Assignment 1

Serviços de Tempo Real em Linux

Cada teste consistiu na execução do respetivo programa durante 30 segundos. Ao fim deste período o programa foi cancelado e o seu output foi registado nas tabelas apresentadas. ***Testes sem "load"***, correspondem a testes que foram realizados sem nenhum programa que usasse intensivamente os recursos do CPU a correr em simultâneo com o programa de teste (periodicTask.c ou schedDeadlineTask.c). ***Testes com "load"*** correspondem a testes onde foi visualizado um video 4k e foi executado o programa **stress** em simultâneo com o programa de teste

b) *Launch the process and then other processes, concurrently, in different terminals and observe the behavior. Use, in particular, processes that generate intensive I/O operations (e.g. backup a big folder with tar, watch youtube videos). Repeat the operation several times and annotate the results.*

Teste sem load	
min	**max**
99999784	99999784
99999295	99999784
99990250	99999784
99988832	100004935
99988832	100009360
99981302	100016290
99980098	100019213

Teste com load	
min	**max**
100000512	100000512
99998506	100001988
99997718	100002862
97910214	102089925
94659522	104749776
87824585	112170248

Pela a análise do programa com diferentes cargas podemos concluir que o inter-arrival time varia mais (máximo é maior e o mínimo é menor) quando existe tarefas consideradas "CPU intensive" a serem executadas em simultâneo com o programa de teste.

d) *Modify the program in such a way that the periodic thread receives a given fixed real-time priority. Repeat now the experiments made in b). Make sure you use a similar interfering load (i.e., the same applications are used).*

Prioridade 1

Teste sem load, prioridade 1	
min	**max**
100000147	100000147
99999651	100000147
99997508	100000147
99995326	100003485
99990107	100010240
99987208	100012267
99974859	100029794
99973424	100029794
99968791	100029794

Teste com load, prioridade 1	
min	**max**
100001407	100001407
100000067	100001407
99998834	100001407
99996479	100001407
99995244	100005593
99992269	100006411
99990757	100006675
99987927	100012151
99985275	100019881
99977681	100019881
94451448	105539256

Prioridade 50

Teste sem load, prioridade 50	
min	**max**
99994538	99994538
99990193	100011065
99989896	100011065
99988283	100013017
99987586	100013017

Teste com load, prioridade 50	
min	**max**
100013738	100013738
99951758	100091232
99935924	100091232
99925967	100108302
99896710	100186290
99817646	100186290
99814207	100202108
99764379	100241883
99751245	100285284
99529169	100315600

Prioridade 99

Teste sem load, prioridade 99	
min	**max**
99999027	99999027
99998561	100000823
99995688	100002381
99990113	100013207
99988931	100013207
99973414	100028921
99970437	100031715
99964372	100031715

Teste com load, prioridade 99	
min	**max**
99841731	99841731
99766485	100191119
99646465	100421413
99545451	100456998

Com realtime fixed priority podemos notar que o inter arrival time varia muito menos que em b) uma vez que usa prioridades estáticas, que têm precedência ao timesharing.

e) *Modify the program to allow passing priorities via command line, as an argument. Execute several instances of the application, in different terminals, with different priorities. Analyze the results.*

priority 1	
min	**max**
99834184	99834184
99528506	100522785

priority 50	
min	**max**
99900472	99900472
99857798	100173692
99311461	100692120
99209793	100714484

priority 99	
min	**max**
100136925	100136925
99877645	100136925
99845781	100136925
99770856	100229533
99755341	100439577
99751777	100439577
99740241	100439577
99724608	100439577

Quanto maior a prioridade menor é a variação do max time no inter-arrival time. Isto deve-se ao facto das tasks com maior prioridade serem executadas antes de tasks com menor prioridade.
Por outro lado, no caso de ambas as tasks terem a mesma prioridade, é aplicado na nossa implementação uma FIFO para decidir qual será executada primeiro. Neste caso, a tarefa a ser executada será a mais antiga.

f) *Try the EDF example. Vary the task load and see the impact on the CPU utilization*

Ao aumentar o tempo de runtime, período e deadline existe uma maior carga no cpu. Tal deve-se ao facto das tarefas do deadline scheduler não serem preempted a não ser que se atinja o tempo de deadline ou que o runtime tenha acabado. Para além disso o aumento do período faz com que a tarefa ocorra mais frequentemente (f=1/T), resultando daí uma maior taxa de ocupação do cpu.

g) *Evaluate the regularity of the deadline scheduler. Compare it with the one of RR and FIFO.*

EDF without Load	
min	max
921410097	921410097
921410097	951451440
921410097	981427053
11444939	981427053
1423495	981427053

O deadline scheduler possui uma maior irregularidade quando comparado ao RR e FIFO em que o minimal arrival time é muito menor e o max time pouco varia. Tal deve-se ao facto de existir uma deadline, que faz com que o max time seja limitado superiormente e para além disso este scheduler tem maior prioridade em relação aos restantes, resultando daí um min time mais variável entre o runtime e o deadline.

h) *Execute concurrently EDF and RR/FIFO tasks and conclude on the mutual interference.*

O deadline scheduler tem maior prioridade que qualquer outro scheduler, e por isso as tasks da FIFO e RR apenas serão executadas quando é ultrapassado o deadline, quando não está a ser executada uma task no deadline ou quando não está outra task em deadline à espera. Como tal o min irá variar pouco no FIFO e RR e o max irá crescer mais rapidamente.