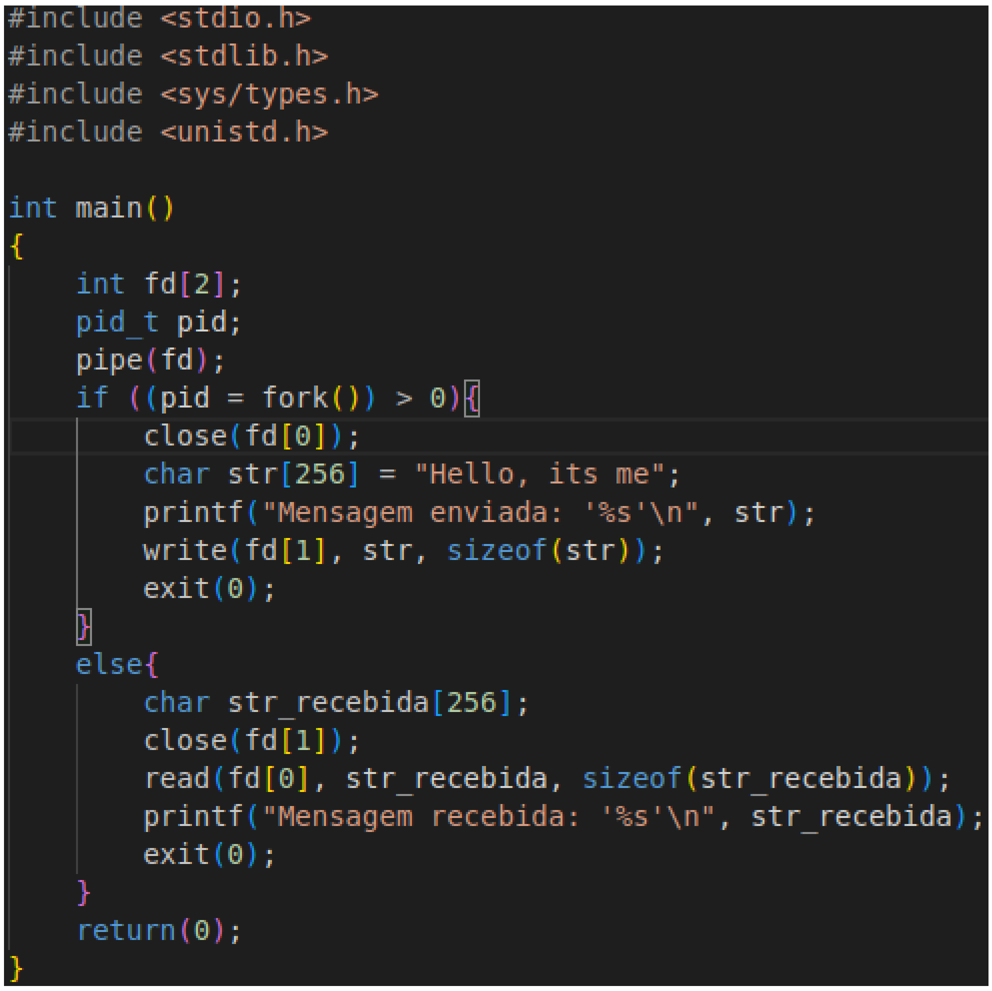
Concurso 3

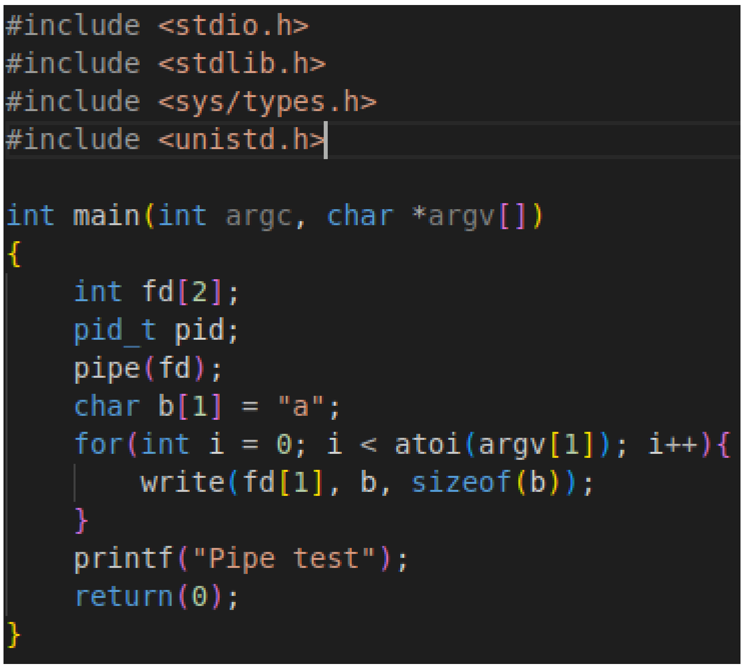
EX1.

1.



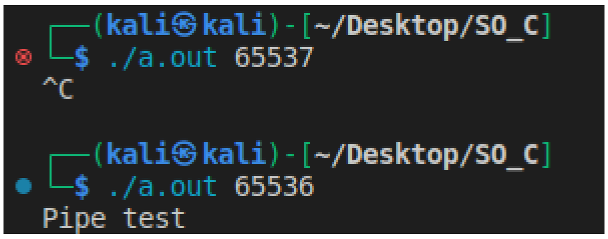
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd[2];
    pid_t pid;
    pipe(fd);
    if ((pid = fork()) > 0){
        close(fd[0]);
        char str[256] = "Hello, its me";
        printf("Mensagem enviada: '%s'\n", str);
        write(fd[1], str, sizeof(str));
        exit(0);
    }
    else{
        char str_recebida[256];
        close(fd[1]);
        read(fd[0], str_recebida, sizeof(str_recebida));
        printf("Mensagem recebida: '%s'\n", str_recebida);
        exit(0);
    }
    return(0);
}
```

2.



```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int fd[2];
    pid_t pid;
    pipe(fd);
    char b[1] = "a";
    for(int i = 0; i < atoi(argv[1]); i++){
        write(fd[1], b, sizeof(b));
    }
    printf("Pipe test");
    return(0);
}
```

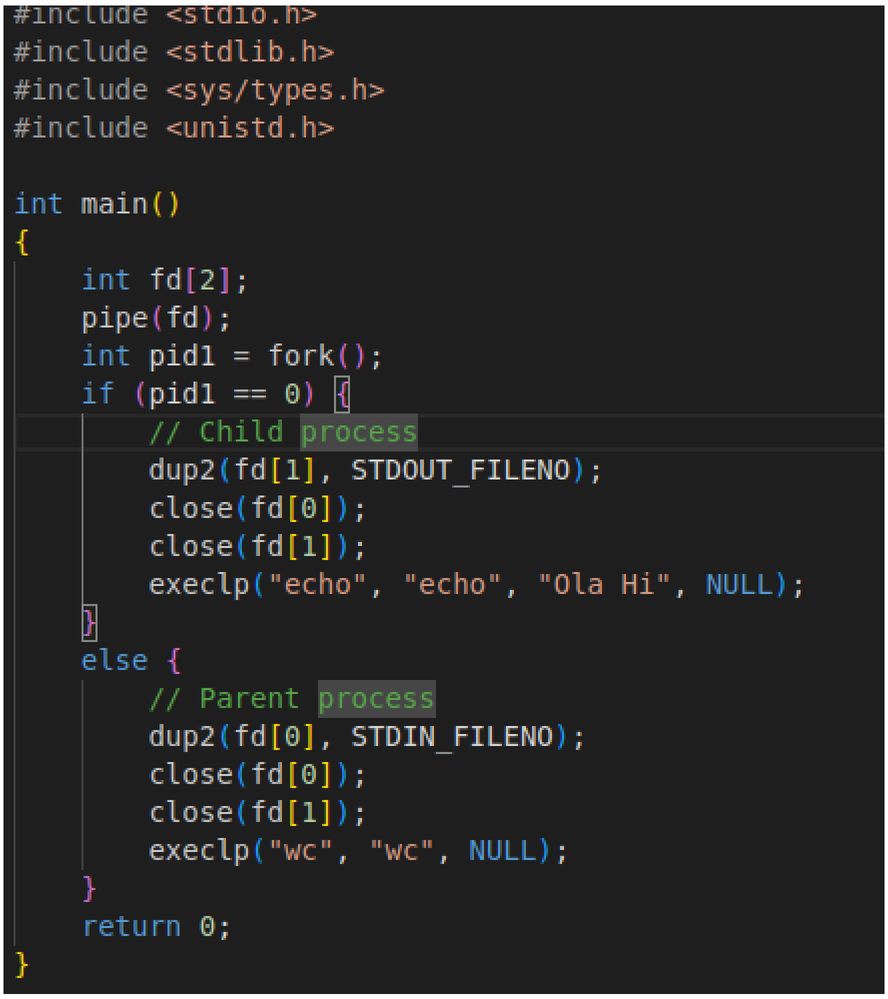
3.



```
┌──(kali㉿kali)-[~/Desktop/SO_C]
└─$ ./a.out 65537
^C

┌──(kali㉿kali)-[~/Desktop/SO_C]
└─$ ./a.out 65536
Pipe test
```

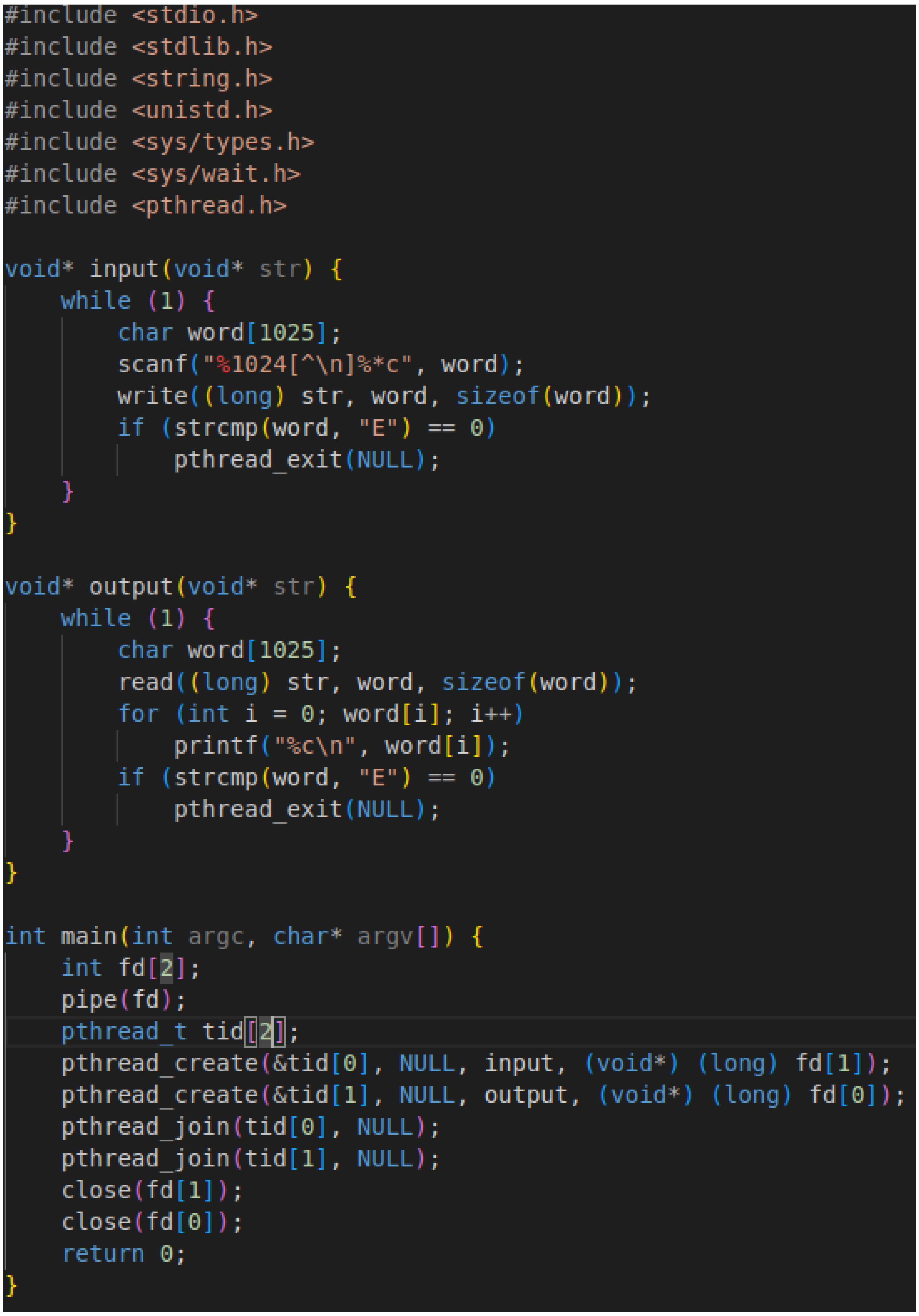
Podemos então concluir que o limite de um pype é 65536.

4.



```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd[2];
    pipe(fd);
    int pid1 = fork();
    if (pid1 == 0) {
        // Child process
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]);
        close(fd[1]);
        execlp("echo", "echo", "Ola Hi", NULL);
    }
    else {
        // Parent process
        dup2(fd[0], STDIN_FILENO);
        close(fd[0]);
        close(fd[1]);
        execlp("wc", "wc", NULL);
    }
    return 0;
}
```

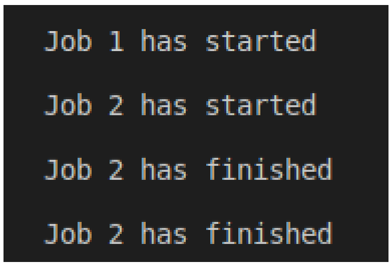
EX2.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>

void* input(void* str) {
    while (1) {
        char word[1025];
        scanf("%1024[^\n]%*c", word);
        write((long) str, word, sizeof(word));
        if (strcmp(word, "E") == 0)
            pthread_exit(NULL);
    }
}

void* output(void* str) {
    while (1) {
        char word[1025];
        read((long) str, word, sizeof(word));
        for (int i = 0; word[i]; i++)
            printf("%c\n", word[i]);
        if (strcmp(word, "E") == 0)
            pthread_exit(NULL);
    }
}

int main(int argc, char* argv[]) {
    int fd[2];
    pipe(fd);
    pthread_t tid[2];
    pthread_create(&tid[0], NULL, input, (void*) (long) fd[1]);
    pthread_create(&tid[1], NULL, output, (void*) (long) fd[0]);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    close(fd[1]);
    close(fd[0]);
    return 0;
}
```

EX3.

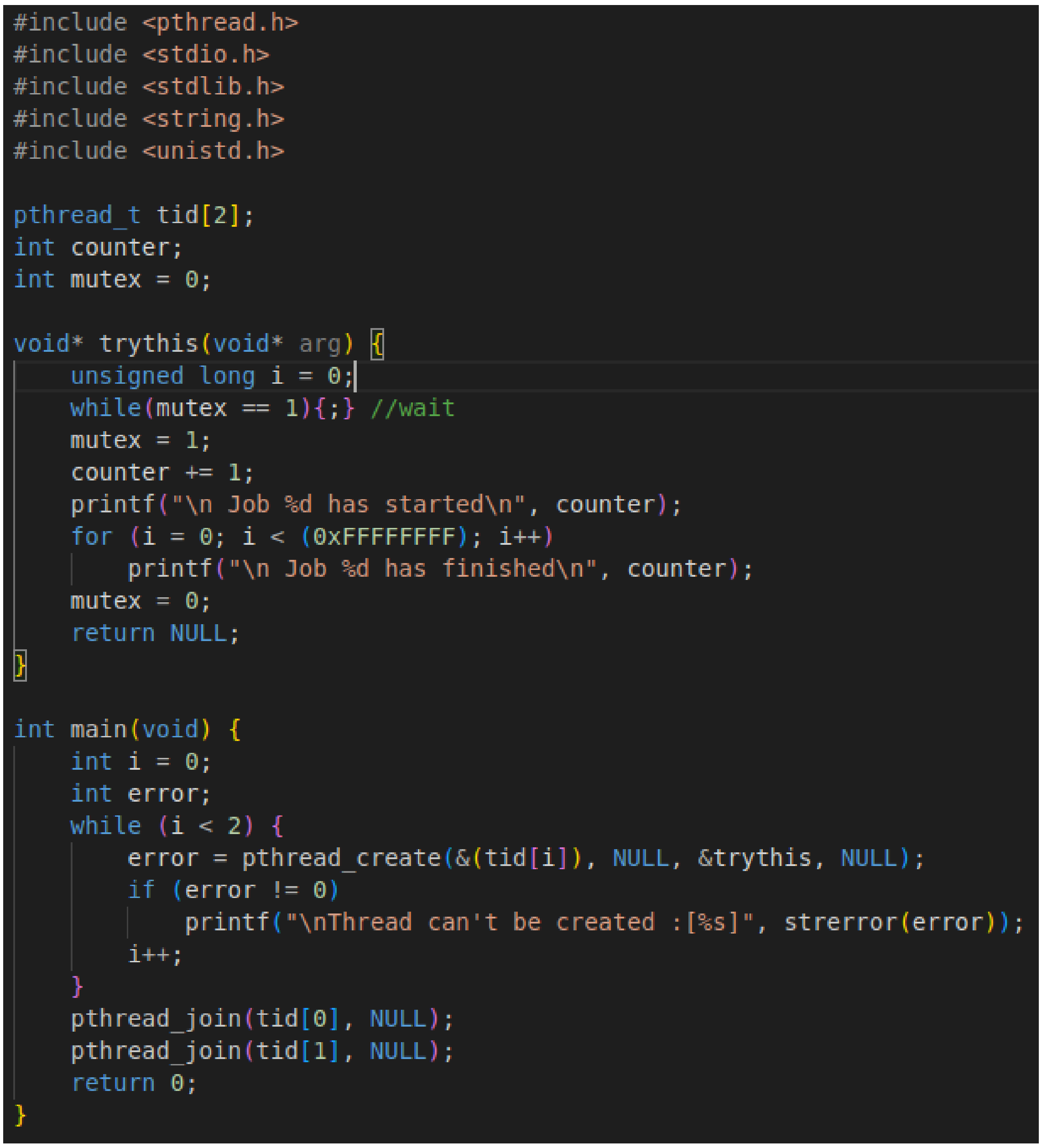
1.



```
Job 1 has started

Job 2 has started

Job 2 has finished

Job 2 has finished
```

2. São criadas duas threads onde as duas executam o mesmo método. É impresso a primeira a frase “Job 1 has started” pois incrementamos um no counter antes, logo em seguida é impresso a mesma frase mas com um counter com valor 2. Em seguida o programa não consegue imprimir as “0Xffffffff” vezes “Job 2 has finished” então so é dado 2 prints com o counter com valor 2.

3.



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

pthread_t tid[2];
int counter;
int mutex = 0;

void* trythis(void* arg) {
    unsigned long i = 0;
    while(mutex == 1){;} //wait
    mutex = 1;
    counter += 1;
    printf("\n Job %d has started\n", counter);
    for (i = 0; i < (0xFFFFFFFF); i++)
        printf("\n Job %d has finished\n", counter);
    mutex = 0;
    return NULL;
}

int main(void) {
    int i = 0;
    int error;
    while (i < 2) {
        error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created :[%s]", strerror(error));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    return 0;
}
```