Trabalho prático 1

Fila, pilhas e complexidade

Hygor Hernane Telles e Silva
Matricula: 2014122665
hygorhernane@gmail.com

1. Introdução

Nesse trabalho foi pedido a implementação de estruturas de dados muito utilizadas na computação, essas estruturas funcionam de maneira dinâmica e tem o objetivo basicamente de guardar informação na memória e a ordem de inserção e remoção dos elementos importa, ou seja, no caso da primeira estrutura **Fila**, temos que o elemento que foi inserido primeiro, será o primeiro a ser retirado, já na segunda estrutura **pilha**, temos que o último elemento que foi inserido, será o primeiro a sair. E ambas as estruturas não permitem a retirada nem inserção de elementos no meio. Com isso a ordem de chegada importa para que o elemento seja retirado.

Essas estruturas para a aplicação do trabalho devem ser dinâmicas, ou seja, não tem tamanho máximo definido inicialmente. Com isso, a cada elemento novo que será guardado na memória, esse espaço deverá ser reservado no próprio programa, para que assim, o consumo de memória seja dinâmico, de acordo com a inserção de elementos e retirada.

Essas estruturas são muito úteis na maioria das aplicações implementadas e também para muitas camadas dos sistemas operacionais desenvolvidos, já que muitas vezes trabalhamos com endereçamento da memória e quando existem chamadas recursivas, por exemplo, a estrutura pilha é muito indicada, já que a última referência de endereço na memória, será a próxima em que o processo deverá retornas, dessa maneira facilita o funcionamento das chamadas do sistema. Entre várias outras aplicações, até mesmo para armazenamento simples de informação, em que a ordem tem algum valor semântico.

2. Desenvolvimento

Para o desenvolvimento desse trabalho, foi utilizado a linguagem **C** e os testes foram realizados no sistema operacional Ubuntu 14.04LTS. As estruturas geradas foram:

```
typedef struct Pessoa{
  int id;
  int entrouCantina;
  int saiuCantina;
}Pessoa;

typedef struct Node{
  Pessoa *p;
  struct Node *next, *prev;
}Node;

typedef struct Generica{
  Node *primeiro, *ultimo;
  int tam;
}Generica;
```

Figura 1. Estruturas de dados

A primeira estrutura **Pessoa**, foi criada para funcionar como cada usuário que utilizará a cantina durante o almoço, portanto terá um *id* para sua identificação, além das variáveis *entrouCantina e saiuCantina*, que guardará o tempo de entrada e saída do usuário na cantina, respectivamente, para o calculo da média de tempo de atendimento calculado posteriormente.

A segunda estrutura **Node**, funciona como um encapsulamento da estrutura **Pessoa**, ou seja, cada Node servirá basicamente para guardar uma **Pessoa**, porém estaremos trabalhando com uma "Lista encadeada", uma estrutura de dados em que cada **Node** será inserido de maneira a guardar o endereço em que o próximo Node e o anterior estão, caso seja duplamente encadeada. Na nossa implementação, esses ponteiros levam o nome de *next e prev*.

A terceira estrutura **Generica**, faz o papel apenas de guardar o endereço do primeiro **Node** inserido e o último. Além de guardar a quantidade de **Nodes** inseridos na variável *tam*. Com isso, basta guardar o endereço dessa estrutura, que conseguimos acessar todos os **Nodes**, consequentemente todas as **Pessoas** inseridas na "Lista".

Dessa maneira teremos as funções abaixo para a implementação do funcionamento dessas estruturas:

```
Pessoa* criaPessoa(int id);
Generica* CriaGenericaVazia();
void insereElemento(Generica *l, Pessoa *p);
Pessoa* desempilha(Generica *l);
Pessoa* desenfileira(Generica *l);
void imprimeGenerica(Generica *l);
void liberaGenerica(Generica *l);
```

As duas primeiras funções **criaPessoa e criaGenerica** foram feitas para a alocação na memória para essas estruturas, e alterar o parâmetro *id* da pessoa criada, a estrutura **Generica** é criada com *tam = 0* e inicialmente terá um **Node**, ou seja, quando a **Generica** estiver vázia, ela já terá um **Node**, que é chamado de Nó sentinela. Para evitar que os ponteiros fiquem apontando para **NULL**.

As funções foram desenvolvidas de maneira que a mesma estrutura pode funcionar como **Fila ou Pilha**, isso funciona de acordo com as funções de inserção e remoção de elemento. Dessa maneira a função **insereElemento**, sempre criará um **Node** e ele guardará a pessoa que está sendo inserida. Depois disso, esse **Node** é inserido na "Lista". Para isso, sempre inserimos o **Node** na última posição e como a lista implementada é circular, teremos que o **Node** estará antes do primeiro **Node**, no caso o Nó sentinela, e depois do último elemento que foi inserido. Com isso temos a seguinte função:

```
void insereElemento(Generica *l, Pessoa *p){
  Node *novo = (Node *) malloc(sizeof(Node));
  novo->p = p;
  novo->next = l->primeiro;
  novo->prev = l->ultimo;
  l->ultimo->next = novo;
  l->ultimo = novo;
  l->primeiro->prev = l->ultimo;
  l->tam++;

}
```

Dessa maneira, quando inicializamos uma nova **Generica** e inserimos os elementos, todos estarão seguindo essa ordem de inserção. Com isso, se quisermos utilizar essa estrutura como **Fila**, usaremos a função **desenfileira**, que vai retirar o **Node** que entrou a mais tempo na estrutura, ou seja, aquele que está logo depois do Nó sentinela. Dessa maneira:

```
Pessoa* desenfileira(Generica *l){
  Node *ptr = l->primeiro->next;
  Pessoa *aux = ptr->p;
  ptr->next->prev = ptr->prev;
  ptr->prev->next = ptr->next;
  if(ptr == l->ultimo)
    l->ultimo = ptr->prev;

  l->tam--;
  free(ptr);

  return aux;

}
```

E quando queremos utilizar a estrutura como **Pilha**, iremos sempre retirar o último elemento que entrou na estrutura, ou seja, aquele que está sendo apontado como último e está antes do sentinela, dessa maneira:

```
Pessoa* desempilha(Generica *l){
  Node *ptr = l->ultimo;
  Pessoa *aux = ptr->p;
  ptr->next->prev = ptr->prev;
  ptr->prev->next = ptr->next;
  l->ultimo = ptr->prev;
  l->tam--;
  free(ptr);

  return aux;

}
```

Por fim, temos as funções que irá imprimir todos os elementos dessa estrutura Generica, ou seja, imprimindo o id de cada pessoa que foi inserida. E a função que irá liberar toda a memória utilizada por aquela estrutura. Dessa maneira é possível utilizar as estruturas e ter o melhor aproveitamento da memória, sem causar nenhuma perda de espaço.

3. Análise

A complexidade da estrutura implementada deve ser analisada de acordo com as suas funções implementadas, dessa forma as funções de criação, inserção de elemento, desenfileira e desempilha terão complexidade **O(1)**, ou seja, levará sempre um tempo constante para a sua execução. Já a função de liberação de memória, tem complexidade **O(n)**, em que n é o número de elementos que estão inseridos na estrutura. Além disso existem as funções abaixo para o funcionamento da aplicação do trabalho:

```
void setEntrou(Pessoa *p, int tempo);
void setSaiu(Pessoa *p, int tempo);
int tamGenerica(Generica *g);
void tiraBandeja(Generica *pilha);
void colocaBandeja(Generica *pilha, int qtd);
int temBandeja(Generica *pilha);
int temGente(Generica *fila);
```

Essas funções são respectivamente, para inserir o horario que a pessoa entrou na cantina, inserir o horário que a pessoa terminou de ser atendida, verificar quantas pessoas existem na lista, retirar uma bandeja da pilha de bandejas, colocar mais bandejas na pilha de bandejas, verificar se existe bandejas na pilha de bandejas e verificar se existem pessoas em alguma fila. Com isso, todas essas funções tem complexidade **O(1)**, já que suas implementações também trabalham com tempo constante.

Por último temos a função que calcula o tempo médio que foi gasto para os usuários serem atendidos em um período de funcionamento da cantina. Essa função foi implementada da seguinte forma:

```c
int calculaTempoMedio(Generica *l){
  Node *ptr = l->primeiro->next;
  int soma = 0;
  while(ptr != l->primeiro){
    soma += (ptr->p->saiuCantina - ptr->p->entrouCantina);

    ptr = ptr->next;
  }

  return (soma/l->tam);
}
```

A Fila será percorrida e para cada usuário que estiver nela, será somado o tempo em que este ficou na cantina, ou seja, utilizando as variáveis *entrouCantina* e *saiuCantina*, teremos esse tempo de atendimento, com a soma total do tempo gasto para todos os usuários que foram atendidos, dividido pela quantidade de usuários atendidos, teremos a média do tempo gasto. Com isso conseguiremos responder a questão principal do trabalho prático proposto. E essa função terá a complexidade **O(n)**, em que n é o número total dos usuários na fila, já que o método terá que percorrer toda a fila, lembrando que filas e pilhas não podem ser percorridas e seus elementos alterados, a menos em casos como esse, em que a aplicação precisa realizar alguma operação com os elementos inseridos.

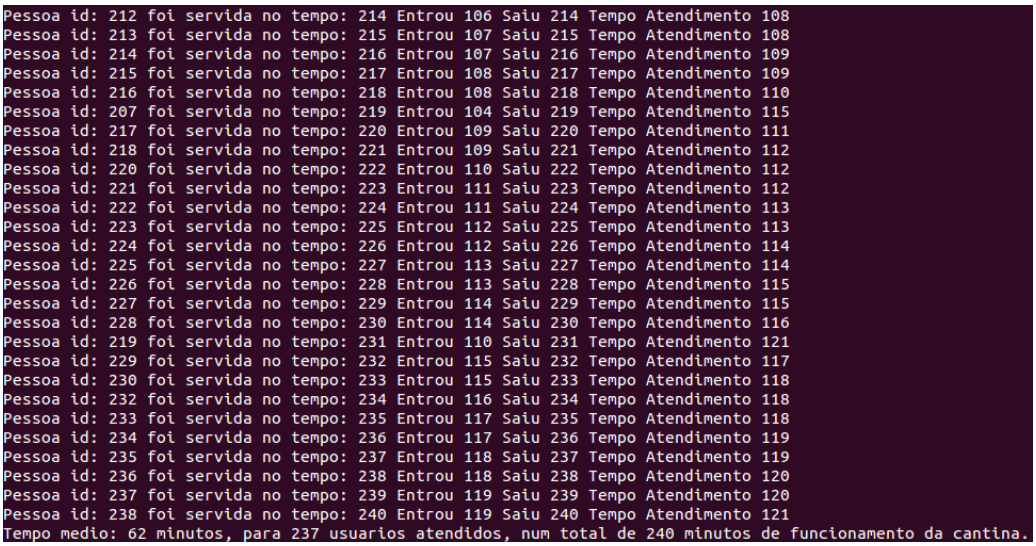
4. Resultados

O tempo de funcionamento do horário de almoço utilizado foram 4 horas, ou seja, 240 minutos. Com isso para a configuração inicial, que a cada minuto teremos duas pessoas chegando na fila. O resultado obtido, como podemos ver na figura abaixo:

```
Pessoa id: 185 foi servida no tempo: 201 Entrou 93 Saiu 201 Tempo Atendimento 108
Pessoa id: 186 foi servida no tempo: 202 Entrou 93 Saiu 202 Tempo Atendimento 109
Pessoa id: 187 foi servida no tempo: 203 Entrou 94 Saiu 203 Tempo Atendimento 109
Pessoa id: 188 foi servida no tempo: 206 Entrou 94 Saiu 206 Tempo Atendimento 112
Pessoa id: 189 foi servida no tempo: 207 Entrou 95 Saiu 207 Tempo Atendimento 112
Pessoa id: 190 foi servida no tempo: 208 Entrou 95 Saiu 208 Tempo Atendimento 113
Pessoa id: 191 foi servida no tempo: 209 Entrou 96 Saiu 209 Tempo Atendimento 113
Pessoa id: 192 foi servida no tempo: 210 Entrou 96 Saiu 210 Tempo Atendimento 114
Pessoa id: 193 foi servida no tempo: 211 Entrou 97 Saiu 211 Tempo Atendimento 114
Pessoa id: 194 foi servida no tempo: 212 Entrou 97 Saiu 212 Tempo Atendimento 115
Pessoa id: 195 foi servida no tempo: 213 Entrou 98 Saiu 213 Tempo Atendimento 115
Pessoa id: 196 foi servida no tempo: 214 Entrou 98 Saiu 214 Tempo Atendimento 116
Pessoa id: 197 foi servida no tempo: 215 Entrou 99 Saiu 215 Tempo Atendimento 116
Pessoa id: 198 foi servida no tempo: 218 Entrou 99 Saiu 218 Tempo Atendimento 119
Pessoa id: 199 foi servida no tempo: 219 Entrou 100 Saiu 219 Tempo Atendimento 119
Pessoa id: 200 foi servida no tempo: 220 Entrou 100 Saiu 220 Tempo Atendimento 120
Pessoa id: 201 foi servida no tempo: 221 Entrou 101 Saiu 221 Tempo Atendimento 120
Pessoa id: 202 foi servida no tempo: 222 Entrou 101 Saiu 222 Tempo Atendimento 121
Pessoa id: 203 foi servida no tempo: 223 Entrou 102 Saiu 223 Tempo Atendimento 121
Pessoa id: 204 foi servida no tempo: 224 Entrou 102 Saiu 224 Tempo Atendimento 122
Pessoa id: 205 foi servida no tempo: 225 Entrou 103 Saiu 225 Tempo Atendimento 122
Pessoa id: 206 foi servida no tempo: 226 Entrou 103 Saiu 226 Tempo Atendimento 123
Pessoa id: 207 foi servida no tempo: 227 Entrou 104 Saiu 227 Tempo Atendimento 123
Pessoa id: 208 foi servida no tempo: 230 Entrou 104 Saiu 230 Tempo Atendimento 126
Pessoa id: 209 foi servida no tempo: 231 Entrou 105 Saiu 231 Tempo Atendimento 126
Pessoa id: 210 foi servida no tempo: 232 Entrou 105 Saiu 232 Tempo Atendimento 127
Pessoa id: 211 foi servida no tempo: 233 Entrou 106 Saiu 233 Tempo Atendimento 127
Pessoa id: 212 foi servida no tempo: 234 Entrou 106 Saiu 234 Tempo Atendimento 128
Pessoa id: 213 foi servida no tempo: 235 Entrou 107 Saiu 235 Tempo Atendimento 128
Pessoa id: 214 foi servida no tempo: 236 Entrou 107 Saiu 236 Tempo Atendimento 129
Pessoa id: 215 foi servida no tempo: 237 Entrou 108 Saiu 237 Tempo Atendimento 129
Pessoa id: 216 foi servida no tempo: 238 Entrou 108 Saiu 238 Tempo Atendimento 130
Pessoa id: 217 foi servida no tempo: 239 Entrou 109 Saiu 239 Tempo Atendimento 130
Tempo medio: 62 minutos, para 217 usuarios atendidos, num total de 240 minutos de funcionamento da cantina.
```

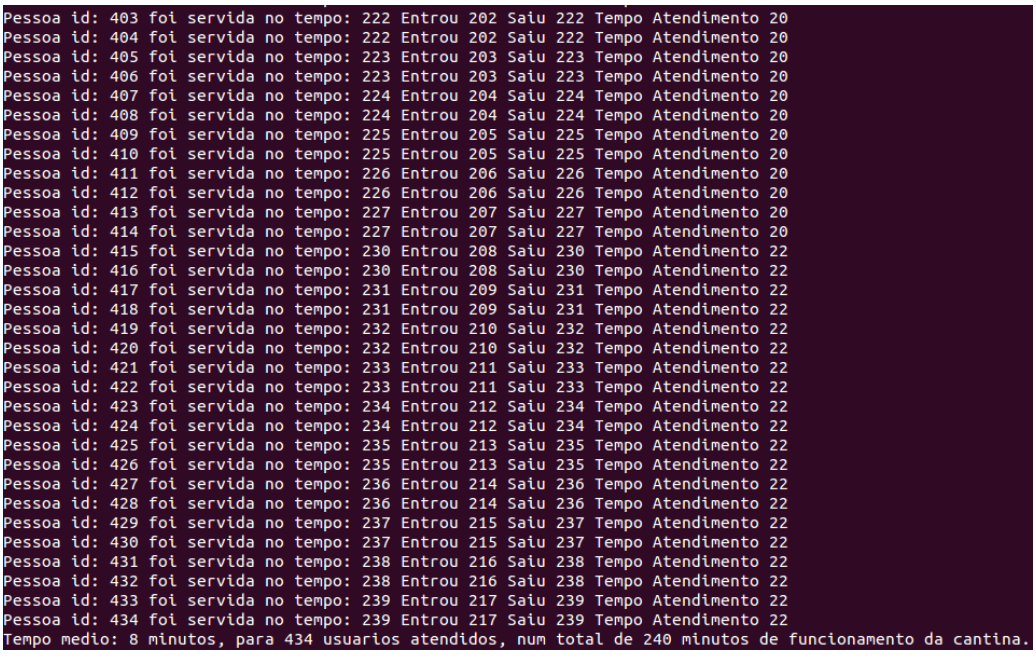
Ou seja, em média cada pessoa teve que esperar 62 minutos para ser atendida e tivemos 217 usuários atendidos.

No caso de termos duas filas para comprar ficha e duas pilhas de bandejas, temos o seguinte resultado:



Ou seja, o tempo médio de atendimento não foi muito alterado. Porém foi possível atender 237 usuários, ou seja, tivemos um aumento na quantidade de usuários, mas podemos perceber que o provável gargalo no tempo é a fila de colocar alimentos.

Dessa forma foi feito o teste acima, porém com duas filas para colocar alimento, ou seja, teria que ser colocado duas mesas com as comidas para que duas filas parar servir fosse passado. Abaixo temos os resultados:



Com essa configuração o tempo médio de atendimento cairia para 8 minutos e teríamos 434 usuários sendo atendidos. Com isso é possível perceber que o maior gargalo do atendimento da cantina é a fila de alimentos.

5. Conclusão

Nesse trabalho foi possível inferir como implementar estruturas de dados e utilizá-las, aumentou todo o conhecimento na linguagem C e como trabalhar com endereços e ponteiros, além de alocação dinâmica da memória.

Foi possível solidificar os conhecimentos em Fila e Pilha e como utilizar essas estruturas de forma combinadas para gerar situações do dia a dia, para solucioná-las.

Além disso foi de grande aproveito a parte de documentação, para que fosse possível a análise de tudo que foi desenvolvido, além de verificar vários cenários de um ambiente realístico vivenciado todos os dias em nosso próprio prédio.