Execução multi-processo postergada

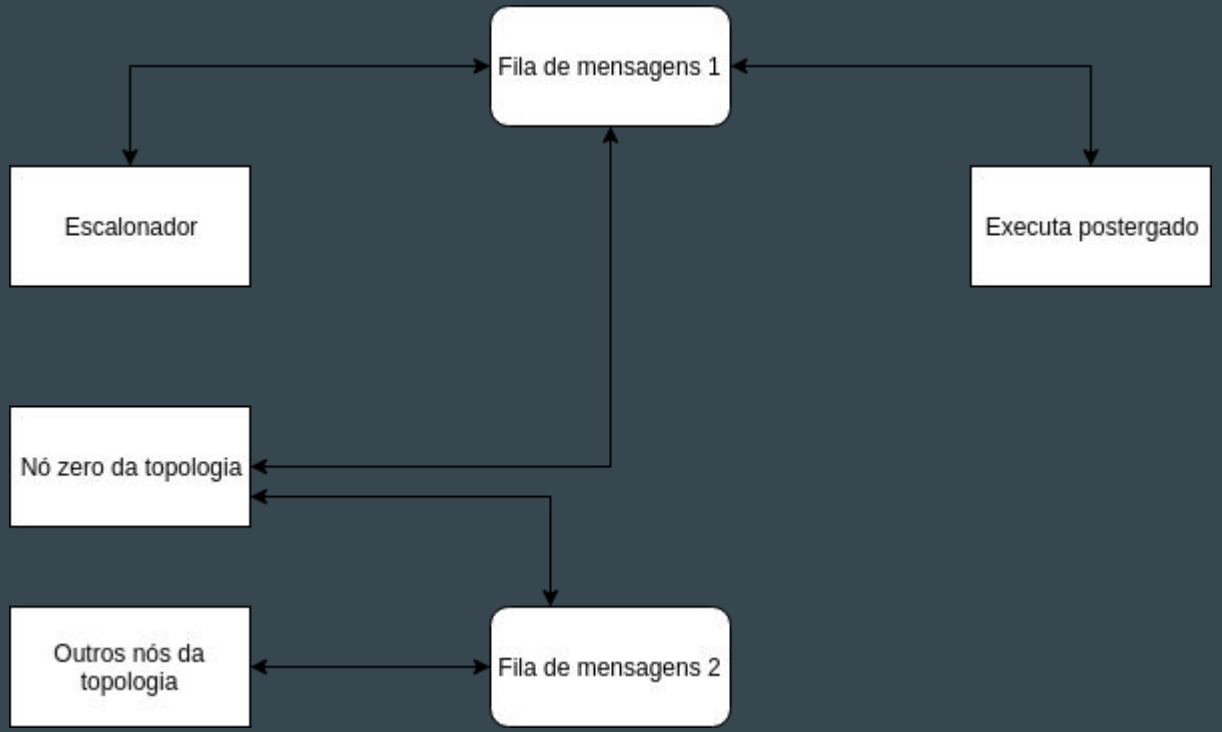
André Luís Souto Ferreira 14/0016261
Otto Kristian von Sperling 12/0131510
Victor Fabre Figueiredo 15/0022948

Introdução

Implementação de um gerente de execução de processos múltiplos de maneira postergada.

A proposta do trabalho é fazer um mecanismo de função semelhante ao comando *at* existente no Unix. O sistema possui um processo escalonador multi-processo e 16 nós (15 no caso da topologia *fattree*), cada nó com um processo gerente de execução de processos.

Diagrama do projeto



Principais processos

A) executa_postergado.c

Solicita a execução de um processo com um delay determinado. Esse processo faz uso de fila de mensagens para poder realizar a solicitação da execução de outro processo e para informar para o escalonador o delay mínimo de execução.

Chamada: ./executa_postergado -f arquivo_executável -d delay

Mecanismos de mensagem usados

- Fila de mensagens: as mensagens enviadas contém o nome do arquivo executável e o delay de execução.

B) escalonador.c

Responsável por gerenciar a ordem de execução dos processos solicitados tendo como base o delay de cada processo. Cria uma fila de execução que é atualizada cada vez que um processo novo é inserido ou que um processo é executado e removido da fila. Também responsável por criar os gerentes de processo e determinar a topologia que será usada. Toda a comunicação é feita por troca de mensagens. É executado em *background*.

Também é responsável pelo *shutdown* dos processos e por guardar as estatísticas de execução.

Chamada: ./escalonador -t topologia &

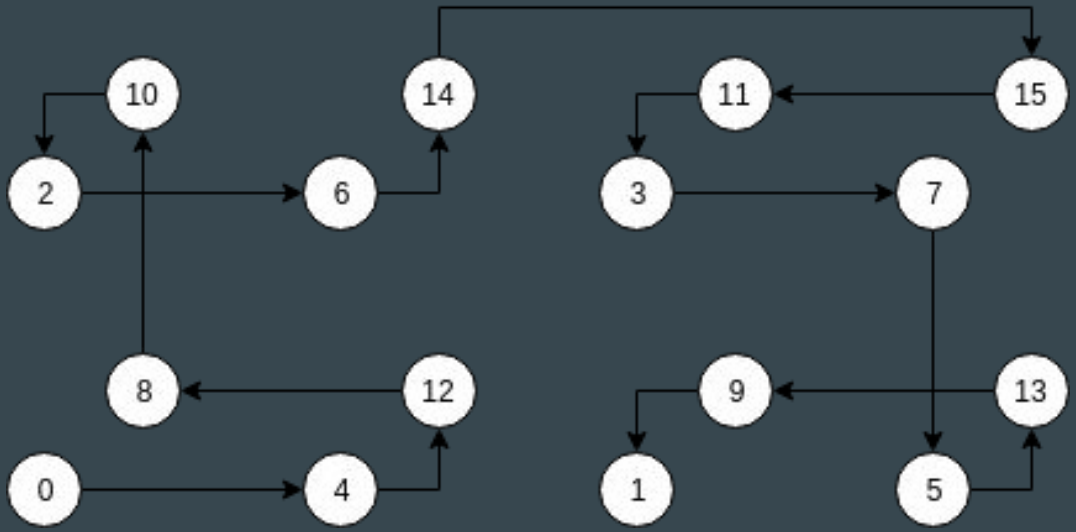
Topologias

A) Hypercube

Topologia que contém 16 nós, onde o nó zero se comunica com o escalonador recebendo o nome do arquivo que deverá ser executado e é responsável por passar essa informação para os outros nós, assim como passar as informações de tempo de execução de todos os nós para o escalonador. Todas as informações são trocadas via fila de mensagens, o tipo da mensagem é o número de cada nó, dessa forma, pode-se usar a mesma fila para todos os nós e não ocorrem problemas no consumo das mensagens na fila.

A) Hypercube

O caminho das mensagens entre os nós é feito como na imagem abaixo, todas as conexões são bidirecionais.

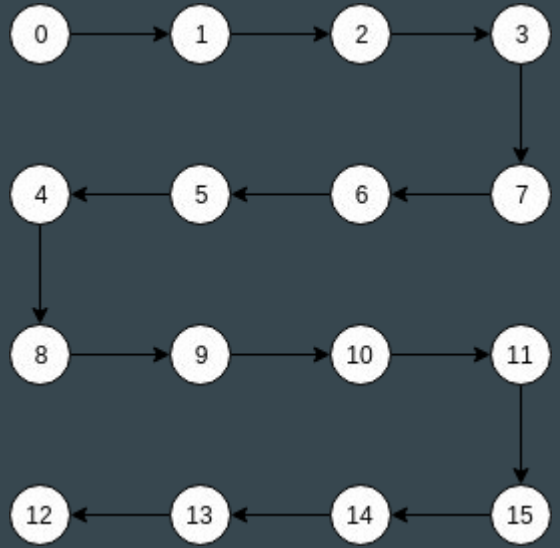


B) Torus

Topologia que contém 16 nós, onde o nó zero se comunica com o escalonador recebendo o nome do arquivo que deverá ser executado e é responsável por passar essa informação para os outros nós, assim como passar as informações de tempo de execução de todos os nós para o escalonador. Todas as informações são trocadas via fila de mensagens, o tipo da mensagem é o número de cada nó, dessa forma, pode-se usar a mesma fila para todos os nós e não ocorrem problemas no consumo das mensagens na fila.

B) Torus

O caminho das mensagens entre os nós é feito como na imagem abaixo, todas as conexões são bidirecionais.



C) Fattree

Topologia que contém 15 nós, onde o nó zero se comunica com o escalonador recebendo o nome do arquivo que deverá ser executado e é responsável por passar essa informação para os outros nós, assim como passar as informações de tempo de execução de todos os nós para o escalonador. Todas as informações são trocadas via fila de mensagens, o tipo da mensagem é o número de cada nó, dessa forma, pode-se usar a mesma fila para todos os nós e não ocorrem problemas no consumo das mensagens na fila.

C) Fattree

O caminho das mensagens entre os nós é feito como na imagem abaixo, todas as conexões são bidirecionais.