Individual Assignment 2

Weilong Chen + weilong

Problem 1

A

The provided code is an example of a multi-tasking system using FreeRTOS, where three tasks (Task 1, Task 2, and Task 3) are created and scheduled to run concurrently. The tasks are synchronized using two mutex semaphores, $resource_a$ and $resource_b$.

A deadlock can occur when multiple tasks acquire resources in a way that creates a circular dependency, preventing any task from proceeding. Let's analyze the code to identify potential deadlock scenarios.

In the given code, there is a possibility of a deadlock if the following conditions are met:

Task A acquires $resource_a$ and then attempts to acquire $resource_b$. Task B acquires $resource_b$ and then attempts to acquire $resource_a$. If both Task A and Task B reach these points simultaneously, a circular dependency is created:

Task A holds $resource_a$ and waits for $resource_b$. Task B holds $resource_b$ and waits for $resource_a$. Both tasks will be stuck indefinitely, unable to proceed, resulting in a deadlock. For figure 1, if one task is at state p1 and another at state p4. The deadlock will happen as below:
Task 1: $p0->p1$ take r_A
Task 1: $p1->p2$ take r_B
Task 1: $p2->p3$
Task 1: $p3->p4$ give r_A
Task 2: $p0->p1$ take r_A
Deadlock

B

In a deadlock situation, one or more processes are blocked while requesting resources that are held by other processes, creating a chain of dependencies. This chain forms a cycle because each blocked process is waiting for a resource held by another blocked process, resulting in a perpetual loop of blocked states. The resource-allocation graph for the deadlock situation is shown in Figure 2.

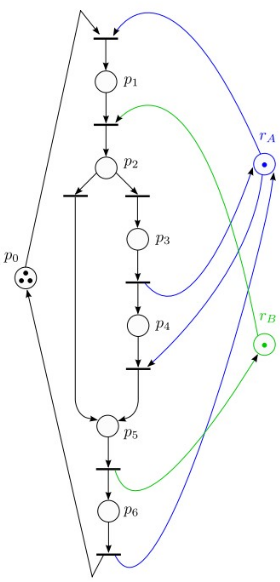

Figure 1: Example of a Petri net built for a part of the BIND software

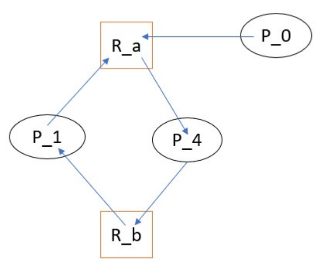

Figure 2

C

A resource C is added to avoid deadlock as shown in Figure 3. The program is updated on github.

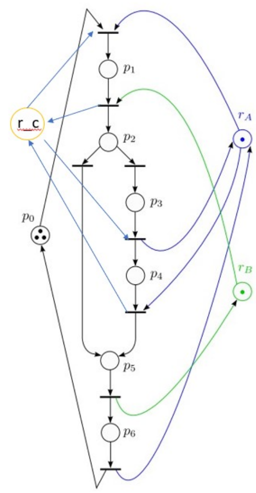

Figure 3

Problem 2

B

The Liu-Layland criteria:
$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{1/N} - 1\right)$ exhibit complete reliability when applied to a single task. However, when dealing with multiple tasks, the criteria can only determine if the tasks are schedulable but cannot guarantee the opposite scenario. For our case, it classifies 5 cases as SCHED_UNKNOWN.

The response time analysis is implemented in Python, which is shown below.

```python
def calculate_response_time(task_list):
    schedulable = "SCHED_YES"

    for i in range(len(task_list)):
        response_old = 0
        response_new = task_list[i]["WCET"]

        while response_new != response_old:
            response_old = response_new
            response_new = task_list[i]["WCET"]

            for j in range(len(task_list)):
```

```
                if task_list[j]["priority"] > task_list[i]["priority"]:
                    response_new += (
                        (response_old // task_list[j]["period"]) + 1
                    ) * task_list[j]["WCET"]

        if response_new > task_list[i]["deadline"]:
            schedulable = "SCHED_NO"
            break

    return schedulable
```

Listing 1: response time analysis

```
Response time test 1: SCHED_YES
Response time test 2: SCHED_NO
Response time test 3: SCHED_NO
Response time test 4: SCHED_YES
Response time test 5: SCHED_YES
```

Listing 2: output