MODULE 3

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

user press ctrl+C

kill() system call is invoked

CPU tried to execute an illegal instruction

When the program access the unassigned memory

When the user presses Ctrl+C: SIGINT (Signal Number 2)

When the kill() system call is invoked: SIGKILL (Signal Number 9)

When the CPU tries to execute an illegal instruction: SIGILL (Signal Number 4)

When the program accesses unassigned memory: SIGSEGV (Signal Number 11)

2. List the gdb command for the following operations

To run the current executable file

To create breakpoints at

To resume execution once after breakpoint

To clear break point created for a function

Print the parameters of the function in the backtrace

1. run
2. break <line\_number>

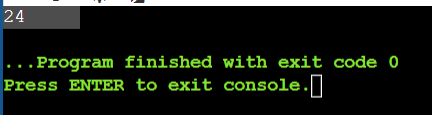
break <function\_name>

1. continue
2. clear <function\_name>
3. backtrace

3. Guess the output for the following program.

2 2

4. Guess the output for the following program.



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 <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void \*print\_hello(void \*arg) {

printf("Hello\n");

pthread\_exit(NULL);

}

void \*print\_world(void \*arg) {

printf("World\n");

pthread\_exit(NULL);

}

int main() {

pthread\_t thread1, thread2;

if (pthread\_create(&thread1, NULL, print\_hello, NULL) != 0) {

fprintf(stderr, "Error creating thread 1\n");

return 1;

}

pthread\_join(thread1, NULL);

if (pthread\_create(&thread2, NULL, print\_world, NULL) != 0) {

fprintf(stderr, "Error creating thread 2\n");

return 1;

}

pthread\_join(thread2, NULL);

return 0;

}

6. How to avoid Race conditions and deadlocks?

* Use synchronization mechanisms such as mutexes, semaphores, and condition variables.
* Avoid shared mutable state where possible.
* Utilize thread-safe data structures.
* Be cautious with nested locks to prevent deadlock.
* Avoid busy waiting and use blocking mechanisms.
* Establish a consistent order for resource allocation and deallocation.
* Thoroughly test and debug concurrent code.

7. What is the difference between exec and fork ?

**fork:**

* fork is a system call used to create a new process, known as a child process, which is an exact copy of the calling (parent) process.
* After calling fork, both the parent and the child processes continue executing from the point where the fork call was made.
* The child process receives a copy of the parent process's memory, including all variables and their values. However, the child process has its own process ID (PID) and has a separate memory space from the parent process.
* fork returns different values in the parent and child processes: 0 in the child process and the PID of the child process in the parent process.

**exec:**

* exec is a system call used to replace the current process's memory space with a new program.
* It loads a new program into the current process, effectively terminating the current program and starting the execution of the new program.
* Unlike fork, exec does not create a new process. Instead, it replaces the entire memory image of the current process with a new one.
* After calling exec, the program counter starts executing the code of the new program, and the new program inherits the process ID (PID) of the original process.
* exec does not return to the calling process unless an error occurs during the execution of the new program.

8. What is the difference between process and threads.

Processes are independent units of execution with their own memory space and resources. Threads are lightweight execution units within a process, sharing the same memory space. Processes provide isolation and parallelism, while threads enable concurrency and efficient resource sharing within a single process.

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

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 3

pthread\_mutex\_t mutex; // Declare a mutex variable

void \*thread\_function(void \*arg) {

int thread\_id = \*((int \*)arg);

pthread\_mutex\_lock(&mutex);

printf("Thread %d is accessing the shared resource.\n", thread\_id);

for (int i = 0; i < 1000000; ++i) {

// Do some computation

}

printf("Thread %d has finished accessing the shared resource.\n", thread\_id);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

if (pthread\_mutex\_init(&mutex, NULL) != 0) {

fprintf(stderr, "Mutex initialization failed\n");

exit(EXIT\_FAILURE);

}

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

thread\_ids[i] = i;

if (pthread\_create(&threads[i], NULL, thread\_function, &thread\_ids[i]) != 0) {

fprintf(stderr, "Error creating thread\n");

exit(EXIT\_FAILURE);

}

}

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

if (pthread\_join(threads[i], NULL) != 0) {

fprintf(stderr, "Error joining thread\n");

exit(EXIT\_FAILURE);

}

}

pthread\_mutex\_destroy(&mutex);

return 0;

}