Trabalho Prático de AEDS2

Servidor de Impressão

Douglas Rodrigues de Almeida
douglasralmeida@live.com

1. Introdução

O objetivo deste trabalho é implementar um simulador de impressão com o intuito determinar qual a melhor política e forma de implementação a ser utilizada. O sistema tem suporte a gerenciamento de contas e permite força impressões com o uso de prioridade. Possui três políticas de escalonamento de tarefas de impressões diferentes conforme descrito a seguir:

- Primeiro a chegar, primeiro a sair: As impressões são feitas na ordem cronológica.
- Usuários com prioridade: Os usuários com maios prioridade são beneficiados durante o processamento das impressões. Usuários com chave de prioridade menor possui maior prioridade que usuários com chave de prioridade menor.
- Combinação das prioridades: Usuários com prioridade 0 tem prioridade total sobre os demais. Para os outros usuários, é feito um calculo para determinar o nível de prioridade da tarefa:

prioridade = prioridade_usuario * 0.2 + prioridade_impressao* 0.6 + tempoespera / tempomaxespera * 0.2.

2. Implementação

Estrutura de Dados

Para executar foi criado um Tipo Abstrato de Dados Servidor com a seguinte implementação:

- ArquivoEntrada: Responsável por auxiliar no processamento do arquivo com as entradas.
- ArquivoSaida: Responsável por auxiliar no processamento da saída com o resultado das impressões.
- HoraAtual: Grava a hora atual durante o processamento das entradas e da fila de impressão.
- Impressora: Guarda a impressora cadastrada no sistema.
- Relatorio: Guarda os dados estatiticos para geração de relatórios de uso.
- Usuarios: Guarda os usuários cadastrados no sistema.

Funções e Procedimentos

O TAD Servidor possui as seguintes funções:

```
TServidor* Tservidor_Criar(void);
```
Aloca o TAD Servidor na memória.

```
void TServidor_Destruir(TServidor** PServidor);
```
Desaloca oTAD Servidor da memória e limpa a variável.

```
bool TServidor_Analisar(TServidor* Servidor);
```
Analisa as entradas disponíveis.

```
bool TServidor_CadastrarImpressora(TServidor* Servidor, char* Impressora,
const int Capacidade, const int Escalonador);
```
Cadastra uma impressora no sistema com as propriedades solicitadas.

```
bool TServidor_ChecarImpressao(TServidor* Servidor, TImpressao* Impressao);
```
Verifica se a impressão pode ser processada. Impressões com tempo limite vencido são consideradas impressões perdidas. Impressões de usuários não cadastrados no sistema são consideradas impressões rejeitadas.

```
void TServidor_FinalizaFila(TServidor* Servidor);
```
Processa todas as tarefas que estão na fila de impressão.

```
void TServidor_Finalizar(TServidor* Servidor);
```
Finaliza uma análise de entrada salvando o resultado num arquivo de saída.

```
bool TServidor_Preparar(TServidor* Servidor, const char* NomeArquivoEntrada,
const char* NomeArquivoSaida);
```
Prepara o servidor para analisar ao arquivo com as entradas de dados e salvar os resultados no arquivo de saída.

```
void TServidor_ProcessarImpressao(TServidor* Servidor);
```
Analisa a impressão a recebida imprimindo-a ou enfileirado-a.

```
void TServidor_RecebeImpressao(TServidor* Servidor; Timpressao* Impressao);
```
Recebe uma nova impressao do arquivo de entrada.

```
void TServidor_Relatorio(TServidor* Servidor);
```
Gera um relatório estatístico das impressões.

```
void TServidor_UsuarioExcluir(TServidor* Servidor, const char* Nome);
```
Exclui um usuário do sistema.

```
void TServidor_UsuarioNovo(TServidor* Servidor, const char* nome, const int
Prioridade);
```
Adiciona um novo usuário no sistema.

O TAD Servidor ainda possui alguns TADs auxiliares para o processamento das impressões. São eles o TAD Impressão que armazena os dados das tarefas de impressão e compara o nível de prioridade conforme a política de impressão escolhida, o TAD Impressora que armazena a impressora cadastrada no sistema e processa uma impressão, o TAD Relatório que armazena as estatísticas de uso do sistema e o TAD Usuário que armazena os usuários cadastrados no sistema.

TAD Fila com Prioridade

O processamento das prioridades das tarefas é realizado pelo TAD Fila com Prioridade. Tal estrutura é responsável por controlar a fila de impressão mantendo as tarefas ordenadas de acordo com sua prioridade de impressão.
As operações utilizadas sobre o TAD Fila com Prioridade são as seguintes:

- Criar: Aloca a fila de prioridade na memória.
- Destruir: Desaloca a fila de prioridade da memória.
- Desenfileirar: Retira o item da fila com maior prioridade.
- Enfileirar: Insere um novo item na fila.
- Limpar: Limpa a fila.

- Tamanho: Retorna a quantidade de itens da fila.

A fila prioridade é implementada com uma estrutura de dados chamada heap. Um heap é definido como uma sequência de itens cujas chaves respeitam a seguinte regra:

$$c[i] >= c[2i]$$
$$c[i] >= c[2i+1]$$

Essa ordem pode ser visualizada como uma árvore binária completa, onde cada nó possui uma chave com maior ou igual prioridade que as chaves dos filhos deste nó, se eles existirem. Assim, a chave do nó raiz é a maior chave da árvore.

A árvore binária completa é representada por um vetor de itens respeitando as regras descritas acima. Esta representação é extremamente compacta e permite caminhar pelos nós da árvore facilmente. Os filhos do nó i estão nas posições 2*i e 2*1+1, caso existam. O pai de um nó i está na posição i / 2. O item com a maior prioridade sempre está na posição 1 do vetor.

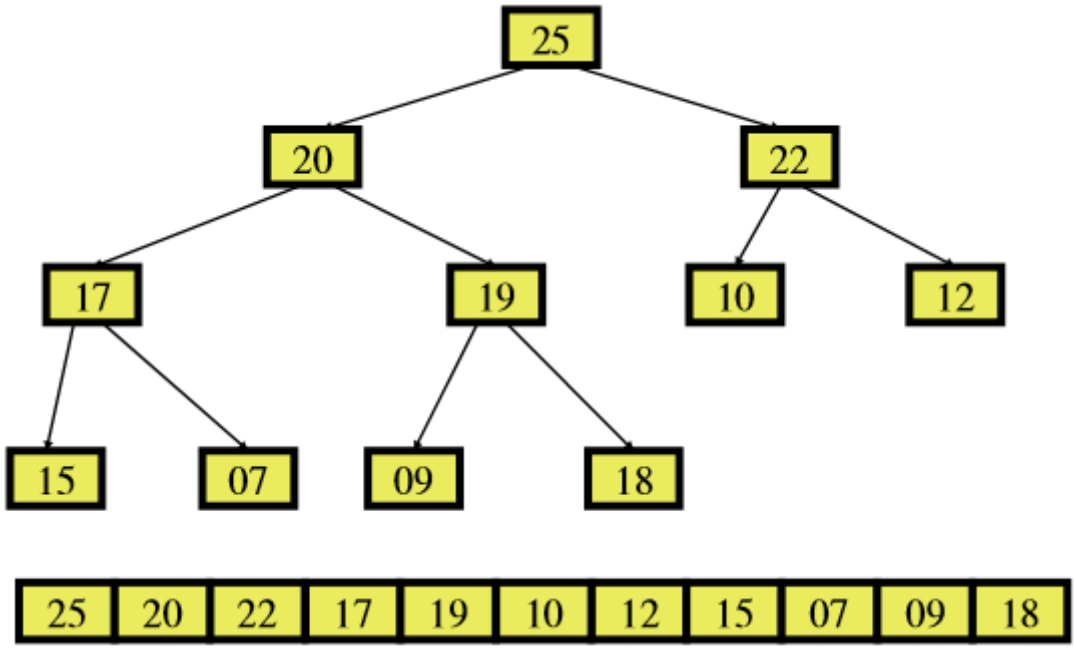

Figura 1: Representação do heap como árvore e seu vetor correspondente

TAD Lista

O TAD Lista implementa uma lista de dados com encadeamento entre si através de ponteiro. Ele é usado para armazenar os usuários do sistema. Sua implementação é bastante simples. Cada item possui um ponteiro apontando para o próximo item da lista. O último item da lista não aponta para nada. A vantagem do uso de ponteiro aparece na inserção e remoção de um item, que é O(1). Entretanto, acessar um item na posição i da lista pode ter O(n), no pior caso.

Um item da lista é chamado de nó. Ele possui dois campos. O primeiro campo representa o dado armazenado. O segundo campo representa o ponteiro para o próximo nó da lista.

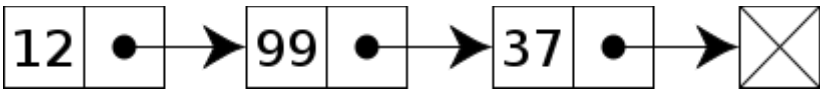

Figura 2: Demonstração de uma lista encandeada.

Programa Principal

O programa principal recebe dois argumentos do aviso de comando com nomes de dois arquivos. O primeiro com a entrada de dados para simulação do servidor. Já o segundo, com o arquivo onde serão gravados os relatórios estatísticos do uso do sistema. Em seguida, é criado o TAD Servidor para processar os dados da entrada, que é processado linha a linha.

A primeira linha do arquivo de entrada traz os dados para a impressora que será cadastrada no sistema. Neste momento, é informado qual política de impressão será utilizada. As demais linhas são comandos para cadastrar e remover usuários, enviar tarefas de impressão para o servidor, exibir relatórios e finalizar o aplicativo.

Cada linha é lida e enviada para um buffer na memória onde é processada. O tamanho máximo suportado pelo buffer é 128 caracteres.

Quando uma tarefa de impressão é recebida pelo sistema, ela é enviada para impressora para ser processada. Entretanto, se a impressora estiver ocupada com outra tarefa de impressão, a impressão recebida entra na fila de espera. O algoritmo que faz esse processamento é descrito na imagem abaixo.

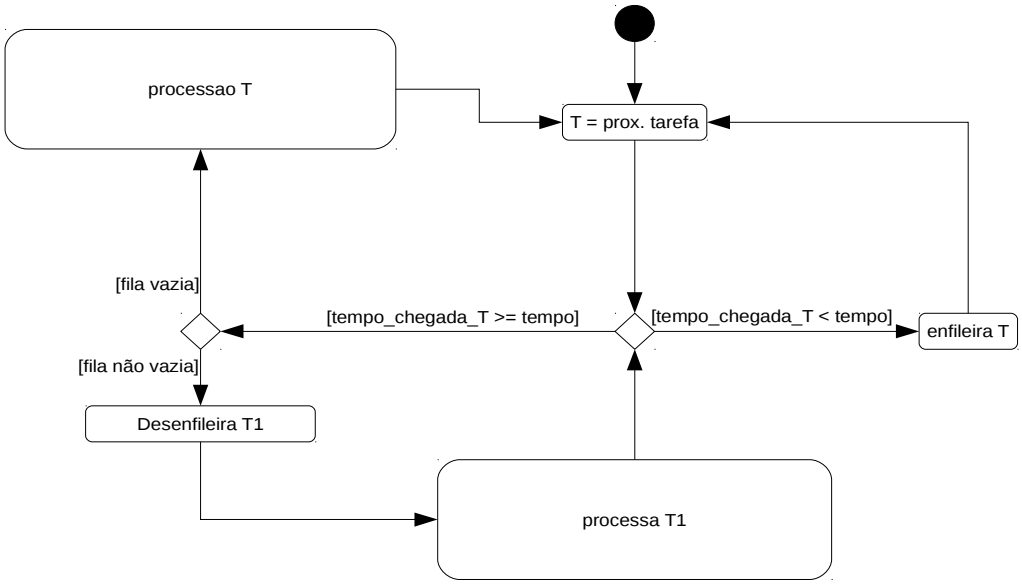

Figura 3: Fluxograma do processamento das impressões

Ao iniciar o processamento, uma variável grava a hora atual do sistema. Quando chega uma nova tarefa, caso seu tempo de chegada seja inferior ao tempo atual, ela é enfileirada. Caso contrário, o sistema checa se a fila de impressão está vazia. Caso positivo, a tarefa é processada, o tempo atual é alterado para o tempo da chegada da tarefa e o sistema aguardará a próxima tarefa de impressão. Caso a fila não esteja vazia, a próxima tarefa da fila é processada e, ao tempo atual, é somado o

tempo gasto na impressão. Esse passo repete-se até que o tempo atual seja maior que o tempo de chegada da última tarefa recebida ou até que a fila de impressão fique vazia.
Após receber todas as tarefas, o sistema processa todas as tarefas da fila, caso existam, e encerra o processamento.

Organização do código

O código está dividido em 17 arquivos principais. Boolutils.h implementa o tipo booleano para facilitar a programação e core.h dá suporte a métodos de callback[1] para os TADs Lista e Fila com Prioridade. O arquivo principal.c implementa o programa principal.

A fila com prioridade é implementada nos arquivos filaprior.h e filaprior.c. A lista é implementada nos arquivos list.h e list.c. O servidor é implementado nos arquivos servidor.h e servidor.c. O supote a análise dos arquivos de entrada é implementado nos arquivos servidor_arquivo.h e servidor_arquivo.c. A impressão e impressora são implementados nos arquivos servidor_impressao.h e servidor_impressao.c. O suporte a relatórios é implementado nos arquivos servidor_relatorio.h e servidor_relatorio.c. Por fim, o suporte a usuários é implementado nos arquivos servidor_usuario.h e servidor_usuario.c.

O compilador utilizado foi o GCC 4.8 no sistema operacional CentOS 7.1.

Para executar o programa, digite no arquivo de comando:

./main <arquivoentrada> <arquivosaida>

3. Análise de Complexidade

A lista de usuários inclui usuário em O(n) e os remove em O(1). Isso acontece porque antes da inserção é realizada uma pesquisa na lista para evitar usuários duplicados. A pesquisa de usuários é O(n).
As funções de enfileirar e desenfileirar na fila com prioridade são O(log n). Após a inserção de um novo item ou após a remoção de um item da lista, é necessário reordená-la. Essa reordenação tem um custo de O(log n), já que o heap funciona como uma árvore binária. Assim, varrer o vetor tem custo idêntico ao de varrer uma árvore no sentido raiz até o nó mais externo ou vice-versa.
O algoritmo de analisar as tarefas de impressão terá O(1) no melhor caso, quando a fila está vazia e a tarefa processada é a tarefa recebida. No pior caso e no caso médio, que acontece quando a fila não está vazia ou o tempo de chegada da tarefa é superior ao tempo atual, as funções de enfileirar e/ou desenfileirar são executadas o que contribuem com um custo de O(log n) para algoritmo de análise.
A geração de relatórios tem custo O(1) pois os dados estatísticos são armazenados na estrutura de dados a cada processamento de tarefa. Gerar um relatório é simplesmente lê o valor das variáveis armazenadas.

4. Conclusão

As filas com prioridade possuem um grande número de aplicações. Sua implementação é simples e sua forma de se manter ordenada, geralmente, é eficiente. Por isso, a sua escolha durante a criação do servidor de impressão.

---

1 Método de callback é um pedaço de código executável que é passado como parâmetro para algum método, onde será executado em algum momento.

A implementação do TAD Fila com Prioridades foi a parte mais difícil do trabalho e que demandou bastante tempo. O algoritmo que analisa as tarefas da entrada, enfileira e desenfileira também demandou tempo e análise, inclusive com a necessidade do uso de fluxogramas para auxiliar sua construção.
Nos testes realizados, a política 3 se mostrou mais eficiente que as demais políticas, chegado a processar até 10% a mais de tarefas de impressões do que as política 2, por exemplo.

Referências

Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição, Editora Cengage Learning, 2009.

Loudon, K., Mastering Algorithms with C, Editora O'Reilly, 1999.

Priority queue. (2016, Abr 13). In Wikipedia, The Free Encyclopedia. Acessado em 01/06/2016.

Anexos

Listagem dos arquivos:
principal.c
servidor.c
servidor_arquivo.c
servidor_impressora.c
servidor_relatorio.c
servidor_usuario.c
filaprior.c
fila.c
lista.c
servidor.h
servidor_arquivo.h
servidor_impressora.h
servidor_relatorio.h
servidor_usuario.h
filaprior.h
fila.h
lista.h
booltutils.h
core.h