

Embedded Operating System Assignment 01

Dep. AI Convergence Engineering

Student ID: 2023214019

Name: Md Habibur Rahman

Code your own simplified shell program

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

#define MAXLINE 1024 // Maximum input line length
#define MAXARGS 128 // Maximum arguments

// Function to parse the command line into arguments
void parsecmd(char * cmdline, char ** argv) {
    char * token;
    int i = 0;

    // Tokenize the command line based on spaces
    token = strtok(cmdline, " \\t\\n");
    while (token != NULL) {
        argv[i++] = token;
        token = strtok(NULL, " \\t\\n");
    }
    argv[i] = NULL; // Set the last argument to NULL
}

// Function to check if a command is built-in (e.g., exit or custom
// commands)
int builtin_command(char ** argv) {
    if (strcmp(argv[0], "exit") == 0) {
        exit(0);
    } else if (strcmp(argv[0], "owner") == 0) {
        printf("habib\\n");
        return 1; // Return 1 to indicate a built-in command was executed
    }
    return 0;
}

int main(void) {
    char cmdline[MAXLINE];
    char * argv[MAXARGS];
    pid_t pid;
    int status;

    while (1) {
        // Display a prompt and get the command from the user
        printf("my_shell> ");
        if (fgets(cmdline, MAXLINE, stdin) == NULL) {
            perror("Error reading input");
            exit(1);
        }

        // Remove the newline character from the command line
        cmdline[strcspn(cmdline, "\\n")] = '\\0';
    }
}
```

```

        // Parse the command line into arguments
        parsecmd(cmdline, argv);

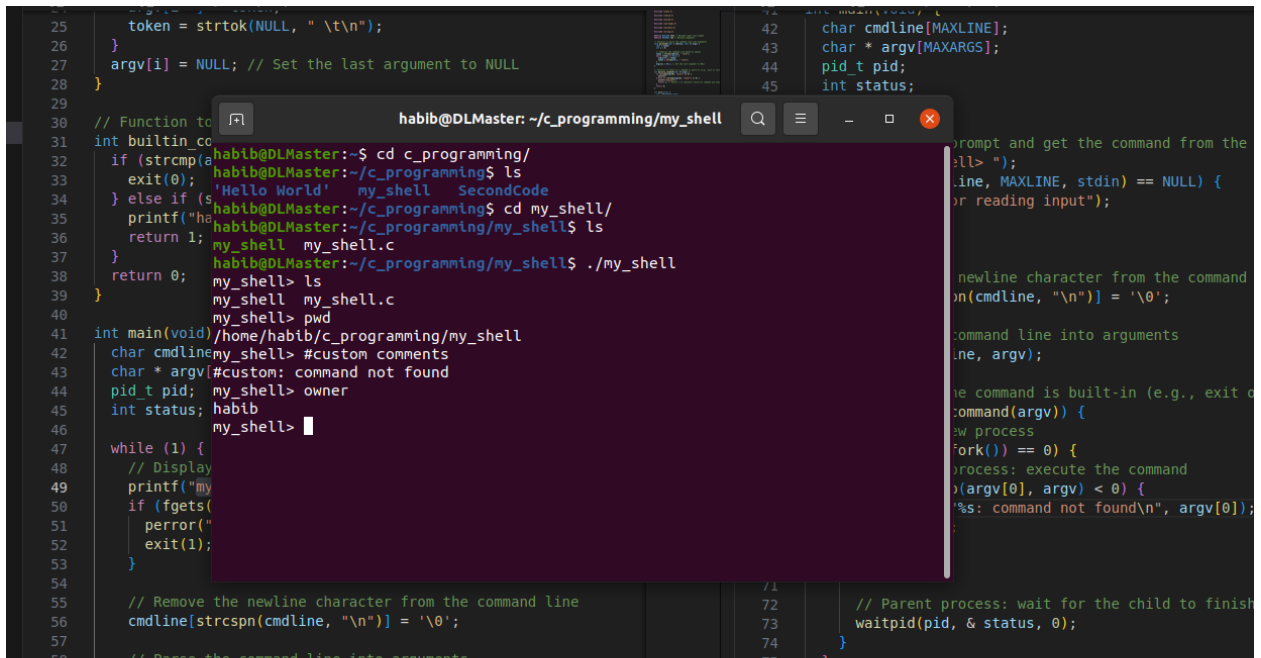
        // Check if the command is built-in (e.g., exit or custom
commands)
        if (!builtin_command(argv)) {
            // Fork a new process
            if ((pid = fork()) == 0) {
                // Child process: execute the command
                if (execvp(argv[0], argv) < 0) {
                    printf("%s: command not found\n", argv[0]);
                    exit(0);
                }
            }

            // Parent process: wait for the child to finish
            waitpid(pid, & status, 0);
        }

        return 0;
}

```

Output



```

25     token = strtok(NULL, " \t\n");
26 }
27 argv[i] = NULL; // Set the last argument to NULL
28 }
29
30 // Function to
31 int builtin_co
32 if (strcmp(a
33     exit(0);
34 } else if (s
35     printf("ha
36     return 1;
37 }
38 return 0;
39 }
40
41 int main(void)
42     char cmdline;
43     char * argv;
44     pid_t pid;
45     int status;
46
47     while (1) {
48         // Display
49         printf("my
50         if (fgets(
51             perror("
52             exit(1);
53     }
54
55     // Remove the newline character from the command line
56     cmdline[strlen(cmdline, "\n")] = '\0';
57
58     // Parse the command line into arguments
59
60     // Check if the command is built-in (e.g., exit or custom
61     commands)
62     if (!builtin_command(argv)) {
63         // Fork a new process
64         if ((pid = fork()) == 0) {
65             // Child process: execute the command
66             if (execvp(argv[0], argv) < 0) {
67                 printf("%s: command not found\n", argv[0]);
68                 exit(0);
69             }
70         }
71
72         // Parent process: wait for the child to finish
73         waitpid(pid, & status, 0);
74     }
75
76     return 0;
77 }

```

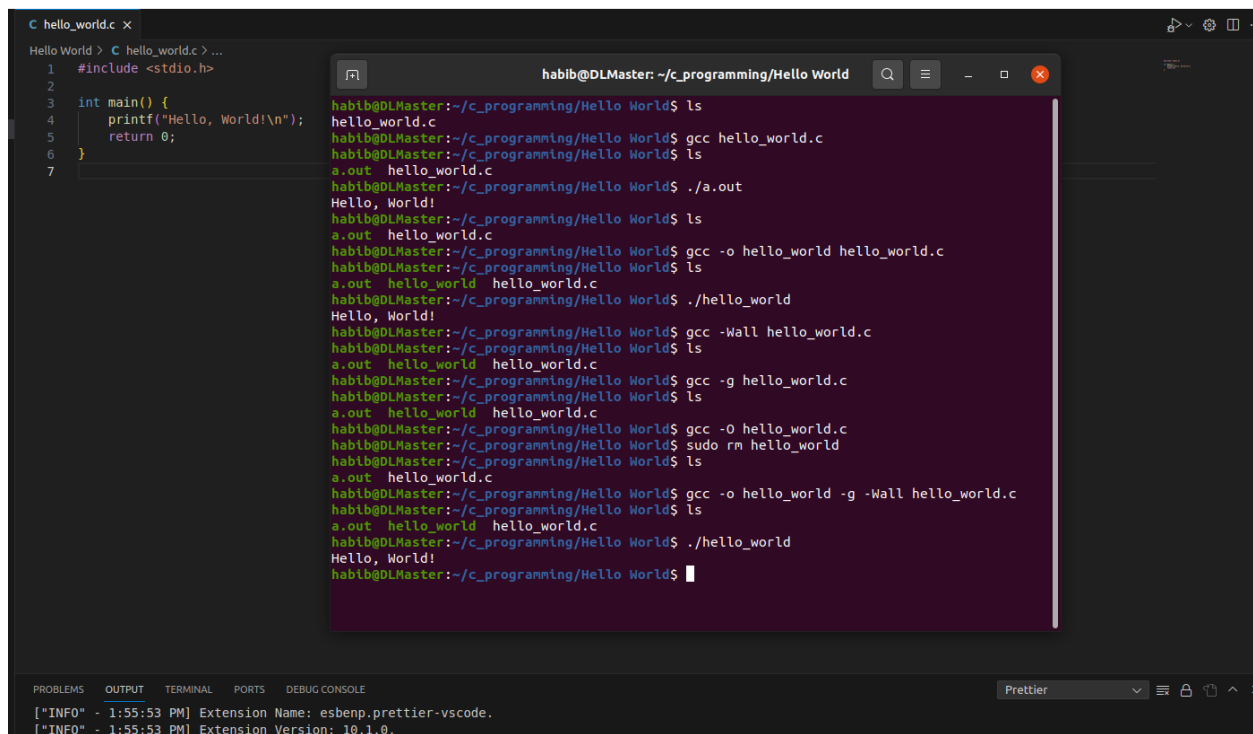
```

habib@DLMaster: ~/c_programming/my_shell
habib@DLMaster:~$ cd c_programming/
habib@DLMaster:~/c_programming$ ls
Hello World  my_shell  SecondCode
habib@DLMaster:~/c_programming$ cd my_shell/
habib@DLMaster:~/c_programming/my_shell$ ls
my_shell  my_shell.c
habib@DLMaster:~/c_programming/my_shell$ ./my_shell
my_shell> ls
my_shell  my_shell.c
my_shell> pwd
/home/habib/c_programming/my_shell
my_shell> #custom comments
my_shell> #custom: command not found
my_shell> owner
my_shell> habib
my_shell>

```

Read and code the Lab tutorial section

1. gcc hellow_world.c # Create a a.out file for execution of the code
2. ./a.out # Execute the code
3. gcc -o hw hw.c # -o: to specify the executable file name
4. gcc -Wall hw.c # -Wall: gives much better warnings
5. gcc -g hw.c # -g: to enable debugging with gdb
6. gcc -O hw.c # -O: to turn on optimization



The image shows a VS Code editor with a C program named `hello_world.c` and a terminal window showing the execution of the program. The C program is as follows:

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello, World!\n");
5     return 0;
6 }
7
```

The terminal window shows the following commands and output:

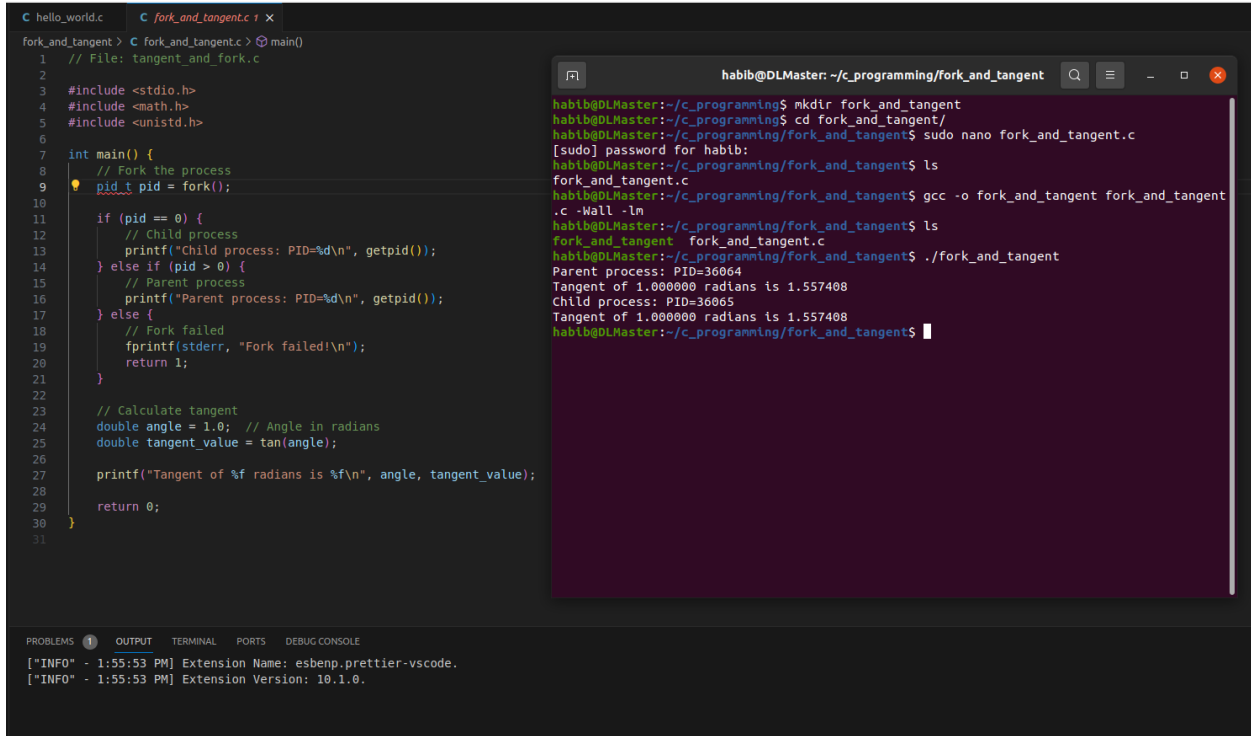
```
habib@DLMaster:~/c_programming/Hello World$ ls
hello_world.c
habib@DLMaster:~/c_programming/Hello World$ gcc hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ./a.out
Hello, World!
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world.c
habib@DLMaster:~/c_programming/Hello World$ gcc -o hello_world hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ./hello_world
Hello, World!
habib@DLMaster:~/c_programming/Hello World$ gcc -Wall hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world hello_world.c
habib@DLMaster:~/c_programming/Hello World$ gcc -g hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world hello_world.c
habib@DLMaster:~/c_programming/Hello World$ gcc -O hello_world.c
habib@DLMaster:~/c_programming/Hello World$ sudo rm hello_world
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world.c
habib@DLMaster:~/c_programming/Hello World$ gcc -o hello_world -g -Wall hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ls
a.out hello_world hello_world.c
habib@DLMaster:~/c_programming/Hello World$ ./hello_world
Hello, World!
habib@DLMaster:~/c_programming/Hello World$
```

The bottom status bar of VS Code shows the following information:

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
["INFO" - 1:55:53 PM] Extension Name: esbenp.prettier-vscode.
["INFO" - 1:55:53 PM] Extension Version: 10.1.0.
```

Linking with Libraries

1. `gcc -o fork_and_tangent fork_and_tangent.c -Wall -lm` # This will include the math library as header file and create an executable file
2. `./tangent_and_fork` # This will execute the code.



The image shows a VS Code editor with a C program named `fork_and_tangent.c` and a terminal window showing the execution of the program.

Code in `fork_and_tangent.c`:

```
1 // File: tangent_and_fork.c
2
3 #include <stdio.h>
4 #include <math.h>
5 #include <unistd.h>
6
7 int main() {
8     // Fork the process
9     pid_t pid = fork();
10
11     if (pid == 0) {
12         // Child process
13         printf("Child process: PID=%d\n", getpid());
14     } else if (pid > 0) {
15         // Parent process
16         printf("Parent process: PID=%d\n", getpid());
17     } else {
18         // Fork failed
19         fprintf(stderr, "Fork failed!\n");
20         return 1;
21     }
22
23     // Calculate tangent
24     double angle = 1.0; // Angle in radians
25     double tangent_value = tan(angle);
26
27     printf("Tangent of %f radians is %f\n", angle, tangent_value);
28
29     return 0;
30 }
31
```

Terminal Output:

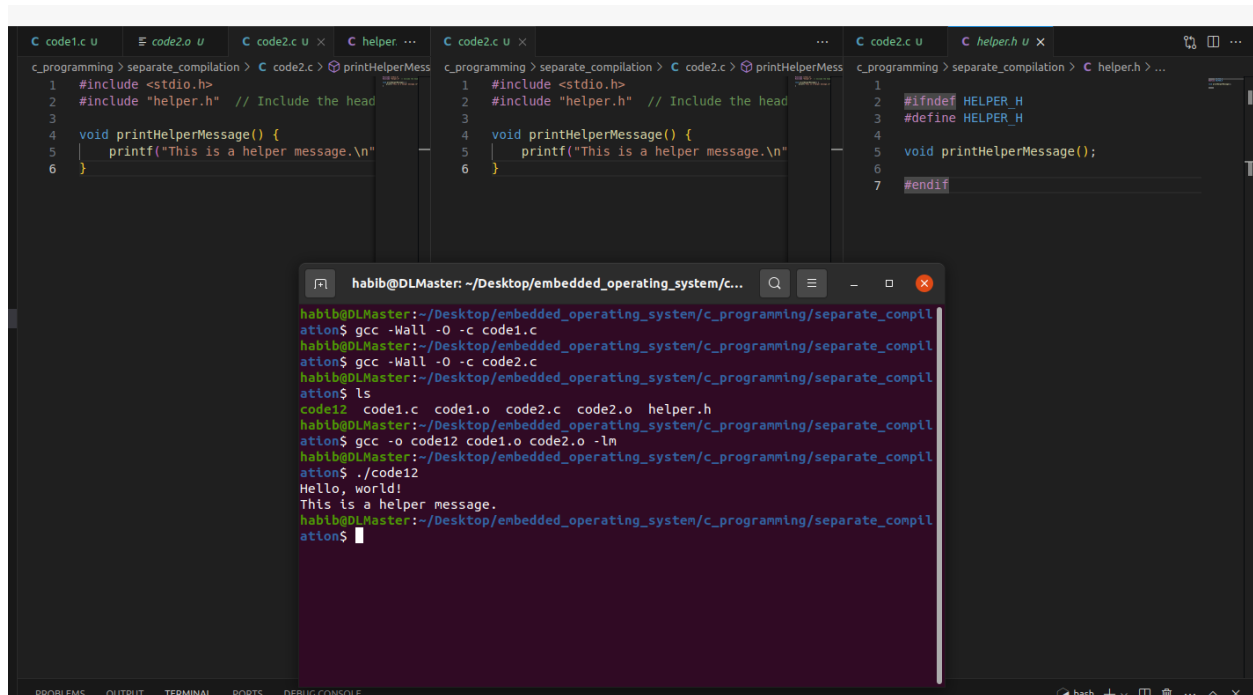
```
habib@DLMaster: ~/c_programming/fork_and_tangent
habib@DLMaster:~/c_programming$ mkdir fork_and_tangent
habib@DLMaster:~/c_programming$ cd fork_and_tangent/
habib@DLMaster:~/c_programming/fork_and_tangent$ sudo nano fork_and_tangent.c
[sudo] password for habib:
habib@DLMaster:~/c_programming/fork_and_tangent$ ls
fork_and_tangent.c
habib@DLMaster:~/c_programming/fork_and_tangent$ gcc -o fork_and_tangent fork_and_tangent.c -Wall -lm
habib@DLMaster:~/c_programming/fork_and_tangent$ ls
fork_and_tangent  fork_and_tangent.c
habib@DLMaster:~/c_programming/fork_and_tangent$ ./fork_and_tangent
Parent process: PID=36064
Tangent of 1.000000 radians is 1.557408
Child process: PID=36065
Tangent of 1.000000 radians is 1.557408
habib@DLMaster:~/c_programming/fork_and_tangent$
```

PROBLEMS:

```
["INFO" - 1:55:53 PM] Extension Name: esbenp.prettier-vscode.
["INFO" - 1:55:53 PM] Extension Version: 10.1.0.
```

Separate Compilation

1. `sudo nano helper.h` # create a header file
2. `gcc -Wall -O -c code1.c`
3. `gcc -Wall -O -c code2.c`
4. `gcc -o code12 code1.o code2.o -lm` # This will make a execute file from two .c code
5. `./code12` # this will execute the combined file



The screenshot displays a code editor with three open files: `code1.c`, `code2.c`, and `helper.h`. The `code1.c` and `code2.c` files both include `helper.h` and define a `printHelperMessage()` function that prints "This is a helper message.\n". The `helper.h` file defines the `printHelperMessage()` function prototype and uses `#ifndef` and `#define` to prevent multiple definitions.

Below the code editor, a terminal window shows the execution of the compilation steps:

```
habib@DLMaster: ~/Desktop/embedded_operating_system/c...  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$ gcc -Wall -O -c code1.c  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$ gcc -Wall -O -c code2.c  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$ ls  
code12  code1.c  code1.o  code2.c  code2.o  helper.h  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$ gcc -o code12 code1.o code2.o -lm  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$ ./code12  
Hello, world!  
This is a helper message.  
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil  
ation$
```

Makefiles

1. Sudo nano Makefile # create a make file
2. make # Build the executable using the Makefile
3. ./code12 # Run the executable
4. make clean # Clean up the generated files

The screenshot shows a VS Code editor with three open files: `code1.c`, `code2.c`, and `helper.h`. The `code1.c` file contains the following code:

```
1 #include <stdio.h>
2 #include "helper.h" // Include the header file
3
4 void printHelperMessage() {
5     printf("This is a helper message.\n");
6 }
```

The `code2.c` file contains the following code:

```
1 #include "helper.h"
2
3 int main() {
4     printHelperMessage();
5     return 0;
6 }
```

The `Makefile` file contains the following code:

```
1 # Makefile
2
3 CC = gcc
4 CFLAGS = -Wall -O
5 LDFLAGS = -lm
6
7 SRC1 = code1.c
8 SRC2 = code2.c
9 OBJS = $(SRC1:.c=.o) $(SRC2:.c=.o)
10 TARGET = code12
11
12 all: $(TARGET)
13
14 $(TARGET): $(OBJS)
15     $(CC) -o $@ $^ $(LDFLAGS)
16
17 %.o: %.c
18     $(CC) $(CFLAGS) -c -o $@ $<
19
20 .PHONY: clean
21
22 clean:
23     rm -f $(OBJS) $(TARGET)
```

The terminal window shows the following commands and output:

```
habib@DLMaster: ~/Desktop/embedded_operating_system/c...
u habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil
u ation$ ls
u code12 code1.c code1.o code2.c code2.o helper.h Makefile
u habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil
u ation$ make clean
u rm -f code1.o code2.o code12
u habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil
u ation$ make
gcc -Wall -O -c -o code1.o code1.c
gcc -Wall -O -c -o code2.o code2.c
gcc -o code12 code1.o code2.o -lm
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil
u ation$ ./code12
Hello, world!
This is a helper message.
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/separate_compil
u ation$
```

Debugging

1. gcc -g -o buggy buggy.c # compile this program with debugging information using the -g flag
2. gdb buggy # use gdb to debug the program and analyze the segmentation fault
3. (gdb) run # run the program and see a segmentation fault
4. (gdb) print p # see the segmentation fault
5. (gdb) break main # break main function
6. (gdb) run # run program to check step by step
7. (gdb) next # check the next line and go on you will encounter a segmentation fault

```
habib@DLMaster: ~/Desktop/embedded_operating_system/c_programming/debugging
c_programming > debugging > C debugging.c > ...
1  #include <stdio.h>
2
3  struct Data {
4      int x;
5  };
6
7
8  int main(int argc, char *argv[]) {
9      struct Data *p = NULL;
10     printf("%d\n", p->x); // Dereferencing a null pointer
11     return 0;
12 }

habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/debugging$ gcc -g -o buggy debugging.c
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/debugging$ ls
buggy debugging.c
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/debugging$ gdb buggy
GNU gdb (Ubuntu 9.2-0ubuntu1-20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from buggy...
(gdb) run
Starting program: /home/habib/Desktop/embedded_operating_system/c_programming/debugging/buggy

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555168 in main (argc=1, argv=0x7fffffffdda8) at debugging.c:10
10     printf("%d\n", p->x); // Dereferencing a null pointer
(gdb) print p
$1 = (struct Data *) 0x0
(gdb) break main
Breakpoint 1 at 0x555555555149: file debugging.c, line 8.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/habib/Desktop/embedded_operating_system/c_programming/debugging/buggy

Breakpoint 1, main (argc=21845, argv=0x0) at debugging.c:8
8     int main(int argc, char *argv[]) {
(gdb) next
9     struct Data *p = NULL;
(gdb) next
10    printf("%d\n", p->x); // Dereferencing a null pointer
(gdb) next
Program received signal SIGSEGV, Segmentation fault.
0x0000555555555168 in main (argc=1, argv=0x7fffffffdda8) at debugging.c:10
10    printf("%d\n", p->x); // Dereferencing a null pointer
(gdb) next
Program terminated with signal SIGSEGV, Segmentation Fault.
The program no longer exists.
(gdb)
```


Measure costs of a system call

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

#define NUM_ITERATIONS 1000000

int main() {
    struct timeval start, end;
    double elapsed_time;
    int i;

    // Start measuring time
    gettimeofday(&start, NULL);

    for (i = 0; i < NUM_ITERATIONS; i++) {
        // Perform a 0-byte read as a simple system call
        read(0, NULL, 0);
    }

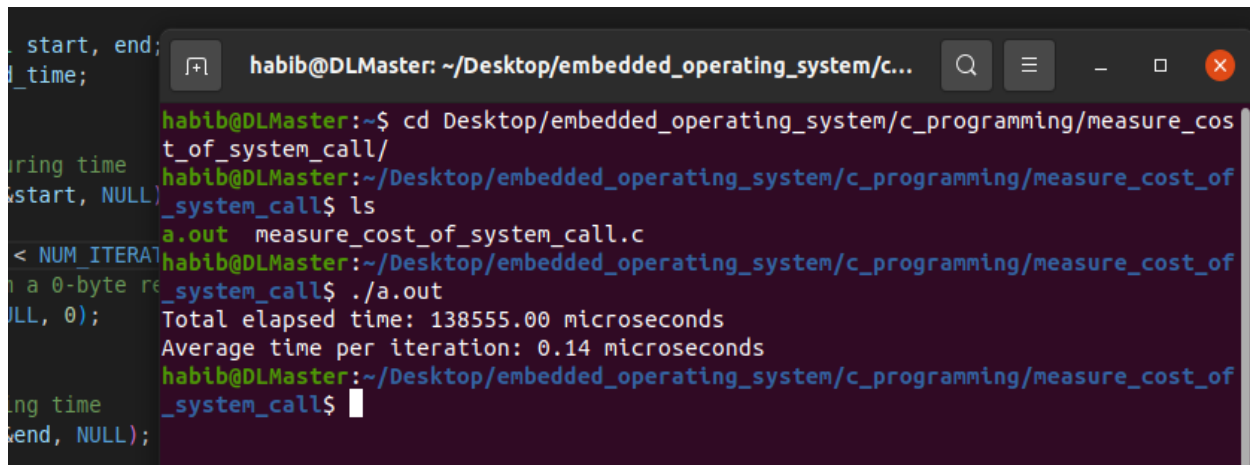
    // End measuring time
    gettimeofday(&end, NULL);

    // Calculate elapsed time in microseconds
    elapsed_time = (end.tv_sec - start.tv_sec) * 1000000.0;
    elapsed_time += (end.tv_usec - start.tv_usec);

    // Calculate average time per iteration
    double average_time_per_iteration = elapsed_time / NUM_ITERATIONS;

    printf("Total elapsed time: %.2f microseconds\n", elapsed_time);
    printf("Average time per iteration: %.2f microseconds\n",
average_time_per_iteration);

    return 0;
}
```



The screenshot shows a terminal window with the following commands and output:

```
habib@DLMaster: ~/Desktop/embedded_operating_system/c...
habib@DLMaster:~$ cd Desktop/embedded_operating_system/c_programming/measure_cos
t_of_system_call/
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/measure_cost_of
_system_call$ ls
a.out measure_cost_of_system_call.c
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/measure_cost_of
_system_call$ ./a.out
Total elapsed time: 138555.00 microseconds
Average time per iteration: 0.14 microseconds
habib@DLMaster:~/Desktop/embedded_operating_system/c_programming/measure_cost_of
_system_call$
```

Calculate and explain the average turnaround time and response time

1. A and B need 50ms of CPU time each.
2. A runs for 10ms and then issues an I/O request
 - a. I/Os each take 10ms
3. B simply uses the CPU for 50ms and performs no I/O
4. The scheduler runs A first, then B after

Answer:

1. STCF Scheduling Algorithm:

- a. Process A Turnaround Time (TAT) = Completion Time–Arrival Time
 $= 90 - 0$
 $= 90$
- b. Process B Turnaround Time (TAT) = Completion Time–Arrival Time
 $= 140 - 0$
 $= 140$

$$\begin{aligned}\text{Average Turnaround Time} &= (90+140)/2 \\ &= 230/2 \\ &= 115\end{aligned}$$

- c. Process A Response Time (RT) = the time it takes for a process to start executing once it's given access to the CPU
 $= 0$
- d. Process B Response Time (RT) = the time it takes for a process to start executing once it's given access to the CPU
 $= 90$

$$\begin{aligned}\text{Average Response Time} &= (0+90)/2 \\ &= 90/2 \\ &= 45\end{aligned}$$

2. Round Robin Scheduling Algorithm:

- a. Process A Turnaround Time (TAT) = Completion Time–Arrival Time
 $= 90 - 0$
 $= 90$
- b. Process B Turnaround Time (TAT) = Completion Time–Arrival Time
 $= 100 - 0$
 $= 100$

$$\begin{aligned}\text{Average Turnaround Time} &= (90+100)/2 \\ &= 190/2 \\ &= 95\end{aligned}$$

c. Process A Response Time (RT) = the time it takes for a process to start executing once it's given access to the CPU
= 0

d. Process B Response Time (RT) = the time it takes for a process to start executing once it's given access to the CPU
= 10

Average Response Time = $(0+10)/2$
= $10/2$
= 5