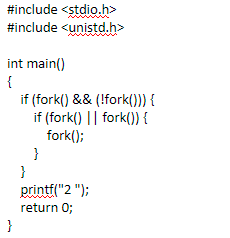
ADVANCED C PROGRAMMING

MODULE 3 ASSIGNMENTS

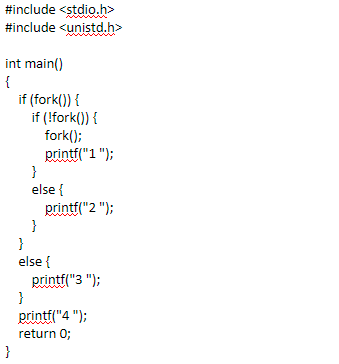
1. Which signals are triggered, when the following actions are performed?
2. User presses ctrl+C – SIGINT signal
3. Kill() system call is invoked – SIGTERM
4. CPU tried to execute an illegal instruction – SIGILL
5. When the program accesses the unassigned memory – SIGSEGV

1. List the gdb command for the following operations
2. To run the current executable file – run [args]
3. To create breakpoints at – break [func\_name]
4. To resume execution once after breakpoint – continue
5. To clear break point created for a function – clear [func\_name]
6. Print the parameters of the function in the backtrace – info args
7. Guess the output for the following program.



OUTPUT: 2

1. Guess the output of the following program



OUTPUT: 2 4

1. 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 <pthread.h>

void \*printHello(void \*arg) {

printf("Hello ");

pthread\_exit(NULL);

}

void \*printWorld(void \*arg) {

printf("World\n");

pthread\_exit(NULL);

}

int main() {

pthread\_t thread1, thread2;

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

perror("pthread\_create");

return 1;

}

if(pthread\_join(thread1, NULL) != 0) {

perror("pthread\_join");

return 1;

}

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

perror("pthread\_create");

return 1;

}

if(pthread\_join(thread2, NULL) != 0) {

perror("pthread\_join");

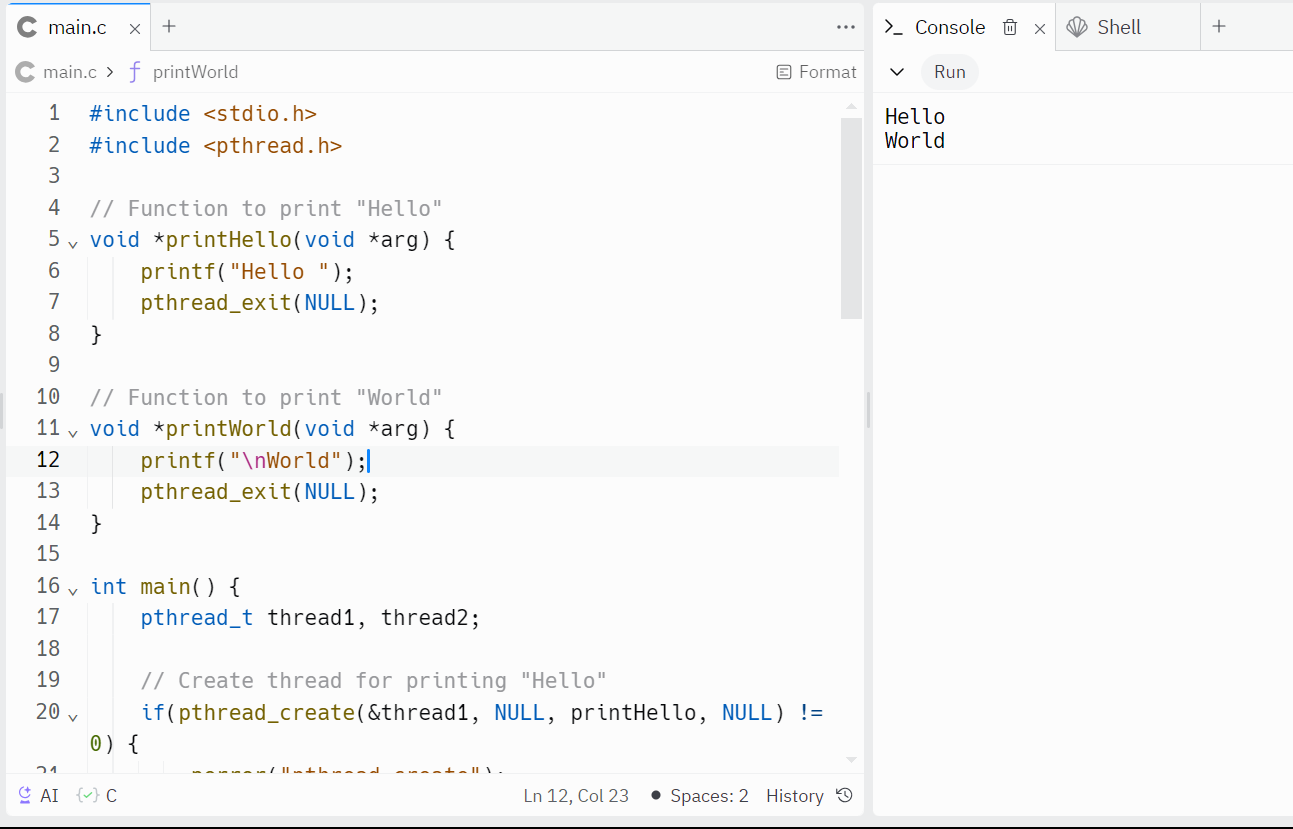
return 1;

}

return 0;

}

OUTPUT:



1. How to avoid Race conditions and deadlocks?

* To avoid race conditions, we can use mutexes, semaphores etc. Mutex is a locking mechanism used to make sure that only one thread can access a shared resource at a particular time. Semaphores are variables that control and coordinate the activities of multiple threads. Timeout mechanisms can also be implemented to avoid dependencies.

1. What is the difference between exec and fork?

`fork()` and `exec()` are both system calls used in Unix-like operating systems, especially in C programming. They serve different purposes in the creation of new processes:

1. fork():

- `fork()` is used to create a new process by duplicating the calling process.

- After a successful `fork()`, two processes are created: the parent process (the original process that called `fork()`) and the child process (the new process created by `fork()`).

2. exec():

- `exec()` is used to replace the current process image with a new process image.

- It loads a new program into the current process's memory space, overwriting the existing program and its data.

- After a successful `exec()`, the current process ceases to exist as it is replaced by the new program.

1. What is the difference between process and threads?

Process:

When a program is executed, it is termed as a process. In a process, context switching takes more time and the other processes will be obstructed or terminated.

Threads:

Threads are small segments of a process. In threads, context switching takes less time. If one thread is affected if one thread is obstructed.

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

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 5

int counter = 0;

pthread\_mutex\_t mutex;

void \*threadFunction(void \*arg) {

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

pthread\_mutex\_lock(&mutex);

printf("Thread %d: Counter = %d\n", thread\_id, ++counter);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

int i;

pthread\_mutex\_init(&mutex, NULL);

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

thread\_ids[i] = i;

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

perror("pthread\_create");

return 1;

}

}

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

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

perror("pthread\_join");

return 1;

}

}

pthread\_mutex\_destroy(&mutex);

return 0;

}

