

ADVANCED C PROGRAMMING ASSESSMENT

MODULE 3

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

1. user press ctrl+C
2. kill() system call is invoked
3. CPU tried to execute an illegal instruction
4. When the program access the unassigned memory

Solution:

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

2. List the gdb command for the following operations

Solution:

1. To run the current executable file
run [args] eg: run 10
2. To create breakpoints at
break [function_name] or break [line_number] or break *[address] or break [file_name] : [line_number] or break [above arguments] if condition or above can be written by replacing break with 'b' .
3. To resume execution once after breakpoint

continue or c / c [repeat_count]

4. To clear break point created for a function

clear [FUNCTION_NAME]

5. Print the parameters of the function in the backtrace

bt/backtrace

Frame/up/down

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;
}
```

Solution:

2 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;
}

```

Solution:

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

Solution:

```

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

void *mythreadfun1(void *args1)
{
    sleep(1);
    printf("Hello\n");
    return NULL;
}

void *mythreadfun2(void *args2)
{
    sleep(1);
    printf("World!");
    return NULL;
}

```

```

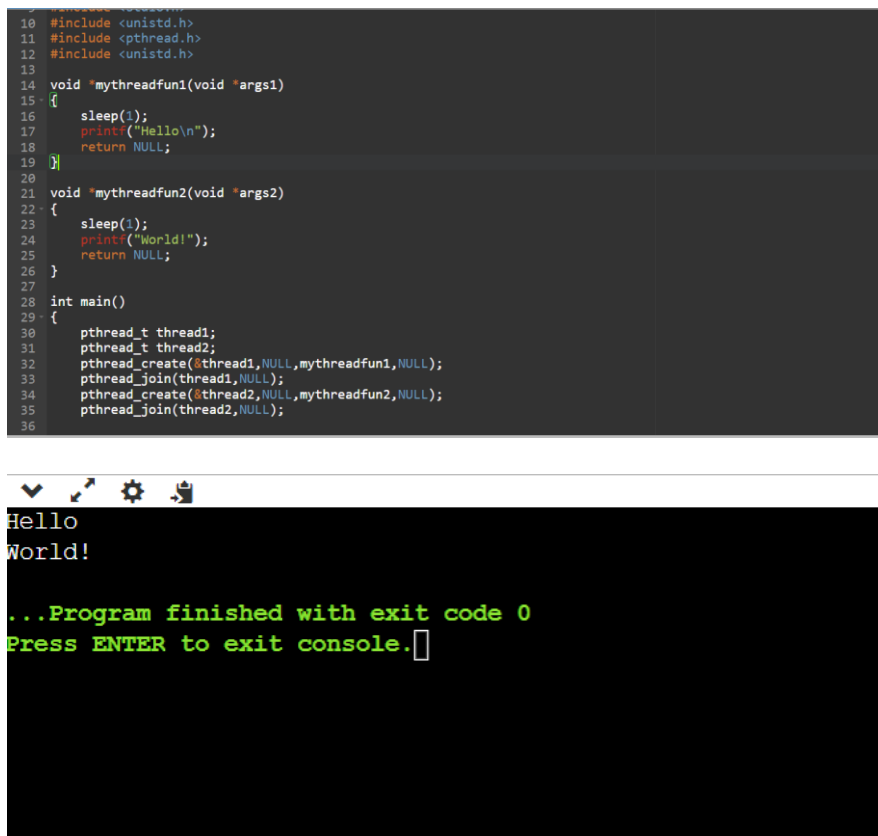
int main()
{
    pthread_t thread1;
    pthread_t thread2;
    pthread_create(&thread1,NULL,mythreadfun1,NULL);
    pthread_join(thread1,NULL);
    pthread_create(&thread2,NULL,mythreadfun2,NULL);
    pthread_join(thread2,NULL);

    return 0;
}

```

OUTPUTS:

**Hello
World!**



The image shows a code editor window with a dark background and light-colored text. The code is a C program that creates two threads, each printing a message and then returning NULL. The main function creates both threads, joins them, and returns 0. Below the code editor is a terminal window with a black background and green text. The terminal output shows 'Hello' and 'World!' on separate lines, followed by a message indicating the program finished with exit code 0 and a prompt to press ENTER to exit the console.

```

9 //include <unistd.h>
10 #include <unistd.h>
11 #include <pthread.h>
12 #include <unistd.h>
13
14 void *mythreadfun1(void *args1)
15 {
16     sleep(1);
17     printf("Hello\n");
18     return NULL;
19 }
20
21 void *mythreadfun2(void *args2)
22 {
23     sleep(1);
24     printf("World!");
25     return NULL;
26 }
27
28 int main()
29 {
30     pthread_t thread1;
31     pthread_t thread2;
32     pthread_create(&thread1,NULL,mythreadfun1,NULL);
33     pthread_join(thread1,NULL);
34     pthread_create(&thread2,NULL,mythreadfun2,NULL);
35     pthread_join(thread2,NULL);
36
37

```

```

Hello
World!

...Program finished with exit code 0
Press ENTER to exit console.

```

6. How to avoid Race conditions and deadlocks?

Solution:

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?

Solution:

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.

Solution:

Process	Thread
1.Processes are independent, they don't share memory. 2.The program is made up of multiple processes. 3.Processes have separate data and code segments. 4.Creation and termination time of processes is more.	1.Threads are dependent, they share memory. 2.Each process consist of multiple threads, building block of process. 3.Threads share the same data, code, OS files with its peer threads. 4.Creation and termination time of threads is less.

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

Solution:

```
#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;
}

```

OUTPUTS:

Thread 1 process has started
 Total savings:300
 Thread 1 process has finished
 Thread 2 process has started
 Total savings:300
 Thread 2 process has finished

```

10 #include <unistd.h>
11 #include <pthread.h>
12 #include <unistd.h>
13 #include <string.h>
14
15
16 pthread_t threadid[2];
17 int c=0;
18 pthread_mutex_t lock;
19
20 void *mythreadfun(void *args1)
21 {
22     pthread_mutex_lock(&lock);
23     c+=1;
24     printf("Thread %d process has started\n",c);
25     int deposit=200;
26     int balance=500;
27     int expense=balance-deposit;
28     printf("Total savings:%d\n",expense);
29     printf("Thread %d process has finished\n",c);
30     pthread_mutex_unlock(&lock);
31     return NULL;
32 }
33
34 }
35

```

Thread 1 process has started
 Total savings:300
 Thread 1 process has finished
 Thread 2 process has started
 Total savings:300
 Thread 2 process has finished

 ..Program finished with exit code 0
 Press ENTER to exit console.