COMO CONTROLAR DE MANEIRA COMPUTACIONAL O TEMPO DE ACESSO DE VEÍCULOS EM UM CRUZAMENTO DE VIAS

Jéferson Costa, Natália Saraiva, Victor Hugo Borges

Curso de Bacharelado em Ciência da Computação – Centro Universitário de Brasília (UNICEUB) – Campus de Taguatinga
Taguatinga - DF - Brasil

jefersonc.neves@gmail.com, natalia_saraiva@outlook.com, victor.hugoborcas18@gmail.com

Abstract. *This article describes the main elements used in constructing a real-time system, using a platform Electronic prototyping hardware free and single board – arduino - to simulate the computer control of vehicle access time in a roads crossing protected by the use of traffic lights. In this system, it is possible to view, analyze and determine the setting of traffic lights time.*

Resumo. *Este artigo descreve os principais elementos utilizados na construção de um sistema de tempo real, utilizando uma plataforma de prototipagem eletrônica de hardware livre e de placa única – arduino- para simular o controle computacional do tempo de acesso de veículos em um cruzamento de vias protegidos pelo uso de semáforos. A partir deste sistema é possível visualizar, analisar e determinar o ajuste dos tempos de um semáforo.*

1. Introdução

O controle do tráfego de veículos nas grandes cidades é um dos sérios problemas enfrentados constantemente, principalmente nos horários de maior fluxo. Segundo o Departamento Nacional de Trânsito - Denatran (1979), nas grandes cidades cerca de 50% dos tempos de viagens e 30% do consumo de gasolina são gastos com os veículos parados nos cruzamentos, esperando que o sinal libere o acesso à via.

Este artigo consiste na apresentação de uma simulação do controle de tráfego no cruzamento de três vias por meio de semáforos, utilizando a arquitetura arduino. O semáforo consiste em controlar o tempo de acesso dos veículos a uma via, a fim de evitar colisões que podem proporcionar grandes acidentes.

Este artigo está organizado em cinco partes. Na seção 2 é apresentado alguns dos principais conceitos abordados no desenvolvimento do simulador. A seção 3 contém todos os processos da implementação do simulador. A seção 4 apresenta os resultados obtidos com a execução do simulador. Por fim a seção 5 apresenta uma conclusão dos resultados obtidos.

2. Base Teórica

Os Sistemas de Tempo Real são sistemas que operam sobre restrições de tempo, os quais consistem de subsistemas de controle e subsistemas controlados interagindo através de triagem de amostras, processamento e respostas.

Os Sistemas de Tempo Real são classificados em Hard Real Time e Soft Real Time. Este artigo abordará o sistema Hard Real Time que segue as restrições de tempo para evitar consequências catastróficas que estão normalmente relacionadas a vida das pessoas, deste modo controlando o tempo de determinadas atividades em ambientes diferentes.

Um meio de controlar o tempo é utilizando o semáforo, que é uma variável especial protegida que tem como função o controle de acesso a recursos compartilhados (por exemplo, um espaço de armazenamento) num ambiente multitarefa.

Segundo Tanenbaum (2009), iniciada uma operação de semáforo, nenhum outro processo pode ter acesso ao semáforo até que a operação tenha bloqueado ou sido bloqueada. Isso torna eficientes as resoluções dos problemas de sincronização e evita que aconteça às condições de corrida.

Operações de Up e Down (generalização de wakeup e sleep, respectivamente), sugeridos por Dijkstra (1965), em que o valor 0 sugere que nenhum sinal de acordar foi salvo ou algum valor positivo se um ou mais sinais de acordar estivessem pendentes[Tanenbaum, 2009].

O sistema que será implementado o semáforo tem que ser tolerante a falhas e operar adequadamente mesmo após falhas a alguns componentes. Capaz de oferecer garantias de correção temporal para o fornecimento de todos seus serviços que apresente restrição temporal.

Segundo o artigo publicado pelo Centro de Informática da UFPE (2016), para que haja uma confiança com o sistema é necessário à tolerância a falhas que é um conjunto de técnicas utilizadas para detectar, mascarar e tolerar falhas no sistema.

Segundo esse mesmo artigo, o tempo faz-se necessário quanto à obtenção do resultado esperado, com isso há a sua redundância que consiste em executar a mesma computação, em instantes distintos de tempo, a fim de verificar a existência de falhas temporais no sistema.

Deste modo haverá restrição de tempo para garantir que, dentro de certos limite, as suas restrições temporais serão atendidas, respeitando o tempo para que tenha um comportamento temporal desejado ou necessitado para que as tarefas sejam executadas cada uma no seu tempo, sem interromper a próxima tarefa .

Para que se tenha um comportamento temporal desejado, é necessário que todas as restrições temporais sejam respeitadas, pois as tarefas de tempo real estão limitadas a prazos, mais especificamente os seus “deadlines” (tempo máximo que uma tarefa deve ser executada).

Tratando-se do controle de vias através de semáforos, as tarefas executadas são tratadas como críticas, pois ao serem completadas após seu deadline pode causar falhas catastróficas, o que, deste modo, associa-se como uma tarefa periódica, pois há uma regularidade e, portanto, uma previsibilidade [De Oliveira 2000].

A forma como o programador vai solucionar a restrição de tempo ou qualquer outro problema, está interligado com o paradigma de programação, onde o programador irá decidir qual linguagem se adequa melhor para solucionar o problema.

Segundo o artigo de Sampaio e Maranhão (2008), desde o surgimento da primeira linguagem de programação de alto nível, Fortran, na década de 1950, uma grande variedade de linguagens de programação tem sido proposta, como consequência de domínios de aplicação distintos, avanços tecnológicos e interesses comerciais, dentre outros aspectos. Algumas linguagens compartilham características em comum e são ditas pertencerem a um mesmo paradigma: um modelo, padrão ou estilo de programação.

Segundo o artigo publicado pela Faculdade de Informática de Taquara – FIT (2008), um paradigma de programação determina uma forma particular de abordar os problemas e de formular respectivas soluções, os quais são classificados enquanto seu conceito base, podendo ser: Imperativo, funcional, lógico, orientado a objetos e estruturado. Contudo, uma linguagem de programação pode combinar dois ou mais paradigmas para potencializar as análises e soluções. Com isso, o programador é o responsável por escolher o paradigma mais adequado para resolver e analisar cada problema.

Sabendo-se disso, para determinar se um conjunto de tarefas é escalonável, se faz necessário o teste de escalonabilidade, o que verifica se existe para um conjunto de tarefas uma escala realizável. Esses testes variam conforme os modelos de tarefas e politicas definidas em um problema de escalonamento.

O conceito de utilização serve de base para testes simples. A ideia central nesses testes é que a soma dos fatores de utilização (Ui) das tarefas do conjunto não ultrapasse o número de processadores disponíveis para suas execuções:

$$U = \sum_{i}^{n} U_i \leq m$$

Figura 1. Esta Figura representa soma dos fatores de utilização (Ui) das tarefas que não ultrapassam o numero de processadores (m).

Em que: Ui = Ci/Pi (sendo a tarefa periódica)

Ci: tempo de computação da tarefa;

Pi: Período ou duração da tarefa;

Com isso, necessita-se, na utilização de semáforo de trânsito, um escalonamento preemptivo, dirigido a prioridade, em que nenhum outro algoritmo de mesma classe pode escalonar um conjunto de tarefas que não seja escalável pela taxa monotônica ("Rate Monotonic")[Jean_Marie, Joni e Rômulo 2000].

A análise de escalonabilidade do RM, é baseada no cálculo de utilização da CPU. Para que *n* tarefas tenham o atendimento de suas restrições temporais quando escalonadas pelo RM. O teste a seguir define uma condição *suficiente:*

$$U = \sum_{i}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

Figura 2. Esta figura representa o cálculo de teste de condição suficiente para o atendimento de tarefas, onde U é a utilização de CPU e n representa o número de tarefas.

3. Desenvolvimento

Quando pistas de trânsito se encontram existe a necessidade de controlar o fluxo de veículos entre elas, para evitar colisões. Neste caso, em um cruzamento de três vias, existe a necessidade de três sinais de trânsito sincronizados para controlar este fluxo de veículos. Este sincronismo entre os sinais significa que enquanto um estiver aberto, luz verde, ou em estado de atenção, luz amarela, o outro deve estar fechado e vice versa. Neste projeto vamos construir um protótipo que faz exatamente este controle, simulando três sinais de trânsito sincronizados.

Para o desenvolvimento foram utilizados os seguintes equipamentos:

- Um Arduino Uno R3;
- Uma protoboard;
- 12 leds, sendo 3 vermelhos, 6 amarelos e 3 verdes;
- 13 Fios jumper;
- 12 Resistores de 100 ohms;
- Um computador com a IDE e drivers do Arduino instalados;
- Um cabo USB para conectar o Arduino no computador;

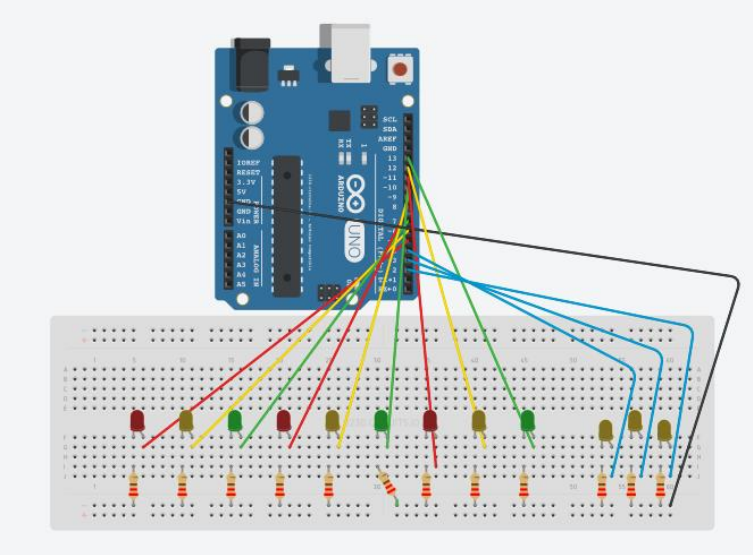

Figura 3. Esta figura ilustra o modelo do projeto

Os componentes foram conectados da seguinte forma:

Tabela 1. Conexão dos componentes do primeiro semáforo de trânsito.

Primeiro Sinal			
Led	Perna Led Menor	Perna Led Maior	Pino Arduino
Vermelho	5	6	5
Amarelo	10	11	6
Verde	15	16	7

Tabela 2. Conexão dos componentes do segundo semáforo de trânsito.

Segundo Sinal			
Led	Perna Led Menor	Perna Led Maior	Pino Arduino
Vermelho	20	21	8
Amarelo	25	26	9
Verde	30	31	10

Tabela 3. Conexão dos componentes do terceiro semáforo de trânsito.

Terceiro Sinal			
Led	Perna Led Menor	Perna Led Maior	Pino Arduino
Vermelho	35	36	11
Amarelo	40	41	12
Verde	45	46	13

- Todos os resistores de 100 ohms devem ser conectados na mesma coluna das pernas menores de cada um dos *leds* e com a linha que receberá a energia;
- Todos os fios jumper devem ser conectados na mesma coluna das pernas maiores de cada um dos *leds;*
- O outro fio dever ser conectado na linha que deseja passar a energia e no pino GND do arduino.
- Optamos por utilizar 3 leds amarelos para simular o Trânsito, conectados da mesma forma que os sinais, porém sua lógica sendo direcionada ao tráfego dos carros.

Para que haja a comunicação e a concorrência entre os semáforos foram criados modelos de tarefas com restrições de tempo, utilizando-se algoritmos em C para a comunicação com o Arduino .

Os tempos foram definidos para que o Deadline de cada tarefa coincida com seu período e o tempo de computação de cada uma seja conhecida e constante, que servem como premissas para o escalonamento do RM.

Seguindo este parâmetro o modelo de tarefas criado segue as seguintes restrições de tempo:

Tabela 4. Tarefas e seus respectivos parâmetros

Restrições de tempo				
Tarefas	Pi	Di	Ci	Ji
A	20000	20000	4000	0
B	17500	17500	4000	0
C	15000	15000	4000	0

em que:

- Ci: Tempo de computação (execução) da tarefa;
- Pi: Período ou duração da tarefa;
- Di: Deadline, ou tempo máximo que a tarefa pode durar;
- Ji: Tempo de ativação no pior caso.

Os tempos de computação foram definidos de modo que os automóveis ultrapassem o sinal de trânsito, de maneira igualitária, pois cada uma das três vias são independentes com o mesmo fluxo.

4. Resultados

O Semáforo de trânsito apresenta-se como um bom exemplo de sistema de tempo real, pois o mesmo deve atender as restrições de tempo, controlar o tráfego de veículos e consequentemente prevenir possíveis acidentes.

Conforme demonstrado na Figura 4. os tempos definidos seguem de forma que assim que ocorra o deadline (linhas em vermelho), a próxima tarefa execute, porem algumas tarefas executam juntamente com o deadline de sua antecessora. Isso ocorre devido, a prioridade definida pelo menor período, conforme a definição do algoritmo de taxa monotonica.

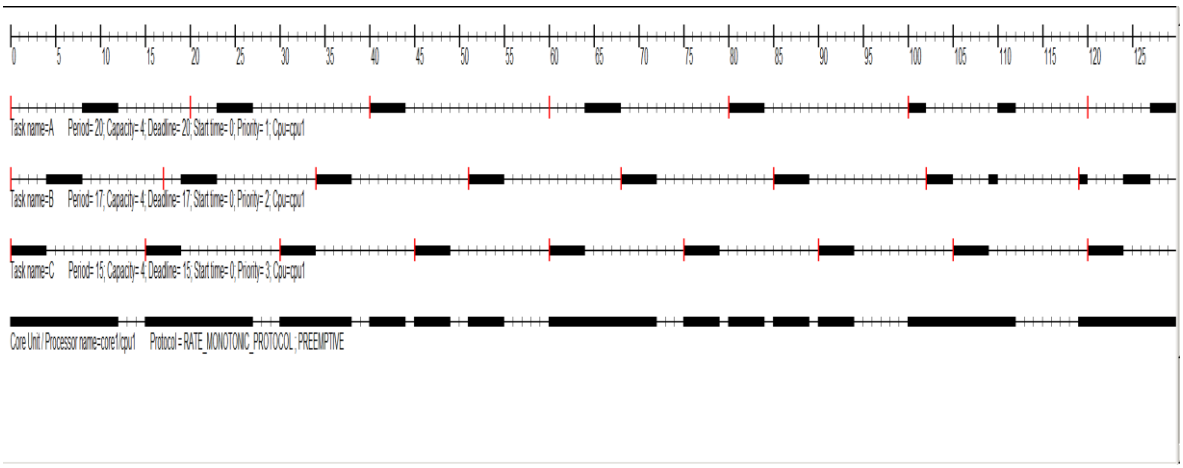

Figura 4. Representação de tempo de cada tarefa

Aplicando o teste de escalonabilidade obtêm-se aos seguintes resultados:

Tarefa 1 => U = 0,2 (Ci/Pi = 4000/20000)

Tarefa 2 => U = 0,23 (Ci/Pi = 4000/17500)

Tarefa 3 => U = 0,26 (Ci/Pi = 4000/15000)

U = Ut1 + Ut2 + Ut3 < 1

U = 0,2 + 0,23 + 0,26 <1

U = 0,69 < 1 (Somatório da utilização da CPU pelas tarefas, que não deve ultrapassar 1).

Além disso com o teste de condição do algoritmo de taxa monotônica obteve-se o seguinte resultado:

- Ut = 0,69 (Somatório da utilização da CPU pelas tarefas)
- $n(2^{1}/n - 1)$, onde n corresponde ao número de tarefas, ou seja, n = 3, logo o resultado é 0,75.

Com isso valida-se a utilização do algoritmo de escalonamento de taxa monotônica, pois: Ut <= $n(2^{1}/n - 1)$, logo 0,69 <= 0,75, o que é condição suficiente para atender as tarefas.

5. Conclusão

Tendo em vista que o controle do tráfego de veículos nas grandes cidades é um dos sérios problemas enfrentados constantemente, os resultados obtidos através do projeto, identificamos uma maneira de tentar amenizar o fluxo em um cruzamento de três vias independentes. Através do algoritmo de escalonamento implementado foi observado que existe a possibilidade de simular o funcionamento de um sinal de trânsito para cada uma das vias.

É possível observar que cada sinal de trânsito tem seu tempo de acesso pré-definido, com base no teste de escalonabilidade do algoritmo de escalonamento Taxa Monotônica (RM) realizado e os resultados do experimento mostram que através desse algoritmo implementado e da utilização dessa plataforma foi possível chegar a uma solução para o problema encontrado, simulando os três semáforos, evitando que os automóveis possam colidir e cause algum tipo de acidente, correlacionando um semáforo á outro, em que um fica aberto e os demais fiquem fechados. Sabendo-se que assim que um sinal é aberto a passagem de carros é liberada, com um certo *delay(atraso)*, para que em quanto os demais terminem de ultrapassar o sinal aberto, os que estão prestes a entrar atividade, não se choquem, o que mantem uma resposta satisfatória, considerando o tempo em que essa atividade é executada.

Outra possível solução que poderá ser utilizada para trabalhos futuros, é a implementação de sensores ao sistema, para detectar o fluxo de veículos em uma via, e desse modo, ajustar o tempo de acesso de acordo com o tráfego da via no momento.

Referências

TANENBAUM, Andrew S. Sistemas operacionais modernos. 3 ed. São Paulo: Pearson, 2009. 625 p.

LEITE, Glleddson Fryttys Menezes; ALVES, Antônio César Baleeiro. IMPLEMENTANDO UM SIMULADOR DE TRÁFEGO URBANO PARA UMA INTERSEÇÃO COM SEMÁFOROS. Scielo, Goiânia, v. 1, n. 1, p. 95-107, set./abr. 2016. Disponível em: <http://wsmartins.net/ermacs/trabalho_18.pdf>. Acesso em: 28 abr. 2016.

CIN. Sistemas de Tempo Real. Disponível em: <http://www.cin.ufpe.br/~jvob/introducao.html>. Acesso em: 29 abr. 2016.

ROMULOSILVADEOLIVEIRA. Sistemas de Tempo Real. Disponível em: <http://www.romulosilvadeoliveira.eng.br/livro-tr.pdf>. Acesso em: 29 abr. 2016.

SAMPAIO, Augusto; MARANHÃO, Antônio. Conceitos e Paradigmas de Programação via Projetos de Interpretadores. Disponível em: <http://www.cin.ufpe.br/~in1007/transparencias/jai/Jai2008Augusto.pdf>. Acesso em: 30 abr. 2016.

JUNGTHON, Gustavo; GOULART, Cristian Machado. Paradigmas de Programação. Disponível em: <https://fit.faccat.br/~guto/artigos/Artigo_Paradigmas_de_Programacao.pdf>. Acesso em: 30 abr. 2016.

CIN. Sistemas Operacionais. Disponível em: <http://www.cin.ufpe.br/~if728/sistemas_tempo_real/livro_farines/capa.pdf>. Acesso em: 17 jun. 2016.