MTE241, Spring 2021, Test 2 Coding Problem

Instructions:

- Do your own work and do not discuss the test with others. If you have questions post them as private questions on Piazza. Do not post any public questions.
- Your solution shall conform to the C99 standard. You may include any C99 standard library header file. You can use any C99-compliant compiler/IDE (e.g. clang, gcc, mingw, Code::Blocks, uVision5, vsCode).
- Make sure that you initialize all variables (they default to 0 in some systems but not all).
- Marks are based solely on test case output. Solutions must compile to receive any marks.
- Upload your source file to the Learn Dropbox.
- Remember that all code will be checked with MOSS for copying.

Download t2Code `.zip` from Learn which contains these files: `sched.c, sched.h, schedMain.c, readylist.c, readylist.h`.

`schedMain.c` contents:

```
int main(void) {
	tcb_t tcbs[] = { { "task0", P_LOW }, // task control blocks
		{ "task1", P_LOW },
		{ "task2", P_MED },
		{ "task3", P_MED },
		{ "task4", P_MED },
		{ "task5", P_HI } };
	const int nTask = sizeof(tcbs)/sizeof(tcb_t);
	action_t actions[] = {
		TIMESLICE, END, TIMESLICE, TIMESLICE, TIMESLICE, END, END, END,
		TIMESLICE, TIMESLICE, TIMESLICE, END, END, TIMESLICE, END };
	const int nAction = sizeof(actions)/sizeof(action_t);

	// release all tasks
	for(int i=0; i<nTask; i++)
		release(&tcbs[i]);
	printf("%s", running->name); // print running task's name

	// do actions
	for(int i=0; i<nAction; i++) {
		switch(actions[i]) {
			case TIMESLICE: printf(" ^ "); timeslice(); break;
			case END: printf(" ! "); terminate(); break;
		}
		printf("%s", running->name);
	}
	printf("\n");
}
```

The test code creates 6 tasks of varying priority. There is also an idle task initialized in `sched.c`. Tasks are created by calls to `release()`. If a newly released task has higher priority than the running task, the new task will run instead. Once the tasks are created, the testbench variously calls `timeslice()` and `terminate()`. `timeslice()` simulates the end of a timeslice and the next task of highest priority in round robin order should run. `terminate()` simulates ending the running task.

The expected output for the above test cases is:

```
task5 ^ task5 ! task3 ^ task4 ^ task2 ^ task3 ! task4 ! task2 !
task1 ^ task0 ^ task1 ^ task0 ! task1 ! idle ^ idle ! idle
```

'^' indicates a timeslice event. '!' indicates a termination event. Task 5 has highest priority so it is the only task to run until it is terminated. Then tasks 2, 3, 4 (medium priority) execute in round-robin order until they are each terminated. Then tasks 0, 1 (low priority) execute in round-robin order until they are each terminated. Finally the idle task executes. The call to terminate the idle task is ignored and it continues to execute.

The `running` pointer in `sched.c` points to the TCB of the running task. The ready lists are fully implemented in `readylist.c` - don't change them. Use the readylist functions: `dequeue(), enqueue(), isEmpty()` to interact with them.

Edit only `sched.c`. Implement these functions: `release(), terminate(), timeslice()`. Their functionality, including inputs and outputs is documented in `sched.h`. You may assume all inputs are valid.

Submit only `sched.c`.

Notes

- Marking will be based solely on test case output.
- You may add #include directives, typedefs, global variables and helper functions as needed to `sched.c`.
- Different test cases than the one shown will be used for marking. It is important that you test your code carefully.

Learning Objectives

- Learn to manipulate the running task and ready lists used in Fixed Priority Preemptive schedulers.