Fundamentos de Sistemas Operacionais
Projeto Final - Simulador de um Sistema Operacional

Caio Peluti, 19/0085312
Gabriel Fontenele, 15/0126760
Ana Clara Jordão Perna, 19/0084006

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0205 - Fundamentos de Sistemas Operacionais - Turma 01 - Prof. Aletéia Patrícia

1. Introdução

Este trabalho consiste na implementação de um simulador de um sistema operacional que possui quatro gerências principais: gerência de processos, de memória, de arquivos e de E/S.

Os processos se classificam em processos de tempo real e de usuário. Os de tempo real possuem prioridade 0 e por isso, o gerenciador de processos possui uma fila própria em que se escalona por meio de FIFO. Já os de usuários possuem três filas com realimentação que atualizam a prioridade dos processos dinamicamente.

A memória RAM possui uma divisão de 64 blocos para processos de tempo real e 960 para os de ueários, sem necessidade de implementação de memória virtual, somente uma alocação contígua de processos.

O gerenciador de arquivos consiste em administrar os arquivos armazenados em disco, tanto na hora de criá-los, quanto na hora de deletá-los.

Por fim, o gerenciador de E/S administra a utilização de diversos dispositivos, são eles: um scanner, um modem, duas impressoras e dois dispositivos SATA.

2. Ferramentas Utilizadas

O projeto foi feito utilizando a linguagem de programação C++, modularizando o programa em classes e, dessa forma, deixando-o mais organizado e legível.

3. Descrição do Algoritmo Desenvolvido

3.1. Classe Process

A classe Process representa um processo e tem como atributos o pid (seu identificador), o run_time (marcador que guarda o tempo de processamento que o processo já teve), o init_time (tempo que o processo é criado), a priority (indica a prioridade do processo, sendo quanto menor, mais prioritário ele é), o exec_time (o tempo que ele precisa ficar na CPU para finalizar), alloc_mem_blocks (guarda o número de blocos que o processo quer alocar na memória RAM), printer_code (indica se o processo requisita alguma impressora), scan_req (indica se o processo requisita o scanner), modem_req (indica se o processo requisita o modem), disk_num (indica se o processo requisita algum dispositivo SATA), e por fim, wait (indica o tempo de espera do processo).

Entre seus métodos há o incrementador do tempo de espera, o método de redefinir o run_time para zero e o de atualizá-lo, e o de definição de novas prioridades para o

processo. Em relação aos dispositivos de entrada e saída, são encontrados os métodos para requisitar os dispositivos de E/S e para liberá-los.

3.2. Classe Operation

A classe Operation representa uma operação de criação ou deleção de um arquivo. Ela possui os seguintes atributos: id (que consiste na identificação da operação para diferenciá-las umas das outras), pid (consiste na identificação do processo que quer fazer esta operação), opcode (é o código da operação, se for 0, o processo quer criar um arquivo, se for 1, quer deletar), file_name (nome do arquivo que será criado ou deletado), file_size (indica o tamanho do arquivo em blocos dos arquivos que serão criados) e, por fim, status (quando é NONE a operação não existe, quando é WAITING a operação está esperando para ser executada, quando é EXECUTING indica que ela está sendo executada, quando FAILED indica que a operação falhou, e quando SUCCESS indica que ela foi bem-sucedida). Quando o processo é retirado da CPU e uma operação não terminou ainda, seu status continua sendo EXECUTING para que quando ela voltar para a CPU, ela cotinue de onde parou.

3.3. Função Main

A função main é responsável por fazer o parser dos arquivos passados como argumentos para o programa e inicializar as estruturas. Ela chama uma classe de métodos e estruturas chamada Parser para analisar cada arquivo de entrada e em seguida instancia as estruturas que representarão os componentes do hardware, como o disco e a memória.

Em seguida, o gerenciador de memória, o gerenciador de processos e o sistema de arquivos são instanciados, sendo essas suas únicas instâncias durante todo o programa. Ademais, ela chama o Dispatcher que de fato irá executar as operações de cada gerência afim de simular o sistema.

3.4. Classe Dispatcher e Integração das Gerências

O Dispatcher é uma classe que representa o processo principal do sistema. Inicialmente ele recupera do Parser os dados referentes dos processos e instancia as estruturas que serão utilizadas por eles quando estiverem sendo processadas na simulação, a saber, objetos da classe Process.

Em seguida, o Dispatcher adiciona os candidatos a processos numa fila de alocação da memória. Isso significa que eles apenas serão considerados processos quando conseguirem alocar um segmento de memória. Essa fila é ordenada pelo valor de tempo de chegada na estrutura de cada processo.

O Dispatcher olha então para os dados de operações analisados pelo Parser e instância objetos de Operation, que ficarão guardados até que um processo com os PIDs referenciados sejam criados (alocados na memória). Assim, o dispatcher incia um laço de execução de instruções que simulam o funcionamento do pseudo-so.

O Dispatcher vai tirando cada instância de processo do início da fila e passando para a função de alocação do gerenciador de memória. Caso o processo seja alocado, recebe um vetor de operações referentes ao seu PID e entra na fila global do gerenciador de processos. Caso contrário a instância fica esperando até que haja espaço suficiente

na memória para alocação, porém independente desta espera o simulador continua sua execução.

Um recém alocado processo é passado para a fila global do gerenciador de processos, que após um ciclo de quantum retorna o dados referentes ao estado de um processo que terminou ou está em execução. Esses dados são usados para finalização dos processos, para a finalização do sistema operacional simulado e, por fim, para a criação do log de saída.

Quando um processo é finalizado, o segmento de memória dedicado a ele é desalocado com uma chamada a função de desalocar do gerenciador de memória. Quando todos os processos forem finalizados e nenhuma instância estiver esperando na fila de alocação, o laço é interrompido e o log de saída é mostrado na tela, encerrando a simulação de execução do sistemna.

3.5. Gerenciador de Memória

O gerenciador de memória, representado pela classe MemoryManager, possui duas funções principais, que são chamadas pelo Dispatcher: alocar e desalocar. Além disso, conta com uma lista encadeada de segmentos e uma tabela de processos alocados em memória.

Um segmento consiste em uma estrutura com dados referentes a se o segmento está livre, o endereço de memória do início do segmento, o tamanho do segmento em blocos e um ponteiro para o endereço do próximo segmento. Já a tabela de processos, é um mapa, cujas chaves representam o PID de um processo o elemento da chave guarda o endereço de início em que o processo está alocado e o tamanho do processo em blocos alocados na memória.

Quando a função de alocar é chamada, ela recebe uma instância de processo e confere se há espaço livre contíguo na memória através da lista encadeada. Se houver, a instância recebe um PID, o vetor da memória é preenchida com esse PID, a lista encadeada é atualizada e a tabela de processos recebe uma nova entrada referente a instância recebida.

Durante a alocação, a prioridade do processo é conferida. Se for zero, o processo é em tempo real, e será alocado nos primeiros 64 blocos da memória, ou seja, o PID do processo serpa escrito nas primeiras 64 posições do vetor de memória. Se a prioridade for maior que zero, o processo será alocado a partir do sexagésimo quinto bloco. O vetor de memória tem um tamanho total de 1024 posições.

Quando a função de desalocar é chamqada, ela recebe apenas o PID do processo, confere o endereço de memória dele através da tabela de processos e desaloca o segmento de memória dedicado a ele na lista encadeada dos segmentos.

3.6. Gerenciador de Processos

O gerenciador de processos é a parte do pseudo-so que gerencia tanto as filas como a execução dos processos (apesar de os processos terem sua rotina de execução interna, é o gerenciador que chama ela).

Para estabelecer as prioridades, foram feitos 4 vetores de processos dentro de um vetor maior, onde o de índice zero é a fila de processos de tempo real que é tratado com

FIFO. Quando é chamada a função de *run* do gerenciador, ele rotaciona as filas, ou seja, faz com que o contador de wait e de running_time seja incrementado em 1 e, caso o processo em execução seja de prioridade 0 (tempo-real) ele não irá retirá-lo da execução antes que todas as suas operações tenham sido finalizadas (com sucesso ou falha).

A cada chamada da função de *run*, o gerenciador irá olhar cada processo na fila e irá checar o contador wait de cada um e irá verificar se ele é maior que o max_wait definido na instanciação do gerenciador. Caso ele seja maior, o gerenciador irá mudar esse processo para uma fila com prioridade igual a prioridade_do_processo-1; após o processo mudou de fila ser executado por um ciclo, ele terá seu tempo de wait zerado e irá voltar para sua fila de prioridade original no próximo quantum.

Além disso, o *run* retorna o processo que foi executado, desse modo permitindo que o dispatcher consiga verificar se a execução foi terminada para que seja possível desalocar a memória e criar novos processos.

3.7. Gerenciador de Arquivos

A classe do gerenciador de arquivos é chamada FileSystem. Possui três atributos atributos que são disco, tabela de arquivos e, por fim, o log.

O disco armazena, em cada posição, pares de string e inteiro, representando o nome do arquivo armazenado e qual processo o criou. A tabela de arquivos armazena os arquivos em disco em um map que possui como chave o nome do arquivo associado a um par de inteiros que representam a primeira posição do arquivo em disco e a quantidade de blocos armazenados para aquele arquivo. O log guarda flags de cada operação, demonstrando se foram executadas ou não, ele é uma tupla de quatro inteiros e um booleano: o primeiro número guarda o id do processo que fez a operação, o segundo indica o código da operação, o terceiro indica o momento em que o erro ocorreu se a operação falhou, caso tenha sucedido, este número será -1. O quarto inteiro guarda a posição do primeiro bloco alocado no disco, e por fim o booleano será false se a operação falhar e true, caso ela seja bem-sucedida.

O construtor da classe recebe como parâmetros o disco que é inicializado na main e o log. Com o disco é possível inicializar a tabela de arquivos ao percorrê-lo e documentando na tabela quais são os arquivos existentes.

O primeiro método da classe é o findOwnerOfFile que recebe o nome do arquivo como parâmetro e tem como função encontrar o processo que o criou utilizando somente o nome do arquivo. Para isso, ele busca na tabela de arquivos a primeira posição do arquivo no disco, com isso ele consegue ter acesso o segundo elemento do par que é o pid do processo.

O segundo método é o doOperation que tem como função fazer uma das duas operações possíveis no disco: criar ou deletar um arquivo. Ele retorna um booleano que indica se a operação falhou ou foi bem-sucedida e recebe como parâmetro um objeto Operation e a prioridade do processo que está pedindo pra fazer a operação.

Dentro do método doOperation há um switch que escolhe um fluxo baseado no código da operação desejada. Se o opcode é 0, a operação desejada é criar um arquivo no disco, se é 1, a operação é deletar um arquivo. Caso o código seja qualquer outro número, é colocado no log um código de erro e retornado false, pois essa operação não existirá.

No caso da operação ser criar um arquivo, o disco será percorrido para descobrir se há espaço suficiente para alocar o arquivo. Para isso, se a posição atual do disco for 0, incrementa-se o contador e guarda-se o index da possível primeira posição alocada. As posições seguintes são checadas até o contador ter o número de blocos desejados ou o disco acabar. Caso a próxima posição não estiver vazia, então o contador é redefinido para seu valor inicial. Se o disco possuir espaço suficiente para alocar o arquivo, a tabela de arquivos e o log são atualizados utilizando a primeira posição alocada. Se não foi possível criar o arquivo, somente o log é atualizado.

Caso a operação seja 1, ele deve checar se o arquivo realmente existe na tabela de arquivos. Se não existir, log é atualizado com essa informação. Se existe, é verificado se o processo que pediu para deletar o arquivo é o mesmo que criou ou se é um processo de tempo real, que tem permissão de deletar qualquer arquivo. Se nenhuma desses requisitos se confirmar, o processo não tem permissão de deletar o arquivo e essa informação é colocada no log. Se ele possuir permissão para fazer a operação, então ele encontra a primeira posição alocada utilizando a tabela de arquivos e substitui o nome do arquivo por 0 no disco em todos os blocos alocados, no fim o log e a tabela são atualizados.

O terceiro método somente imprime na tela o estado do disco, percorrendo suas posições e mostrando os arquivos guardados. Por fim, os métodos get de todos os atributos privados encontram-se no fim da classe.

3.8. Gerenciador de E/S

O gerenciador de Entrada e Saída basicamente contém variáveis de controle dos dispositivos que armazenam o PID dos processos que estão utilizando o dispositivo. Além disso, o gerenciador possui uma fila de pids para cada dispositivo, caso o dispositivo esteja ocupado, ele adiciona o pid que requisitou para a fila e, quando o dispositivo for liberado o gerenciador preeche o dispositivo com o primeiro pid da fila.

Quando um processo tenta ocupar um dispositivo, ele chama o metodo de ocupar passando o proprio pid e o id do dispositivo; caso seja possível ocupar, a função retorna true; caso contrário, false. Além disso, o dispositivo fica ocupado até que o processo libere o dispositivo; normalmente o processo irá manter o dispositivo ocupado enquanto tiver operação para executar e, uma vez que termine, chama a rotina de liberar dispositivo que escreve -1 ou coloca o primeiro da fila.

4. Conclusão

Com o fim da implementação do simulador do sistema operacional foi possível colocar em prática os conhecimentos adquiridos em sala de aula e entender mais profundamente como um software de base tão importante funciona.

O maior desafio encontrado se deu pela integração dos módulos do sistema, pois pode-se observar a necessidade de organizar e conectar as funções de cada de forma que o sistema funcione de acordo com o esperado e ofereça abstrações para os detalhes internos a cada gerência e ofereça uma interface que responda aos arquivos de entrada com mensagens de saída condizentes aos processos criados e às operações requisitadas.

Referências

1. TANENBAUM, A. S. , Sistemas Operacionais Modernos. Quarta Edição, Pearson, 2015.

Anexos

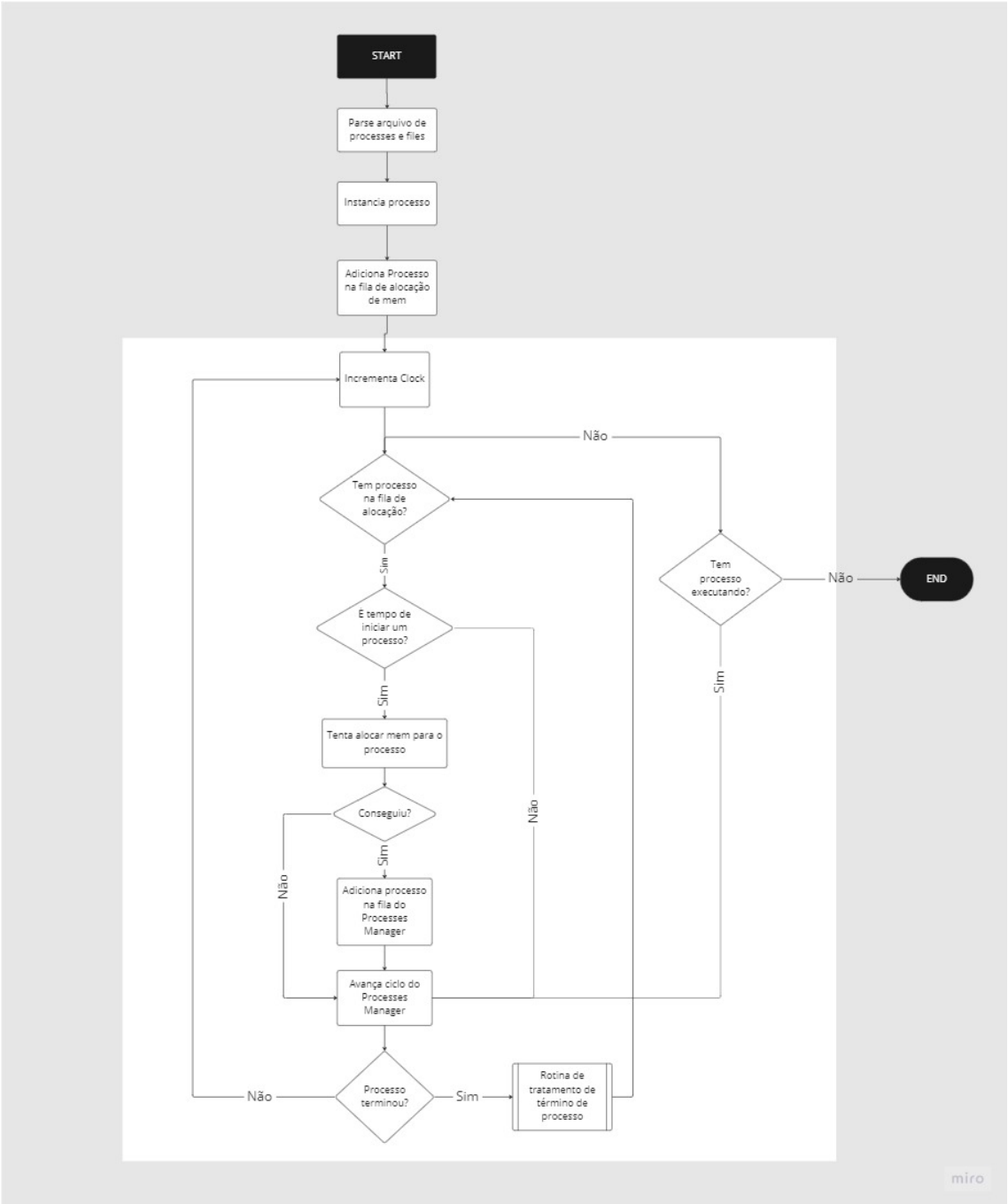

Figure 1. Fluxograma do dispatcher

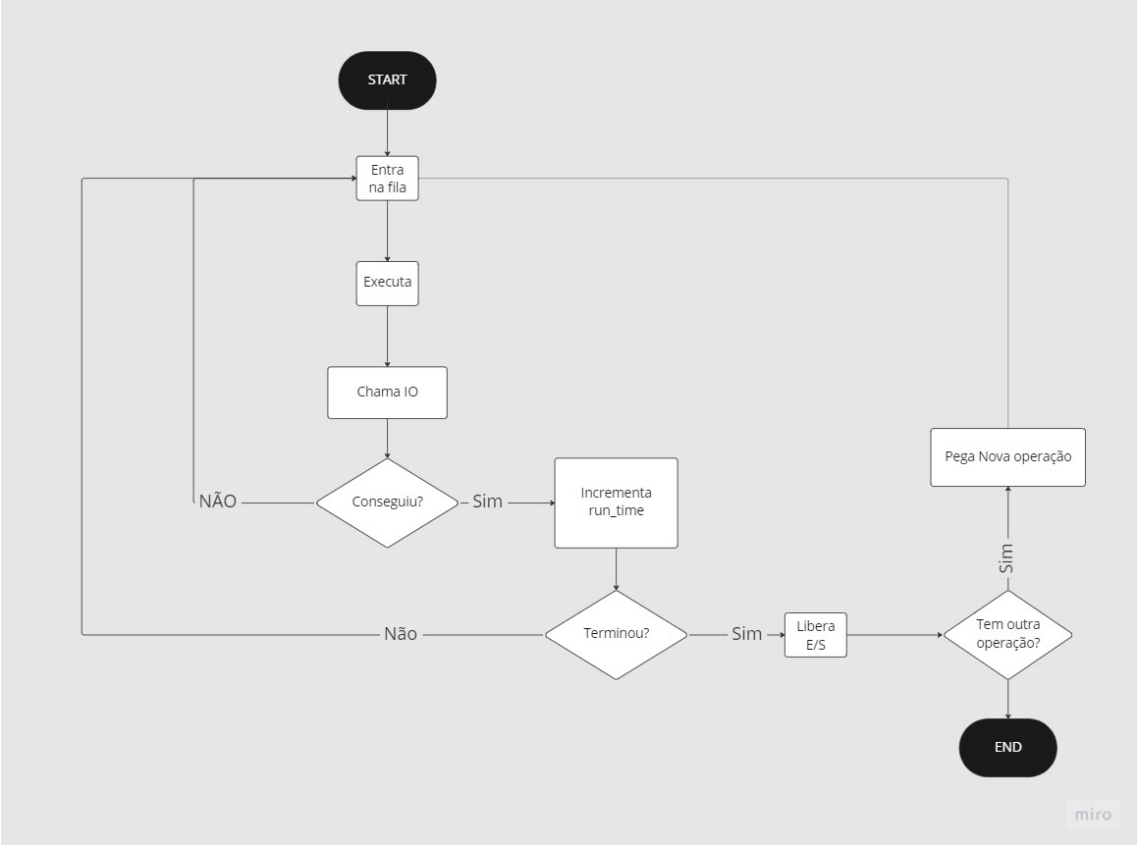

Figure 2. Fluxograma do ciclo de vida de um processo