PseudoOS - Projeto SO - 1/2023

Gabriel N. Halabi -150010290, Gabriel C. Moretto -150154917

Julho de 2023

1 Ferramentas e linguagens usadas

- python3 para implementação
- make para simplificar a execução
- pylint para verificação estática do código
- Github para versionamento
- Overleaf para o desenvolvimento do relatório

2 Descrição teórica e prática da solução

Para implementar o protótipo de pseudo-sistema operacional para o projeto, foram desenvolvidos 5 classes de gerenciadores e uma classe de processo básica com o fim de abstrair os recursos que normalmente seriam de hardware e manejar os processos. O projeto foi desenvolvido de forma modularizada e orientada à objetos, definindo cada um dos gerenciadores como instâncias de uma classe, permitindo uma certa escalabilidade virtual do sistema visto que é possível instanciar mais de um gerenciador. Os gerenciadores e recursos também foram desenvolvidos de forma mais versátil permitindo configurações iniciais que alteram seus comportamentos.

2.1 Processos

Os processos recebem um conjunto de inteiros como argumento em sua inizialização, definindo o ID único do processo, momento de chegada na CPU, nível de prioridade de execução, número de instruções que precisa executar e número de blocos de memória que precisa alocar para ser executado. Por meio de um de seus métodos, processos podem também definir um valor inteiro que é utilizado como bitmap para informar quais dispositivos do sistema ele requisita para ser executado.

Representar os dispositivos como um bitmap auxilia na requisição sequencial de dispositivos removendo a necessidade de guardar cada dispositivo em um

atributo dedicado e implementar um if para checar cada um destes atributos, pois na forma de bitmap basta mapear cada bit para um dispositivo e realizar um operação AND entre a lista de dispositivos requisitados e a potência de 2 referente ao dispositivo para saber se aquele está sendo requisitado ou não, permitindo também que o número de dispositivos definidos posteriormente no gerenciador de dispositivos seja mais versátil caso definido de forma diferente durante a inicialização dos objetos.

Os processos também possuem atributos que informam o estado de dormência e se devem ser encerrados. Quando um processo requisita um recurso para poder ser executado e não consegue acesso a este, eles deixam um ponteiro para si mesmos em uma fila de requisição do gerenciador de memória ou de dispositivos e se coloca em estado de dormência para não ficar repetindo as buscas por recurso todo ciclo. Desta forma, fica como responsabilidade dos gerenciadores "acordarem" o processo quando algum recurso for liberado para que o processo possa requisita-los novamente. O segundo atributo, uma flag de kill process, é utilizada pelo dispatcher para saber quando algum erro aconteceu na alocação de recursos e o processo tentou requisitar algum recurso que o sistema nunca vai possuir, portanto os gerenciadores mudam esta flag no processo para o dispatcher removê-los da fila de preparo.

2.2 Dispatcher

A classe do dispatcher é uma coleção de listas de processos e operações de sistema de arquivos. Nesta classe, são definidos métodos para realizar o parsing dos arquivos de texto e gerar os processos e operações de sistema de arquivos, salvando cada uma das entradas em uma lista global ordenada por ID unico destes objetos. Ponteiros para os processos são movidos para uma fila de chegada ordenada por ciclo de chegada dos processos para que o dispatcher possa ir movendo os processos para a fila de peraro. A fila de preparo, por sua vez, é onde os processos tentam requisitar os recursos necessários para a sua execução antes de serem passados à frente para as filas de execução do escalonador.

2.3 Escalonador

O escalonador é composto, principalmente, por um conjunto de filas de processos para representar os níveis de prioridade de execução. O escalonador executa as filas em ordem, iterando da primeira até a ultima fila, checando se possui algum processo e caso sim, executando esse e interrompendo a iteração.

A primeira fila, representada pela prioridade 0 é acessada por iterador de mesmo valor, é uma fila escalonada de forma não preemptável com uma lógica de FIFO (First In - First Out) simples.

As filas seguintes são filas com realimentação escalonadas por algoritmo de Round Robin com seus time slices definidos durante a instanciação do objeto escalonar para permitir versatilidade, portanto cada fila possui seu própio time slice para definir quantos quantum vai alocar para cada processo na fila. Quando um processo termina de utilizar seu time slice e ainda não foi concluído, ele é

removido da fila e realimentado na próxima fila de menor prioridade ou simplesmente movido para o fim da fila caso ja esteja na fila de menor prioridade.

2.4 Gerenciador de Memória

O gerenciador de memória é composto principalmente por uma string utilizada para representar a versão abstraída da memória principal do sistema. Esta string tem um determinado comprimento infromado durante a inicialização do objeto do gerenciador e pode ser ocupada com os characteres '0' e '1' para representar blocos livres e ocupados. Pode ser informado também um número variável de blocos reservados para processos de alta prioridade.

Antes de alocar memória para um processo, o gerenciador de memória primeiro verifica se foram requisitados mais blocos do que o sistema possui, definindo uma kill flag do processo para que este não fique ocupando a fila de preparo indefinidamente. Depois, o gerenciador verifica se possui memória livre suficiente para o processo antes de buscar pelos blocos contíguos para alocar, desta forma pode-se evitar iterar através da string quando a memória estiver muito cheia.

Para alocar a memória, o gerenciador itera através da string em busca de uma quantidade de '0' contíguos e separa a string em duas partes antes e após o padrão procurado. Para definir que aqueles blocos foram alocados basta concatenar a parte anterior com um número de '1' contíguos e a parte seguinte da string. É retornado então o tamanho da parte anterior, equivalente ao offset da posição na string onde a memória foi alocada, e -1 caso não tenha encontrado os blocos contíguos.

Quando um processo falha ao alocar memória, ele coloca um ponteiro para si mesmo na fila de espera do gerenciador de memória e se coloca em estado de dormência. Toda vez que memória é liberada, o gerenciador de memória itera através da fila de espera acordando os processos.

2.5 Gerenciador de Recursos / Dispositivos

O gerenciador de recursos é composto por uma lista de pares de uma string com o nome do dispositivo e uma fila de requisição. A busca por um dado dispositivo é feita à partir de sua posição na fila em relação ao bitmaps de requisição dos processos, primeiro bit representando um booleano se o primeiro dispositivo registrado está sendo requisitado ou não, e assim sucessivamente para cada dispositivo.

As filas de requisição de cada dispositivo guardam ponteiros para os processos que os requisitam e a primeira posição representa o processo que está atualmente em posse do recurso. Toda vez que um recurso é liberado, a posição seguinte da fila passa para a frente o gerenciador de dispositivos acorda o processo, similarmente ao que o gerenciador de memória faz, para avisá-lo que agora está em posse do dispositivo requerido para tentar se mover para as filas de execução.

2.6 Gerenciador de Sistema de Arquivos

O gerenciador de sistema de arquivos utiliza um método semelhante ao do gerenciador de memória, representando o espaço de disco com uma string de '0's e substituindo estes por outros caracteres contíguos que representam um determinado arquivo alocado naqueles blocos.

Há uma segunda estrutura de dados importante na classe que representa o superbloco do sistema de arquivos, com um conjunto de informações para arquivo como um caractere único, número de blocos que ocupa, offset e ID do processo que o criou. Este superbloco é utilizado para garantir que um processo não delete um arquivo para o qual não possua permissão para deletar e para garantir que arquivos novos tenham um nome único.

O sistema de arquivos define também um modo de execução de suas operações, que é usado pelo escalonador para identificar em que momento deve tentar executar as operações do sistema de arquivos. Os modos definidos foram:

- Modo 'batch': executa todas as operações associadas a um determinado processo de uma vez assim que o processo termina suas instruções na CPU.
- Modo 'synchronous': executa cada operação associada a um processo em uma de suas instruções, em ordem, podendo por vezes não chegar a ser executado quando o processo possuir menos instruções de cpu do que operações de FS a executar.
- Modo 'asynchronous': executa todas as operações de FS de uma vez, em ordem de definição, quando o escalonador terminar de executar todos os processos.

3 Principais dificuldades na implementação

Na especificação do projeto, algumas funcionalidades não ficaram claras em relação ao comportamento esperado. O pincipal exemplo disto foi o momento em que uma operação do sistema de arquivos é executado.

Outro ponto de maior reflexão na implementação do projeto foi como representar os recursos de hardware como memória, dispositivos e disco.

Com a limitação de tempo, também, e seguindo as sugestões da especificação, o gerenciamento de recursos não ficou particularmente eficiente podendo em alguns casos levar certos processos a altos tempos de espera.

4 Soluções para as dificuldades

A linguagem escolhida para o desenvolvimento deste projeto foi python precisamente para ajudar a abstrair os recursos e processos por meio de listas e strings. A memória e disco, por exemplo, utilizam strings pois o typo ja possui métodos nativos para buscar padrões de caracteres e separar o objeto em volta deste padrão, simplificando a alocação contígua.

O uso de filas de requisição na alocação de memória e recursos ajuda a reduzir o número de requisições que um mesmo processo tem de fazer para conseguir os recursos que precisa para ser executado, garantindo que em um sistema real onde não há uma quantidade infinita de processos chegando, que, eventualmente, todos os processos serão executados. Porém, a alocação de memória pode levar por vezes a grandes períodos de espera quando um processo precisa de muito memória para ser executado, o que requerem menos serão executados primeiro.

O escalonador e gerenciador de sistema de arquivos foram implementados com múltiplos modos de operação para alterar um pouco seus comportamentos, enquanto os outros gerenciadores também possuem uma inicialização mais versátil como consequência. O sistema pode ter quantidades variáveis de recursos mas ainda conseguindo satisfazer o definido na especificação do projeto.

Para fins de debugging e demonstração, cada módulo possui também níveis de 'verbose' ou 'log' que definem quanta informação é disponibilizada no terminal a cada ciclo da CPU.

5 Funções por aluno

5.1 Gabriel Carvalho Moretto

- recursos.py
- arquivos.py
- memoria.py
- Documentação
- Relatório

5.2 Gabriel Nazareno Halabi

- dispatcher.py
- filas.py
- processos.py
- Refatorção orientada a objetos
- Pylint coding standards compliance
- Manual de instalação, compilação e execução

6 Bibliografia

Tanenbaum, A. S., Bos, W. Sistemas Operacionais Modernos. Person, São Paulo, 4° ed., 2016.