Concepts and Definitions

Aperiodic task: runs at irregular intervals.

Aperiodic deadline can be

- hard, with the pre-condition
  - a lower bound on minimum event inter-arrival time
  - an upper bound on worst case computing time for the aperiodic request

- soft, a requirement on average response time
  - no special requirement on inter-arrival time, typical assumption: exponential inter-arrival time (Poisson Process)
  - no special requirement on worst case execution, typical assumption: exponentially execution time

The Fundamental Idea

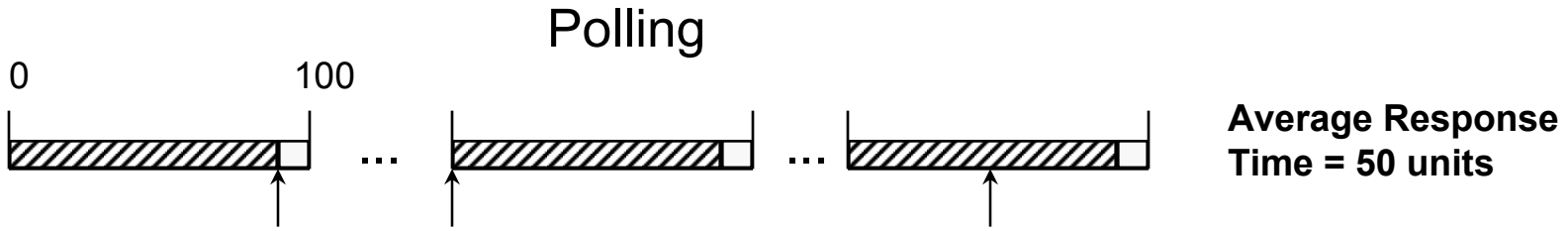
- Rate monotonic scheduling is a periodic framework. To handle apperiodics, we must convert the aperiodic event service to into a periodic framework.

- Except in the case of using interrupt handler, the basic idea is to periodically allocate CPU cycles to each stream of aperiodic requests.
  - polling
  - Handle it within an interrupt handler
  - sporadic server

Polling

- The simplest approach is polling:
  - buffer all aperiodic requests
  - serve the buffered requests periodically with
    - a budget C
    - and a period T
    - the priority is assigned according to the service periodic (higher rate, higher priority just like periodic tasks)
  - The utilization of a polling server is simply C/T
  - NOTE: each time, the server will keep serve buffered request until either
    - all the buffered requests are serviced, unused budget, if any, will be discarded
    - or the budget C runs out. In this case, the service suspends until the beginning of next period with a new C budget again

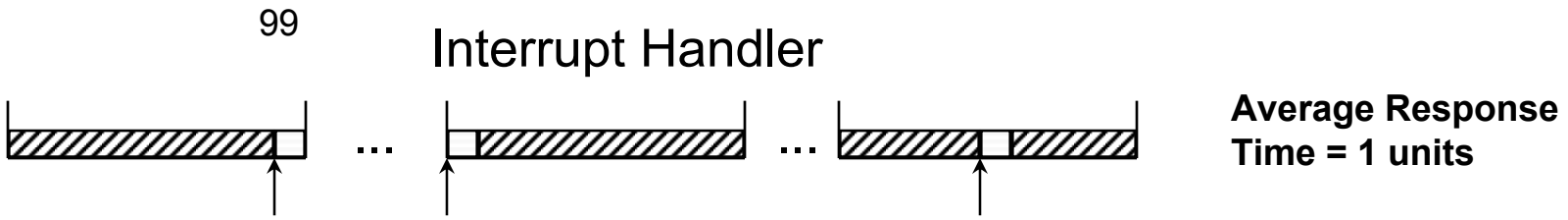
Performance of a Polling Server



Service delay of a polling server is, on average, roughly half of the the period.

- polling looks at request buffers only once per period. (like a city bus)
- higher polling rate will give better response time.
- low polling will have lower overhead

Using Interrupt Handler

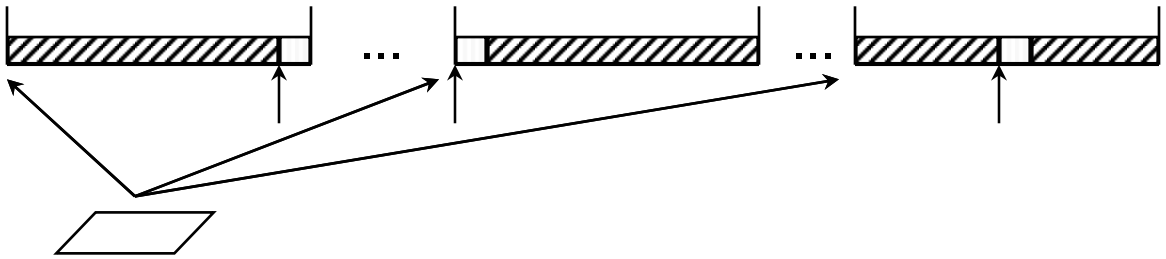


- Handle aperiodic requests within interrupt handler gives the best performance, since interrupt handlers runs at priority higher than applications
- Precisely for the same reason, a larger amount of such interrupts would cause deadlines of periodic tasks to missed.
- It is a solution with serious side effects. Use it ONLY as a last resort for short fuse hard deadline aperiodic request such as power failure warning.

Sporadic Server

- Sporadic server is similar to paying an employee a monthly (periodic) salary but each person to spend his/her money whenever they like (aperiodic usages)
- The idea is to deposit a budget the beginning of each period and aperiodic requests, when arrive, can use the budget if there is any.
- (If there is no budget left, wait until there is)
- Unused budget cannot be carry over to next period

Sporadic Server

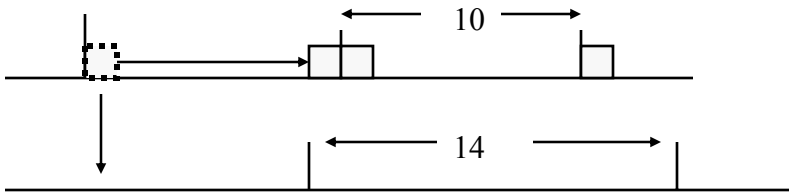


Average Response Time = 1 units

Budget deposited at beginning of period.

Sporadic Server Implementation Issues

- The idea is to make the service of aperiodic events to behave like a regular periodic task, i.e. if the sporadic server has a budget C and a period of T_s, then it should preempt any lower priority task, of period T, ceil(T/T_s) times.



•Because of the deferred usage of the budget, a straightforward implementation would create more than ceil(T/T_s) occurrences of the preemption.

• In the example above, it creates 3 preemptions, more than ceiling (14/10)

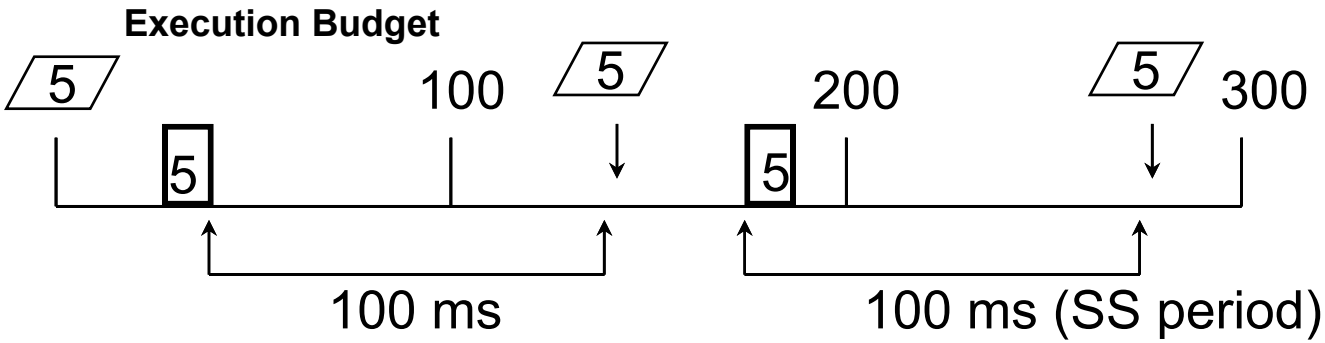
Sporadic Server (SS)

Modeled as periodic tasks

- Fixed execution budget (C)
- Replenishment interval (T)

Priority is based on T, adjusted to meet requirements

The Simplest replenishment method is to delay one “period” after it is used up. (Like refill your motor oil if it is low). More aggressive replenishment is possible and would give somewhat better response time.



Improving the Replenishment

- The delay 1 period after use up works. But in fact it generates less preemption that a regular periodic task. An improved version is as follows
- Whenever the server becomes active, i.e. moved from wait queue to ready queue by the OS, set server_start_time = current time.
- When the budget used up, delay 1 period from server_start_time and replenish the budget.
- This mimics periodic task better and it is still easy to implement. Fancier versions are still possible but the gain is small.

Implementing Period Transformation

- Recall that period transformation is a useful techniques to ensure:
  - stability under transient overload
  - improve system schedulability
- But it is undesirable to slice up the program codes. (Thou shalt separate timing concerns with functional concerns.)
- For example, a task with period T and exception time C, can be transformed as a sporadic task with a budget C/2 and period T/2. This is transparent to the applications.

Handling Hard Aperiodic Requests

- Most hard aperiodic request are trigger by threshold based warnings, for exampe, a sensor checks the temperature of a device every 50 ms
  - if the temperature ok, do nothing
  - if the temperature is too high, send out an warning until it is cool down,

- We do not know when the warning will come. But we know that if the waning comes, it will be
  - each warning is separated by some interval, 50 ms in this case
  - the handling time of warning must be bounded, say 5 ms.
  - So we can just tailor make a sporadic task with period 50 and budget 5 ms.
  - Question: what is the advantage of sporadic sever vs polling in this example?

Handling Soft Real Time Requests

- The basic tool here are 1) queueing theory and 2) simulation. We will limit ourselves to the simples form of queueing theory model in this class.

- M/M/1 queue:
  - Poisson arrival with average arrival rate λ, say 10 arrivals on average per sec.
  - Exponential service time $1/\mu$, say 0.1 sec on average.
  - The workload $r = \lambda/\mu$, the product of average arrival rate times average service time.
  - The average response time (waiting for service time)
    - $w = (1/\mu)/(1 - r)$

Sample Problems: Aperiodic

General Sporadic Server (GS) Design guideline: Give it as high a priority as possible and as many "tickets" as possible, without causing periodic tasks missing deadlines:

For get a rough initial estimation, we may use queuing theory formula M/M/1, provided that the sporadic server executes at a high priority. For example

Average execution time 1 msec with average inter-arrival time 100 msec creates a 1% average workload on the CPU.

- Execution Budget, C = 5 and replenish Interval T = 100 gets a server with 5% CPU capacity.
- The workload on the server is (workload/server_capacity) = 0.01/0.05 = 0.2.
- NOTE: We should NOT use 0.001 as the workload, since we have only a fraction of the CPU (5%) assigned to the sporadic server.

- The M/M/1 approximation for sporadic server is:

  $w = (1/m)/(1 - r) = 1 / (1 - 0.2) = 1.25.$

To estimate the average response time accurately, simulation is a must.