FMU – FACULDADES METROPOLITANAS

NATHAN RODRIGUES DE SOUZA - RA:6617073

PEDRO MINORU IZUMIDA DE ALMEIDA - RA: 7264016

SIMULADOR PARA O ESCALONAMENTO DE PROCESSOS

São Paulo

2020

NATHAN RODRIGUES DE SOUZA

PEDRO MINORU IZUMIDA DE ALMEIDA

SIMULADOR PARA O ESCALONAMENTO DE PROCESSOS

Trabalho simulador de escalonamento de processos apresentado a professora Maria Inês Lopes Brosso Pioltine requisito parcial para a obtenção do título Bacharel em Ciência da Computação.

São Paulo

2020

Sumário

RESUMO

Através do desenvolvimento de um simulador de escalonamento de processos busca-se a exemplificação do funcionamento deste mecanismo dentro do sistema operacional.

Através de pesquisas bibliográficas para o aprofundamento referente ao escalonamento de processos busca-se transpor as abstrações que a temática possa oferecer. Para cada principal parte do processo foram criadas funções que são apresentadas com nome explicativos traçando paralelo a parte teórica esquivando-se assim de um campo imaginário para o campo mais concreto que a linguagem C possa trazer.

Para que seja possível a manipulação dos processos suas respectivas informações devem ser alocadas sendo assim utilizadas as tradicionalmente intituladas pilhas. Entretanto devido ao volume de informações, guardar estes dados de maneira estruturada é imprescindível para que o sistema operacional trabalhe de maneira eficaz, evitando redundâncias, falhas em consultas ou guarnição das informações relacionadas aos processos, esta importante relação descreve o ponto culminante deste projeto, trata-se do trabalho em conjunto do bloco de controle de processos e o escalonador

ABSTRACT

Through the development of a process scheduling simulator, the primal focus is to exemplify the functioning of this mechanism inside of the operating system.

Through bibliographic researches to understand more specificly about scheduling of processes, the objective's project is to transpose the abstractions that subject may offer. For each main part of the process, functions were created that are named with explanatory titles, setting parallel to the theoretical part, thus avoiding an imaginary field for the more concrete field that the C language can bring to us.

To be able to manipulate the processes, their respective informations must be allocated, being used the called traditionally piles. However, due to the volume of information, keeping this data in a structured way is essential for the operating system work effectively, avoiding redundancies, queries failures or garnishing information related to processes, this important relationship describes the culmination of this project, the combination of the control block of processes and the scheduler working together.

INTRODUÇÃO

Ao tratarmos de processos, a associação dos processadores ao assunto é imediata, no entanto, como chegam ao seu destino para serem devidamente executadas? O processador organiza-se de alguma forma a fim de realizar suas atividades de maneira mais efetiva/eficaz ou trata-se de algum tipo de mecanismo que é responsável por esta questão. Os questionamentos mencionados anteriormente serão aprofundados e exemplificados de maneira mais detalhada através de um simulador desenvolvido na linguagem C.

OBJETIVO GERAL

Busca-se a exemplificação do funcionamento de um escalonador de processos bem como a interação das áreas/métodos envolvidos para que as tarefas sejam devidamente direcionadas ao processador.

Para contribuir a compreensão acerca do escalonamento de processos, a linguagem C foi utilizada, através da qual é possível a construção de blocos de código de maneira hierárquica que se assemelham aos mecanismos existentes no sistema operacional, ilustrando assim, o campo teórico de frequente confusão.

OBJETIVO ESPECÍFICO

A Linguagem C é utilizada a fim de ilustrar a comunicação bem como comportamento ao que aborda o escalonamento de processos. Para tal funções que simulam as tarefas por parte do usuário, processos de sistema bem como bloco de controle de processos e o próprio escalador são criados.

A princípio para que seja esclarecido os tipos de processos e suas relações, é realizada a criação de duas funções “Processes”, “systemProcesses” das quais unem-se em um bloco de controle, simulando assim a junção de tarefas oriundas do próprio sistema operacional e a necessidade/iniciativa do usuário.

Se faz presente a guarnição dos dados referentes aos processos tais como P.I.D, peso, prioridade do contrário não haverá modo de manipular as tarefas para escalonamento, sendo assim necessária a elucidação, o desenvolvimento de uma função que simule o B.C.P.

Por fim, é imprescindível a produção de uma unidade capaz de determinar quais tarefas deverão ser direcionadas a CPU bem como sua ordem, neste projeto uma função denominada “Scheduler” é responsável pela lógica por traz dos métodos de escalonamento.

Ao estabelecer a comunicação e passagem de informações dentre os recursos listados, realiza-se a criação de uma barra de progresso que exemplifique ao usuário o processamento de cada tarefa.

METODOLOGIA DE PESQUISA

Através de pesquisas bibliográficas foram aprimorados conceitos referentes aos sistemas operacionais com foco ao escalonamento de processos. Palestras bem como aulas, foram igualmente examinadas, sendo fonte de conhecimento a fim de maior assertividade e clareza acerca da temática. Através da análise de informações reunidas é possível a distinção de pontos, lacunas ou lapsos responsáveis pela dificuldade no entendimento dos paradigmas referentes ao escalonamento de processos.

Foram utilizadas as metodologias experimental e pesquisa-ação, sendo aplicado o conhecimento e experiencia acumulados durante o processo de criação do simulador.

DESENVOLVIMENTO

Abordaremos incialmente as funções principais das quais este projeto é composto, para assim focar na união e parte central do projeto que será a função main, pois além de apresentar as informações de maneira hierárquica, auxilia para uma rápida compreensão, pois faz alusão a construção dos próprios códigos em na linguagem C (tradicional declaração de funções antes do main).

A princípio temos a função sysProcess, das quais as variáveis são criadas na função main, assim como um sistema operacional as gera em uma parte central, mas que as direciona para seus respectivos setores de funcionamento/utilização. Esta função tem como objetivo simular processos próprios do sistema operacional. Ponteiros são passados como parâmetro a esta função, entretanto seus valores não são escolhidos pelo usuário, pois possui como intuito evidenciar tarefas que são nativas do próprio sistema operacional fugindo da manipulação do utilizador do sistema.

Para que haja a simulação de processos gerados pelo usuário, isto é, abrir um programa ou executar uma tarefa, temos a função “process”. Ao executar o simulador, uma pergunta é exibida ao usuário para que assim digite o PID do processo, peso, status e prioridade, vale a ressalva, os processos serão alocados em um bloco de controle, para tal um laço de repetição irá atribuir através de ponteiros os valores informados pelo usuário. Este laço se iniciaria em 8 pois alocará somente uma pequena parcela de suas informações cedidas pelo usuário.

Anteriormente vetores foram desenvolvidos, para a área a seguir um array bidimensional é criado para simular a alocação das diversas pilhas de informação em um bloco de controle único, mais conhecido como B.C.P, para fácil entendimento assim denominou-se esta função. No array 2D batizado como “bcp_Table”, aloca-se os principais números referentes aos processos tais como, memória a ser utilizada, prioridade, status e identificação dos processos. Sendo composto por 4 colunas (P.I.D, memory, process status e process priority) e 10 linhas reservadas a parcela de processos de sistema (sysProcesses) e usuário (process). A organização dos dados nesta área envolve a permutação entre os índices das matrizes para alocar cada informação de maneira que representem o modo em que o escalonador ordena as informações no bloco de controle, tornando a função de direcionar dados ao processador de maneira mais eficaz, segundo a lógica proposta.

Com o intuito de clarear o campo abstrato referente aos métodos de escalonamento, 4 opções são exibidas ao usuário (Priority,”F.I.F.S”,“L.I.F.S” e Ticket.) para que assim escolha o método de escalonamento que deseja examinar.

Partimos hierarquicamente para a função scheduler onde a princípio simulará um processo de escalonamento a partir da prioridade (caso a opção 1 seja digitada), sendo assim processos de menor número possuem maior prioridade aos que portam números mais levados, logo um processos de prioridade 1 será processado de maneira antecedente ao de prioridade 10. Em seguida temos o método de escalonamento denominado F.I.F.S (First in First Service), para tal é usado o peso das tarefas como parâmetro, ou seja, processos mais leves possuem alta tendência de chegarem ao processador antes dos demais. Logo após temos o L.I.F.S que trata-se de um método antagônico ao F.I.F.S, logo os últimos a chegarem serão os primeiros a serem tratados. Por fim, temos o método de Loteria (Ticket), onde são sorteados quais processos serão tratados, tarefas de maior prioridade recebem um maior número de tickets, tendo assim probabilidade maior que as demais de ser direcionada ao processador, pois a decisão de processamento ocorre através do sorteio dos bilhetes. Para este método, um vetor é desenvolvido e alocado como 5ª coluna do vetor 2D B.C.P, a fim de maior controle bem como exemplificação da manipulação dos dados para o processo que envolve o escalonamento de tarefas.

Focando a parte didática, uma função responsável pela criação de uma barra de progresso é desenvolvida, evidenciando assim a execução de cada processo ao usuário. Para os métodos de escalonamento uma barra somente se inicia ao término de sua antecessora, pois trata-se de uma coletânea de processos não preemptivos. O peso de memória de cada processo é inserido em uma função “Sleep” elucidando ao usuário de que as tarefas de maior peso consecutivamente tomarão maior tempo ao obterem suas respectivas conclusões. Esta função possui um laço FOR, seu limite de repetição representará a quantidade de caracteres escritos (para este caso ALT + 219, “  ”) definindo assim o tamanho da barra de processamento.

Por fim chegamos à função “main”, é responsável pela união de cada secção deste projeto, conectando as funções para que o processo de escalonamento ocorra. Os vetores são todos criados nesta região, a fim de exemplificar ao usuário de que os “objetos” são criados em uma unidade central, mas compartilhados entre diversas outras áreas para que objetivo maior seja concluído, isto é, o processamento das tarefas.

RESULTADOS OBTIDOS

Através do desenvolvimento constante de um algoritmo capaz de simular o escalonamento de processos além de aplicação de testes, saltos ocorreram referentes a velocidade de raciocínio para a construção lógica bem como o refinamento de técnicas voltadas a programação, sendo reforçados ao surgimento de cada paradigma, parte oriunda da própria linguagem C.

Houve melhor compreensão do funcionamento relacionado a ordenação e estruturação das informações para a execução das instruções no processo de direcionado a CPU, possibilitando assim o enriquecimento do projeto.

Por fim, o êxito foi obtido tendo em vista que a construção do simulador proporcionou a abstração que a temática pode oferecer, tornando assim mais claros os conceitos referentes ao escalonamento de processos.

CONCLUSÃO

Através da construções de funções que representem os setores de um sistema operacional é possível distinguir as áreas bem como etapas envolvidas no escalonamento de processos.

A princípio é preciso que seja clara a distinção entre programas e processos, sendo um a execução do outro. Em seguida, deve-se haver a diferenciação entre processos oriundos da parte sistema/usuário, assim como temos os chamados CPU Bound e I/O Bound, dos quais se assemelham por tratarem de diferentes prioridades.

Como manipular os processos de maneira estruturada? A partir deste questionamento entende-se que a construção de pilhas auxiliam quanto a este quesito, seja exibição dos valores, passada entre funções e afins, entretanto estas pilhas trabalham de maneira isoladas, um método mais eficaz é a junção dessas pilhas para um local de controle, ilustrando assim a função do Bloco de Controle de Processos (B.C.P).

A área que detém a algoritmo por trás do método de escalonamento de processos chama-se scheduler, diferentes logicas podem ser aplicadas tendo em vista que métodos antigos não são necessariamente deixados de lados, pois desde um monitor de sinais vitais a um computador necessitam de um sistema operacional, entretanto trabalharão de maneiras distintas, pois em sua gênese possuem objetivos diferentes. O método F.I.F.S não é recomendado a um sistema operacional de multiprocessamento reservado a desktops, entretanto a hardwares mais simples a utilização pode ser extremamente adequada. Aprimorou-se a compreensão referente a rápida substituição dos métodos como F.I.F.S ou L.I.F.S com os avanços da área, caindo seu uso por possuírem como critério de execução primeiros ou últimos processos de um fila, adiando parte das tarefas que, por sua vez , poderão nunca ser executadas, exemplificando assim o conceito Starvation.

Por fim torna-se clara como a utilização de ponteiros facilita além do ganho relacionado a eficácia quanto a passagem de informações entre as diferentes áreas que compõem o sistema operacional.

Para implementações futuras, busca-se igualmente a simulação de sistemas preemptivos para tal se faz necessária a criação de threads para que os laços de repetição FOR sejam processados simultaneamente, gerando assim barras de progresso que sejam operem paralelamente, entretanto com diferentes durações, tendo em vista que o tempo de conclusão varia entre cada tarefa devido suas especificidades.

Para uma interface mais amigável, busca-se a criação de um novo vetor para a alocação referente aos nomes dos processos, tal como é realizado no sistema operacional, onde apesar de existir um identificação (P.I.D) ainda assim é atribuído nome a cada um.

REFERENCIAS BIBLIOGRAFICAS

TANENBAUM, Andrew S. e BOS, Herbert. Sistemas Operacionais Modernos. 4. ed. São Paulo: Person Education do Brasil, 2016 (Disponível na Biblioteca Virtual- Person).

Maxwell Anderson, Estevam Pessoa, Josemary marcionila, Sistema Simulador Escalonamento de Processos em Sistemas Operacionais. Disponível em:<https://periodicos.ifpb.edu.br/index.php/principia/article/viewFile/295/252>, Acesso em: 12/10/2020.

Emerson L. de Moraes, Henrique Y.O. Asakura, Ricardo P. Bonfiglioli, Murilo da S. Dantas. Simulador Web de Algoritmos para Escalonamento de Processos em um Sistema Operacional. Disponível em: <http://plutao.sid.inpe.br/col/dpi.inpe.br/plutao/2012/06.21.21.52.18/doc/Simulador%20Web%20de%20Algoritmos%20para%20Escalonamento%20de%20Processos%20em%20um%20Sistema%20Operacional_Emerson-Henrique-Ricardo-Murilo.pdf>. Acesso em: 12/10/2020.

Carlos Eduardo Lino. Simulador de escalonamento de processos. Disponível em:<https://www.vivaolinux.com.br/script/Simulador-de-Escalonamento-de-Processos>. Acesso em: 12/10/2020.

UNIVESP TV. Sistemas operacionais – Aula 04 – Processos. Disponível em:<https://www.youtube.com/watch?v=Yb0rViAwVQ0&list=PLxI8Can9yAHeK7GUGxMsqoPRmJKwI9Jw&index=5>. Acesso em: 12/10/2020.

How to code/create a Loading Bar in C/C++ using (codeblocks). Disponível em: <https://www.youtube.com/watch?v=KnbYuOuHhW0>. Acesso em: 12/10/2020.

Loading Bar in C++. Disponível em: <https://www.youtube.com/watch?v=5Sn9yDDo7OA>. Acesso em: 12/10/2020.

Loading Screen In C Programming Without Using Graphics (C Projects). Disponível em: <https://www.youtube.com/watch?v=ZgsHKwdslIQ>. Acesso em: 12/10/2020.

Ordenação de vetor em C. Disponível em:<https://www.youtube.com/watch?v=llWdKjFtXuM>. Acesso em: 12/10/2020

.

APÊNDICE – CÓDIGO DO SIMULADOR DE ESCALAMENTO DE PROCESSOS

```
#include <stdio.h>

#include <windows.h>

#include <stdlib.h>

#include <time.h>


#define PIDE        0

#define MEMO        1

#define STAT        2

#define PRIO        3

#define COL         5

#define ROW         10


void sysProcess(int *p_PI,int *p_M,int *p_S,int *p_PR, int *p_T){

    int i;

    srand(time(NULL));


    *(p_PI+0) = 5587;

    *(p_PI+1) = 4877;

    *(p_PI+2) = 9951;

    *(p_PI+3) = 5583;

    *(p_PI+4) = 1287;

    *(p_PI+5) = 5229;

    *(p_PI+6) = 1927;

    *(p_PI+7) = 1723;
```

```
*(p_M+0) = 2;

*(p_M+1) = 4;

*(p_M+2) = 8;

*(p_M+3) = 32;

*(p_M+4) = 64;

*(p_M+5) = 128;

*(p_M+6) = 256;

*(p_M+7) = 512;


*(p_S+0) = 1;

*(p_S+1) = 0;

*(p_S+2) = 2;

*(p_S+3) = 1;

*(p_S+4) = 2;

*(p_S+5) = 0;

*(p_S+6) = 0;

*(p_S+7) = 1;


*(p_PR+0) = 37;

*(p_PR+1) = 16;

*(p_PR+2) = 13;

*(p_PR+3) = 50;

*(p_PR+4) = 6;
```

```
        *(p_PR+5) = 10;
        *(p_PR+6) = 12;
        *(p_PR+7) = 22;

        for(i=0;i<ROW;i++){
                *(p_T+i)= rand()%100;
        }

}

int process(int *p_PI,int *p_M,int *p_S,int *p_PR){
        int i;

        for(i=8;i<ROW;i++){
                printf("Type PID Value: ");
                scanf("%d",(p_PI+i));

                printf("Type Memory Weight: ");
                scanf("%d",(p_M+i));

                printf("Type Status Number: ");
                scanf("%d",(p_S+i));

                printf("Type Priority Number: ");
```

```
            scanf("%d",(p_PR+i));
            printf("\n");
        }

}

void BCP(int (*bcp_Table)[COL],int *p_PI,int *p_M,int *p_S,int *p_PR,int *p_T){
        int i,z;
        for(i=0;i<=ROW;i++){
            for(z=0;z<COL;z++){
                *(*(bcp_Table+i)+0) = *(p_PI+i);
                *(*(bcp_Table+i)+1) = *(p_M+i);
                *(*(bcp_Table+i)+2) = *(p_S+i);
                *(*(bcp_Table+i)+3) = *(p_PR+i);
                *(*(bcp_Table+i)+4) = *(p_T+i);
            }
            printf("\n");
        }
}

void scheduler(int *p_PI,int *p_M,int *p_S,int *p_PR,int *p_T, int opt){
        int i,X,Z,change;

        void changeValues(){
```

```
            change = p_PI[X];
            p_PI[X] = p_PI[Z];
            p_PI[Z] = change;

            change = p_M[X];
            p_M[X] = p_M[Z];
            p_M[Z] = change;

            change = p_S[X];
            p_S[X] = p_S[Z];
            p_S[Z] = change;

            change = p_PR[X];
            p_PR[X] = p_PR[Z];
            p_PR[Z] = change;

            change = p_T[X];
            p_T[X] = p_T[Z];
            p_T[Z] = change;
    }

    for(X=0;X<ROW;X++){
            for(Z=X;Z<ROW;Z++){
                    if(opt == 1){
```

```
                    if(p_PR[X] > p_PR[Z]){
                        changeValues();
                    }
                }
                else if(opt == 2){
                    if(p_M[X] > p_M[Z]){
                        changeValues();
                    }
                }
                else if(opt == 3){
                    if(p_M[X] < p_M[Z]){
                        changeValues();
                    }
                }
                else{
                    if(p_T[X] < p_T[Z]){
                        changeValues();
                    }
                }
            }
        }

}
```

```
void loadBar(int *p_PI, int *p_M){
	int i,Z;

	for(i=0;i<ROW;i++){
		printf("Process: %d",*(p_PI+i));
		printf("\n");

		for(Z=0;Z<40;Z++){
			Sleep(*(p_M+i));
			printf("%c",219);
		}

		printf("\n\n");
	}

}

int printBCP(int (*bcp_Table)[COL]){
	int i,j;
	for(i=0;i<ROW;i++){
		for(j=0;j<COL;j++){

			if(j == 0){
				printf("PID[%d][%d] = %d",i,j, *(*(bcp_Table+i)+j));
```

```
                }

                else if(j == 1){
                        printf("Memory[%d][%d] = %d",i,j,
*(*(bcp_Table+i)+j));
                }

                else if(j == 2){
                        printf("Status[%d][%d] = %d",i,j, *(*(bcp_Table+i)+j));
                }

                else if(j == 3){
                        printf("Priority[%d][%d] = %d",i,j,
*(*(bcp_Table+i)+j));
                }
                else{
                        printf("Ticket[%d][%d] = %d",i,j, *(*(bcp_Table+i)+j));
                }
                printf("; ");
        }
        printf("\n");
    }
}

int main(){
```

```
int i,j,opt,bcp_Table[ROW][COL];;

int p_PID[ROW],p_MEM[ROW], p_STA[ROW], p_PRI[ROW], p_TIC[ROW];

sysProcess(p_PID,p_MEM,p_STA,p_PRI,p_TIC);

process(p_PID,p_MEM,p_STA,p_PRI);

BCP(bcp_Table, p_PID,p_MEM,p_STA,p_PRI,p_TIC);

printf("\n_____BEGIN BCP Table without any method_____\n");

printBCP(bcp_Table);

printf("\n_____END BCP Table without any method_____\n");

printf("\nEscolha um dos tipos de escalonamento\n\n");

printf("\n1 - Priority\n2 - FIFO\n3 - LIFO\n4 - Ticket\n\nInsert the value:  ");

scanf("%d", &opt);

scheduler(p_PID,p_MEM,p_STA,p_PRI,p_TIC, opt);

BCP(bcp_Table, p_PID,p_MEM,p_STA,p_PRI,p_TIC);

printf("\n_____BEGIN OF SCHEDULER WORK _____\n");
```

```
            printBCP(bcp_Table);
            printf("\n_____END OF SCHEDULER
WORK_____\n");

            system("Pause");

            loadBar(p_PID, p_MEM);

        system("Pause");
        return(0);
    }
```

TRABALHOS RELEVANTES

TANENBAUM, Andrew S. e BOS, Herbert. Sistemas Operacionais Modernos. 4. ed. São Paulo: Person Education do Brasil, 2016 (Disponível na Biblioteca Virtual- Person). Livro utilizado durante as aulas (Sistemas Operacionais Modernos), trata-se da organização de estrutura de dados em um SO, tais teorias e conceitos citados da bibliografia, foram testados e utilizados como principal fonte para a construção do simulador.

Carlos Eduardo Lino. Simulador de escalonamento de processos. Disponível em: <https://www.vivaolinux.com.br/script/Simulador-de-Escalonamento-de-Processos>

Trata-se de um projeto em linguagem C, diferentes tipos de escalonamento com nível de prioridade de execução aos processos de menor peso, que consequentemente, entrarão na CPU e sairão primeiro (Round-Robin,SJF,FCFS).

Alex Onsman.What is Process Control Block (PCB)?. <https://www.tutorialspoint.com/what-is-process-control-block-pcb>. Site utilizado como fonte para a construção e compreensão do BCP (Process Control Block).