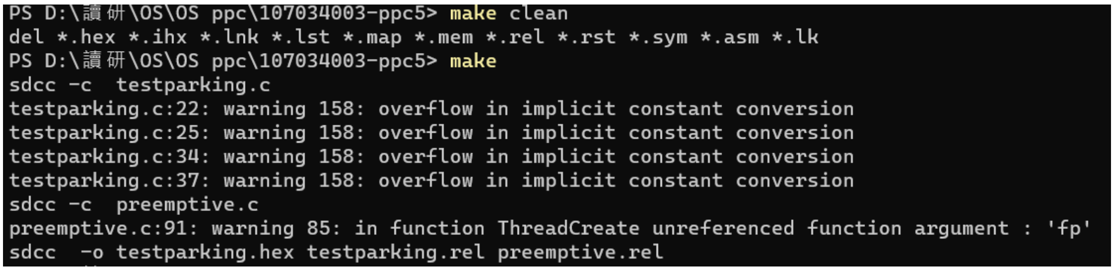
107034003 游鈞皓



```
PS D:\讀研\OS\OS ppc\107034003-ppc5> make clean
del *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lk
PS D:\讀研\OS\OS ppc\107034003-ppc5> make
sdcc -c  testparking.c
testparking.c:22: warning 158: overflow in implicit constant conversion
testparking.c:25: warning 158: overflow in implicit constant conversion
testparking.c:34: warning 158: overflow in implicit constant conversion
testparking.c:37: warning 158: overflow in implicit constant conversion
sdcc -c  preemptive.c
preemptive.c:91: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
sdcc  -o testparking.hex testparking.rel preemptive.rel
```

- Based on the above requirement, state your choice of time unit and provide your justification for how you think you can implement a `delay()` that meets the requirement above.

  Let time unit= 8 x 8051 timer

  Beacause we only have 4 threads and we use RR to scheduling, each thread can be executed once every 4 x 8051 timer which is half of our time unit. That is , even in the worst case, all threads' delay complete in the same time_unit, one can continue in that time_unit and the others can continue in the following 3 8051 timers, which is less than 0.5 time_unit.As a result, the delay will never rounds to n+1 time_units.

- what does your timer-0 ISR have to do to support these multiple delays and now()?

  Just repeating counting from 0 to 7 and increasing the time_unit by 1 each 8 8051 timers. The table named time_limit will record deadlines of threads.

- what if all threads call delay() and happen to finish their delays all at the same time? How can you ensure the accuracy of your delay? (i.e., between *n* and *n*+0.5 time units)?

  Same as the first question.

- How does the worst-case delay completion (i.e., all threads finish delaying at the same time) affect your choice of time unit?

  Same as the first question.

- when a car gets the parking spot (what time, which spot)

  I print the number of time_units in octal as long as it increases , so it's easy to see when a car gets a parking lot or exits one. The output format of each event will be like:

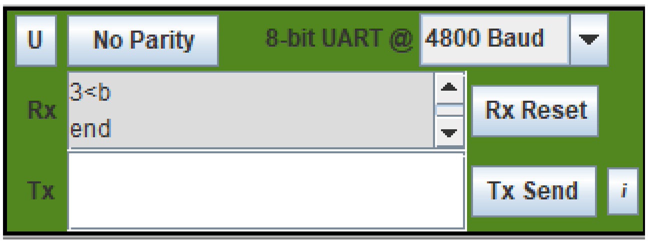
{ car number } { direction } { lot }.
Car number is an integer in [1,5].
Direction is '<' meaning exit a parking lot or '>' meaning gets one.
Lot is either a or b to represent 2 parking lots.

- when a car exits the parking lot (what time)
  Same as last question.

More details can be found in the code comments.

Executing example:

The last event is the car number 3 exits the parking lot b. Then, the main thread exits and gets in a infinte loop after printing 'end'.



```
3<b
end
```