UNIVERSIDADE FEDERAL DO CEARÁ - UFC
CAMPUS CRATEÚS
CIÊNCIA DA COMPUTAÇÃO

Disciplina: Estrutura de Dados
Prof. Roberto Cabral
Dupla: José Tarcísio De Sousa Araújo Neto **Matrícula:** 402542

RELATÓRIO TÉCNICO

1.INTRODUÇÃO(Descrição):

O objetivo do trabalho é implementar um sistema que simule o funcionamento de dois escalonadores FCFS, e o JSF, utilizando estruturas de dados Pilha, Fila e Lista, logo cada processo seleciona aleatoriamente um computador, cpu e o disco, com filas em cada recurso, assim a rede é o unico recurso compartilhado entre todos os processos, com uma fila que funciona de acordo com o escalonador desejado.

Logo, ao final da simulação, serão mostrados as informações de cada processo e o computador selecionado, como também as informações referentes ao tempo médio de execução, espera e as taxas de processamento.

2. RESOLUÇÃO DO PROBLEMA:

Primeiramente foi implementada uma lista duplamente encadeada, pois em alguns casos foi necessário percorrer a lista, e retirar elementos de acordo com sua demanda de cpu, disco e rede. Logo as filas foram implementadas com a lista, também foi criada uma lista normal de elementos que representam os computadores, assim cada nó da lista contém a fila da cpu, e a do disco.

Foi implementado um TAD chamado neolook, que possui funções que são responsáveis por gerenciar e realizar a simulação de acordo com o escalonador escolhido pelo usuário.

3. Complecidade das funções:

Disco.c:

Funções	Complexidade
Disco* disco_cria(int i);	O(1)
Void disco_libera(Disco *d);	O(1)
char getDadoDisco(Disco *d);	O(1)
int getIdDisco(Disco *d);	O(1)

Processo.c:

Funções	Complexidade
Processo* cria_processo();	O(1)
void inserir_demandas(Processo *p, int *v, int id, int idComp);	O(1)
int getIdProcesso(Processo *p);	O(1)
int getTempoInicial(Processo *p);	O(1)
int getCpu(Processo *p);	O(1)
int getDisco(Processo *p);	O(1)
int getRede(Processo *p);	O(1)
int getEspera(Processo *p);	O(1)
int getIdDiscoUsado(Processo *p);	O(1)
int getIdComputador(Processo *p);	O(1)
char getDados(Processo *p);	O(1)
void setIdDisco(Processo *p, int id);	O(1)
void setDado(Processo *p, char d);	O(1)
void setEspera(Processo *p, int espera);	O(1)
void setTempoInicial(Processo *p, int i);	O(1)
void destroi_processo(Processo *p);	O(1)
void imprimir_processo(Processo *p);	O(1)
void imprimir_processo_espera(Processo *p);	O(1)

Lista.c

Funções	Complexidade
int lista_vazia(Lista *l);	O(1)
Lista* lista_insere_ini(Lista *l, Processo *processo);	O(1)
Lista* lista_insere_fim(Lista *l, Processo *processo);	O(1)
Lista* lista_retira_ini(Lista *l);	O(1)
Lista* lista_retira_fim(Lista *l);	O(n)
int getInicioLista(Lista *l);	O(1)
int verificar_maior(Lista *l, Processo *p);	O(1)
Lista* lista_inserir_ordenado(Lista *l, Processo *p);	O(n)

Processo* getProcesso(Lista *l);	O(1)
Lista* lista_cria();	O(1)
void lista_libera(Lista *l);	O(n)
void lista_imprimir(Lista *l);	O(n)
void lista_imprimir_espera(Lista *l);	O(n)

Fila.c

Funções	Complexidade
Fila* fila_cria();	O(1)
int fila_vazia(Fila *f);	O(1)
void fila_inseri_fim(Fila *f, Processo *p);	O(1)
void fila_inseri_ordenado(Fila *f, Processo *p);	O(n)
Processo* fila_retira_ini(Fila *f);	O(1)
void fila_libera(Fila *f);	O(n)
void fila_imprimir(Fila *f);	O(n)
void fila_imprimir_espera(Fila *f);	O(n)

Rede.c

Funções	Complexidade
Rede* cria_rede();	O(1)
int rede_vazia(Rede *r);	O(1)
void inserir_fila_rede(Rede *r, Processo *p);	O(1)
void inserir_fila_rede_ordenado(Rede *r, Processo *p);	O(n)
Processo* retira_fila_rede(Rede *r);	O(1)
Processo* retira_menor_demanda_rede(Rede*c);	O(3n-1)
int rede_cheia(Rede *r);	O(1)
void rede_push(Rede *r, char c);	O(1)
char rede_pop(Rede *r);	O(1)
void libera_rede(Rede *r);	O(n)
void imprimir_rede(Rede *r);	O(3n)

Computador.c

Funções	Complexidade
Comp* comp_cria(int i);	O(1)
int comp_fila_vazia(Comp *c);	O(1)
int comp_fila_disco_vazia(Comp *c, int id);	O(1)

void comp_insere_fila_rede(Comp *c, Processo *p);	O(1)
void comp_insere_fila_rede_ordenado(Comp *c, Processo *p);	O(n)
void comp_insere_filaD1(Comp *c, Processo *p);	O(1)
void comp_insere_filaD2(Comp *c, Processo *p)	O(1)
void comp_fila_insere(Comp *c, Processo *p)	O(1)
Processo* retira_processo_filaD1(Comp *c);	O(1)
Processo* retira_processo_filaD2(Comp *c);	O(1)
Processo* retira_processo_fila_rede(Comp *c);	O(1)
Processo* retira_processo_fila(Comp *c);	O(1)
Processo* retira_menor_demanda_cpu(Comp *c){	O(3n–1)
Processo* retira_menor_demanda_disco(Comp *c, int idDisco)	O(3n-1)
void comp_imprimi_fila(Comp *c);	O(n)
void comp_imprimi_fila_discos(Comp *c);	O(n)
void setRede(Comp *c, Rede *rede);	O(1)
void libera_comp(Comp *c, int i);	O(n)
int getIdComp(Comp *c);	O(1)
char getDadoCompD1(Comp *c);	O(1)
char getDadoCompD2(Comp *c);	O(1)
int getIdDiscoD1(Comp *c);	O(1)
int getIdDiscoD2(Comp *c);	O(1)
Rede* getRedeComp(Comp *c);	O(1)

ListaComp.c

Funções	Complexidade
ListaComp* lista_comp_cria();	O(1)
ListaComp* lista_comp_insere(ListaComp *lc, int id);	O(1)
void adicionar_rede_lista(ListaComp *l, Rede *r);	O(1)
int lista_comp_vazia(ListaComp* l);	O(1)
void lista_comp_libera(ListaComp *lc);	O(n^2)
Comp* busca_comp(ListaComp *lc, int x);	O(n)

void setProx(ListaComp *l, ListaComp *p);	O(1)
ListaComp* getProx(ListaComp *l);	O(1)

Neolook.c

Funções	Complexidade
int escalonador_fcfs(char *s);	O(n)
int escalonador_sjf(char *s);	O(n)
void fila_cpu(Comp *c, int n);	O(n(3n-1)+n)
void fila_disco_auxiliar(Comp *c, int n, int idDisco);	O(n(3n-1)+n)
void fila_disco(Comp *c, int n);	O(2(n(3n-1)+n))
Fila* fila_rede_FIFO(Comp *c, int n);	O(n(3n-1))
void informacoes_processo(Fila *f1, Fila *f2);	O(n)
ListaComp* cria_computadores(ListaComp *l, int num_comp, Rede *r);	O(n)
static float qtd_processos(Fila *f);	O(2n)
static void informacoes(float x1, float x2);	O(1)
void taxas_processamento(Fila *f);	O(2n)
void libera_memoria_neolook(ListaComp *l, Fila *f);	O(n^2+n)
void falha();	O(1)
void neolook_escalonador(ListaComp *lc, Comp *c, int num_comp, char *escalonador, Fila *f1)	O(n(n(3n-1)+2n+ 2(n(3n-1)+n))+n+n(3n-1))
void neolook(char *text, char *escalonador, int num_comp);	O(n(n/4)+n+(n(n(3n-1)+2n+ 2(n(3n-1)+n))+n+n(3n-1)))

4. MANUAL: COMO FUNCIONA O SISTEMA:

O sistema funciona com uma simulação de dois algoritmos de escalonamento, bastando somente, compilar os arquivos .c, gerar um arquivo executável, logo, em seguida execute-o no terminal, passando antes o escalonador do programa.