
Machine Project Assignment

mp5: Watchdog Timer

Assigned: 25 April 2023

Due: 10 May 2023 End of day

I. OBJECTIVES:

Embedded systems use a watch dog timer to recover from program malfunctions. During faulty software operations (e.g. a program gets stuck in an infinite loop), expiration of a watchdog timer causes the system to reboot. In embedded applications, these timers can be implemented in either software or hardware. This assignment teaches students on how to implement a watchdog timer in software. Copy files for this exercise from `/courses/cs341/s23/hefeiqiu/mp5/` to your `cs341/mp5` directory. You will use timing functions provided in `timepack_sapc.c`.

II. Operations of a Watchdog Timer:

One can imagine if an embedded system installed in a remote area goes haywire, it may be impossible or very costly to send a service personnel to reset the system. See an example application on the [NASA Deep Space Probe](#) and a more recent one on the [Mars helicopter flight](#). A watchdog timer can automatically reset the system without human intervention in the event of a system failure.

In theory, a [watch dog timer](#) is a timer that is set to expire at a pre-determined time interval. During a faulty condition (e.g. software is stuck in an infinite loop), timer expiry reboots the entire system. If the watchdog timer is implemented in external hardware, its logic will send a signal to the reset circuit (e.g. power up restart) of the embedded system when the timer expires. If implemented in software, the interrupt service routine of the timer will run a reboot function that restarts the application from scratch.

Since rebooting an application causes a major interruption to the operation of the system (similar to the powering up your computer), you don't want to reboot it if this is not needed. In most cases, we don't know exactly when the problem will occur, but we may have some estimate on the time when the problem will likely to occur. Then you set the watchdog timer to expire at or a little after that estimated time. For those times that the software is running without fault, you reset the watchdog timer before it expires to prevent interrupt from happening. This action is known as "kicking the dog"

III. Software implementation of a Watchdog timer:

a) Part 1: Simulating the coding problem:

We use a similar test set up as in mp4. The test program for this mp is `charGen1.c` and is provided in the `cs341/mp5/` directory. scp the file to your vserver VM and **build the charGen1 executable at the vserver VM** using:

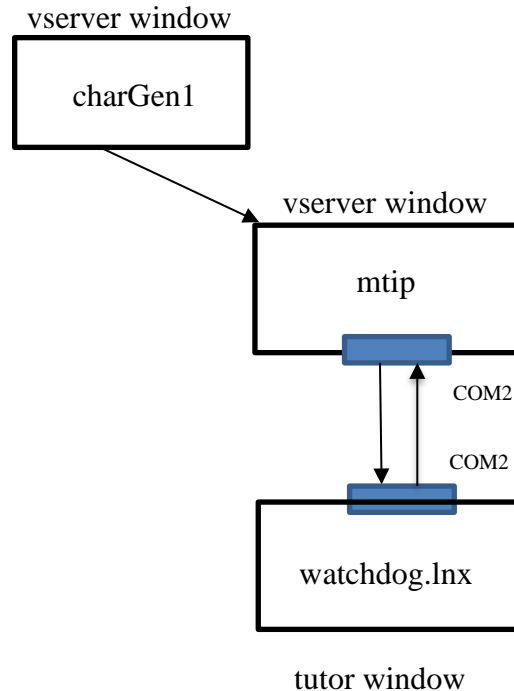
```
gcc -o charGen1 charGen1.c
```

Machine Project Assignment

Build the software application watchdog.lnx on users.cs.umb.edu (pe15) using the provided makefile, watchdog.c, reboot.s and timepack_sapc.c files. scp the watchdog.lnx to your vserver VM and load it into the tutor VM using mtip and ~d. Run the programs in this sequence:

- i) Start the watchdog.lnx using tutor command: go 100100
- ii) Open up a remote ssh window at vserver VM and run charGen1 to generate test characters to COM2 at 1 second intervals using:

```
sudo ./charGen1  
(use cs444 as your sudo password)
```



The provided code in watchdog.c will print out characters received from the charGen1 driver. After 10 characters, the system will get stuck in a simulated infinite loop and no characters will be printed out. However, the serial port interrupts are ongoing. The program continues to execute the serial port ISR and process the received characters, but the main() is not printing them out.

Capture a run of your program in the tutor VM with a typescript file named script_part_1.

Machine Project Assignment

b) Part 2a: Reset the system using a watchdog timer

Code the watchdog timer using functions provided in `timepac_sapc.c`. Set the count value = 0 to generate an interrupt every 55msec (this is longest duration between interrupts you can set with a 16-bit counter hardware). When the watch dog timer expires, the ISR calls a reboot function (the provided `reboot.s`) that restarts the `watchdog.lnx` program. If you want to wait a longer time to do `reboot()`, you have to modify the ISR software as follows:

```
void irq0inthandc()
{
    ....
    tickcount++;
    if (tickcount % 300 == 0) reboot(); // this will do reboot every 300* 55ms
    ...
}
```

Cat your programs (`watchdog.c`, `timepack_sapc.c` and `reboot.s`) and capture the run of your program in the tutor VM with a typescript file named `script_part_2a`.

c) Part 2b: Implement the “Kick the dog” function

In `watchdog.c`, at the end of the `do_work()` function, implement a `kick_dog()` to call the `set_timer_count` function to reset the count back to 0. This will prevent the timer from expiring under normal circumstances. If the program gets stuck in the middle of `do_work()` and `kick_dog()` is not run, then the timer will expire and the program will get rebooted at the timer ISR.

Cat your programs (`watchdog.c`, `timepack_sapc.c` and `reboot.s`) and capture the run of your program in the tutor VM in a typescript file named `script_part_2b`.

Any private functions or variables you need should be declared static in "`timepack_sapc.c`", so that the calling program can never accidentally use a variable or function with the same name, or interfere with yours.

FINAL NOTE:

Leave working versions of the source files and the script files in your `mp5` project directory. The grader or I may rebuild them and/or run them to test them. A copy of the rubric sheet for grading `mp5` is included. In the event that you are unable to correctly complete this assignment by the due date, do not remove the work you were able to accomplish - partial credit is always better than none.