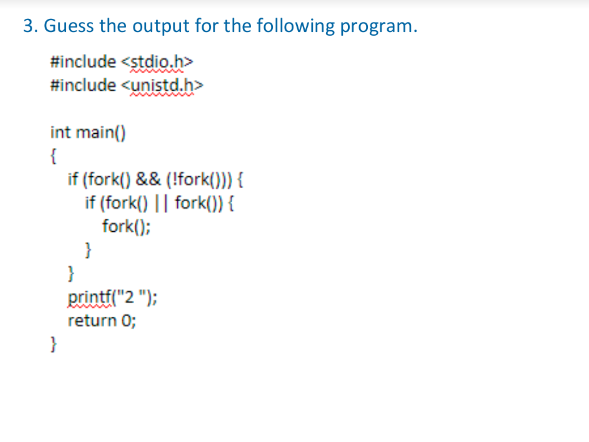
1. **Which signals are triggered, when the following actions are performed.**

* **user press ctrl+C:** SIGTERM (Terminates the entire program)
* **kill() system call is invoked:** SIGKILL (Immediate program termination)
* **CPU tried to execute an illegal instruction:** SIGILL (invalid instruction access)
* **When the program access the unassigned memory**: SIGSEGV (outside memory)

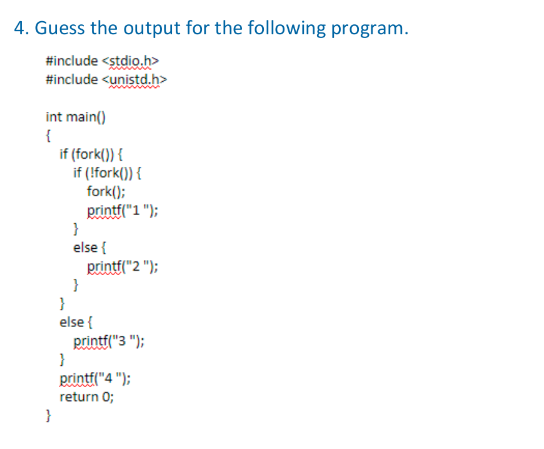
1. **List the gdb command for the following operations**

* **To run the current executable file:** run[args]
* **To create breakpoints at:** break [function\_name] or break [line\_number] or break \*[address] or break,[file\_name] : [line\_number] or break [above arguments]
* **To resume execution once after breakpoint:** continue
* **To clear break point created for a function:** clear[functionname]
* **Print the parameters of the function in the backtrace:** bt/backtrace

info args



**Answer: 2 2**

****

**ANSWER :**

3 4 2 4 ( when child process is executed first)

2 4 3 4 ( when parent process is executed first)

**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 <unistd.h>

#include <pthread.h>

#include <unistd.h>

void \*mythread1(void \*args1)

{

sleep(1);

printf("Hello\n");

return NULL;

}

void \*mythread2(void \*args2)

{

sleep(1);

printf("World!");

return NULL;

}

int main()

{

pthread\_t thread1;

pthread\_t thread2;

pthread\_create(&thread1,NULL,mythread1,NULL);

pthread\_join(thread1,NULL);

pthread\_create(&thread2,NULL,mythread2,NULL);

pthread\_join(thread2,NULL);

return 0;

}

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

**Race Condition:** Occurs when a program depends on the timing of one or more events to function correctly thus multiple threads sharing the same resources, data or access shared variable at the same time. This can be avoided by mutex which provides thread synchronization thus limits the access to a shared resources when we have multiple threads of execution. It is a lock that is set before accessing a shared resources and releasing it after the usage, during the period of lock no other thread can access the locked region.

**Deadlocks:** When two or more processes try to access the critical section at the same time and fails to access simultaneously while accessing the critical section. It can be avoided by the deadlock avoidance policy which grants the resource request only if it can establish that it cannot provide any deadlock in near future. Each process specifies the maximum number of resources of each unit of class that is required, and it is granted only if they are fewer number of resources, and the worst case is analysed which is used to check the possibility of deadlocks immediately and in future.

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

|  |  |
| --- | --- |
| **EXEC** | **FORK** |
| 1. Helps in making of processes 2. Allows in copying of processes. 3. The parent and child processes are in different physical address spaces. 4. When fork() is called there are child and parent processes. | 1. Helps in creation of processes. 2. Creates new process and replaces it with the existing one. 3. The child address space replaces the parent address space. 4. When exec() is called there is only a child process. |

**8. What is the difference between process and threads**

|  |  |
| --- | --- |
| **PROCESS** | **THREADS** |
| 1. Processes are independent 2. The program is made of multiple processes 3. Creation and termination time of   processes is more | 1. Threads are dependent 2. Each process consist of multiple threads,building block of process. 3. Creation and termination time of threads is less |

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

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <unistd.h>

#include<string.h>

pthread\_t threadid[2];

int c=0;

pthread\_mutex\_t lock;

void \*mythreadfun(void \*args1)

{

pthread\_mutex\_lock(&lock);

c+=1;

printf("Thread %d process has started\n",c);

int deposit=200;

int balance=500;

int expense=balance-deposit;

printf("Total savings:%d\n",expense);

printf("Thread %d process has finished\n",c);

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main()

{

int i=0;

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

{

printf("failed to initialize the mutex");

}

while(i<2)

{

pthread\_create(&threadid[i],NULL,&mythreadfun,NULL);

i++;

}

pthread\_join(threadid[0],NULL);

pthread\_join(threadid[1],NULL);

pthread\_mutex\_destroy(&lock);

return 0;

}