Universidade Federal de Roraima
Centro de Ciências e Tecnologia
Departamento de Ciência da Computação

Professor: Herbert Rocha
Aluno: William Juan da Silva Melo

Laboratório de Sistemas Operacionais
2019.1

Ferramentas utilizadas: simulador SoSim e uma IDE para programar em C, no caso eu usei o Codeblocks 17.12. Sistema: Windows 10 SL

Questão 1

PRÁTICA A:

Observe que em alguns momentos existem processos no estado de pronto porém nenhum em estado de execução. Explique o porquê dessa situação:

Resposta: Isso ocorre porque o escalonador primeiro verifica se está livre para mandar alguém para execução. Se estiver ocupado ele deixar em modo pronto. Se estiver desocupado ele escalona o processo de maior prioridade para execução. Qual o processo termina de ser executado ele é colocado em estado de pronto no final da fila, com isso ficam dois processos na fila e nada em execução. Porque o escalonador tem que verificar se está livre ou não para escalonar outro processo para execução, estando livre ele escalona outro processo e manda para execução.

PRÁTICA B:

- Por que o problema do starvation pode ocorrer?
- Cite duas ações que o administrador do sistema pode realizar quando é identificada a situação de starvation em um processo?

OBS: Foram criados dois processo um CPU-bound prioridade 4 e outro processo I/O-bound prioridade 3 e escalonamento circular com prioridade estática.

Resposta: Segundo a análise que podemos fazer com o SOSIM é que o processo I/O-bound nunca é escalonado para execução. Pois, o processo cpu-bound tem prioridade

maior que o processo i/o-bound sendo 4 e 3. Como o escalonador sempre dá prioridade para processo de maior prioridade, logo só escalona o processo cpu-bound que tem prioridade maior. Com isso, o processo i/o-bound sofre com o starvation pois nunca consegue acessar o recurso compartilhado porque o outro processo impede que ele seja escalonado por ter maior prioridade. O que o administrador do sistema pode fazer é matar o processo cpu-bound para que o outro processo i/o-bound possa ser escalonado para execução ou diminuir a prioridade do processo cpu-bound ou aumentar a prioridade do processo i/o-bound para que ele possa ser escalonado para execução.

PRÁTICA C:

- Qual o espaço de endereçamento real máximo de um processo?

Resposta: Corresponde a quantidade de memória principal e memória virtual juntas.

- Qual o espaço de endereçamento real mínimo de um processo?

Resposta: Corresponde a o tamanho mínimo da tabela de mapeamento carregada.

- Qual o tamanho da página virtual?

Resposta: Pode variar devido a diversos fatores como: processador, arquitetura de hardware, além de em alguns sistemas operacionais ser possível configurá-lo manualmente.

OBS: foram criados dois processos cpu-bound com prioridade 0, escalonamento circular, ou clicado em PCB.

PRÁTICA D:

- Quais os critérios utilizados pelo simulador para selecionar o processo a ser transferido para o arquivo de paginação (swap out)?

Resposta: ele seleciona o processo com menor chance de execução pelo processador. Neste caso vários algoritmos de escalonamento de CPU, podem ser utilizados. Ou seja, manda salva na memória de paginação o bloco que está a mais tempo sem ser usado para dar lugar a outro processo.

- Quando o processo deve ser transferido novamente para a memória principal (swap in)?

Resposta: quando necessita reescalonar um processo para execução. Ou seja, continuar com a execução de onde parou.

2) Com relação ao problema de Deadlock. Pesquisa e descreva o algoritmo do banqueiro (criado por Dijkstra) que pode ser utilizado para evitar impasses. Sempre

que recursos são solicitados, o algoritmo avalia se atender à solicitação levará a um estado inseguro e se isso ocorrer, ela não é atendida. Adicionalmente, escreva o algoritmo do banqueiro em C/C++ e apresente alguns exemplos de sua execução.

O Algoritmo do banqueiro é um algoritmo de alocação de recursos com prevenção de impasses desenvolvido por Edsger Dijkstra. Ele é executado pelo sistema operacional quando um processo de computação requisita recursos.

O algoritmo impede o impasse, ao negar ou adiar o pedido se ele determinar que aceitar o pedido pode colocar o sistema em um estado inseguro (onde um impasse poderia ocorrer). Quando um novo processo entra em um sistema, ele deve declarar o número máximo de instâncias de cada tipo de recurso que não pode exceder o número total de recursos no sistema. Para o algoritmo do banqueiro trabalhar, ele precisa saber três coisas:

- Quanto de cada recurso cada processo poderia solicitar.
- Quanto de cada recurso cada processo atualmente detém.
- Quanto de cada recurso o sistema tem disponível.

Quando o sistema recebe um pedido de recursos, é executado o algoritmo do banqueiro para determinar se ele é seguro com o propósito de se deferir o pedido. O algoritmo é bastante simples uma vez a distinção entre os estados seguros e inseguros ter sido compreendida.
- Pode o pedido ser concedido?
- * Se não, o pedido é impossível e deve ser negado ou colocado em lista de espera
- Suponha que o pedido é concedido
- O novo estado é seguro?
- * Sendo assim deferir o pedido
- * Se não, negar o pedido, ou colocá-lo em uma lista de espera

Se o sistema negar ou adiar um pedido impossível ou inseguro é uma decisão específica do sistema operacional.

3) Com relação a problemas clássicos de comunicação entre processos. Escreva o algoritmo do barbeiro (visto em sala de aula), usando threads, em C/C++ e apresente alguns exemplos de sua execução.

Descrição do algoritmo: um barbeiro, assentos para os clientes com capacidade N. Quando não tem ninguém na barbearia o barbeiro dorme da cadeira de barbear, caso um cliente chegue ele acorda o barbeiro para fazer sua barba, se chegar outro cliente e ainda tiver lugar para sentar na cadeira de espera que tem capacidade N, ele fica. Caso contrário vai embora por está cheio o estabelecimento.

Sobre a minha implementação: para a implementação do problema do barbeiro são usados Threads para representar o barbeiro e os clientes. E usando técnicas de exclusão mútua para gerenciar os recursos com otimização.