IMPLEMENTATION DETAILS - AYUSH JAIN [jain207@purdue.edu]

Specific questions asked in lab6 for implementation are at the end

Steps taken for the implementation

HPET Device Implementation

a) In order to store the hook function which will be registered with this device, a pointer of type [void (*)(void)] is stored in the HPET control block. This pointer is set to NULL during HPET initialization and set to the function pointer whenever a new hook function (say alarm_trigger) is registered using hpetcontrol method and control action as `HPET_CTRL_HOOK_SET` overriding the previous value

b) The **init, control and intr entries** of HPET device type in Configuration file are set to hpetinit, hpetcontrol and hpetdispatch respectively.

c) Using SoC quark sheet, it is seen that bit 2 is set/clear to enable/disable the Timer 1 interrupt and bit 1 is used to acknowledge the timer 1 interrupt while handling the interrupt.

d) Thus, t1cc_1 register is used in struct of HPET Memory mapper registers to enable/disable Timer 1 interrupt and timer 1 interrupt is acknowledged by writing 1 in the Timer1 bit of the General Interrupt Status (gis) register in the same structure.

ALARM System Implementation

a) All the data structures used for implementation of alarm system are declared in alarm.h and initialized in alarm_init.c.

b)
```
struct aentry {          /* One per process plus two per list      */
uint32  akey;            /* Key on which the queue is ordered      */
qid16   anext;           /* Index of next process or tail  */
qid16   aprev;           /* Index of previous process or head      */
};
```

An alarm table alarmtab of type struct aentry is used to store multiple alarms in the form of list where Timer1 interrupt is set based on the head of this list. The key in this struct element refers to the delay (relative delay for all keys except the head key)

c) This list is similar to the delta list implementation except the fact that the head key of this list is updated during alarm insertion/removal in list instead of key of the head item in the sleepq being decremented every 1 ms.

d) The maximum size of this alarmtab is set to NALARMS + 2 . The additional 2 is used to store the head and tail of this list. Along with this, alarmqid is used to store the queueid of this list which is basically the index of head in the alarmtab. Tail of this list is given the next index after head in this alarmtab.

e) Also, alarmqsize is used to store the current size of this alarmtab which is updated during every insertion/removal of alarm in the list.

f) The callback function and its argument is stored in a separate table infotab of type struct struct alarm_info{

```
    void (*callbackfn)(int32 arg);
    int32 arg;
```

}; and size of NALARMS.

By intializing the values in infotab to NULL for callfunction and args = -1 respectively, we are able to reuse the alarm space whenever they have successfully completed or have been deregistered. This is done by setting these default values after termination/completion.

g) Each index in the alarmtab (except head and tail index) corresponds to the **unique alarm id** just like processid in the delta list. This alarmid is returned whenever an alarm is registered and allocated space in both alarmtab and infotab. If the alarm is added at the head of list, the timer1 interrupt is updated according to the requirements using Remaining time of interrupt control action in hpet interrupt.

h) During alarm registration, insert_alarm in alarm_init.c is used to add the alarm in the alarmtab as well as store the corresponding callbackfunc and its arg in infotab. The alarm id of this new alarm is retrieved using newaid() similar to newpid() function by checking empty slot in info tab.

EXTRA CREDIT IMPLEMENTATION

a) The alarm deregistration is done using remove_alarm method in alarm_init. Initially, the tti value is used to update the head key and then the element corresponding to aid is removed from alarmtab with its delay added to the successive key as the delays are relative in the list.

b) It is ensured that interrupts are updated accordingly if head of the list has been removed from the alarmtab.

SPECIFIC QUESTIONS ASKED IN LAB

a) Explain how you computed the value of ticks_for_1ms i.e. the HPET main counter ticks needed for passage of 1 ms

Since 1 tick in main counter happens every **69.841279 ns,** 1 ms (=10^6 ns) will take around (1000000/69.841279) number of ticks **equal to** 14318 (lower ceiling value) for every 1 ms.

b) Go through the HPET_CTRL_TTI control function for the HPET device that is implemented for you and explain how it correctly computes the value of "time to interrupt" in ms.

This function returns the time in milliseconds (through the passed pointer) until the next interrupt will happen (or -1 if Timer1 interrupt is disabled)

i) This control function checks first whether Timer1 interrupt is disabled or not . In case it is disabled, it returns -1
ii) It subtracts the Timer1 counter value (that has been set by the interrupt set control action for a specific amount of delay) with the current main counter value to return the remaining time to interrupt in ticks. This count is then divided by 14318 to find the remaining time in ns.

TEST CASES

a) Test cases have been written to check alarm_register, alarm_deregister independently and then mingled with each other with alarms added/removed at various locations of the alarmtab list at various instances of time.

b) Along with this, the alarm system initialization, alarm_trigger and HPET device system implementation is checked by debugging statements for the values that are initialized or updated at various instances.

c) Existing system calls like sleep are checked to ensure that they are not affected in any manner.

Comments specific to each test case written in main.c