Advanced C Training Program

Module 3 - Assignment

Qn 1: Which signals are triggered when the following actions are performed

1.1 user press Ctrl+C
    SIGINT (2) → this interrupt signal is invoked when we press Ctrl+C

1.2 kill() system call is invoked
    When the kill() system call is commonly used to send signals. The kill function has two arguments, pid and sig

    Kill(pid,sig)

    This invoked the SIGTERM signal

1.3 CPU tried to execute an illegal instruction

SIGILL (4)

1.4 When the program access the unassigned memory
    SIGSEGV (11)

Qn 2: list the gdb command for the following operations:

to run the current executale file → r (or) run

to create breakpoints:

at beginning of program → b main

at current line → b

at line N → b N

at beginning of function → b fn

to resume execution once after breakpoint → c (or) continue

to clear breakpoint created for a function → d N

print the parameters of the function in the backtrace → bt

Qn 3: What is the output of the following code?

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork() && (!fork())) {
        if (fork() || fork()) {
            fork();
        }
    }
    printf("2 ");
    return 0;
}
```

Output: 2 2 2 2 2 2 2

Qn 4: What is the output of the following code?

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    if (fork()) {
        if (!fork()) {
            fork();
            printf("1 ");
        }
        else {
            printf("2 ");
        }
    }
    else {
        printf("3 ");
    }
    printf("4 ");
    return 0;
}
```

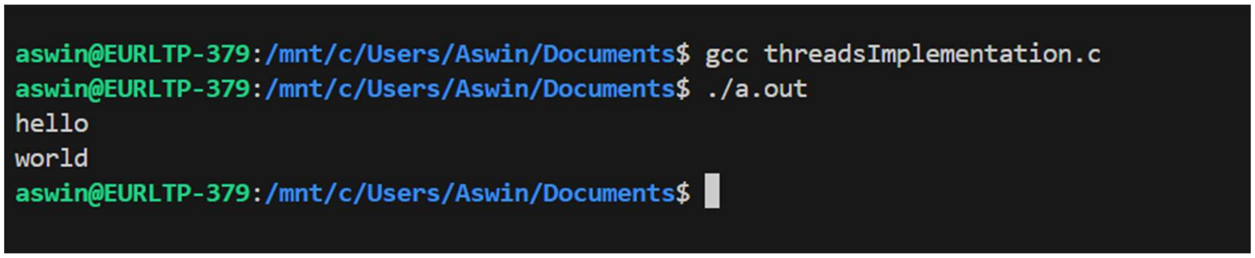
Output: 2 4 1 4 1 4 3 4

```
}
```

Qn 5: Create two thread functions to print hello and world separately and create threads for each and execute them one after other in C.

Code:

```
#include <stdio.h>

#include <pthread.h>



void* start_routine1(){

  printf("hello\n");


}


void* start_routine2(){

  printf("world\n");


}

int main(){

  pthread_t thread1,thread2;

  pthread_create(&thread1,NULL,start_routine1,NULL);

  pthread_join(thread1,NULL);


  pthread_create(&thread2,NULL,start_routine2,NULL);

  pthread_join(thread2,NULL);

}
```



```
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$ gcc threadsImplementation.c
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$ ./a.out
hello
world
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$
```

Qn 6: How to avoid race around conditions and deadlocks?

The race around and deadlock conditions can be prevented by using Mutexes. The purpose of mutex is to lock a particular shared resources to a single thread thereby preventing other threads to access the same shared resource.

Qn 7: Difference between exec and fork:

- Fork() starts a new process that is a copy of the one that calls it.
- exec() replaces the current process image with another (different) one.
- Both parent and child processes are executed simultaneously in the case of fork().
- Control never returns to the original program unless there is an exec() error.

Qn 8: Difference between process and threads

- A process is any program in execution whereas the thread is a part of a process
- Multiple processes uses different memory space, but multiple threads uses same memory space.
- A process takes more time to create and terminate, whereas threads take less time to create or terminate.

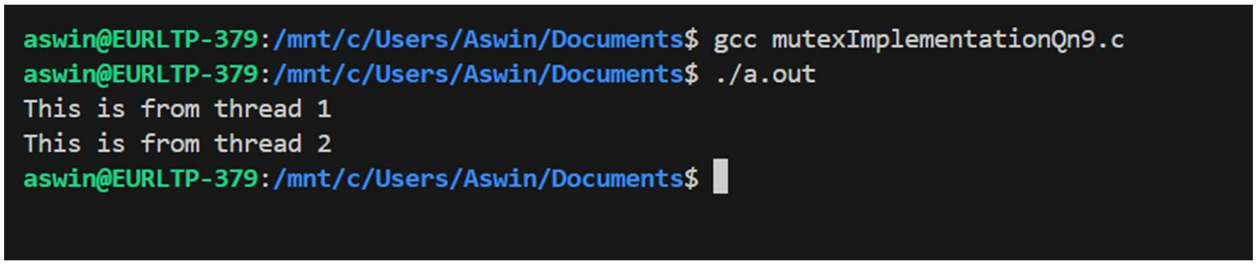
Qn 9:

Program:

```
#include <stdio.h>

#include <pthread.h>
```

```
#include <unistd.h>


pthread_t thread[2];

pthread_mutex_t lock;


void* threadFunc(void* arg){

  int* i = (int*)arg;

  pthread_mutex_lock(&lock);

  printf("This is from thread %d\n",*i);

  pthread_mutex_unlock(&lock);


}


int main(){

  pthread_mutex_init(&lock,NULL);


  for(int i=0;i<2;i++)

    pthread_create(&thread[i],NULL,threadFunc,(void*)&i);


  pthread_join(thread[0],NULL);

  pthread_join(thread[1],NULL);


  pthread_mutex_destroy(&lock);

}
```

Output:



```
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$ gcc mutexImplementationQn9.c
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$ ./a.out
This is from thread 1
This is from thread 2
aswin@EURLTP-379:/mnt/c/Users/Aswin/Documents$
```