MODULE 3 ASSIGNMENT

1. Which signals are triggered, when the following actions are performed.

a) User press ctrl+C = **SIGINT**
b) Kill() system call is invoked = **SIGTERM**
c) CPU tried to execute an illegal instruction = **SIGILL**
d) When the program access the unassigned memory = **SIGSEGV**

2. List the gdb command for the following operations

a) To run the current executable file = **run (or) run[args]**

b) To create breakpoints at = **break [filename] : [linenumber]**

c) To resume execution once after breakpoint = **continue**

d) To clear break point created for a function = **clear function_name**

e) Print the parameters of the function in the backtrace = **bt, info args**

3. Guess the output for the following program.

```
#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

4. Guess the output for the following program.

```
#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

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

```
#include <pthread.h>
#include <stdio.h>

void* helloFunction(void* arg)
{

	printf("Hello ");
	return NULL;
}

void* worldFunction(void* arg)
{
	printf("World");
	return NULL;
}

int main()
{
	pthread_t hello_thread;
	pthread_create(&hello_thread, NULL, helloFunction, NULL);
	pthread_t world_thread;
	pthread_create(&world_thread, NULL, worldFunction, NULL);
```

```
    pthread_join(hello_thread, NULL);
    pthread_join(world_thread, NULL);

    return 0;
}
```

6. How to avoid Race conditions and deadlocks?

Race conditions can be avoided by using mutex locks and semaphores.Mutex locks can be used ensure that only one thread can access a critical section of code at a time.Lock the code before entering the critical section and unlock it after exiting.Semaphores can be used to control access to a shared resource by limiting the number of threads that can access it simultaneously.

Deadlocks can be avoided by using time frames and Banker's Algorithm.The banker's algorithm is a resource allocation and deadlock avoidance algorithm that simulates resource allocation for predetermined maximum possible amounts of all resources before performing an "s-state" check to look for potential activities and determining whether allocation should be permitted to continue.

7. What is the difference between exec and fork ?

The fork generates new processes while simultaneously preserving its parent process. The exec creates new processes but doesn't preserve the parent process simultaneously.

8. What is the difference between process and threads.

Process is a program in execution while thread is a smallest segment of instructions that can be handled independently by a scheduler.The process takes more time to terminate while the thread takes less time to terminate.If one process is obstructed then it will not affect the operation of another process but if one thread is obstructed then it will affect the execution of another process.

9. Write a C program to demonstrate the use of Mutexes in threads synchronization

```
#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

pthread_t threads[3];

int counter;
```

```
pthread_mutex_t lock;
void* lockFunction(void* arg)
{
	pthread_mutex_lock(&lock);
	unsigned long i = 0;
	counter += 1;
	printf("\n Job %d has started\n", counter);
	for (i = 0; i < 100; i++);
	printf("\n Job %d has finished\n", counter);
	pthread_mutex_unlock(&lock);
	return NULL;
}

int main(void)
{
	int i = 0;
	int error;
	if (pthread_mutex_init(&lock, NULL) != 0) {
		printf("mutex init has failed");
		return 1;
	}
	while (i < 3) {
		error = pthread_create(&(threads[i]), NULL, &lockFunction, NULL);
		i++;
	}
```

```
    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);
    pthread_join(threads[2], NULL);
    pthread_mutex_destroy(&lock);
    return 0;
}
```