(1,buffer1,j);
	write(1,buffer,i);
	pclose(rd);
	pclose(rd1);
}
```

Terminal Screenshot:



122CS0067

2. Create a basic chat application using pipes where the parent and child processes can send and receive messages in a loop until a specific termination message (e.g., "exit") is sent.

4_2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#define B_SIZE 1024

void parent_process(int write_fd, int read_fd);
void child_process(int write_fd, int read_fd);

int main() {
    int pipe1[2];
    int pipe2[2];

    if (pipe(pipe1) == -1) {
        perror("pipe1");
        exit(EXIT_FAILURE);
    }
    if (pipe(pipe2) == -1) {
        perror("pipe2");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        close(pipe1[1]);
        close(pipe2[0]);
        child_process(pipe2[1], pipe1[0]);
        close(pipe1[0]);
        close(pipe2[1]);
    } else {
        close(pipe1[0]);
        close(pipe2[1]);
        parent_process(pipe1[1], pipe2[0]);
        close(pipe1[1]);
        close(pipe2[0]);
        wait(NULL);
    }

    return 0;
}

void parent_process(int write_fd, int read_fd) {
    char buffer[B_SIZE];
    while (1) {
        printf("Parent: ");
        if (fgets(buffer, B_SIZE, stdin) == NULL) {
            perror("fgets");
            exit(EXIT_FAILURE);
        }
        buffer[strcspn(buffer, "\n")] = '\0';
```

```
        write(write_fd, buffer, strlen(buffer) + 1);

        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        ssize_t bytesRead = read(read_fd, buffer, B_SIZE);
        if (bytesRead <= 0) {
            perror("read");
            exit(EXIT_FAILURE);
        }
        buffer[bytesRead] = '\0';
        printf("Child Sent: %s\n", buffer);

        if (strcmp(buffer, "exit") == 0) {
            break;
        }
    }
}

void child_process(int write_fd, int read_fd) {
    char buffer[B_SIZE];
    while (1) {
        ssize_t bytesRead = read(read_fd, buffer, B_SIZE);
        if (bytesRead <= 0) {
            perror("read");
            exit(EXIT_FAILURE);
        }
        buffer[bytesRead] = '\0';
        printf("Parent Sent: %s\n", buffer);

        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        printf("Child: ");
        if (fgets(buffer, B_SIZE, stdin) == NULL) {
            perror("fgets");
            exit(EXIT_FAILURE);
        }
        buffer[strcspn(buffer, "\n")] = '\0';
        write(write_fd, buffer, strlen(buffer) + 1);
        if (strcmp(buffer, "exit") == 0) {
            break;
        }
    }
}
```

Terminal Screenshot: