Operating Systems Lab Report

Muhammad Shafeen
Student ID: 22P-9278

Lab 7: Operating Systems

1 Exercise 2 :

1.1 Question :

Model a fork() call in C/C++ so that your pro- gram can create a total of EXACTLY 6 pro- cesses (including the parent). (Note: You may check the number of processes created using the method from Exercise 1 (Section-3.2.1.1 by using a sleep of 60 seconds or more and enter- ing eithe ps, or pstree in another terminal).

1.2 Answer :

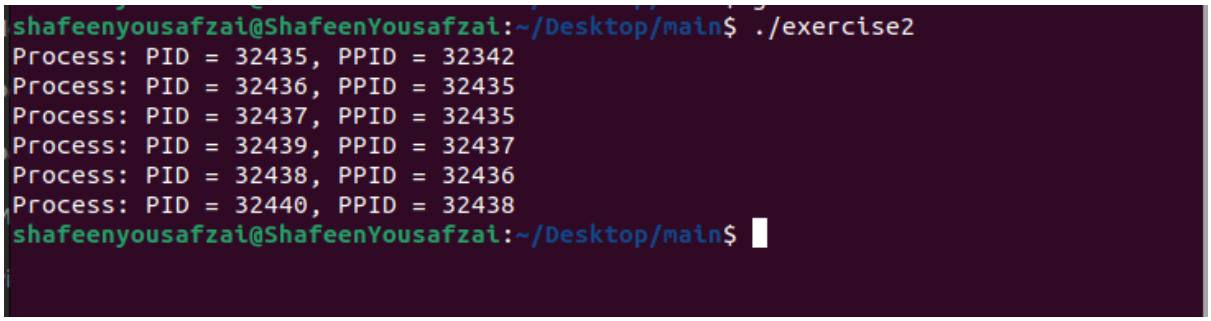


Figure 1: This is the output for 1 parent and 5 child processes , total 6

2 Running States

Code

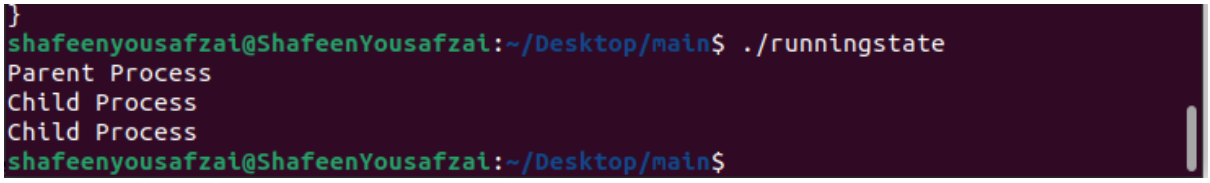


Figure 3: Output of the code

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat lab7_runningstate.c
#include<unistd.h>
#include<stdio.h>
int main()
{
 int p;
char *arg[]={"/user/bin/ls",0};
p = fork();
if(p==0)
{
printf("Child Process\n");
execv(arg[0],arg);
printf("Child Process\n");
}
else
{
printf("Parent Process\n");
}
}
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 2: Code for Running States

Questions and Answers

Q1: What is the first argument to the `execv()` call? What is its content?

Answer: The first argument to the `execv()` function is the path to the executable file that you want to run. In our case, the path is `"/bin/ls"`, which specifies the `ls` command to list directory contents.

Q2: What is the second argument to it? What is its content?

Answer: The second argument is an array of argument strings (`argv`) passed to the executable. This array must be terminated with a NULL pointer. It typically starts with the name of the executable itself, followed by any additional arguments. For example, `argv[0]` is the program name, and `argv[1]` onwards are the arguments.

Q3: What is `arg`?

Answer: `arg` is the array that contains the command-line arguments and the path to the executable that you want to run. It is used by `execv()` to replace the current process image with a new process image. The array is terminated with a NULL pointer to indicate the end of arguments.

Q4: Look at the code of the child process (`p == 0`). How many times does the statement "Child Process" appear? Why?

Answer: The statement "Child Process" appears only once because after the `execv()` call is executed, the child process is replaced by the new executable. Therefore, the original code (including any further print statements) does not continue to execute in the child process.

3 Waiting States Exercise - 3.2.3.1 Sleep()

Problem

Write a C program that can display a count from 10 to 0 (in reverse order) using a `for` or a `while` loop. Each number should be displayed after a delay of 1 second.

Solution

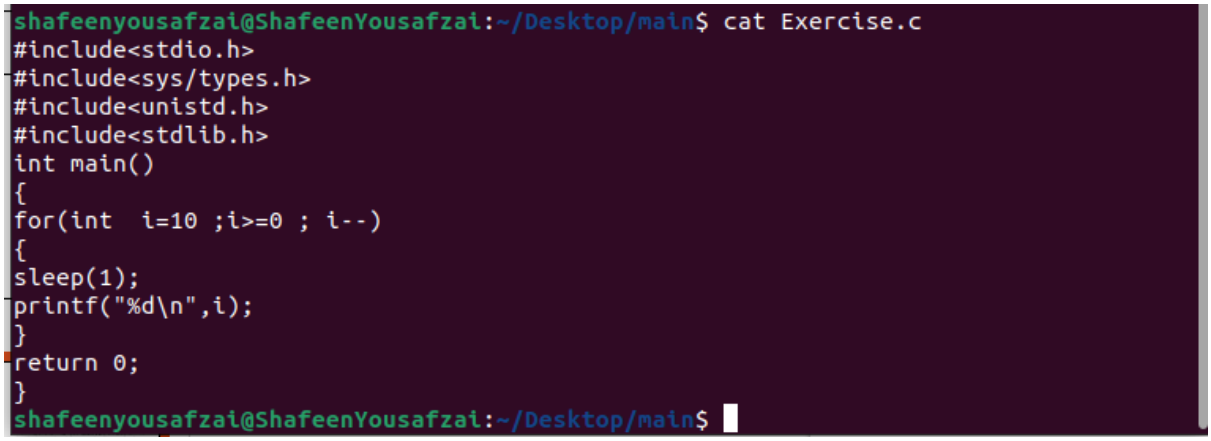


Figure 4: C Program Solution for Waiting States Exercise

Below is the C program that accomplishes the task:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
for(int  i=10 ;i>=0 ; i--)
{
sleep(1);
printf("%d\n",i);
}
return 0;
}
```

4 Exit() Call

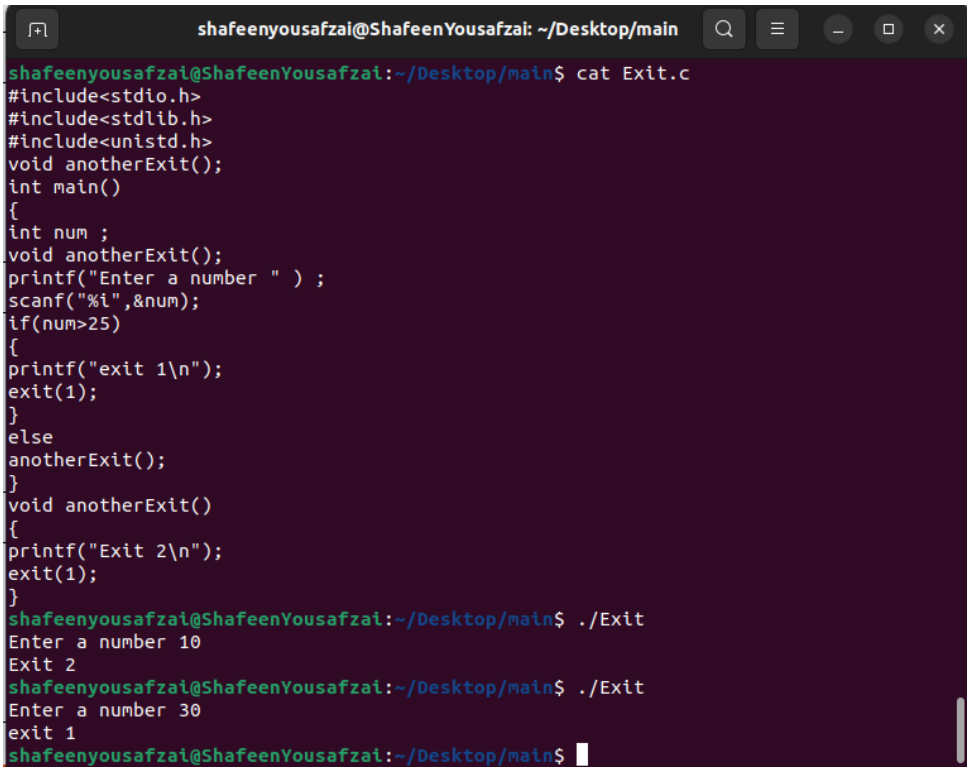


Figure 5: Caption

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void anotherExit();
int main()
{
int num ;
void anotherExit();
printf("Enter a number " ) ;
scanf("%i",&num);
if(num>25)
{
printf("exit 1\n");
exit(1);
}
else
anotherExit();
}
void anotherExit()
{
printf("Exit 2\n");
exit(1);
}
```

Question

Can you identify which `exit()` call is being used each time a program exits?

Answer

Yes, the `exit()` function is called with different status codes based on the input:

- When entering a number below 25, `exit(2)` is called.
- When entering a number above 25, `exit(1)` is called.

These exit codes can be used to indicate different termination reasons to the operating system or calling processes.

5 Atexit() Call

5.1 Code :

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat atexit.c
#include<stdio.h>
#include<stdlib.h>
void f1(void);
void f2(void);
void f3(void);
int main(void) {
void f1(void), f2(void), f3(void);
atexit(f1);
atexit(f2);
atexit(f3);
printf("Getting ready to exit\n");
exit(0);
}

void f1(void) {
printf("In f1\n");
}
void f2(void) {
printf("In f2\n");
}
void f3(void) {
printf("In f3\n");
}
```

Figure 6: Caption

5.2 Execution of Code :

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ./atexit
Getting ready to exit
In f3
In f2
In f1
```

Figure 7: Output of the above code

Explanation

The `atexit()` function allows you to register functions to be called upon normal program termination. This can be useful for performing cleanup tasks such as freeing allocated memory, closing files, or other housekeeping activities.

Example Usage:

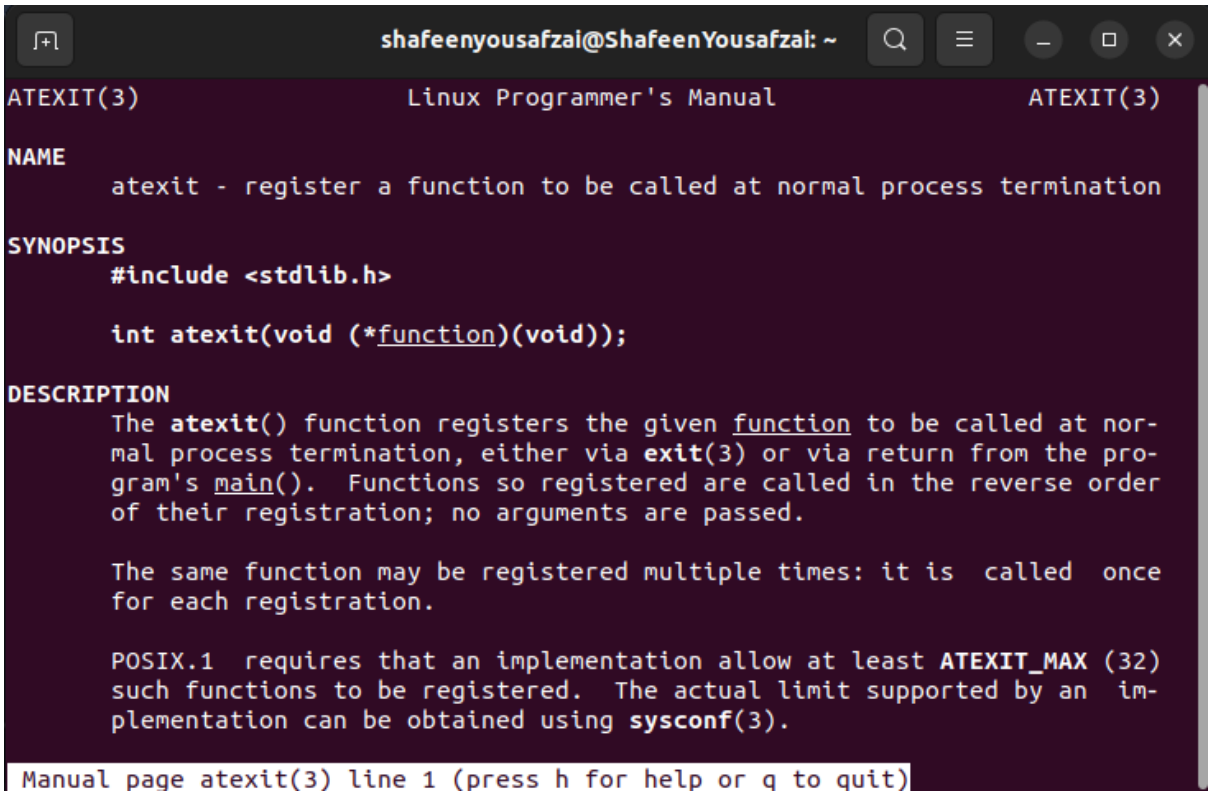
```
#include<stdio.h>
#include<stdlib.h>
void f1(void);
void f2(void);
void f3(void);
int main(void) {
void f1(void), f2(void), f3(void);
atexit(f1);
atexit(f2);
atexit(f3);
printf("Getting ready to exit\n");
exit(0);
}

void f1(void) {
printf("In f1\n");
}
void f2(void) {
printf("In f2\n");
}
void f3(void) {
printf("In f3\n");
}
```

In this example, the `cleanup` function is registered to be called when the program exits normally.

5.3 Question 1 :

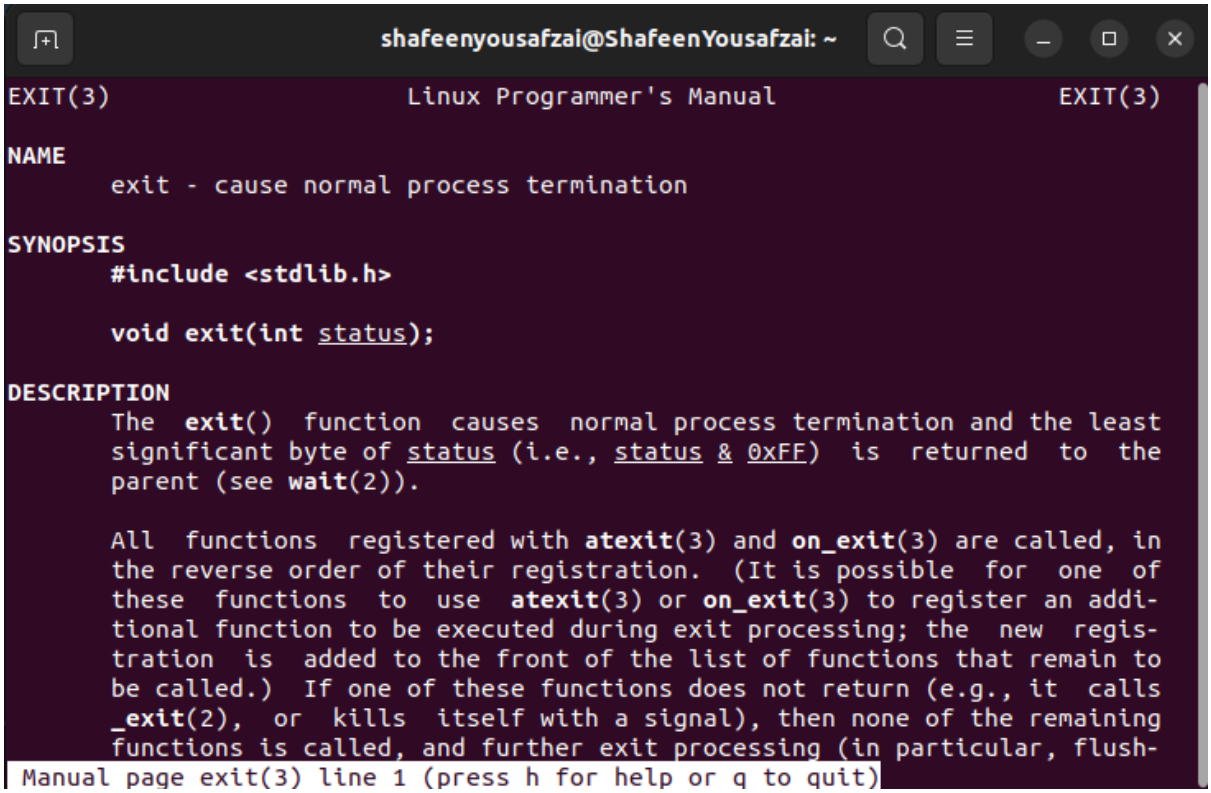
What is the difference between exit() and atexit()? What do they do? (Check man atexit and man 3 exit).

5.4 Answer 1:



```
shafeenyousafzai@ShafeenYousafzai: ~
ATEXIT(3)                Linux Programmer's Manual                ATEXIT(3)

NAME
       atexit - register a function to be called at normal process termination

SYNOPSIS
       #include <stdlib.h>

       int atexit(void (*function)(void));

DESCRIPTION
       The atexit() function registers the given function to be called at nor-
       mal process termination, either via exit(3) or via return from the pro-
       gram's main().  Functions so registered are called in the reverse order
       of their registration; no arguments are passed.

       The same function may be registered multiple times: it is  called  once
       for each registration.

       POSIX.1  requires that an implementation allow at least ATEXIT_MAX (32)
       such functions to be registered.  The actual limit supported by an  im-
       plementation can be obtained using sysconf(3).

 Manual page atexit(3) line 1 (press h for help or q to quit)
```

Figure 8: This is the man for atexit()



```
shafeenyousafzai@ShafeenYousafzai: ~
EXIT(3)                  Linux Programmer's Manual                  EXIT(3)

NAME
       exit - cause normal process termination

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit()  function  causes  normal process termination and the least
       significant byte of status (i.e., status & 0xFF)  is  returned  to  the
       parent (see wait(2)).

       All  functions  registered with atexit(3) and on_exit(3) are called, in
       the reverse order of their registration.  (It is possible  for  one  of
       these  functions  to  use  atexit(3) or on_exit(3) to register an addi-
       tional function to be executed during exit processing; the  new  regis-
       tration  is  added to the front of the list of functions that remain to
       be called.)  If one of these functions does not return (e.g., it  calls
       _exit(2),  or  kills  itself with a signal), then none of the remaining
       functions is called, and further exit processing (in particular, flush-
 Manual page exit(3) line 1 (press h for help or q to quit)
```

Figure 9: This is the man for exit()

exit(): Terminates the program execution.
behaviour:
Performs cleanup such as flushing standard I/O buffers. Calls all functions registered with atexit() in the reverse order of their registration. Returns control to the host environment with the specified exit status.
atexit() : Registers functions to be automatically called upon normal program termination.
behaviour:
(1em) Allows multiple functions to be registered. Registered functions are executed in the reverse order of their registration when exit() is called. Useful for cleanup tasks like freeing resources or saving state.

5.5 Question 2 :

What does the 0 provided in the exit() call mean? What will happen if we change it to 1? (Check manual page for exit)

5.6 Answer 2 :

The 0 in exit(0) indicates successful program termination. Changing it to 1 signals an error or abnormal termination.

5.7 Question 3 :

Q2) What does the 0 provided in the exit() call mean? What will happen if we change it to 1? (Check manual page for exit)

5.8 Answer 3 :

If ‘exit()‘ is called in ‘f1‘, ‘f2‘, or ‘f3‘, the program will terminate immediately, and any remaining ‘atexit()‘-registered functions will not be executed.

5.9 Question 4 :

Why do you think we are getting reverse order of execution of atexit calls?

5.10 Answer 4 :

The ‘atexit()‘ functions are executed in reverse order of registration to ensure proper cleanup, similar to how stack unwinding works. This way, the most recently registered function is called first, ensuring that resources are released in the correct sequence.

6 3.2.4.3 Abort Call

6.1 Question 1 :

Q1 Check the man pages for abort. How does the abort call terminate the pro-gram? What is the name of the particular signal?

6.2 Answer 1 :

```
ABORT(3)                                Linux Programmer's Manual                                ABORT(3)

NAME
       abort - cause abnormal process termination

SYNOPSIS
       #include <stdlib.h>

       void abort(void);

DESCRIPTION
       The  abort()  function  first  unblocks  the SIGABRT signal, and then raises that signal for the calling process (as though
       raise(3) was called).  This results in the abnormal termination of the process unless the SIGABRT signal is caught and  the
       signal handler does not return (see longjmp(3)).

       If  the  SIGABRT  signal  is  ignored,  or  caught by a handler that returns, the abort() function will still terminate the
       process.  It does this by restoring the default disposition for SIGABRT and then raising the signal for a second time.

RETURN VALUE
       The abort() function never returns.

ATTRIBUTES
       For an explanation of the terms used in this section, see attributes(7).

       | Interface | Attribute | Value |
       |---|---|---|
       | abort() | Thread safety | MT-Safe |

CONFORMING TO
       SVr4, POSIX.1-2001, POSIX.1-2008, 4.3BSD, C89, C99.

NOTES
       Up until glibc 2.26, if the abort() function caused process termination, all open streams were closed and flushed (as  with
       fclose(3)).   However,  in  some  cases this could result in deadlocks and data corruption.  Therefore, starting with glibc
       2.27, abort() terminates the process without flushing streams.  POSIX.1  permits  either  possible  behavior,  saying  that
       abort() "may include an attempt to effect fclose() on all open streams".

SEE ALSO
       gdb(1), sigaction(2), assert(3), exit(3), longjmp(3), raise(3)

COLOPHON
 Manual page abort(3) line 1 (press h for help or q to quit)
```

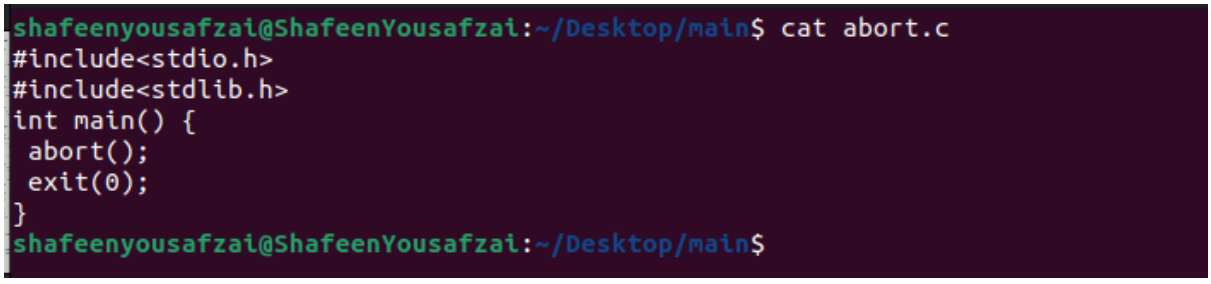
Figure 10: This is the screenshot of man abort

The ‘abort()‘ call terminates the program by generating the **SIGABRT*** signal, which causes an abnormal program termination. This signal is used to indicate a serious program error and usually results in a core dump for debugging purposes.

6.3 Question 2 :

Q2 Execute your program. What is the out- put of our program?

6.4 Answer 2 :



```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat abort.c
#include<stdio.h>
#include<stdlib.h>
int main() {
 abort();
 exit(0);
}
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 11: Runnig the code and checking its output

```
}
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ./abort
Aborted (core dumped)
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 12: output of the running code

The program terminates immediately with an abnormal termination due to the abort() call, so exit(0) is not reached, and no output is produced.

6.5 Question 3 :

Q3 Include the abort call in function f3 in our code provided for Atexit() call. How does our program terminate using this?

6.6 Answer 3 :

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat question3abort.c
#include<stdio.h>
#include<stdlib.h>
void f1(void);
void f2(void);
void f3(void);
int main(void) {
void f1(void), f2(void), f3(void);
atexit(f1);
atexit(f2);
atexit(f3);
printf("Getting ready to exit\n");
exit(0);
}

void f1(void) {
printf("In f1\n");
}
void f2(void) {
printf("In f2\n");
}
void f3(void) {
abort();
printf("In f3\n");
}
```

Figure 13: Showing the code with abort

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ./question3abort
Getting ready to exit
Aborted (core dumped)
```

Figure 14: Executing code

If abort() is called in f3, the program terminates immediately with a SIGABRT signal, preventing any further execution of remaining atexit() functions.

7 3.2.4.4 Kill Call

KILL -l 9)th SIGKILL is a signal in Unix-like operating systems that is used to immediately terminate a process. It has the following characteristics

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat killcode.c
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include <signal.h>
int main()
{
printf("Hello");
kill(getpid(), 9);
printf("Goodbye");
}
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 15: The kill code , Showing the code

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ./killcode
Killed
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 16: This is the demonstration code provided in manual

7.1 Question Code :

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat killcode2.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include<sys/types.h>
int main()
{
int i, pid;
pid = fork();
if (pid > 0)
// Parent
{
sleep(2);
exit(0);
}
else if (pid == 0)
// Child
{
for (i=0; i < 5; i++)
{
printf("My parent is %d\n", getppid());
sleep(1);
}
}
}
```

Figure 17: This is the Questions code

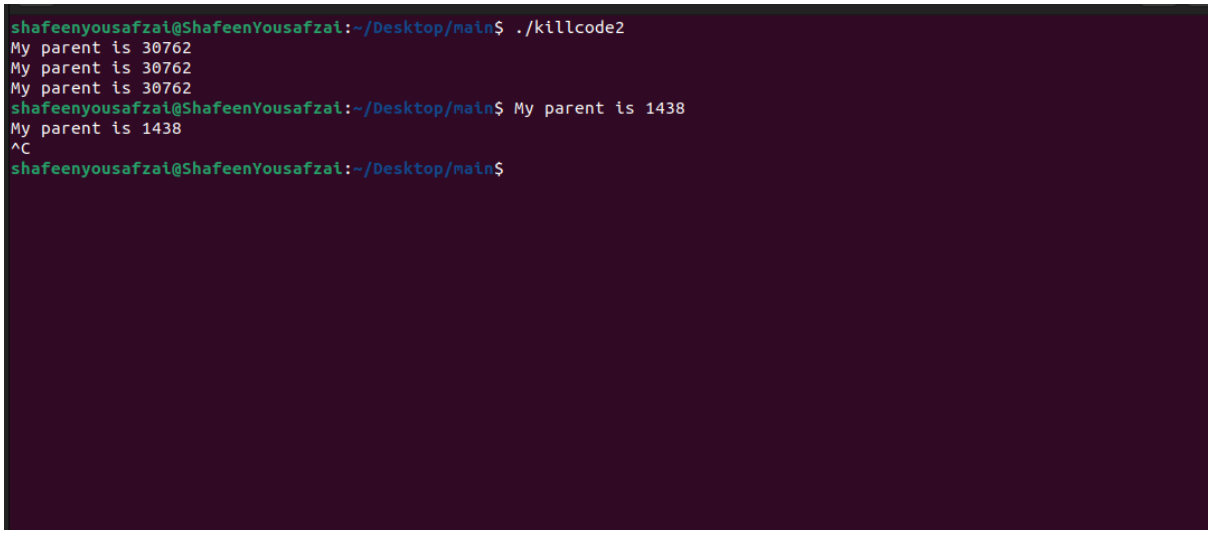


Figure 18: Exection of the code

7.2 Question 1:

What are the PPID values you are receiv-ing from the for loop?

7.3 Answer 1:

The PPID values received from the loop are: 38762,38762,38762,1438,1438

7.4 Question 2 :

What has happened when the numbers of the PPID change?

7.5 Answer 2:

After two seconds (when the parent process calls exit(0)), the child becomes an orphan. The init process (typically PID 1) adopts the child, so the PPID changes to the PID of the init process.

7.6 Question 3 :

What is now PID of the init process?

7.7 Answer 3 :

The PID of the init process is typically 1. Once the parent process exits, the child process's PPID changes to 1, indicating that it has been adopted by the init process. The word mentioned next to signal 9 (‘kill -l‘) is: SIGKILL

Answer:

8 3.3.1 Parent Dies Before Child

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ cat zombiecode.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include<sys/types.h>
int main()
{
int i, pid;
pid = fork();
if (pid > 0)
{
sleep(120);
}
else if (pid == 0)
{
exit(0);
}
}
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 19: Original Code for analysis

8.1 Exection of code

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ps au
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
shafeen+    1525  0.0  0.0 171048  5760 tty2     Ssl+ 14:15   0:00 /usr/libexec/gdm-wayland
shafeen+    1528  0.0  0.0 231700 15616 tty2     Sl+  14:15   0:00 /usr/libexec/gnome-sessi
shafeen+   28927  0.0  0.0  19928  5376 pts/0    Ss   16:37   0:00 bash
shafeen+   31988  0.0  0.0  19796  5504 pts/1    Ss   17:39   0:00 bash
shafeen+   32091  0.0  0.0   2644  1024 pts/0    S+   17:42   0:00 ./zombiecode
shafeen+   32092  0.0  0.0      0     0 pts/0    Z+   17:42   0:00 [zombiecode] <defunct>
shafeen+   32094  0.0  0.0  21328  3456 pts/1    R+   17:42   0:00 ps au
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ps au
```

Figure 20: This is the part before sleep happens

8.2 Both child and parent processes have been killed

```
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$ ps au
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
shafeen+    1525  0.0  0.0 171048  5760 tty2     Ssl+ 14:15   0:00 /usr/libexec/gdm-wayland
shafeen+    1528  0.0  0.0 231700 15616 tty2     Sl+  14:15   0:00 /usr/libexec/gnome-sessi
shafeen+   28927  0.0  0.0  19928  5376 pts/0    Ss+  16:37   0:00 bash
shafeen+   31988  0.0  0.0  19796  5504 pts/1    Ss   17:39   0:00 bash
shafeen+   32095  0.0  0.0  21328  3456 pts/1    R+   17:42   0:00 ps au
shafeenyousafzai@ShafeenYousafzai:~/Desktop/main$
```

Figure 21: This is the part after sleep happens