CG2271 Real Time Operating Systems

Tutorial 5

Question 1

In a co-operative multitasking operating system, tasks surrender control to other tasks on their own accord. I.e. they are never pre-empted. How is such an OS similar to function queue scheduling, and how is different? List down and discuss as many points as possible.

Question 2

C supports a special data type called "void *", and this is commonly used in RTOS. For example in uC/OS-II and FreeRTOS, tasks are usually declared as:

```
void task1(void *param)
{
     !! task1 body
}

void task2(void *param)
{
     !! task2 body
}
```

a. A variable that is declared of type int * (e.g. int *ptr) is essentially a pointer to an integer variable. By the same logic, a variable that is of type void * (e.g. void *ptr) is a pointer to void variable. However there is no such thing as a void variable in C. So what is a void * variable?
b. Discuss why void * variables are useful.
c. What are the pitfalls of using void * variables?

Question 3

In C, a pointer to a function with prototype *void fun(int *)* is defined as:

```
int (*fpointer)(int);
```

So you can for example do this.

```
#include <stdio.h>

int f(int x)
{
       return 2*x;
}

int main()
{
       int (*fptr)(int);

       fptr=f; // Assign function f to pointer fptr

       printf("%d\n", fptr(3)); // Exactly the same as calling
                                  // function f  itself.
       return 0;
}
```

a. Discuss, within the context of operating systems, the different ways that function pointers are useful.

b. Using what you've just learnt about function pointers, implement a priority queue. Use the following structure:

```
typedef struct pq
{
       void (*fp)(void *);
       int priority;
       struct pq *next, *prev;
} TFuncQ;
```

Your priority queue has the following functions to queue and dequeue functions.

```
void enq(void (*fp)(void *), int priority);
TFuncQ  *deq(); // Returns the structure at the head of the queue.
```

You can assume that priority=0 is for highest priority functions and priority=255 is for lowest.

c. Suppose that there are three handler functions adc_func, timer0_func and timer1_func defined as:

```
       void adc_func(void *ptr)
       {
              !! Handle ADC stuff
       }
```

```
void timer0_func(void *ptr)
{
     !! Handle timer 0 stuff
}
void timer1_func(void *ptr)
{
     !! Handle timer 1 stuff
}
```

Write the ISRs for the ADC, Timer 0 and Timer1 compare interrupts that insert the respective handler functions into the priority queue. You only need to show the ISR code, you do not need to show any initialization/startup code for the ADC or the timers. ADC has highest priority, Timer 0 next highest, and Timer 1 the lowest priority.

d. Continuing on with c., implement a full function-queue scheduling system, except for the code to initialize/startup the timers and ADC.