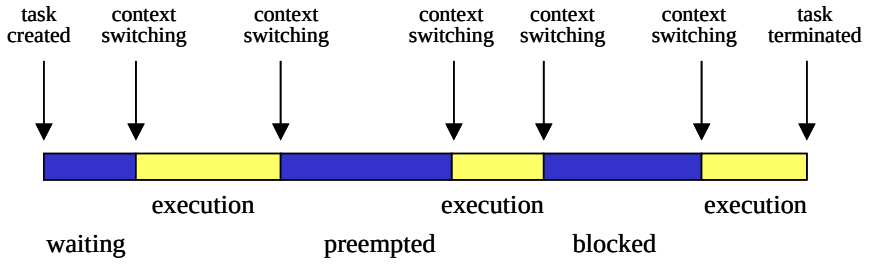
Assignment 4 A Polling Server in Zephyr RTOS (100 points)

In our last assignment, you are required to implement a polling server in Zephyr RTOS. The server is given a budget b_s for every replenishment period p_s. It receives aperiodic job requests from a global queue and processes the requests in the order of their arriving times. When the server's reminding budget is positive, the server runs in its assigned priority. The reminding budget can become zero when consumed, or there is no request to process (the arrival queue is empty). Until the budget is replenished, the server can still process aperiodic job requests in a background mode, i.e., when there is no other active tasks.

Zephyr RTOS doesn't support the notion of periodic and aperiodic tasks. Hence, we used a timer to trigger the repeated execution of a periodic task in Assignment 1. Similarly, with proper control, we should be able to implement a polling server using a Zephyr thread. For instance, to track budget consumption, you will need to measure thread execution time at run time. Note that a thread may be preempted (by high priority tasks) or blocked (due to unavailable resources locked by low priority tasks). To measure thread execution time, we need to know the instants that a threat is started, dispatched to run, switched off to waiting or ready state, and deleted (terminated), as shown in the flowing figure.



With Zephyr tracing hooks that are embedded in the kernel source code, the measurement of thread execution time can be done easily. The hooks are defined in /include/tracing/tracing.h and can be overridden to generate a custom operation. They are used to produce traces for SEGGER SystemView tool.

To track execution time (budget consumption) of a polling server, we only need to consider two tracing hooks:

- sys_port_trace_k_thread_switched_in(), and
- sys_port_trace_k_thread_switched_out().

In this assignment, we assume you will configure SEGGER's SystemView service in Zephyr. Hence, two additional hooks for the polling server are added in the SystemView's tracing functions for the switched_in and switched_out hooks. The patch file for this addition, polling_p4.patch, shows the hooks you need to work with.

All periodic and aperiodic jobs involve independent computation. To simulate the computation, we will use a busy loop of *n* iterations in the assignment, i.e.

```
volatile uint64_t n;
while(n > 0) n--;
```

This piece of code is included in the "*looping*" function which is available in "task_model_p4.h" header file. The header file also includes:

1. The structures to represent the threads for periodic tasks and the polling server.

2. A timer definition and its expiry function for aperiodic requests. In your main program, you need to start this timer by adding the following line of code:

   k_timer_start(&req_timer, K_USEC(ARR_TIME), K_NO_WAIT);

   Note that in the timer's expiry function, new requests are created (with random inter-arrival time and random number of loop iterations) and are put into a message queue. The polling server should get the next request from the queue and perform the requested computation.

3. Two useful functions for random number generation and for computing time difference. It is recommended that you use *k_cycle_get_32()* to track execution time.

With the implemented polling server, you need to use the parameters in task_model_p4.h as a test case to compute the average response time of all aperiodic requests. Note that the average response time should be computed with a maximum budget for polling server, i.e., BUDGET, subject to meeting deadline requirement for all periodic jobs. Then you can make a comparison with the average response time of a background server and show the improvement of polling server.

Due Date

The due date is 11:59pm, April 28.

What to Turn in for Grading

- Your submission is a zip archive, named RTES-LastName-FirstInitial_04.zip, that includes
  - An application folder, named *project_4*, to include all your implementation of the assignment. The folder should contain *CMakeLists.txt, prj.conf, readme.txt*, and a source file directory *src.*
  - A screenshot of SystemView timeline for the first 400ms of execution.
  - The readme text file describes all the commands you use to build and run your program, as well as the average response times of aperiodic requests when served by polling server and background server, respectively.
- Note that any object code or temporary files should not be included in the submission. Submit the zip archive to the course Canvas by the due date and time.
- Please make sure that you comment the source files properly and the readme file includes a description about how to make and use your software. Don't forget to add your name and ASU id in the readme file.
- There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Canvas. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor to drop a submission.
- The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas. The instructor reserves the right to ask any student to explain the work and adjust the grade accordingly.
- Failure to follow these instructions may cause deduction of points.
- Here are few general rule for deductions:
  - Cannot compile or compilation error -- 0 point for the assignment.
  - Must have "–Wall" flag for compilation -- 5-point deduction for each warning.
  - 10-point deduction if no compilation or execution instruction in README file.
  - Source programs are not commented properly -- 10-20-point deduction.
- ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.