**Module - 3 Assessment**

**Vishnu Sankar J**

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

**a)user press ctrl+C**

Sends an interrupt signal, SIGINT (Signal Interrupt), to the currently running process in the terminal.

**b)kill() system call is invoked**

The kill() system call is used to send a signal to a process. The signal can be used to terminate the process, stop it temporarily, or continue it. The kernel checks to see if the calling process has permission to send the signal to the target process. If the calling process has permission, the kernel sends the signal to the target process.

**c)CPU tried to execute an illegal instruction**

When the CPU attempts to execute an illegal instruction, it triggers an interrupt or exception. It is SIGILL (Signal Illegal Instruction).

**d)When the program access the unassigned memory**

When a program tries to access unassigned memory, it results in a segmentation fault. This triggers the SIGSEGV signal (Signal Segmentation Violation), indicating that the program has attempted an invalid memory access.

**2)List the gdb command for the following operations**

**a)To run the current executable file**

cmd: run

**b)To create breakpoints at**

cmd: break

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

cmd: continue

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

cmd: clear function\_name

**e)Print the parameters of the function in the backtrace**

cmd: backtrace

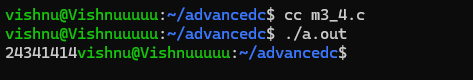
**3) Guess the output for the following program**

**Output:**



1. **Guess the output for the following program**

**Output:**



1. **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 <pthread.h>

#include <stdio.h>

void \*hello(void \*arg) {

printf("Hello\n");

return NULL;

}

void \*world(void \*arg) {

printf("World\n");

return NULL;

}

int main() {

pthread\_t hello\_thread, world\_thread;

pthread\_create(&hello\_thread, NULL, hello, NULL);

pthread\_create(&world\_thread, NULL, world, NULL);

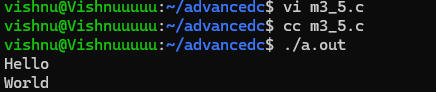
pthread\_join(hello\_thread, NULL);

pthread\_join(world\_thread, NULL);

return 0;

}

**Output:**



1. **How to avoid Race conditions and deadlocks?**

**Answer:**

To avoid race condition, we can use thread synchronization i.e. can execute only one thread at a time. For avoiding we can use either deadlock detection algorithm or deadlock avoidance algorithm.

1. **What is the difference between exec and fork?**

**Answer:**

fork() is a system call used to create a new process in C. It returns a integer value, a fork is created using fork(). The new process is a child process which is duplicating a parent process. When a fork fails to create a process it returns -1.

exec() replaces current process image with a new process image, exec() runs an executable file. The currently executing process terminates and replaced with newly formed process. There is only child process in exec(). exec() makes child process and replace it with parent process.

1. **What is the difference between process and threads?**

**Answer:**

Process is any program that is executed. Process takes more to get created, and it takes more time for termination. Process is less efficient for communication. Multiprogramming does the process of multi-process. The process is isolated and it is called heavyweight process. The process doesn’t share the data with each other.

A thread is a lightweight unit of execution within a process. It is a segment of process. Threads take less time for termination and creation. Multiple threads within a process share the same resources, including memory. Threads within the same process can communicate more easily through shared memory.Threads are more lightweight compared to processes, as they share resources and can be created and terminated more quickly. Threads within the same process can be scheduled independently, allowing for parallel execution.

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

**Code:**

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 5

int sharedCounter = 0;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*incrementCounter(void \*threadID) {

long tid = (long)threadID;

pthread\_mutex\_lock(&mutex);

sharedCounter++;

printf("Thread %ld incremented the counter. Current value: %d\n", tid, sharedCounter);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int rc;

long t;

for (t = 0; t < NUM\_THREADS; t++) {

printf("Creating Thread %ld\n", t);

rc = pthread\_create(&threads[t], NULL, incrementCounter, (void \*)t);

if (rc) {

printf("Error: Return code from pthread\_create() is %d\n", rc);

return 1;

}

}

for (t = 0; t < NUM\_THREADS; t++) {

pthread\_join(threads[t], NULL);

}

printf("All threads have completed. Final counter value: %d\n", sharedCounter);

return 0;

}

**Output:**

