SCR2043 OPERATING SYSTEMS

Name	:	TEH RU QIAN	Marks
Student ID	:	A23CS0191	
Section	:	Section 03	

This lab assessment is designed to test your understanding and skills on some basic concepts and tools related to process monitoring and management in operating system. Please follow the instructions carefully and submit your answers in this word document and rename the file as **os-lab-assessment02-studentname-matricno.docx**.

Essential Steps Before Starting Lab Assessment 2:

1. **Download necessary source codes:**

   Use the `wget` command to retrieve the following source code files to your Linux (or WSL or MacOS) environment:

```
wget -O mainprocess.c https://rebrand.ly/mainprocess_c
wget -O subprocess1.c https://rebrand.ly/subprocess1_c
wget -O subprocess2.c https://rebrand.ly/subprocess2_c
```

2. **Compile the source files:**

   Use the `gcc` compiler to create executable files from the source code.

```
     gcc mainprocess.c -o mainprocess
     gcc subprocess1.c -o subprocess1
     gcc subprocess2.c -o subprocess2
```

3. **Execute the dummy processes:**

   Run all the dummy processes

```
     ./mainprocess &
```

   Press **enter** two times.

4. The dummy processes are running for 2 hours. If you took longer than 2 hours on questions 1-9, please restart the main process with `./mainprocess &`.

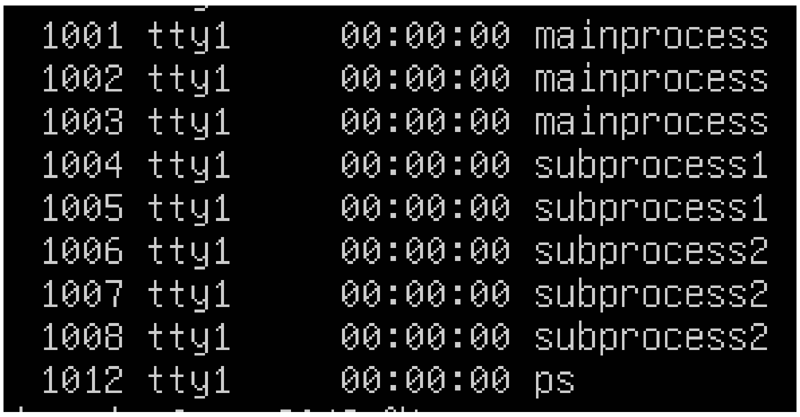
Lab Assessment 2 : Linux Process Monitoring and Management

Instructions:

1. Carefully execute each command as instructed in the questions.
2. Write down the exact command used for each task.
3. Capture a screenshot of the command's output.

Question 1

Use the `ps` command with the appropriate option to display a complete list of all running processes within the Linux operating system.

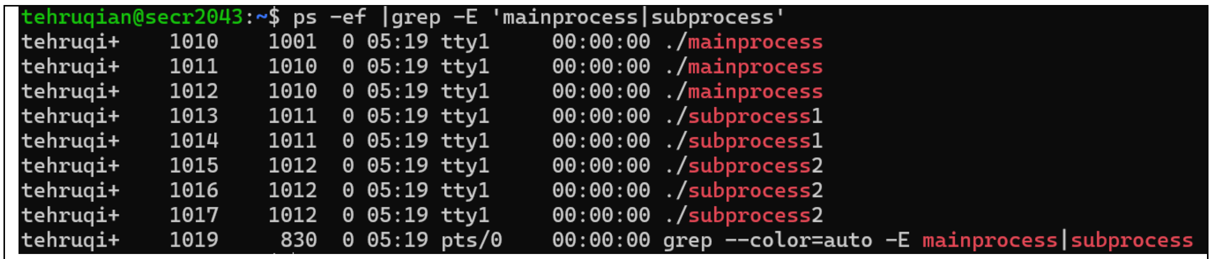
Command
`ps -e`
Output


```
 1001 tty1     00:00:00 mainprocess
 1002 tty1     00:00:00 mainprocess
 1003 tty1     00:00:00 mainprocess
 1004 tty1     00:00:00 subprocess1
 1005 tty1     00:00:00 subprocess1
 1006 tty1     00:00:00 subprocess2
 1007 tty1     00:00:00 subprocess2
 1008 tty1     00:00:00 subprocess2
 1012 tty1     00:00:00 ps
```

Question 2

Employ the `ps` command with necessary options to unveil comprehensive details about each running process.

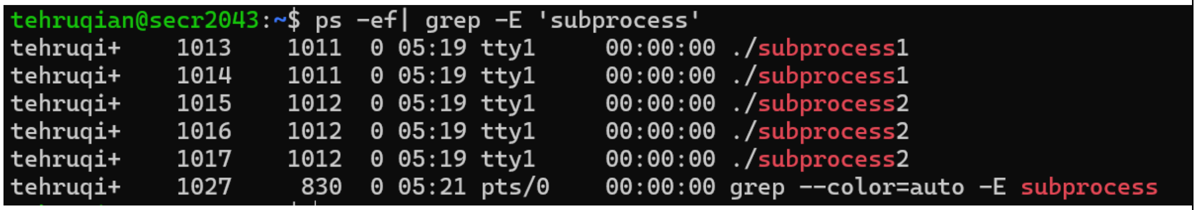
Command
`ps -ef |grep -E 'mainprocess|subprocess'`
Output



```
tehruqian@secr2043:~$ ps -ef |grep -E 'mainprocess|subprocess'
tehruqi+    1010    1001  0 05:19 tty1     00:00:00 ./mainprocess
tehruqi+    1011    1010  0 05:19 tty1     00:00:00 ./mainprocess
tehruqi+    1012    1010  0 05:19 tty1     00:00:00 ./mainprocess
tehruqi+    1013    1011  0 05:19 tty1     00:00:00 ./subprocess1
tehruqi+    1014    1011  0 05:19 tty1     00:00:00 ./subprocess1
tehruqi+    1015    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1016    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1017    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1019     830  0 05:19 pts/0    00:00:00 grep --color=auto -E mainprocess|subprocess
```

Question 3

Use the `ps` command with some tools to only list processes named "subprocess" and show some info about them.

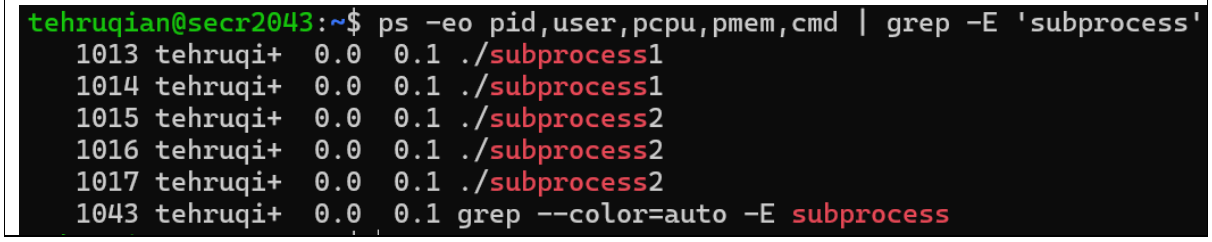
Command
`ps -ef| grep -E 'subprocess'`
Output


```
tehruqian@secr2043:~$ ps -ef| grep -E 'subprocess'
tehruqi+    1013    1011  0 05:19 tty1     00:00:00 ./subprocess1
tehruqi+    1014    1011  0 05:19 tty1     00:00:00 ./subprocess1
tehruqi+    1015    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1016    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1017    1012  0 05:19 tty1     00:00:00 ./subprocess2
tehruqi+    1027     830  0 05:21 pts/0    00:00:00 grep --color=auto -E subprocess
```

Question 4

Execute the `ps` command, specifying options that reveal only the following columns:

- Process ID (`pid`)
- Owner of the process (`user`)
- CPU percentage (`pcpu`)
- Memory percentage (`pmem`)
- Command (`cmd`)

Command
`ps -eo pid,user,pcpu,pmem,cmd | grep -E 'subprocess'`
Output


```
tehruqian@secr2043:~$ ps -eo pid,user,pcpu,pmem,cmd | grep -E 'subprocess'
   1013 tehruqi+  0.0  0.1 ./subprocess1
   1014 tehruqi+  0.0  0.1 ./subprocess1
   1015 tehruqi+  0.0  0.1 ./subprocess2
   1016 tehruqi+  0.0  0.1 ./subprocess2
   1017 tehruqi+  0.0  0.1 ./subprocess2
   1043 tehruqi+  0.0  0.1 grep --color=auto -E subprocess
```

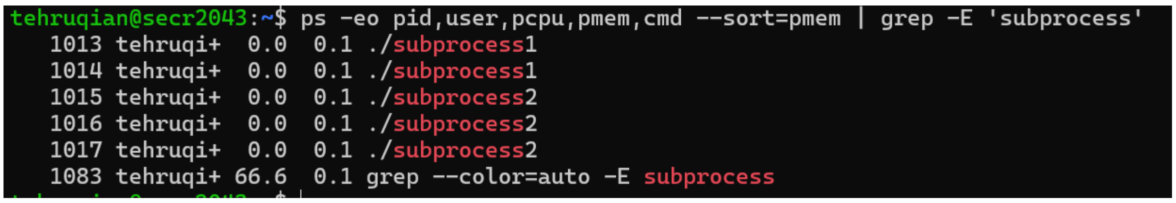
Question 5

Building on the ps command used in Question 4, can you add an option to sort the listed processes by their memory usage (`pmem`)?

Command

```
ps -eo pid,user,pcpu,pmem,cmd --sort=pmem | grep -E
'subprocess'
```

Output



```
tehruqian@secr2043:~$ ps -eo pid,user,pcpu,pmem,cmd --sort=pmem | grep -E 'subprocess'
   1013 tehruqi+  0.0  0.1 ./subprocess1
   1014 tehruqi+  0.0  0.1 ./subprocess1
   1015 tehruqi+  0.0  0.1 ./subprocess2
   1016 tehruqi+  0.0  0.1 ./subprocess2
   1017 tehruqi+  0.0  0.1 ./subprocess2
   1083 tehruqi+ 66.6  0.1 grep --color=auto -E subprocess
```

Question 6

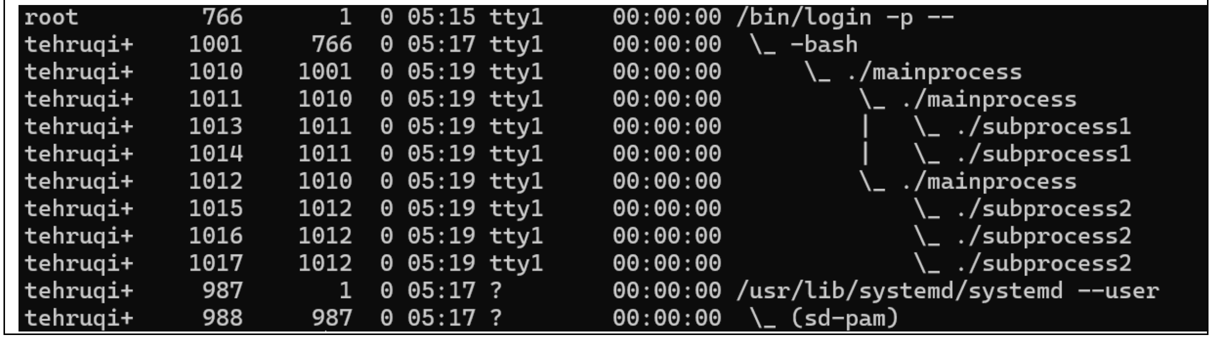
Construct a command using `ps`, suitable options, and any additional tools to visualize the hierarchical structure (tree-like) of the following processes:

- "mainprocess"
- "subprocess1"
- "subprocess2"

Command

```
ps -ef --forest -C mainprocess -C subprocess1 -C subprocess2
```

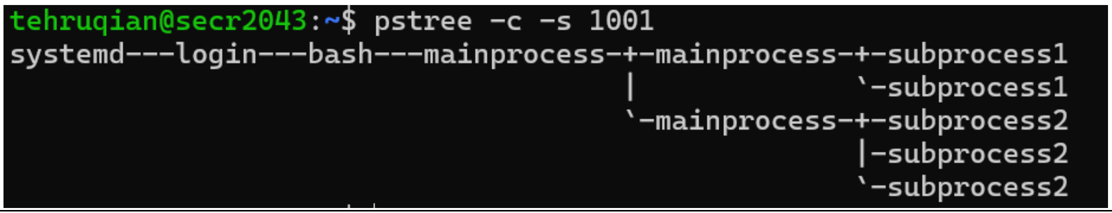
Output



```
root         766       1  0 05:15 tty1     00:00:00 /bin/login -p --
tehruqi+    1001     766  0 05:17 tty1     00:00:00  \_ -bash
tehruqi+    1010    1001  0 05:19 tty1     00:00:00      \_ ./mainprocess
tehruqi+    1011    1010  0 05:19 tty1     00:00:00          \_ ./mainprocess
tehruqi+    1013    1011  0 05:19 tty1     00:00:00          |   \_ ./subprocess1
tehruqi+    1014    1011  0 05:19 tty1     00:00:00          |   \_ ./subprocess1
tehruqi+    1012    1010  0 05:19 tty1     00:00:00          \_ ./mainprocess
tehruqi+    1015    1012  0 05:19 tty1     00:00:00              \_ ./subprocess2
tehruqi+    1016    1012  0 05:19 tty1     00:00:00              \_ ./subprocess2
tehruqi+    1017    1012  0 05:19 tty1     00:00:00              \_ ./subprocess2
tehruqi+     987       1  0 05:17 ?        00:00:00 /usr/lib/systemd/systemd --user
tehruqi+     988     987  0 05:17 ?        00:00:00  \_ (sd-pam)
```

Question 7

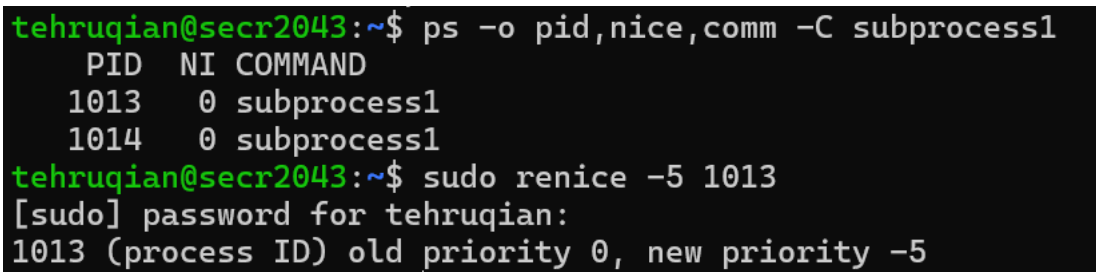
Use `pstree` command with option that show the number of threads to each process.

Command
`pstree -c -s 1001`
Output


```
tehruqian@secr2043:~$ pstree -c -s 1001
systemd---login---bash---mainprocess-+-mainprocess-+-subprocess1
                                     |             `-subprocess1
                                     `-mainprocess-+-subprocess2
                                                   |-subprocess2
                                                   `-subprocess2
```

Question 8

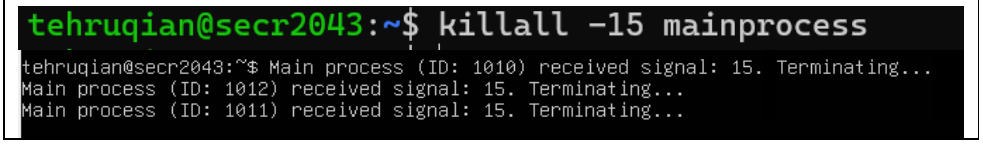
Use `renice` command to change priority level of one of process “subprocess1”.

Command
`sudo renice -5 1013`
Output


```
tehruqian@secr2043:~$ ps -o pid,nice,comm -C subprocess1
    PID  NI COMMAND
   1013   0 subprocess1
   1014   0 subprocess1
tehruqian@secr2043:~$ sudo renice -5 1013
[sudo] password for tehruqian:
1013 (process ID) old priority 0, new priority -5
```

Question 9

Terminate all running processes with the name “mainprocess”.

Command
`killall -15 mainprocess`
Output


```
tehruqian@secr2043:~$ killall -15 mainprocess
tehruqian@secr2043:~$ Main process (ID: 1010) received signal: 15. Terminating...
Main process (ID: 1012) received signal: 15. Terminating...
Main process (ID: 1011) received signal: 15. Terminating...
```

Question 10

Write a short C or Python code (choose only one language) demonstrating multiprocessing with `fork()` and `wait()`. Compile and/or run the code. Show the output.

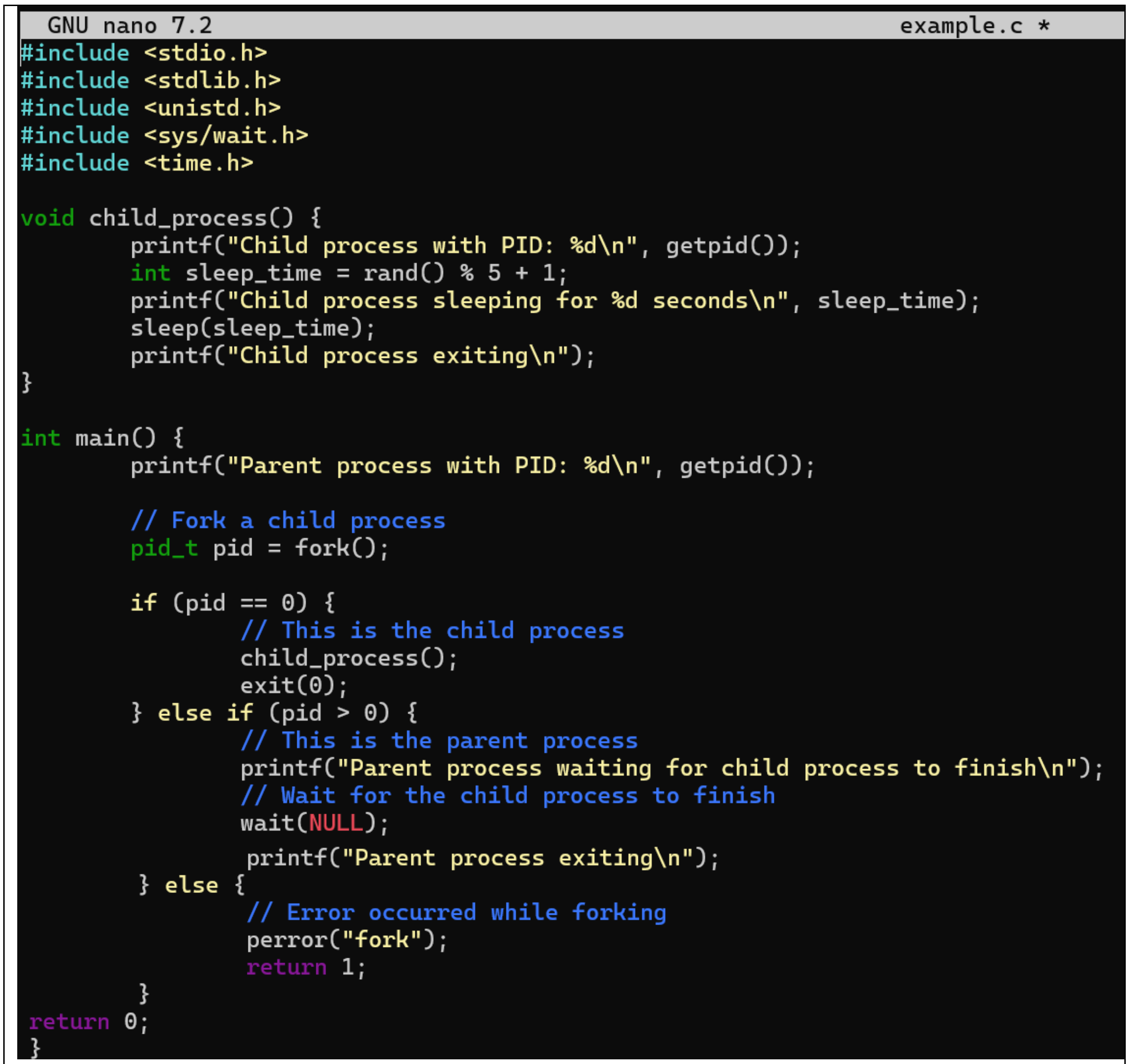
Source Code:

```
nano example.c
gcc example.c -o example
gcc./example
```

example.c

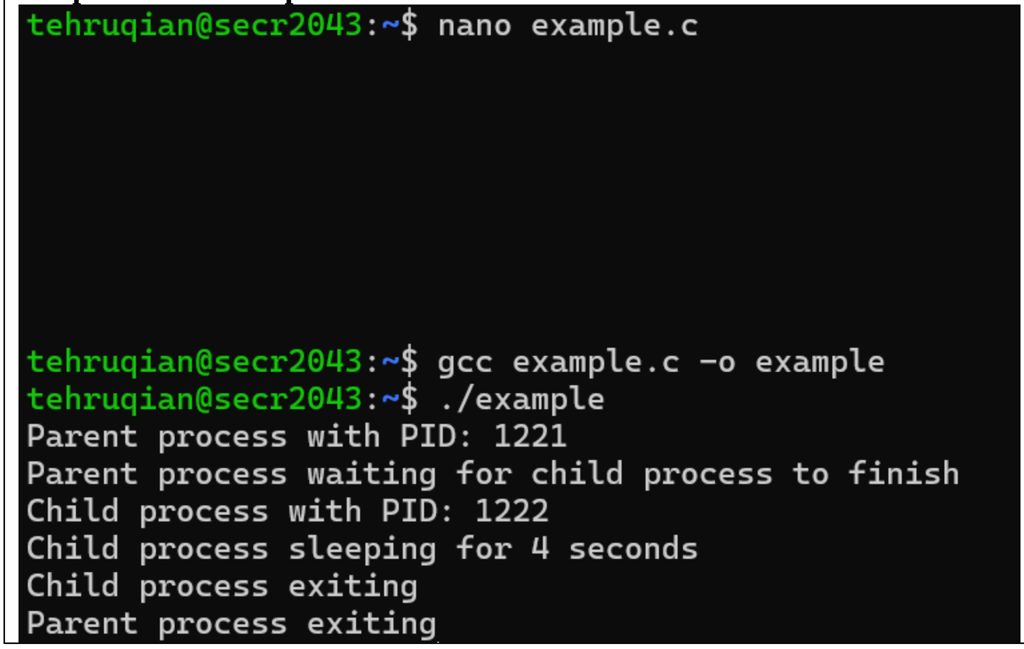
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
void child_process() {
  printf("Child process with PID: %d\n", getpid());
  int sleep_time = rand() % 5 + 1;
  printf("Child process sleeping for %d seconds\n",
sleep_time);
  sleep(sleep_time);
  printf("Child process exiting\n");
}
int main() {
  printf("Parent process with PID: %d\n", getpid());
  // Fork a child process
  pid_t pid = fork();
  if (pid == 0) {
    // This is the child process
    child_process();
    exit(0);
  } else if (pid > 0) {
    // This is the parent process
    printf("Parent process waiting for child process to
finish\n");
    // Wait for the child process to finish
    wait(NULL);
    printf("Parent process exiting\n");
  } else {
    // Error occurred while forking
    perror("fork");
    return 1;
  }
return 0;
}
```

Output:



```
 GNU nano 7.2                                                   example.c *
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

void child_process() {
        printf("Child process with PID: %d\n", getpid());
        int sleep_time = rand() % 5 + 1;
        printf("Child process sleeping for %d seconds\n", sleep_time);
        sleep(sleep_time);
        printf("Child process exiting\n");
}

int main() {
        printf("Parent process with PID: %d\n", getpid());

        // Fork a child process
        pid_t pid = fork();

        if (pid == 0) {
                // This is the child process
                child_process();
                exit(0);
        } else if (pid > 0) {
                // This is the parent process
                printf("Parent process waiting for child process to finish\n");
                // Wait for the child process to finish
                wait(NULL);
                 printf("Parent process exiting\n");
         } else {
                 // Error occurred while forking
                 perror("fork");
                 return 1;
         }
 return 0;
 }
```

Output of the example.c



```
tehruqian@secr2043:~$ nano example.c







tehruqian@secr2043:~$ gcc example.c -o example
tehruqian@secr2043:~$ ./example
Parent process with PID: 1221
Parent process waiting for child process to finish
Child process with PID: 1222
Child process sleeping for 4 seconds
Child process exiting
Parent process exiting
```