**Operating Systems – Lab 04**

**Name:** Hafsa Salman

**Roll no.** 22K-5161

Lab 04 – Tasks 1

Lab Exercises - I

**Task no. 01**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main ()

{

pid\_t child1, child2;

child1 = fork();

if (child1 == 0)

{

printf("Child 01 PID: %d\n", getpid());

exit(0);

}

else

{

child2 = fork();

if (child2 == 0)

{

printf("Child 2 PPID: %d\n", getppid());

exit(0);

}

else

{

wait(NULL);

wait(NULL);

printf("Parent - Child processes have been terminated successfully\n");

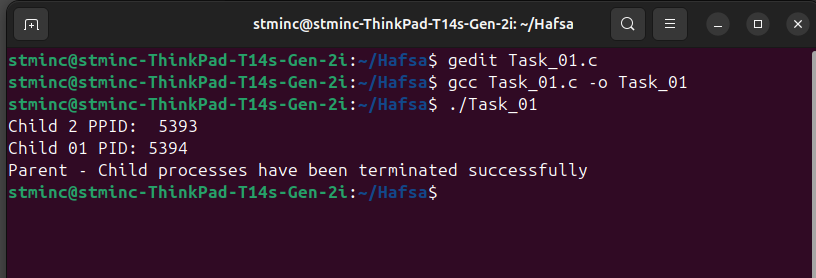
}

}

return 0;

}

Output:



**Task no. 02**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main ()

{

pid\_t child = fork();

if (child == 0)

{

for (int i=1; i<=100; i++)

{

printf("%d. I am a child process.\n", i);

}

exit(0);

}

else

{

for (int i=1; i<=100; i++)

{

printf("%d. I am a parent process.\n", i);

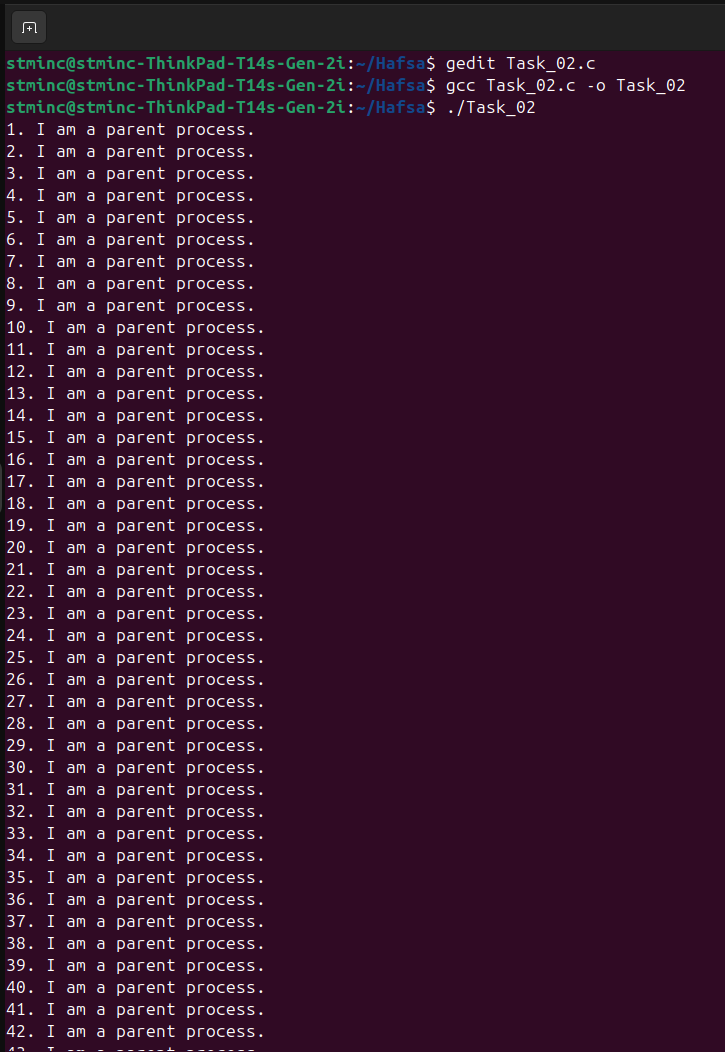
}

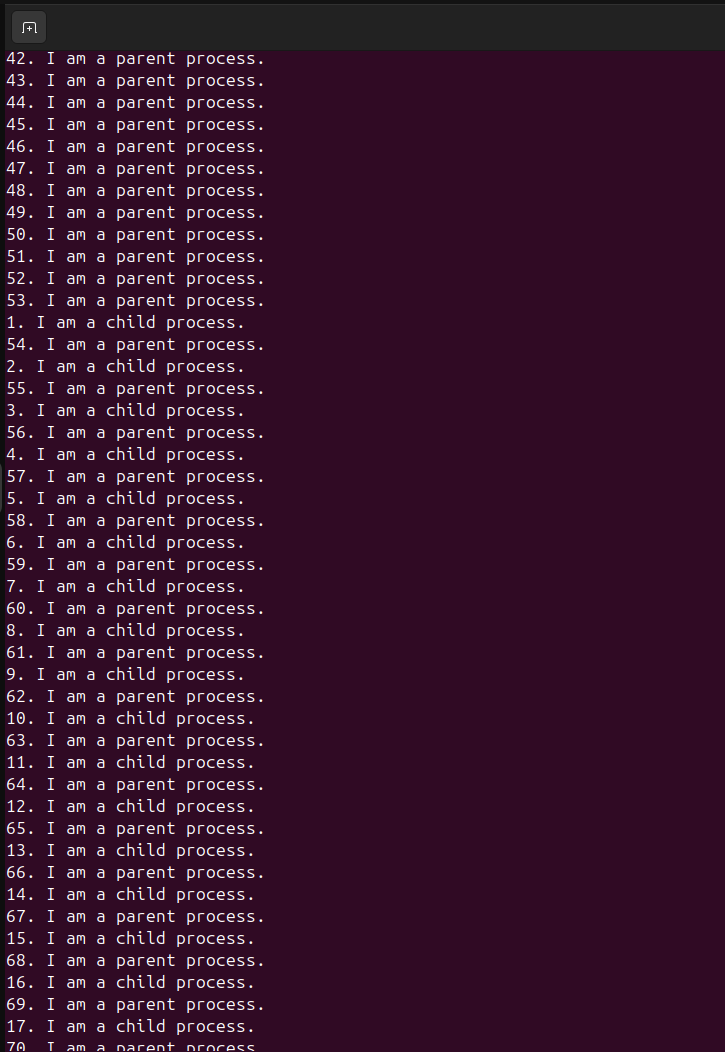
}

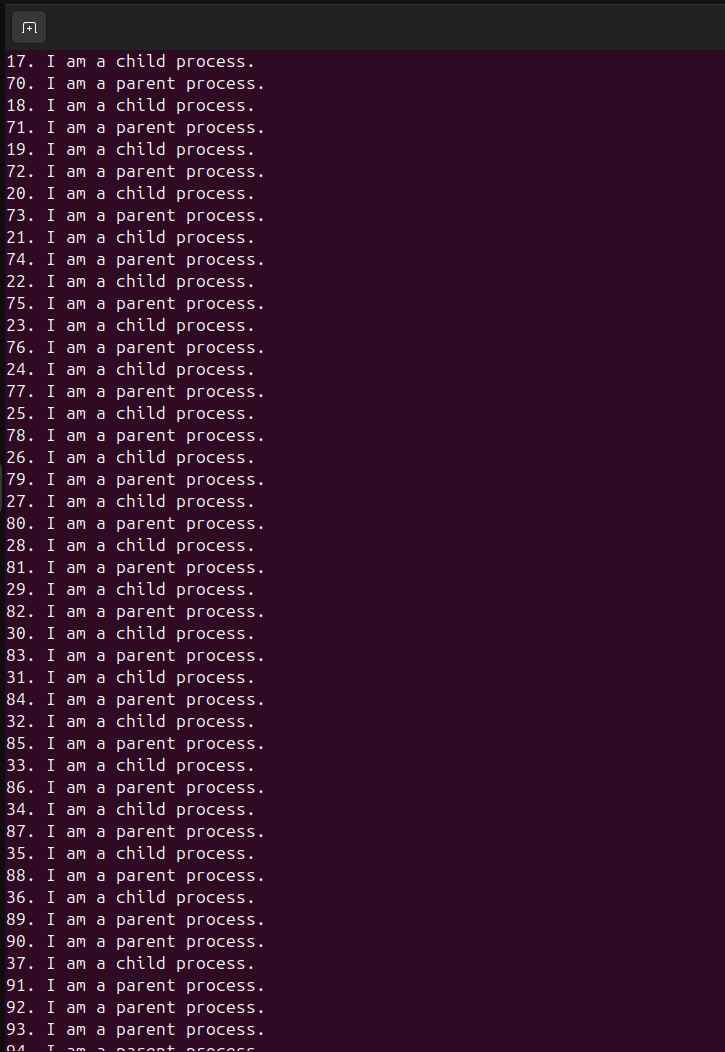
return 0;

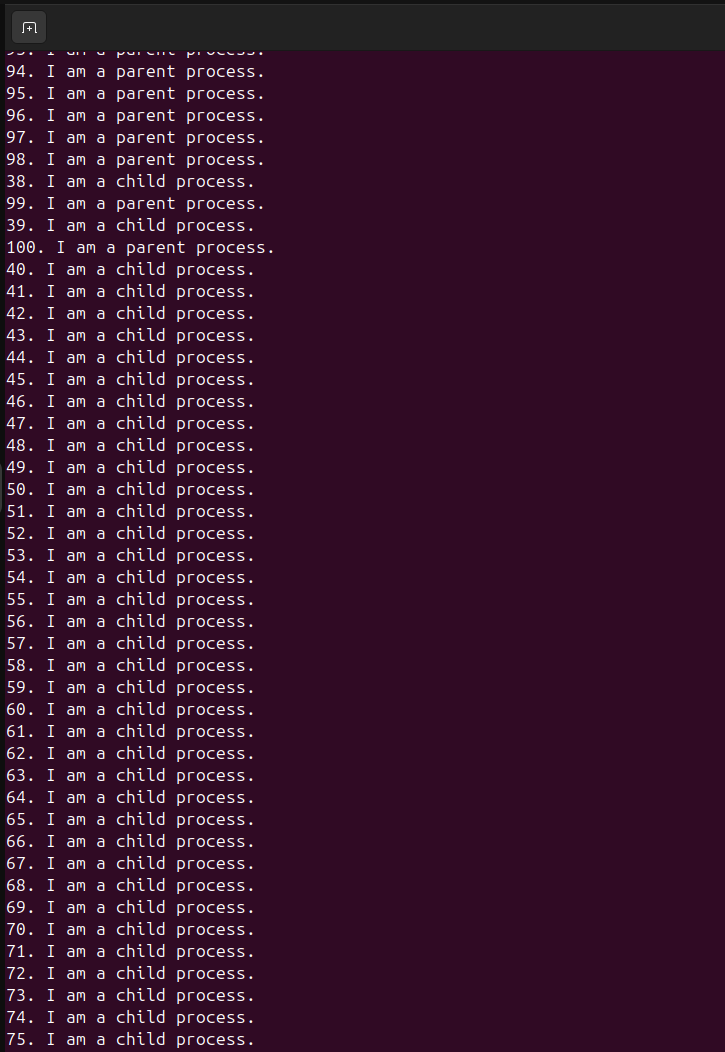
}

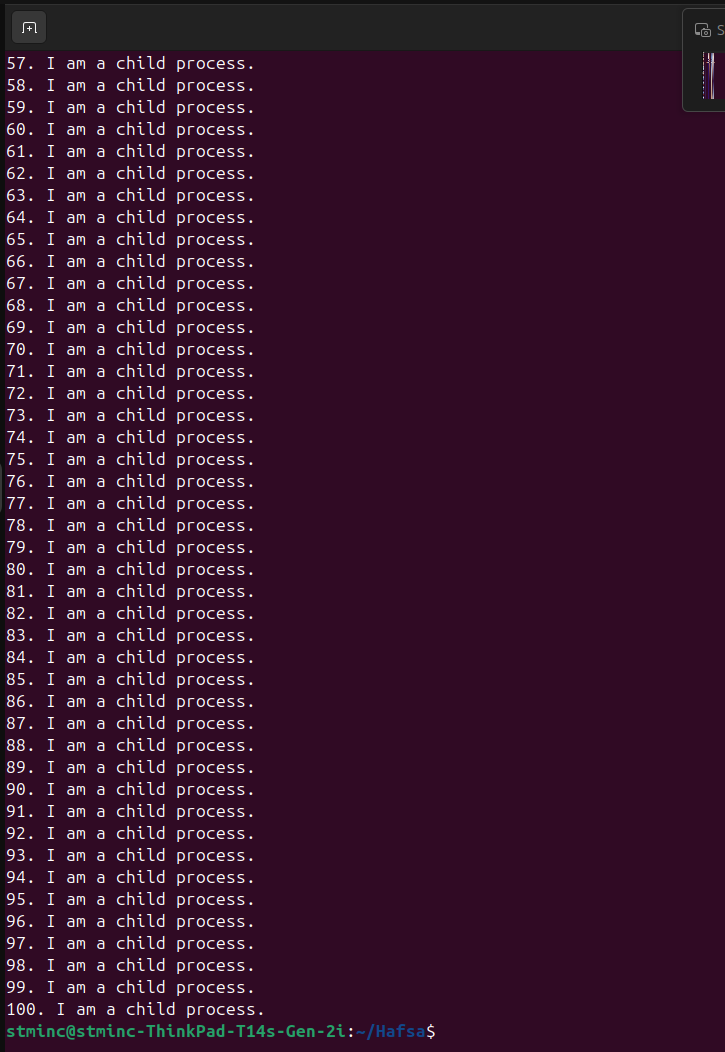
Output:











**Task no. 03**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

void userID()

{

printf("User ID: %d\n", getuid());

}

int main()

{

printf("PID: %d\n", getpid());

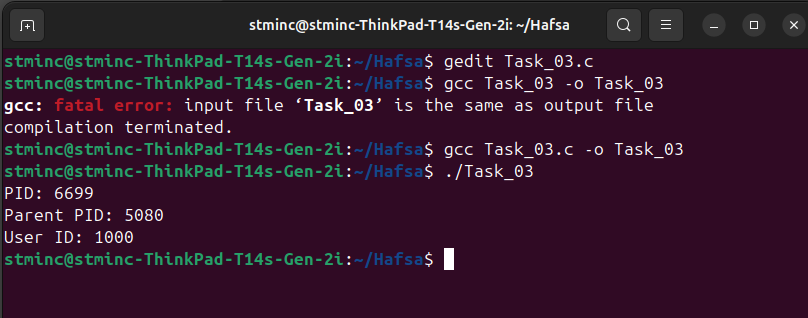
printf("Parent PID: %d\n", getppid());

userID();

return 0;

}

Output:



**Task no. 04**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<sys/wait.h>

#define MAX\_NUMS 100

int parseInput(char \*input, int numbers[])

{

int count = 0;

char \*token = strtok(input, "$");

while (token != NULL && count < MAX\_NUMS)

{

numbers[count++] = atoi(token);

token = strtok(NULL, "$");

}

return count;

}

int main (int argc, char \*argv[])

{

if (argc != 2)

{

fprintf(stderr, "Usage: %s number1$number2$...$numberN\n", argv[0]);

exit(EXIT\_FAILURE);

}

char input[1024];

strncpy(input, argv[1], sizeof(input)-1);

input[sizeof(input) - 1] = '\0';

int nums[MAX\_NUMS];

int n = parseInput(input, nums);

if (n == 0)

{

fprintf(stderr, "Error: No numbers inputted!\n");

exit(EXIT\_FAILURE);

}

pid\_t child1, child2, child3;

child1 = fork();

if (child1 == 0)

{

int sum = 0;

for (int i=0; i<n; i++)

{

sum = sum + nums[i];

}

printf("Child 01: Sum: %d\n", sum);

exit(0);

}

child2 = fork();

if (child2 == 0)

{

int sum = 0;

for (int i=0; i<n; i++)

{

sum = sum + nums[i];

}

float avg = (float)sum/n;

printf("Child 02: Average: %.2f\n", avg);

exit(0);

}

child3 = fork();

if (child3 == 0)

{

int max = nums[0];

for (int i=1; i<n; i++)

{

if (nums[i] > max)

{

max = nums[i];

}

}

printf("Child 03: Max: %d\n", max);

exit(0);

}

wait(NULL);

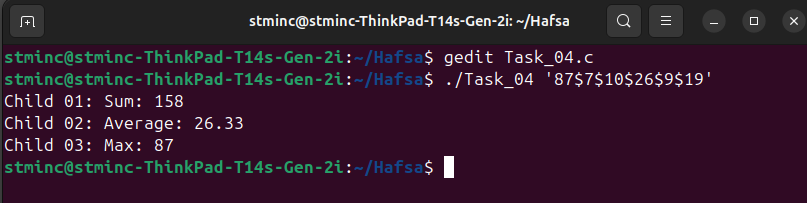
wait(NULL);

wait(NULL);

return 0;

}

Output:



**Task no. 05**

Code:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<fcntl.h>

#include<errno.h>

#define BUFFER\_SIZE 1024

int main (int argc, char \*argv[])

{

if (argc != 3)

{

fprintf(stderr, "Usage: %s <source\_file> <destination\_file>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int src\_fd = open(argv[1], O\_RDONLY);

if (src\_fd == -1)

{

perror("Error opening source file!\n");

exit(EXIT\_FAILURE);

}

int dest\_fd = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (dest\_fd == -1)

{

perror("Error opening destination file!\n");

close(src\_fd);

exit(EXIT\_FAILURE);

}

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read, bytes\_written;

while ((bytes\_read = read(src\_fd, buffer, BUFFER\_SIZE)) > 0)

{

bytes\_written = write(dest\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read)

{

perror("Error writing to destination file!\n");

close(src\_fd);

close(dest\_fd);

exit(EXIT\_FAILURE);

}

}

if (bytes\_read == -1)

{

perror("Error reading from source file!\n");

close(src\_fd);

close(dest\_fd);

exit(EXIT\_FAILURE);

}

close(src\_fd);

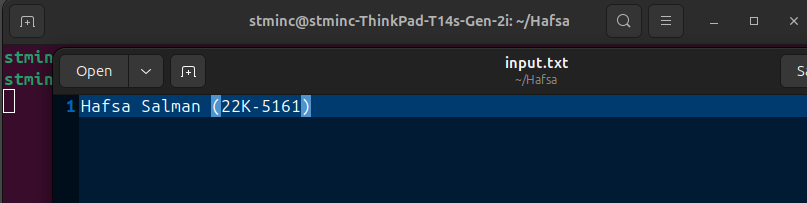
close(dest\_fd);

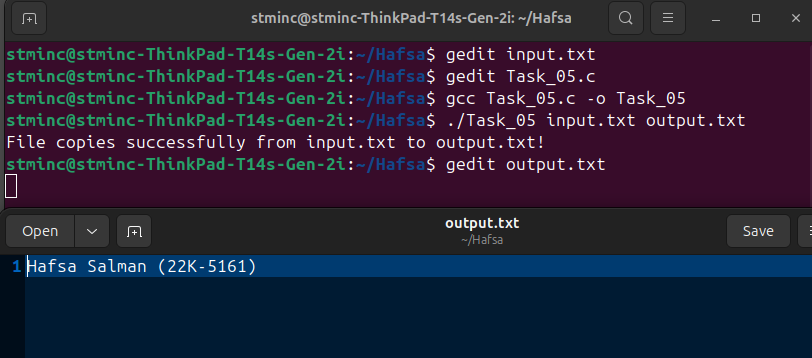
printf("File copies successfully from %s to %s!\n", argv[1], argv[2]);

return 0;

}

Output:





**Task no. 06**

Code:

#include<stdio.h>

#include<stdlib.h>

int main()

{

printf("Making directory 'Task\_6'\n");

system("mkdir Task\_6");

printf("Making file 'Task\_6/22K-5161.txt'\n");

system("touch Task\_6/22K-5161.txt");

printf("Copying file content from 'input.txt' to 'Task\_6/22K-5161.txt'\n");

system("cp input.txt Task\_6/22K-5161.txt");

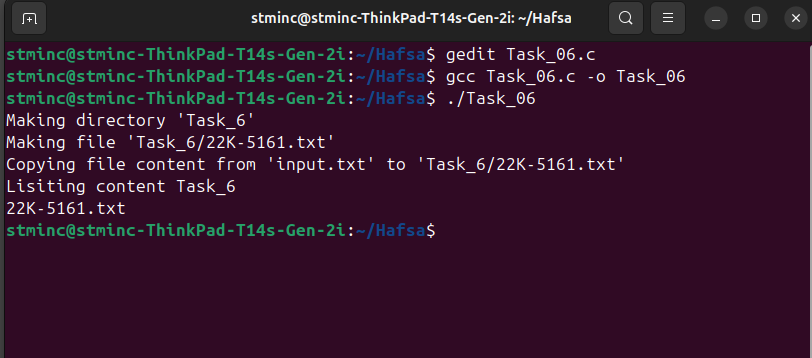
printf("Lisiting content Task\_6\n");

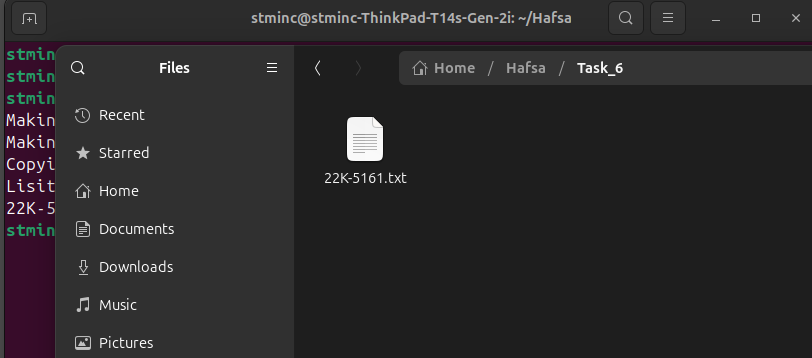
system("ls Task\_6");

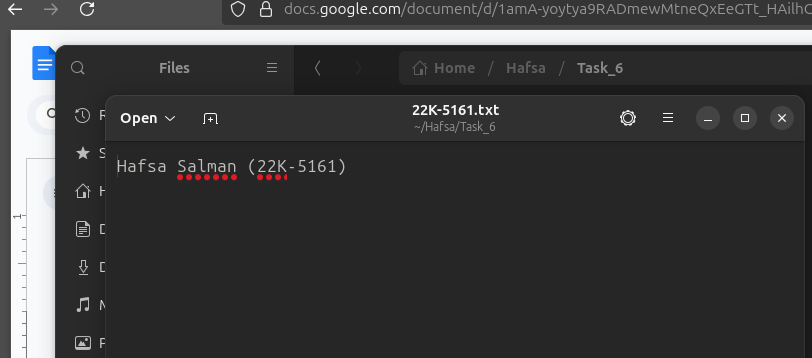
return 0;

}

Output:







**Task no. 07**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t child = fork();

if (child == 0)

{

printf("Child Process is running!\n");

execlp("ls", "ls", NULL);

}

else

{

printf("Parent - Waiting for Child process\n");

wait(NULL);

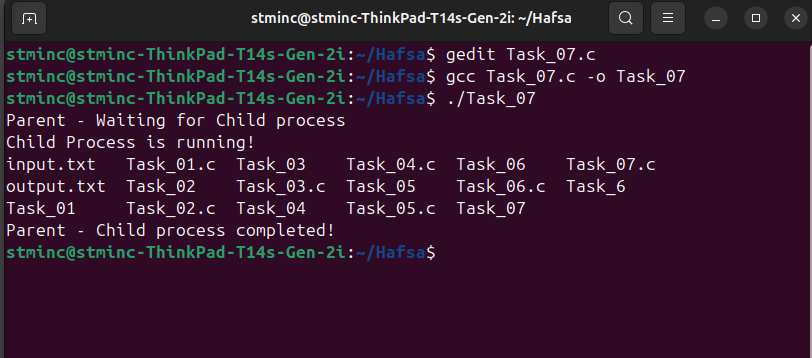
printf("Parent - Child process completed!\n");

}

return 0;

}

Output:



**Task no. 08**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

void Alarm (int sig)

{

printf("Alarm! Exiting Program!\n");

exit(0);

}

int main()

{

signal(SIGALRM, Alarm);

alarm(5);

printf("Alarm for 5 seocnds!\n");

while (1)

{

printf("Sleeping - 1 second\n");

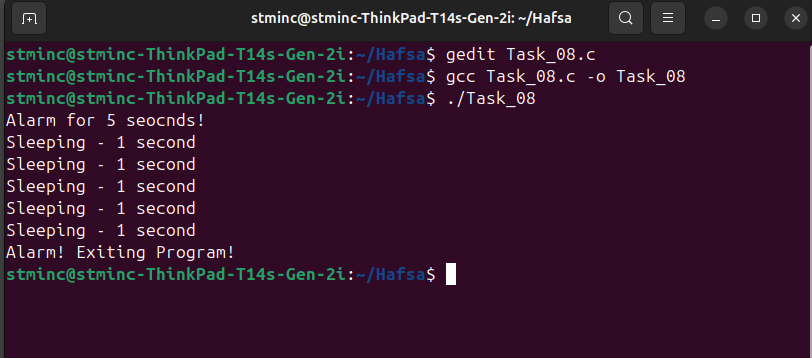
sleep(1);

}

return 0;

}

Output:



Lab Exercises - II

**Task no. 01**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main ()

{

pid\_t child = fork();

if (child == 0)

{

printf("Child - Listing contents...\n");

system("ls");

exit(0);

}

else

{

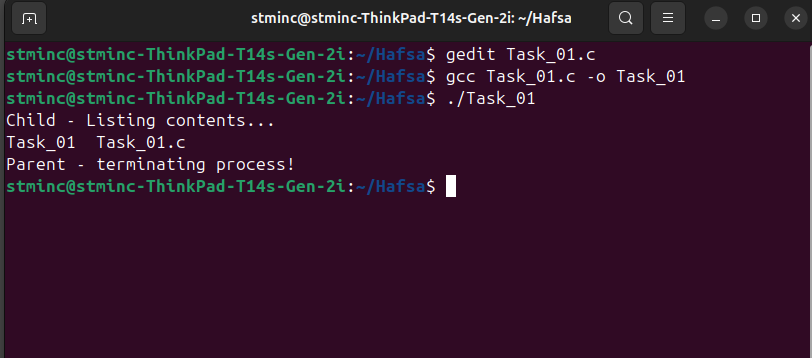
wait(NULL);

printf("Parent - terminating process!\n");

}

}

Output:



**Task no. 02**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

printf("Program's PID: %d\n", getpid());

pid\_t child = fork();

if (child == 0)

{

printf("Child Process: \n");

printf("PID: %d\n", getpid());

printf("Parent PID: %d\n", getppid());

}

else

{

printf("Parent Process: \n");

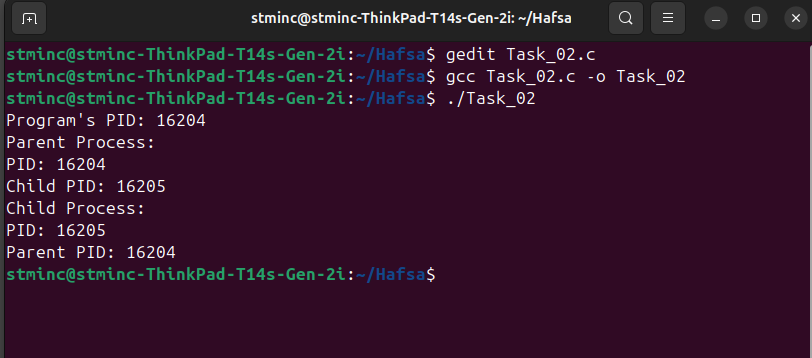
printf("PID: %d\n", getpid());

printf("Child PID: %d\n", child);

}

}

Output:



**Task no. 03**

Code:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<fcntl.h>

#include<errno.h>

#define BUFFER\_SIZE 1024

int main (int argc, char \*argv[])

{

if (argc != 3)

{

fprintf(stderr, "Usage: %s <source\_file> <destination\_file>\n", argv[0]);

exit(EXIT\_FAILURE);

}

int src\_fd = open(argv[1], O\_RDONLY);

if (src\_fd == -1)

{

perror("Error opening source file!\n");

exit(EXIT\_FAILURE);

}

int dest\_fd = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (dest\_fd == -1)

{

perror("Error opening destination file!\n");

close(src\_fd);

exit(EXIT\_FAILURE);

}

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read, bytes\_written;

while ((bytes\_read = read(src\_fd, buffer, BUFFER\_SIZE)) > 0)

{

bytes\_written = write(dest\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read)

{

perror("Error writing to destination file!\n");

close(src\_fd);

close(dest\_fd);

exit(EXIT\_FAILURE);

}

}

if (bytes\_read == -1)

{

perror("Error reading from source file!\n");

close(src\_fd);

close(dest\_fd);

exit(EXIT\_FAILURE);

}

close(src\_fd);

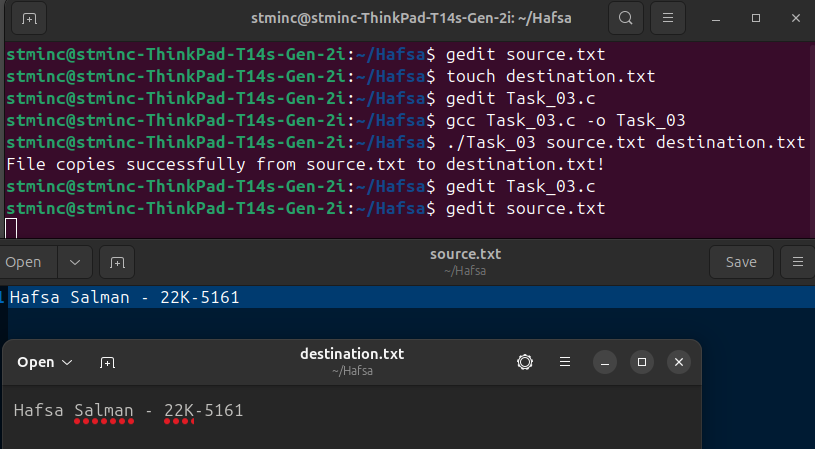
close(dest\_fd);

printf("File copies successfully from %s to %s!\n", argv[1], argv[2]);

return 0;

}

Output:



**Task no. 04**

Code:

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

void list\_dir(const char \*path, int level)

{

DIR \*dir;

struct dirent \*entry;

dir = opendir(path);

if (!dir)

{

perror("opendir failed");

return;

}

while ((entry = readdir(dir)) != NULL)

{

char fullpath[1024];

struct stat statbuf;

if (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

{

continue;

}

snprintf(fullpath, sizeof(fullpath), "%s/%s", path, entry->d\_name);

if (stat(fullpath, &statbuf) == -1)

{

perror("stat failed");

continue;

}

for (int i = 0; i < level; i++)

{

printf(" ");

}

if (S\_ISDIR(statbuf.st\_mode))

{

printf("[DIR] %s\n", entry->d\_name);

list\_dir(fullpath, level + 1);

}

else if (S\_ISREG(statbuf.st\_mode))

{

printf("[FILE] %s\n", entry->d\_name);

}

else

{

printf("[OTHER] %s\n", entry->d\_name);

}

}

closedir(dir);

}

int main()

{

const char \*start\_dir = ".";

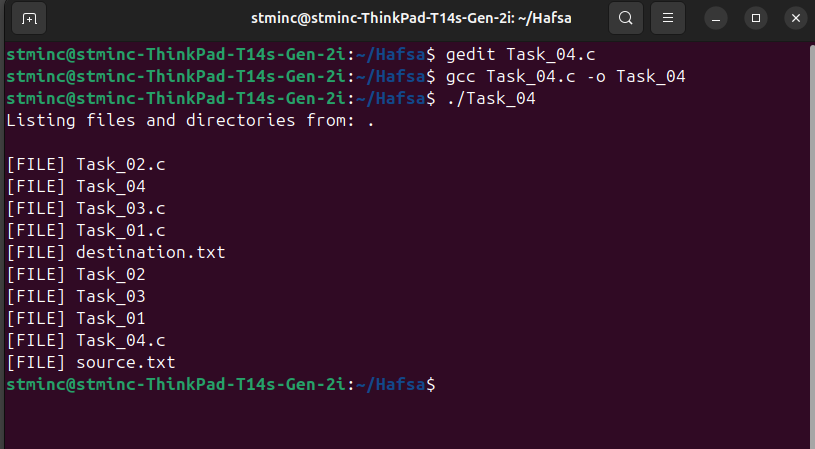
printf("Listing files and directories from: %s\n\n", start\_dir);

list\_dir(start\_dir, 0);

return 0;

}

Output:



Lab 04 -Tasks 2

Lab Exercises - III

**Task no. 01**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

pid\_t child = fork();

if (child < 0)

{

printf("Unsuccessful Child Process Creation!\n");

exit(EXIT\_FAILURE);

}

else if (child > 0)

{

wait(NULL);

printf("Parent - Child process is completed!\n");

}

else

{

printf("Child - Parent ID: %d\n", getppid());

for (int i=1; i<=10; i+=2)

{

printf("%d ", i);

}

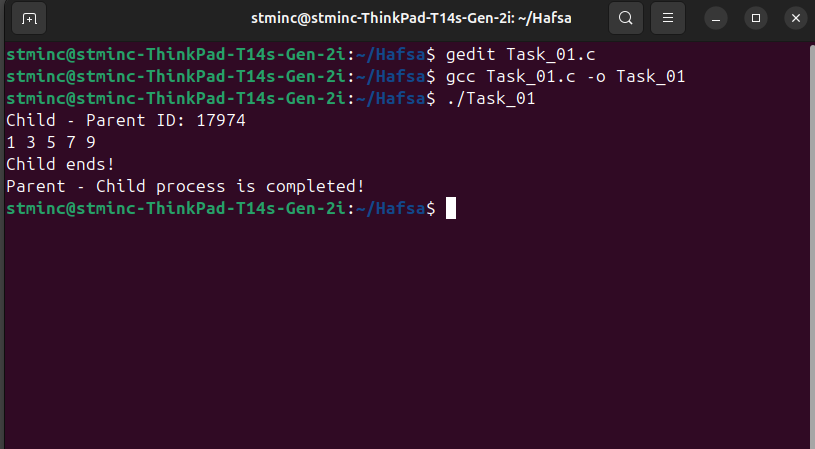
printf("\nChild ends!\n");

exit(EXIT\_SUCCESS);

}

}

Output:



**Task no. 02**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int n;

printf("No. of child processes: ");

scanf("%d", &n);

for (int i=0; i<n; i++)

{

pid\_t child = fork();

if (child == 0)

{

printf("Child %d: \n", (i+1));

printf("PID: %d\n", getpid());

printf("Parent PID: %d\n\n", getppid());

exit(0);

}

}

for (int i=0; i<n; i++)

{

wait(NULL);

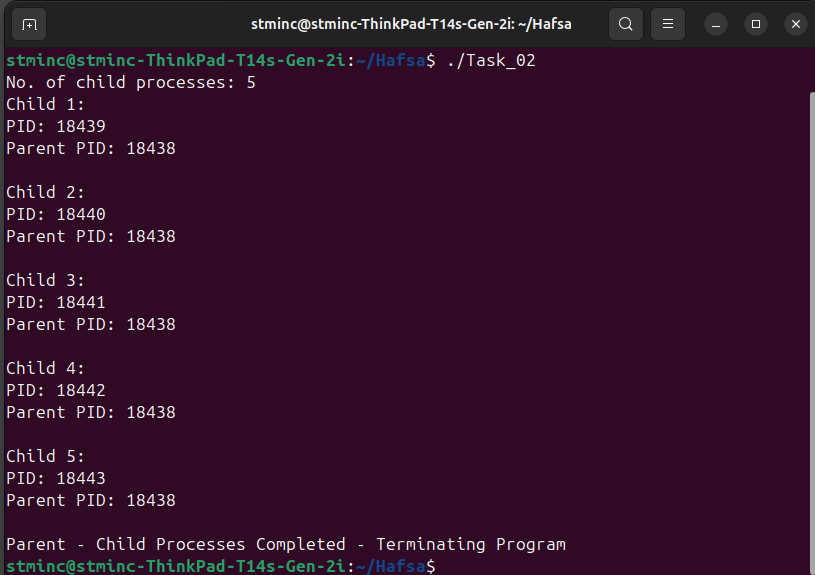
}

printf("Parent - Child Processes Completed - Terminating Program\n");

return 0;

}

Output:



**Task no. 03**

Code:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

int main()

{

pid\_t child1, child2, child3;

child1 = fork();

if (child1 == 0)

{

sleep(3);

printf("Child 01 - Terminating after child 02 and before parent\n");

exit(0);

}

child2 = fork();

if (child2 == 0)

{

sleep(2);

printf("Child 02 - Terminating after child 03 and before child 01\n");

exit(0);

}

child3 = fork();

if (child3 == 0)

{

sleep(1);

printf("Child 03 - Terminating first\n");

exit(0);

}

wait(NULL);

wait(NULL);

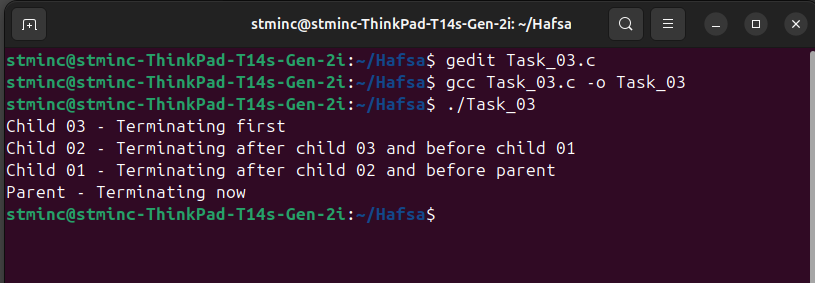
wait(NULL);

printf("Parent - Terminating now\n");

return 0;

}

Output:



**Task no. 04**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int Task\_04(int level, int max)

{

if (level > max)

{

return 0;

}

pid\_t child = fork();

if (child == 0)

{

printf("Process Level %d | PID: %d | PPID: %d\n", level, getpid(), getppid());

Task\_04(level+1, max);

exit(0);

}

else

{

wait(NULL);

printf("Parent at Level %d | PID: %d finished waiting for child.\n", level, getpid());

}

return 0;

}

int main()

{

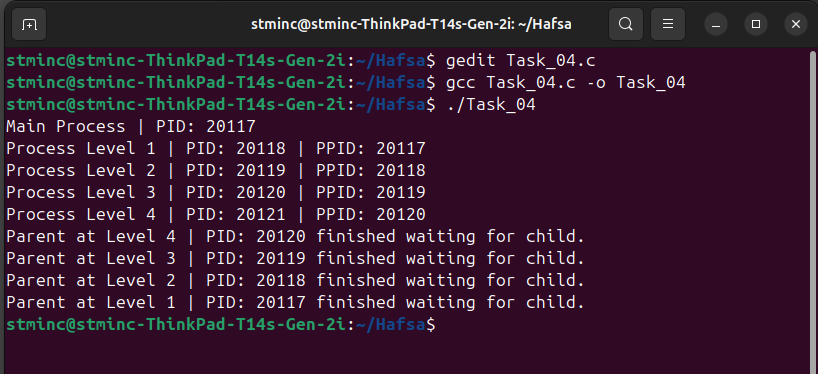
printf("Main Process | PID: %d\n", getpid());

Task\_04(1, 4);

return 0;

}

Output:



**Task no. 05**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int n;

printf("No. of child processes: ");

scanf("%d", &n);

for (int i=0; i<n; i++)

{

pid\_t child = fork();

if (child == 0)

{

printf("Child %d: \n", (i+1));

printf("PID: %d\n", getpid());

printf("Parent PID: %d\n\n", getppid());

exit(0);

}

}

for (int i=0; i<n; i++)

{

wait(NULL);

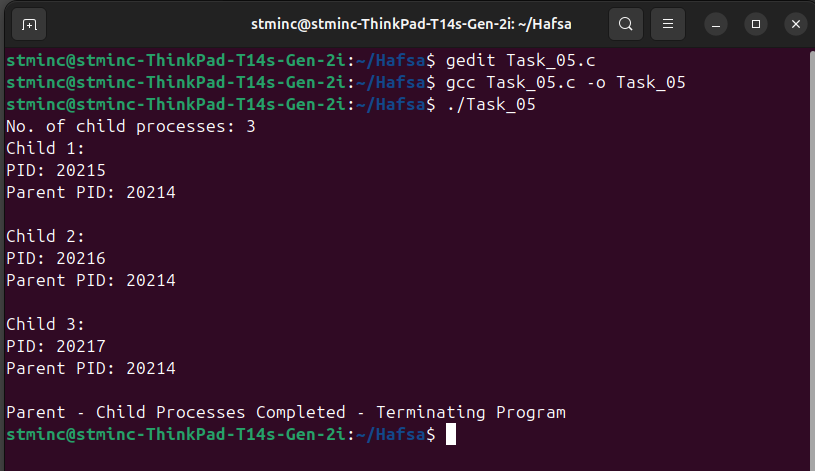
}

printf("Parent - Child Processes Completed - Terminating Program\n");

return 0;

}

Output:



**Task no. 06**

Code:

#include<stdio.h>

#include<unistd.h>

int main()

{

printf("I am process 10 and my PID is %d.\n", getpid());

int i = fork();

if (i == 0)

{

printf("I am process 18 and my PID is %d. PPID: %d\n", getpid(), getppid());

i = fork();

if (i == 0)

{

printf("I am process 21 and my PID is %d. PPID: %d\n", getpid(), getppid());

}

else

{

i = fork();

printf("I am process 15 and my PID is %d. PPID: %d\n", getpid(), getppid());

}

}

else

{

i = fork();

if (i == 0)

{

printf("I am process 6 and my PID is %d. PPID: %d\n", getpid(), getppid());

i = fork();

if (i == 0)

{

printf("I am process 8 and my PID is %d. PPID: %d\n", getpid(), getppid());

}

else

{

i = fork();

printf("I am process 4 and my PID is %d. PPID: %d\n", getpid(), getppid());

}

}

}

return 0;

}

Output:

