Universidade do Vale do Itajaí
Disciplina: Sistemas Operacionais
Professor: Felipe Viel
Trabalho M2 – Escalonamento

Instruções

1. Esta avaliação deve ser feita individualmente ou até em trio.
2. Data de entrega: 27/05/2024 até 19:00. Não serão aceitos trabalhos entregues em atraso.
3. Esta avaliação tem por objetivo consolidar o aprendizado sobre escalonamento.
4. A implementação deverá ser desenvolvida utilizando as bibliotecas e arquivos cedidos. Poderá ser usado o equivalente no Windows. O uso de bibliotecas de terceiros deverá ser explicado com uma justificativa válida que ficará ao critério do professor aceitar ou não.
5. Link para os arquivos: Scheduler
6. O sistema deve ser entregue funcionando corretamente.
7. Deve ser apresentado um relatório eletrônico em formato PDF (em outro formato é descontado 1,5 ponto) que contenha:
    a. Identificação do autor e do trabalho.
    b. Enunciado do projeto.
    c. Explicação e contexto da aplicação para compreensão do problema tratado pela solução.
    d. Resultados obtidos com as simulações.
    e. Códigos importantes da implementação.
    f. Resultados obtidos com a implementação (tabelas, gráficos e etc).
    g. Análise e discussão sobre os resultados finais.
8. Deve ser disponibilizado os códigos da implementação juntamente com o relatório (salvo o caso da disponibilidade em repositório aberto do aluno, que deve ser fornecido o link). Também deve ser apresentado o trabalho em aula para o professor. Na apresentação do trabalho, não é necessário utilizar slides, apenas apresentar o código e sua execução (compilar na apresentação).

Descrição do projeto a ser desenvolvido

Projeto 1

Para consolidar o aprendizado sobre os escalonadores, você deverá implementar dois algoritmos de escalonadores de tarefas (taks). Os escalonadores são o Round-Robin com prioridade (RR_p) e EDF (Earliest Deadline First). Para essa implementação, são disponibilizados os seguintes arquivos (Link Github):

- driver (.c) – implementa a função main(), a qual lê os arquivos com as informações das tasks de um arquivo de teste (fornecido), adiciona as tasks na lista (fila de aptos) e chama o escalonador. Esse arquivo já está pronto, mas pode ser completado.
- CPU (.c e .h) – esses arquivos implementam o monitor de execução, tendo como única funcionalidade exibir (via print) qual task está em execução no momento. Esse arquivo já está pronto, mas pode ser completado.

- list (.c e .h) - esses arquivos são responsáveis por implementar a estrutura de uma lista encadeada e as funções para inserção, deletar e percorrer a lista criada. Esse arquivo já está pronto, mas pode ser completado.

- task (.h) – esse arquivo é responsável por descrever a estrutura da task a ser manipulada pelo escalonador (onde as informações são armazenadas ao serem lidas do arquivo). Esse arquivo já está pronto, mas pode ser completado.

- scheduler (.h) – esse arquivo é responsável por implementar as funções de adicionar as task na lista (função add()) e realizar o escalonamento (schedule()). Esse arquivo deve ser o implementado por vocês. Você irá gerar as duas versões do algoritmo de escalonamento, RR_p e EDF em projetos diferentes.

Você poderá modificar os arquivos que já estão prontos, como o de manipulação de listas encadeada, para poder se adequar melhor, mas não pode perder a essência da implementação disponibilizada. Algumas informações sobre a implementação:

- Sobre o RR_p, a prioridade só será levada em conta na escolha de qual task deve ser executada caso haja duas (ou mais) tasks para serem executadas no momento. Em caso de prioridades iguais, pode implementar o seu critério, como quem é a primeira da lista (por exemplo). Nesse trabalho, considere a maior prioridade como sendo 1. Além disso, é obrigatório o uso de múltiplas filas para a gerência de prioridade.

- Você deve considerar mais filas de aptos para diferentes prioridades. Acrescente duas taks para cada prioridade criada.

- A contagem de tempo (slice) pode ser implementada como desejar, como com bibliotecas ou por uma variável global compartilhada.

- Lembre-se que a lista de task (fila de aptos) deve ser mantida “viva” durante toda a execução. Sendo assim, é recomendado implementar ela em uma biblioteca (podendo ser dentro da próprio schedulers.h) e compartilhar como uma variável global.

- Novamente, você pode modificar os arquivos, principalmente o “list”, mas sem deixar a essência original deles comprometida. Porém, esse arquivo auxilia na criação de prioridade, já que funciona no modelo pilha.

- Para usar o Makefile, gere um arquivo schedule_rr.c, schedule_rrp.c e schedule_fcfs.c que incluem a biblioteca schedulers.h (pode modificar o nome da biblioteca também). Caso não queira usar o Makefile, pode trabalhar com a IDE de preferência ou compilar via terminal.

- Utilize um slice de no máximo 10 unidades de tempo.

- Para os algoritmos, você deverá, via uma primeira thread extra, a simulação da ocorrência do timer em hardware. Essa thread irá fazer a simulação do tempo e gerará a flag de estouro do tempo (para o slice). Além disso, para o algoritmo EDF será necessário avaliar o deadline das tasks e verificar qual das tasks está com o menor deadline.

Pontuação Extra na prova (máximo 1,5 ponto):

- Como uma segunda thread extra, você deverá simular um contador de tempo global que irá simular o sleep das tasks de alta prioridade. Com isso, você deverá marcar as threads de prioridade 1 como real-time e implementar um dos dois algoritmos como escalonador totalmente preemptivo. Essa simulação de sleep permite que você force o escalonador a interromper qualquer task que esteja em execução quando o sleep da task real-time tenha feito ela despertar. Essa thread irá gerar o efeito da interrupção de software/hardware do sleep gerada pelas tasks de real-time.