Implementação e Análise de Thread Periódica no Linux

Thiago Nogiri Igarashi (18102009)
Caroline Laura Bagatini (18103421)
Universidade Federal de Santa Catarina (UFSC)

I. Implementação

Esse trabalho consiste na implementação de tarefas periódicas no Linux. A linguagem utilizada foi C++. Para tal implementação, foram criadas duas classes, uma para contemplar o funcionamento de um timer e outro para representar a tarefa periódica. O timer utilizado é o padrão POSIX, que envia um sinal a cada período completo e o valor do período é inserido pelo usuário no início da execução da tarefa. O construtor da classe Timer inicializa todos os valores necessários para seu funcionamento, mas não inicia o timer. Para iniciar e parar o timer, duas funções foram implementadas, chamadas de timer_enable e timer_disable. Nessa implementação, o deadline da tarefa é igual ao seu período.

No construtor da classe da tarefa, é feita a inicialização das variáveis presentes na classe, além disso o timer é criado e iniciado. Na classe também há uma função chamada *run()*, que corresponde ao código executado pela tarefa. Essa função utiliza a função sigwait, que mantém a tarefa inativa enquanto o sinal especificado em seu parâmetro não for ativado. Nesse caso o sinal que a função espera é o sinal que o timer ativa, SIGALRM. Uma vez ativado o sinal, o corpo da função run executa e ao final do seu ciclo, a função *tempoderesposta()*, implementada também na classe da tarefa, calcula a diferença entre o tempo que o sinal foi ativo e o tempo que a tarefa terminou o ciclo, e imprime em tela esse valor. Se por acaso o período do timer completar enquanto a tarefa ainda não terminou de executar, o handler do timer é ativo, imprimindo em tela que o deadline da tarefa foi perdido.

No arquivo *main.cpp*, é pedido ao usuário que insira 4 informações: período da tarefa em milissegundos, prioridade da tarefa, fator de carga da CPU e política de escalonamento. O período da tarefa é usado para criar o timer e o fator de carga é usado na função *run()* da tarefa. A prioridade da tarefa e a política de escalonamento são usados como parâmetro para a função *sched_setscheduler()* da biblioteca *sched.h*, usada no trabalho. Após configurar o escalonador, a função *main()* cria a tarefa a ser executada e chama a função *run()* da tarefa.

II. Resultados

Ao executar cada tarefa periódica separadamente, é observado uma baixa variação do tempo de resposta, sendo que o deadline só é perdido com a diminuição do valor de entrada do período da tarefa ou com um grande aumento do fator de carga.

Ao executar ambas as tarefas em paralelo no mesmo processador, utilizando o comando taskset no terminal, entrando com os valores de período de 500ms e fator de carga (*load*) de 100 mil, tendo ambas as tarefas a mesma prioridade e política de escalonamento SCHED_RR, é possível observar um aumento no tempo de resposta de ambas as tarefas, comparado à execução individual, e uma variação no range do tempo de resposta de cada tarefa. No próximo teste, onde foi utilizado a mesma política de escalonamento SCHED_RR mas com diferentes prioridades, a tarefa com maior prioridade voltou a ter tempo de resposta semelhante de sua execução individual, pois qualquer outra tarefa executando cederá o processador para ela, enquanto a tarefa com menor prioridade teve um aumento considerável na média do tempo de reposta.

Com esses resultados, pode-se observar alguns pontos:

- um aumento no fator de carga ou diminuição do período da tarefa podem fazer com que ela perca seu deadline, e vice-versa;
- tarefas com prioridade maior possuem menor tempo de resposta;
- para duas tarefas com diferentes prioridades, a política de escalonamento não tem influência no tempo de resposta;
- tarefas com prioridades iguais, fator de carga próximo e política de escalonamento RR possuem tempo de resposta similar
- tarefas com prioridades iguais, fator de carga próximo e política de escalonamento FIFO dependem da ordem de chegada da tarefa, portanto não se pode aferir nada.