Part 1 : Introduction to RTS

1. What is a Real-Time System ?
2. What is the difference between a Typical System Configuration and a Real-Time System configuration?
3. What is a critical system? What is a non-critical system?
4. What does Time-Triggered mean ? What does Event-Triggered mean?
5. Give types of tasks.
6. Give the 3 types of RTOS.

Part 2 : POSIX Threads

Q1. What is the difference between Concurrency and Parallelism?
Q2. What is the difference between Process and thread?
Q3. What is POSIX and What is its utility?
Q4. What is a detached thread, what is it used for and what is the difference from a “joinable” thread?
Q5. What is the difference between fork() and pthread_create() ?

Exercise 1 :
Predict the Output of the following programs:

PROGRAM A :

```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
// make two process which run same
// program after this instruction
fork();
printf("Hello world!\n");
return 0;
}
```

PROGRAM B :

```c
#include<stdio.h>
#include<sys/types.h>
int main()
{
fork();
fork();
fork();
printf("hello\n");
return 0;
}
```

Exercise 2: Correct Answer

PROGRAM A:

What is the output of this program?

```c
#include<stdio.h>
```

```c
#include<pthread.h>
void *fun_t(void *arg);
void *fun_t(void *arg)
{
    printf("Having Fun\n");
    pthread_exit("Bye");
}
int main()
        {
            pthread_t pt;
            void *res_t;
            if(pthread_create(&pt,NULL,fun_t,NULL) != 0)
                perror("pthread_create");
            return 0;
        }
```

a) This program prints the string “Having Fun”
b) This program prints nothing
c) segmentation fault
d) run time error

PROGRAM B:

1. Which one of the following string will be printed first by this program?

```c
#include<stdio.h>
#include<pthread.h>

void *fun_t(void *arg);
void *fun_t(void *arg)
{
    printf("Hello\n");
    pthread_exit("Bye");
}
        int main()
        {
            pthread_t pt;
            void *res_t;
            if(pthread_create(&pt,NULL,fun_t,NULL) != 0)
                perror("pthread_create");
            printf("Linux\n");
            if(pthread_join(pt,&res_t) != 0)
                perror("pthread_join");
            return 0;
        }
```

a) Linux
b) Sanfoundry
c) it cannot be predicted
d) none of the mentioned

Exercise 3: Creating and Waiting for Threads

Question 1: Write a program with the following behavior:
1. Threads are created (their number is passed as parameter when launching the program);
2. Each thread displays a message (for example hello world!);
3. The main thread is waiting for the termination of the different threads created.

Exercise 4: Thread Identification

Question 1: Modify the program from the previous question so that each thread displays:

- its PID (with getpid ());
- The opaque value returned by pthread_self, for example with: `printf("%p\n", (void *) pthread_self ());`

Exercise 5: Passing Parameters and Mutual Exclusion

Question 1: Change the program from the previous question to pass its order number to each thread. Each thread must then display it. Check that the order number displayed by each thread is different (correct your program if necessary).

Question 2: Declare a global variable *sum* initialized to 0. Each thread must, in a loop, add 1000000 times its order number to this global variable (we want to make 1000000 additions
per thread, not just one). View the value obtained after the termination of all threads.

Question 3: With 5 threads (numbered from 0 to 4), we should get (0 + 1 + 2 + 3 + 4) * 1000000 =10000000. Correct your program if it does not display this result.

Question 4: Change your program to no longer use global variables. We must therefore pass the addresses of the different variables needed as arguments to the threads, in a structure.

Exercise 6: Starvation problem

Bob and Allen also are borrowing books from library, but now after Allen borrows a book, he doesn't want to return it to library. So Bob can't borrow this book.

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

pthread_mutex_t book=PTHREAD_MUTEX_INITIALIZER;

void *Allen(void *arg);
void *Bob(void *arg);

int main() {
   pthread_t tid1, tid2;
   pthread_create(&tid1, NULL, *Allen, NULL);
   pthread_create(&tid2, NULL, *Bob, NULL);

   pthread_join(tid1, NULL);
   pthread_join(tid2, NULL);

   return 0;
}

void *Allen(void *arg) {
   printf("I'm Allen, I borrow this book and I won't return it!\n");
   pthread_mutex_lock(&book);
}
```

```
void *Bob(void *arg) {
    pthread_mutex_lock(&book);
    printf("I'm Bob, I can borrow it now\n");
}
```

I think you can solve this problem using pthread mutex, just try it!