UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO (UFRRJ)
INSTITUTO MULTIDISCIPLINAR (IM)
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS

HEBER DOS SANTOS SALES, JOÃO VICTOR DESSUPOIO, RAFAELA FERREIRA FIGUEIREDO

REAL-TIME SCHEDULING: NIGHTMARE ALGORITHM(NTM)

NOVA IGUAÇU - RJ
2022

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO (UFRRJ)
HEBER DOS SANTOS SALES, JOÃO VICTOR DESSUPOIO, RAFAELA FERREIRA FIGUEIREDO

REAL-TIME SCHEDULING:
NIGHTMARE ALGORITHM(NTM)

Relatório apresentado como requisito à obtenção de uma das notas parciais da disciplina de Sistemas Operacionais do curso de Ciência da Computação da Universidade Federal Rural do Rio de Janeiro.

Profa. Juliana M. N. S. Zamith

NOVA IGUAÇU - RJ
2022

Sumário

Páginas

1 Introdução

Neste artigo, apresentamos um método de escalonamento em tempo real e comparamos com outros algoritmos já existentes e conhecidos. Atualmente, mais de vinte e cinco algoritmos de escalonamento estão disponíveis no simulador (SimSo) escolhido para os testes. Nesse simulador foram propostos 4 processos chamados de "task", configurados aleatoriamente, para serem executados em cada um dos métodos de escalonamento.

id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)	Followed by
1	TASK T1	Periodic	No	0	3.0	-	3.0	0.349999	
2	TASK T2	Periodic	No	0	19	-	19	11.244592	
3	TASK T3	Periodic	No	0	6	-	6	0.330771	
4	TASK T4	Periodic	No	0	27	-	27	6.38238	

O algoritmo usado como base para a criaçao do NTM foi o Earliest Deadline First (EDF).

1.1 Earliest Deadline First

O Earliest Deadline First (EDF) define um escalonamento baseado em prioridades. É um esquema de prioridades dinâmicas com um escalonamento on-line. O EDF é um algoritmo ótimo na classe dos escalonamentos de prioridade dinâmica. As premissas que o determinam são:

- As tarefas são periódicas e independentes.
- O deadline de cada tarefa coincide com o seu período (Di=Pi).
- O tempo de computação (Ci) de cada tarefa é conhecido e constante.
- O tempo de chaveamento entre tarefas é assumido como nulo.

Código em Python do EDF:

```
"""
Implementation of the Global-EDF (Earliest Deadline First) for multiprocessor
architectures.
"""
from simso.core import Scheduler


class EDF(Scheduler):
    """Earliest Deadline First"""
    def init(self):
        self.ready_list = []

    def on_activate(self, job):
        self.ready_list.append(job)
        job.cpu.resched()

    def on_terminated(self, job):
        if job in self.ready_list:
            self.ready_list.remove(job)
        else:
            job.cpu.resched()

    def schedule(self, cpu):
        ready_jobs = [j for j in self.ready_list if j.is_active()]

        if ready_jobs:
            # Key explanations:
            # First the free processors
            # Among the others, get the one with the greatest deadline
            # If equal, take the one used to schedule
            key = lambda x: (
                1 if not x.running else 0,
                x.running.absolute_deadline if x.running else 0,
                1 if x is cpu else 0
            )
            cpu_min = max(self.processors, key=key)

            job = min(ready_jobs, key=lambda x: x.absolute_deadline)

            if (cpu_min.running is None or
                    cpu_min.running.absolute_deadline > job.absolute_deadline):
                self.ready_list.remove(job)
                if cpu_min.running:
                    self.ready_list.append(cpu_min.running)
                return (job, cpu_min)
```

1.2 Round Robin

O Algoritmo Round-robin (RR) é um dos algoritmos mais simples de agendamento de processos em um sistema operacional, que atribui frações de tempo para cada processo em partes iguais e de forma circular, manipulando todos os processos sem prioridades.

Por ser totalmente imune a problemas de starvation é usado em projetos de sistemas operacionais multitarefa, e foi projetado especialmente para sistemas time-sharing (tempo compartilhado), pois este algoritmo depende de um temporizador (Timer).

O Round Robin geralmente emprega tempo compartilhado, dando a cada tarefa um tempo definido chamado quantum. A tarefa é interrompida se esgotado o quantum e retomará de onde parou no próximo agendamento. Sem o tempo compartilhado, tarefas grandes poderiam ser favorecidas em detrimento de tarefas menores.

Código em Python:

```
"""
Round Robin Scheduler algorithm for uniprocessor architectures.
"""
from simso.core import Scheduler, Timer
from queue import Queue

class RR(Scheduler):
    def init(self):
        self.ready_list = Queue()
        self.running_job = None
        self.quantum = 1 # ms
        self.timer = Timer(self.sim, RR.reschedule,
                           (self, self.processors[0]), self.quantum, one_shot=False,
                           cpu=self.processors[0])
        self.timer.start()

    def reschedule(self, cpu):
        if not self.ready_list.empty():
            cpu.resched()

    def on_activate(self, job):
        self.ready_list.put(job)
        job.cpu.resched()

    def on_terminated(self, job):
        self.running_job = None
        job.cpu.resched()

    def schedule(self, cpu):
        if not self.ready_list.empty():
            job = self.ready_list.get()
            if self.running_job is not None:
                self.ready_list.put(self.running_job)
            self.running_job = job
        else:
            job = self.running_job
        return (job, cpu)
```

2 Desenvolvimento

O NTM(Nightmare Algorithms) foi criado a partir de uma idéia de motificação do algoritmo já conhecido, o EDF. Com isso, foi herdado as características do mesmo, como ser preemptivo e dinâmico.

O desempenho exposto por esse método de escalonamento foi superior no quesito média de tempo computacional e um número de perdas de deadlines satisfatórias ocorridas no decorrer dos testes aplicados.

Os testes aplicados para a medida de desempenho são os explicitados na Figura 1. Formado por 4 tasks aleatórias, pondo na prática como elas seriam processadas em cada algoritmo escolhido com a duração de 360ms, com 1000000 ciclos/ms e tendo como modelo de tempo de execução o WCET.

O diferencial do algoritmo está no código, que ao invés de só utilizarmos o conceito de priorizar o menor deadline absoluto, propomos uma soma entre o próprio deadline absoluto com o WCET, que no caso seria o tempo de execução.

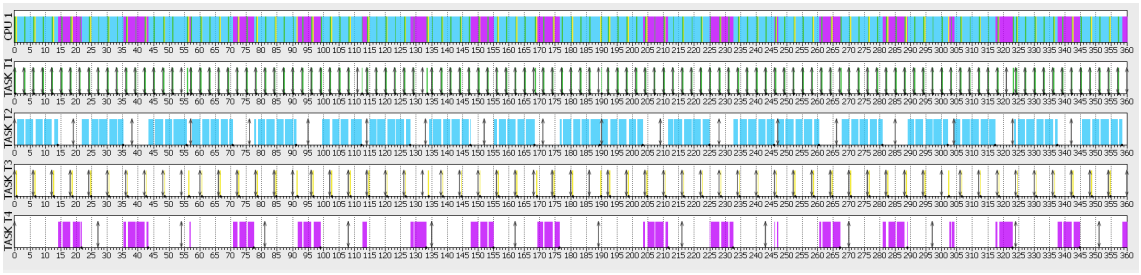
Código do NTM:

```
"""
NIGHTMARE algorithm for uniprocessor architectures.
"""

from simso.core import Scheduler, Timer


class NTM(Scheduler):
    def init(self):
        self.ready_list = []
        self.running_job = None
        self.quantum = 1 # ms
        self.timer = Timer(self.sim, NTM.reschedule,
                           (self, self.processors[0]), self.quantum, one_shot=False,
                           cpu=self.processors[0])
        self.timer.start()

    def reschedule(self, cpu):
        if self.ready_list == 0:
            cpu.resched()

    def on_activate(self, job):
        self.ready_list.append(job)
        job.cpu.resched()

    def on_terminated(self, job):
        self.ready_list.remove(job)
        job.cpu.resched()

    def schedule(self, cpu):
        if self.ready_list:
            # job with the highest priority
            job = min(self.ready_list, key=lambda x: x.wcet + x.absolute_deadline)
        else:
            job = None

        return (job, cpu)
```

3 Resultados

3.1 Teste com o EDF

O EDF é conhecido por ser simples e eficaz e se saiu muito bem nos testes. Executou todas os processos e não ocorreu nenhuma perda de deadline.

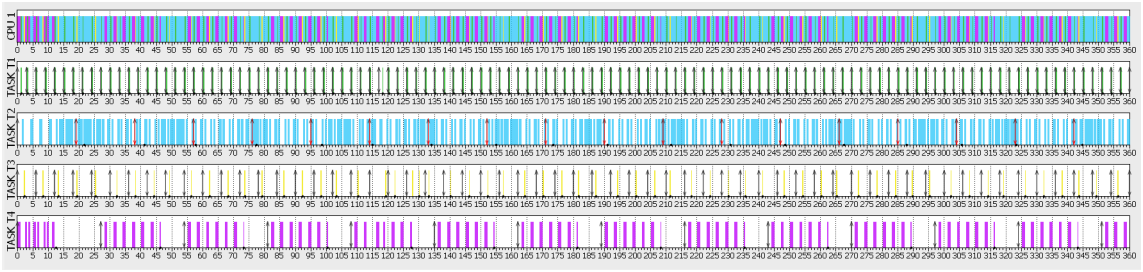


Computation time:

Task	min	avg	max	std dev	occupancy
TASK T1	0.350	0.350	0.350	0.000	0.117
TASK T2	11.245	11.245	11.245	0.000	0.593
TASK T3	0.331	0.331	0.331	0.000	0.055
TASK T4	6.382	6.382	6.382	0.000	0.235

3.2 Teste com o RR

Já o Round Robin não se comportou muito bem nos testes propostos. As linhas vermelhas nas tasks indicam os deadlines ocorridos. A imagem abaixo demonstra a tabela com os 18 perdas de deadlines.



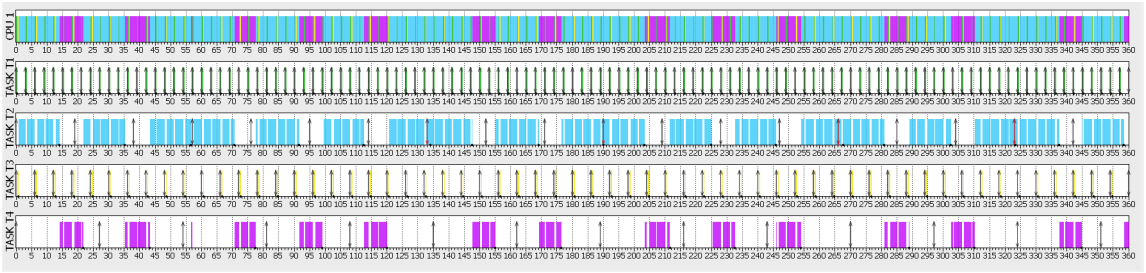
Computation time:

Task	min	avg	max	std dev	occupancy
TASK T1	0.350	0.350	0.350	0.000	0.117
TASK T2	11.245	11.245	11.245	0.000	0.589
TASK T3	0.331	0.331	0.331	0.000	0.055
TASK T4	6.382	6.382	6.382	0.000	0.239

Activation	Start	End	Deadline	Comp. time	Resp. time	CPI	Preemptions	Migrations
0.0000	0.0000	21.7500	19.0000	11.2446	21.7500		13	0
19.0000	21.7500	41.0869	38.0000	11.2446	22.0869		15	0
38.0000	41.0869	57.8062	57.0000	11.2446	19.8062		14	0
57.0000	57.8062	77.5255	76.0000	11.2446	20.5255		15	0
76.0000	77.5255	98.5432	95.0000	11.2446	22.5432		15	0
95.0000	98.5432	114.9317	114.0000	11.2446	19.9317		14	0
114.0000	114.9317	133.9817	133.0000	11.2446	19.9817		13	0
133.0000	133.9817	155.0510	152.0000	11.2446	22.0510		16	0
152.0000	155.0510	173.3879	171.0000	11.2446	21.3879		14	0
171.0000	173.3879	191.1072	190.0000	11.2446	20.1072		15	0
190.0000	191.1072	211.5072	209.0000	11.2446	21.5072		15	0
209.0000	211.5072	230.8441	228.0000	11.2446	21.8441		14	0
228.0000	230.8441	248.5634	247.0000	11.2446	20.5634		15	0
247.0000	248.5634	267.6327	266.0000	11.2446	20.6327		15	0
266.0000	267.6327	287.9696	285.0000	11.2446	21.9696		14	0
285.0000	287.9696	305.6889	304.0000	11.2446	20.6889		15	0
304.0000	305.6889	323.4081	323.0000	11.2446	19.4081		15	0
323.0000	323.4081	344.8082	342.0000	11.2446	21.8082		15	0
342.0000	344.8082		361.0000				12	0

3.3 Teste com o NTM

Já o NTM foi o melhor em quesito velocidade de execução, porém ocorrendo 4 perdas de deadlines.



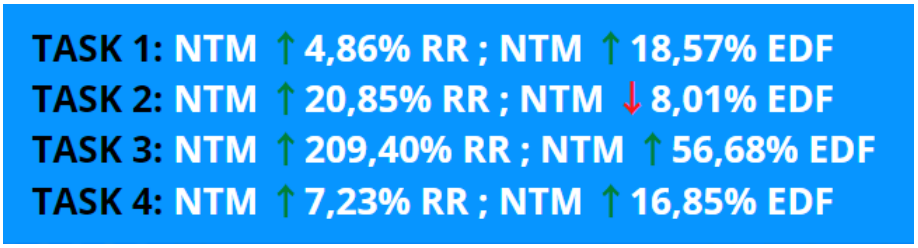
Computation time:

Task	min	avg	max	std dev	occupancy
TASK T1	0.350	0.350	0.350	0.000	0.117
TASK T2	11.245	11.245	11.245	0.000	0.593
TASK T3	0.331	0.331	0.331	0.000	0.055
TASK T4	6.382	6.382	6.382	0.000	0.235

Activation	Start	End	Deadline	Comp. time	Resp. time	CPI	Preemptions	Migrations
0.0000	0.0000	13.9869	19.0000	11.2446	13.9869		4	0
19.0000	19.0000	35.0562	38.0000	11.2446	16.0562		4	0
38.0000	38.0000	56.4562	57.0000	11.2446	18.4562		4	0
57.0000	57.0000	70.6561	76.0000	11.2446	13.6561		4	0
76.0000	76.0000	91.5124	95.0000	11.2446	15.5124		5	0
95.0000	95.0000	112.5817	114.0000	11.2446	17.5817		5	0
114.0000	114.0000	133.9817	133.0000	11.2446	19.9817		5	0
133.0000	133.9817	147.6379	152.0000	11.2446	14.6379		5	0
152.0000	152.0000	169.0379	171.0000	11.2446	17.0379		5	0
171.0000	171.0000	190.1072	190.0000	11.2446	19.1072		5	0
190.0000	190.1072	203.4133	209.0000	11.2446	13.4133		4	0
209.0000	209.0000	224.8134	228.0000	11.2446	15.8134		4	0
228.0000	228.0000	245.8826	247.0000	11.2446	17.8826		4	0
247.0000	247.0000	267.6327	266.0000	11.2446	20.6327		5	0
266.0000	267.6327	280.9388	285.0000	11.2446	14.9388		4	0
285.0000	285.0000	302.3389	304.0000	11.2446	17.3389		4	0
304.0000	304.0000	323.4081	323.0000	11.2446	19.4081		4	0
323.0000	323.4081	337.3951	342.0000	11.2446	14.3950		5	0
342.0000	342.0000	358.4643	361.0000	11.2446	16.4643		5	0

3.4 Comparações

O NTM em modo geral, ofereceu uma velocidade significativa no tempo de execução em relação aos dois outros algortimos apresentados.



4 Conclusão

O NTM tem como objetivo superar dois dos mais de vinte e cinco algoritmos disponíveis na biblioteca do simulador.

O desempenho se mostrou superior aos métodos citados acima nos testes dos quais foram propostos a executar. O número de deadlines nas tasks foi satisfatório, e o tempo computacional de execução foi superior em ambos os algoritmos que foram propostos. Assim, temos que o algoritmo se mostrou eficiente e se tornando uma boa opção como método de escalonamento diante das vastas opções disponíveis nos dias de hoje.

5 Trabalhos Relacionados e Referências

Foram utilizados como base para realização de comparações e exemplos o livro **"Sistemas de Tempo Real - Escola de Computação"**, disponibilizado por Jean Marie Farine.

"Simulador de Escalonamento para Sistemas de Tempo Real" – Universidade Federal da Bahia (UFBA). Giselia Magalhaes Cruz, George Lima.

"SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms," -Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France. Maxime Cheramy, Pierre-Emmanuel Hladik and Anne-Mariee Deplanche;

“DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling,” -in Proc. of ECRTS ’10, 2010. G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt.

“Early-release fair scheduling,” -in Proc. of ECRTS ’00, 2000. J. Anderson and A. Srinivasan.

“Multiprocessor scheduling with few pre-emptions,” -in Proc. of RTCSA, 2006. B. Andersson and E. Tovar.