Universidade da Beira Interior

Sistemas Operativos

Relatório final do projeto prático

Ruben Santos, Engenharia Informática, Nº41308

Gonçalo Domingos, Engenharia Informática, Nº41719

ID do grupo: 1

Link para o projeto GitLab: https://gitlab.com/so-projeto-id1/id-1-so-project-2019-2020

18/06/2020

Índice

1 Introdução

Este projeto dividiu-se em duas partes, onde a primeira consistia em que através de um sistema operativo seja possível desenvolver uma aplicação que simule o funcionamento da gestão, do uso da memória e da execução de processos.

Iremos utilizar, neste conceito, operações de escalonamento do CPU.

Na primeira parte do projeto também será possível a criação de novos processos, bloquear e terminar determinados processos e a comutação de contexto.

Por fim, é possível visualizar as estatísticas globais da execução do simulador e o estado do sistema no "stdout", onde encontramos uma vasta informação que nos permite retirar conclusões e dados relevantes acerca do simulador e dos processos.

Ao longo da primeira parte, criamos e recorremos inúmeras vezes a varáveis globais e a funções criadas por nós ou descritas no enunciado do projeto, que estarão todas definidas e explicadas (o que cada um faz/devolve) neste relatório.

Na parte dois, objetivo consiste em simular e avaliar esquemas de reserva e libertação de memória através de vários algoritmos.

Temos a possibilidade de alocar, desalocar memória e fazer solicitações das mesmas e também contemos um relatório de estatísticas relevantes da simulação.

Por último também acrescentamos mais dois algoritmos de escalonamento, para além do FCFS, agora temos o SJFS e a Prioridade ambos com a opção de serem não preemptivos ou preemptivos.

Utilizamos a linguagem de programação C para o desenvolvimento desta aplicação.

2 Variáveis Globais

Todas as variáveis globais que iremos citar estarão definidas no ficheiro "biblioteca.h".

Variáveis que foram solicitadas no enunciado : Tempo, "CPU_ProgramCounter","PCB_Tabela" (Array com entrada para cada processo utilizado) , "readyQueue "(fila de processos prontos a executar), Bloqueados (fila de processos bloqueados) e "runningState" (estrutura de dados que contêm da PCB_Tabela, do respetivos processo em execução, bem como o seu PID e PC).

Para além das que foram solicitadas, acrescentámos as seguintes variáveis:

Parte I

- Instrução *memoria : Array do tipo de dados instrução.
- File_Process *fila : Array dos processos lidos do "plano.txt".
- Nodo *terminados : Fila que contêm os processos terminados.
- Nodo *execução : Processo que está a executar no momento.
- int flag_terminar_simulador : flag para terminar a execução do simulador, quando é "1" termina.
- int aux_primeiro_endereco : cada vez que entra um processo na memória, esta varável contêm a posição onde será inserido.
- int contador_array_ficheiro_memoria : contador que contêm o tamanho da memória que está a ser utilizada.
- int flag_control : flag para diferenciar quando usar o control ou a stdin.
- int numero_de_processos/ int aux_numero_de_processos : Para não executar processos que não vão existir.
- char buffer_ler_instrucao : Guarda uma instrução lida embora não executada.
- int timeQuantum : TimeQuantum do programa.
- int TEMPO : Variável que guarda o tempo á medida que o programa executa.
- runningState x : Array de processos em execução.

Parte II

- ListaM listaMemoria : Lista do tipo de dados ListaM.
- ListaM inicio : Ponteiro inicial referente ao processo de alocação de memória, ou seja, é a primeira posição onde se vai começar o processo de alocação.
- ListaM *nextFitPointer : Ponteiro específico para o algoritmo NextFit.
- int QUALFIT : Decisão do user referente á escolha dos algoritmos.
- int QUALSOLICITACAO : Decisão do user referente ás solicitações, ou seja, se será aleatoriamente, estaticamente ou a partir de um trace.
- int MemoriaTotal : Contém a memória total fornecida pelo user.
- int n_particoes : Nós que conterá a lista.
- int seed : Semente para gerar os números aleatórios.
- int tamanho_particao : Fornecido pelo user. Usado para o cálculo do número de partições.
- int n_particoes_por_ocupar : Contém o número de partições que não foram ocupadas na lista.
- int resultado_alocacao : Resultado do processo de alocação de memória.
- int numeroDeNos, numeroTotalDeNos : Número de nós e o seu total que serão usados no cálculo das estatísticas.
- int countNEGADOS : Contagem de solicitações negadas.
- int countTOTAL : Contagem total de solicitações.
- int *fragrementos_externos : Número de fragmentos externos.
- int tamanho_fragmento : Tamanho do vetor de fragmentos.
- int *numero_de_nos : Número de nós da lista. Usado no cálculo de estatísticas.
- int tamanho_nos : Tamanho do vetor de nós. Usado no cálculo de estatísticas.
- int OPCAO_ESCALONAMENTO, OPCAO_ESCALONAMENTO2 : Opção do user referente aos algoritmos de escalonamento disponíveis.
- int dentro_execute :
- int indice_para_executar:
- int SJFS_count, tempo_restante_SJFS2, tamanho_Fila_Processos:
- int teminou_processo :
- double fragmentos_medio : Número médio de fragmentos. Usado no cálculo de estatísticas.
- double nos_medio : Número médio de nós. Usado no cálculo de estatísticas.
- double percentagem_vezes : Percentagem de vezes que as solicitações foram negadas. Usado no cálculo de estatísticas.

Estruturas de dados

Estruturas de dados utilizadas (Comentários a descrever cada tipo de dados e a respetiva informação):

Parte I

```
//ESTRUTURAS DE DADOS
typedef struct {
    char ins;// Instrução
    int n;    // Variável N
    char nome[15]; // Nome do programa
} instrucao;

typedef struct {
    char nome[15]; // Nome do processo
    intendereco; //endereço do processo
    int ValorVariavel; // Valor da variável
    int PID_processo; // Program ID do processo
    int PPID_processo; // Program ID do pai
    int prioridade; // Prioridade dos processos
    int program_counter; // Program counter
    int estado; // Estado  if==0 ->ready if==1 ->em execução  if==2 -
>blocked  if==3->terminado
    int tempo_chegada; //Tempo em que chegou à readyQueue
    int tempo_terminada; //Tempo em que terminou
} PCB;

typedef struct NODO{
    PCB *pcb; //Nodo que aponta para um PCB
    struct NODO *nseg;   //Nodo seguinte
}Nodo;

typedef struct {
    int indicePCB;    //Indice do processo em execução
    int PID_exec;    //Program ID
    int PC_exec;    //PC do processo
}runningState;

typedef struct{

    char nome[20];      //Nome do processo
    int tempo;          //Tempo descrito no ficheiro
    int prioridade; //Prioridade descrita no ficheiro

}File_Process;
```

Parte II

```
//PARTE 2
typedef struct LISTA{

    int ID_Process; // Id do processo que vai para a Lista Memória
    struct LISTA *nseg; // elemento seguinte da lista

}ListaM;
```

Funções usadas/criadas

Ao longo deste projeto, usamos diversas funções de forma a minimizar e facilitar a programação do simulador de processos e da memória.

De seguida cada função será descrita e explicada detalhadamente:

Funções Parte I

- char *parsePRG (char *string) : Devolve uma string até encontrar um "." .

- void parse (char *buf, char **args) : Particiona o comando Unix (armazenado em buff) em argumentos.

- void ficheiro_memoria (FILE *f1) : Irá receber o ficheiro "plano.txt" como parâmetro e passa para o array de memória todas as instruções.

- void execute2 (int índice_do_processo) : Recebe como parâmetro o índice do processo que irá executar e executa as instruções que estão em memória.

- void FCFS2() : Aqui caso exista algum processo na readyQueue, este será removido e irá ser executado.

- void report() : Esta função trata de dar print do tempo atual, e das informações dos processos existentes nas filas da readyQueue, dos bloqueados, dos prontos a executar e dos terminados.

- void estatísticas_globais () : Função que calcula “Arrival Time”, “Burst Time”, “Completion Time”, “Turnaround Time” e “Waiting Time” dando print de todas no stdout.

- void gerirProcessosPreemptivo () : Aqui vamos abrir o ficheiro “plano.txt” e preencher a fila de processos com todos os processos do ficheiro. De seguida irá analisar a flag usada no “lerComandos()” e de acordo com a flag decide se vai utilizar o “control.txt” ou a “stdin”. Analisa todos os processos do plano à medida que o tempo passa e quando tiver no tempo pretendido insere-se o processo na readyQueue.

- void control2(char *array, int count) : Esta função irá ler os caracteres de um ficheiro e caso coincidam executa as respetivas ações. (E-> Executa um programa com N unidades de tempo ; I-> Interrompe um processo e bloqueia-o ; D-> Chama o escalonador de Longo prazo ; R-> Chama a função de report e imprime as estatísticas ; T-> Termina o simulador e imprime estatísticas).

- void minha_stdin () : Irá ler da stdin e ver se o carácter coincide com os que estão definidos, caso coincidam vai executar as respetivas ações correspondentes a cada letra referidas em cima.

- void lercomandos () : Nesta função, o utilizador irá tomar a decisão de querer “debugging” ou não, caso deseje utilizar, a função irá retornar flags, 1 se o utilizador decidir usar o ficheiro “control.txt”, 0 se usar a stdin.

- void LongoPrazo () : Esta função irá escolher aleatoriamente uma posição da fila de processos bloqueados e mudá-lo para a readyQueue.

- int sizeFila (Nodo *fila) : Devolve o tamanho de uma fila.

- Nodo *pushFila3 (Nodo **L, Nodo *nv) : Recebe a fila (L) onde vai inserir o elemento que recebeu também por parâmetros (nv).

- Nodo *criarNodo (PCB *pcb) : Cria um apontador para um PCB.

Funções parte II

- void gerirFits() : Ficheiro que contêm todos os “Fits” , ou seja, todos os algoritmos de alocação de memória.

- void criarMemoria() : Cria nodo’s e mete na lista de memória até o número de partições ser zero.

- void FirstFit(int ID, int num_particoes) : Ajusta os dados na memória, digitalizando desde o início da memória disponível até ao fim, até encontrar o primeiro espaço livre que seja grande o suficiente para aceitar os dados.

- void WorstFit(int ID, int num_particoes) : Aloca um processo numa partição que seja a maior partição entre todas as outras disponíveis na memória principal.

- void BestFit(int ID, int num_particoes) : Aloca um processo numa partição que é a partição que mais se adequa ao tamanho do processo, ou seja a menor partição suficiente entre todas as disponíveis.

- void NextFit(int ID, int num_particoes) : Versão semelhante ao first fit mas que quando é chamada novamente começa a pesquisa onde tinha ficado, acabando por não voltar ao início da lista novamente.

- void estatisticasFit() : Função que calcula estatísticas como o número médio de fragmentos externos, o tempo médio de alocação e a percentagem de vezes que uma solicitação é negada.

- void SJFS() : Caso o burst-time do processo que chegou à readyQueue seja menor do que o tempo restante do processo em execução, então há preempção. Corresponde ao SJF preemptivo.

- void SJFS2() : Neste caso, uma vez que a CPU é atribuída a um processo, este não pode ser preempcionado até completar o seu CPU burst por completo. Corresponde ao SJF não preemptivo.

- void correrFilaProcessos() : Na presença de um algoritmo preemptivo corre uma fila de processos e coloca na readyQueue.

- void printList(Nodo *L) : Para testes, onde conseguimos ver se as listas estavam ou não bem preenchidas.

- void printListaMemoria(ListaM *La) : Para testes, onde conseguimos ver se a lista de memória estava ou não bem preenchida.

- void Prioridade() : Consoante a prioridade do processo, esta função vai comparar com as restantes prioridades dos outros processos e executa o que tiver maior prioridade.

- void mudarHead(Nodo **L) : Muda a “cabeça” da lista. Função para testes.

- int deallocate_mem (int process_id) : Desaloca memória que estava alocada para o processo cujo Id era o process_id. Retorna 1 se a desalocação foi bem sucedida, -1 em caso contrário. Fornecida no enunciado.

- int allocate_mem (int process_id, int num_units) : Aloca um certo numero de unidade num processo cujo o Id é igual ao process_id. Retorna o número de nós percorridos na atual lista, caso contrário retorna -1. Fornecida no enunciado.

- int fragment_count () : Retorna o número de furos (fragmentos de 1 ou 2 unidades). Fornecida no enunciado.

- ListaM *criarNodoLista() : Cria um nodo do tipo ListaM.

- ListaM *pushListaMemoria(ListaM *Lista, ListaM *no) : Dá “push” de um elemento para uma lista.
- Nodo *pop2(Nodo **L) : Remove um elemento de uma lista.
- Nodo *obterElemento(Nodo **L, Nodo *nv) : Obter elemento de uma lista.
- Nodo *obterElementoExecucao(Nodo **L) : Obter primeiro elemento de uma lista.
- Nodo *menorBurstTime() : Calcula o burstTime de cada processo e escolhe o menor de todos.
- Nodo *maiorPrioridade() : Compara as prioridades de todos os processos e devolve o que tem maior prioridade.
- Nodo *obterFirst(Nodo **L) : Obter o primeiro elemento que entrou numa lista.

Funcionamento do Programa

Inicialmente ao executar o programa, o utilizador vai inserir o "timequantum" que deseja e terá a opção de escolher qual algoritmo de alocação quer utilizar (1-First Fit ; 2-Worst Fit ; 3- Best Fit ; 4- Next Fit ; 5- Nenhum).

De seguida, o programa irá pedir ao utilizador para introduzir a memória total que o sistema irá conter e também o tamanho das partições.

O número de partições será calculado dividindo a memória total do sistema pelo tamanho das partições (ex: 60 -> Memória Total, 2 -> Tamanho das Partições, 60/2 = 30 partições).

Depois o utilizador tem a possibilidade de escolher 3 tipos diferentes de solicitações, terá a opção de escolher a opção de aleatório, estaticamente ou trace.

Caso a opção seja Aleatório o utilizador terá que introduzir o número da seed para gerar um número "random" a partir da mesma.

Agora, o utilizador terá de escolher o tipo de escalonamento que quer usar,o FCFS, o SJF (caso escolha a opção SJF terá a opção de escolher o modo preemptivo ou não preemptivo) e o algoritmo de Prioridade. De seguida deve dizer se quer "debugging" ou não.

Por fim o utilizador terá que escolher se quer usar o ficheiro "control.txt" ou a stdin para inserir e executar as instruções.

As estatísticas do escalonamento, e as dos algoritmos de alocação serão fornecidas ao utilizador no terminal onde consegue reter informação acerca dos processos (Average Turnaround Time, Average Waiting Time, Burst Time, Arrival Time e Completion Time) e do número médio de fragmentos externos, tempo médio de alocação e percentagem de solicitações negadas, ou seja, estatística dos "Fits".

Exemplos de Execução

Aqui estão disponíveis vários exemplos de execução, onde usamos diferentes algoritmos de escalonamento preemptivos e não preemptivos, bem como vários algoritmos de alocação de memória e diferentes tipos de solicitações.

```
Insira o valor do timequantum
12
Algoritmo a usar:
1- First Fit
2- Worst Fit
3- Best Fit
4- Next Fit
5- Nenhum
1
Memoria Total do Sistema (25):
25
Tamanho das Partições (1):
1
Escolha o tipo de solicitações geradas.
1- Aleatório
2- Estaticamente
3- Trace Não Feita
1
Insira o numero da seed (12481)
11
Escolha um dos algoritmos de escalonamento:
1. Escalonamento de curto prazo
1
1. FCFS
2. SJSF
3. Prioridade
2
```

```
1. Escalonamento de curto prazo
1
1. FCFS
2. SJSF
3. Prioridade
2
1.SJSF Não-Preempetivo
2.SJSF Preempetivo
2
Quer debugging?
1-Sim
2-Nao
2
TEMPO ACTUAL: 17

PROCESSO EM EXECUÇÃO: NÃO EXISTE NENHUM PROCESSO EM EXECUÇÃO.

PROCESSOS BLOQUEADOS: NÃO EXISTE NENHUM PROCESSO BLOQUEADO

PROCESSOS PRONTOS A EXECUTAR:
Fila de processos:
4,0,5,0,12
2,0,1,0,10
-5,-1,-1,-1,-1

PROCESSOS TERMINADOS:
Fila de processos:
5,0,4,293,13
3,0,3,288,10
1,0,2,219,1
Estatísticas Globais:

P#      AT      BT      CT      TAT     WT

P1      1       3       4       3       0
P3      10      4       15      5       1
P5      13      4       18      5       1


Average Turnaround Time = 2.166667
Average Waiting Time = 0.333333

ESTATISTICAS FIT
NUMERO MÉDIO DE FRAGMENTOS EXTERNOS: 0.000000
TEMPO MÉDIO DE ALOCAÇÃO: 8.000000
PERCENTAGENS DE SOLICITAÇÕES NEGADAS: 0.00
```

Neste exemplo foi usado o First Fit, o SJF preemptivo e uma solicitação aleatória.

Agora, usamos o Best Fit, com uma solicitação aleatória, e o algoritmo de escalonamento FCFS.

```
Insira o valor do timequantum
8
Algoritmo a usar:
1- First Fit
2- Worst Fit
3- Best Fit
4- Next Fit
5- Nenhum
3
Memoria Total do Sistema (25):
38
Tamanho das Partições (1):
2
Escolha o tipo de solicitações geradas.
1- Aleatório
2- Estaticamente
3- Trace Não Feita
1
Insira o numero da seed (12481)
231
Escolha um dos algoritmos de escalonamento:
1. Escalonamento de curto prazo
1
1. FCFS
2. SJSF
3. Prioridade
1
Quer debugging?
1-Sim
2-Nao
2
TEMPO ACTUAL: 22

PROCESSO EM EXECUÇÃO: NÃO EXISTE NENHUM PROCESSO EM EXECUÇÃO.

PROCESSOS BLOQUEADOS:
Fila de processos:
2,0,1,0,10

PROCESSOS PRONTOS A EXECUTAR:
Fila de processos:
6,0,6,0,18
5,0,5,0,12
4,0,4,0,13
```

Estatísticas do teste executado em cima (continuação da screenshot anterior).

```
PROCESSOS PRONTOS A EXECUTAR:
Fila de processos:
6,0,6,0,18
5,0,5,0,12
4,0,4,0,13
-5,-1,-1,-1,-1

PROCESSOS TERMINADOS:
Fila de processos:
3,0,3,0,10
1,0,2,219,1
Estatísticas Globais:

P#        AT       BT       CT       TAT      WT

P1        1        3        4        3        0
P3        10       4        23       13       9


Average Turnaround Time = 2.285714
Average Waiting Time = 1.285714

ESTATISTICAS FIT
NUMERO MÉDIO DE FRAGMENTOS EXTERNOS: 0.000000
TEMPO MÉDIO DE ALOCAÇÃO: 19.000000
PERCENTAGENS DE SOLICITAÇÕES NEGADAS: 0.00
```

```
Insira o valor do timequantum
20
Algoritmo a usar:
1- First Fit
2- Worst Fit
3- Best Fit
4- Next Fit
5- Nenhum
3
Memoria Total do Sistema (25):
60
Tamanho das Partições (1):
2
Escolha o tipo de solicitações geradas.
1- Aleatório
2- Estaticamente
3- Trace Não Feita
1
Insira o numero da seed (12481)
5211
Escolha um dos algoritmos de escalonamento:
1. Escalonamento de curto prazo
1
1. FCFS
2. SJSF
3. Prioridade
3
Quer debugging?
1-Sim
2-Nao
2
TEMPO ACTUAL: 23

PROCESSO EM EXECUÇÃO: NÃO EXISTE NENHUM PROCESSO EM EXECUÇÃO.

PROCESSOS BLOQUEADOS: NÃO EXISTE NENHUM PROCESSO BLOQUEADO

PROCESSOS PRONTOS A EXECUTAR:
Fila de processos:
6,0,6,0,18
5,0,5,0,12
4,0,4,0,13
-5,-1,-1,-1,-1
```

Aqui, usamos o Best Fit como algoritmo de alocação de memória, e o algoritmo de escalonamento referente à Prioridade.

```
PROCESSOS TERMINADOS:
Fila de processos:
3,0,3,0,10
2,0,1,0,10
-5,-1,-1,-1,-1
Estatísticas Globais:

P#      AT      BT      CT      TAT     WT

P2      10      3       20      10      7
P3      10      4       24      14      10


Average Turnaround Time = 3.428571
Average Waiting Time = 2.428571

ESTATISTICAS FIT
NUMERO MÉDIO DE FRAGMENTOS EXTERNOS: 0.000000
TEMPO MÉDIO DE ALOCAÇÃO: 30.000000
PERCENTAGENS DE SOLICITAÇÕES NEGADAS: 0.00
```

GIT LAB (Link para o Projeto)

https://gitlab.com/so-projeto-id1/id-1-so-project-2019-2020

Conclusão

Este projeto não dividimos o trabalho mas sim fomos fazendo sempre em conjunto via MicrossoftTeams de maneira a estarmos sempre dentro do mesmo assunto.

A primeira parte do projeto deu-nos uma grande abrangência e uma melhor compreensão de todos este ciclo entre os processos, acerca dos algoritmos de escalonamento e o seu real funcionamento e também sobre a comutação de contexto.

Permite-nos realmente perceber como é que tudo acontece, mesmo muito pormenorizado, o que ajuda na compreensão de como os processos funcionam, como interagem com a memória e entre si e como são escalonados.

Já na segunda parte do projeto, conseguimos interiorizar melhor certos conceitos acerca dos algoritmos de alocação de memória e o seu real funcionamento, conceitos e aspetos importantes face á reserva e libertação de memória e também referente à fragmentação.

Conseguimos implementar e a funcionar corretamente nesta parte dois os algoritmos de escalonamento SJF e o algoritmo de Prioridade com a opção de serem preemptivos ou não preemptivos que não tinham sido pedidos na parte I.

Contudo concluímos que este projeto contribuiu e muito para ambos a nível de conhecimentos referentes à cadeira de Sistemas Operativos e que nos será útil no futuro.