Universidade da Beira Interior
Sistemas Operativos
Projeto 2º Semestre – 2020

Título: Simulador de Gestão de Processos
Escalonamento e Gestão de Memoria
Versão 1.0 23/3/2020

OBJETIVOS

O objetivo deste trabalho é o desenvolvimento de uma aplicação que simule o funcionamento das operações de gestão e execução de processos por um sistema operativo incluindo as operações de escalonamento do CPU, criação e terminação de processos e a comutação de contexto.

O trabalho será descrito dum modo geral para que haja uma grande variedade de interpretações e implementações possíveis!

Faz parte do seu trabalho inventar falhas na descrição do simulador e esclarecê-los no relatório do trabalho e apresentação.

DESCRIÇÃO : ESCALONAMENTO

O trabalho é dividido em duas partes. Na primeira parte descrita aqui neste enunciado criará um simulador de processos e implementará vários algoritmos de escalonamento.
Na segunda parte ampliará o simulador com funcionalidades de gestão de memoria.

Pode usar C, C++, Java ou Ocaml para desenvolver a sua aplicação.

- Deverá desenvolver o seu código duma forma modular.
- Cada elemento do grupo deve criar uma conta individual no gitlab/github. Um membro do grupo poderá criara o projeto inicial e partilhar com os outros membros do seu grupo e com os professores.
- Deverão usar um Gestor de Compilação (build tool): make para C/C++, Ant/Maven para Java etc

Será descrito aqui em baixo o esboço do simulador. O seu simulador poderá ser muito mais interessante incluindo por exemplo a simulação da chegada dinâmica de novos processos usando algum mecanismo aleatório etc. Deverá usar a sua imaginação !

Utilize o fórum de discussão da disciplina para colocar perguntas/dúvidas/comentários e trocar opiniões com colegas.

Simulação dum Processo

Cada processo no simulador consiste na execução dum programa que atualiza o valor duma **única variável** do tipo inteiro.

O estado (**ou contexto**) dum processo consiste no valor deste inteiro e o valor do seu contador do programa (PC) também será incluído no estado dum processo o número de identificação do processo (pid) o identificador do processo progenitor (ppid), a sua prioridade, prazo temporal etc

O processo gestor tem identificador terá o identificador 0

As instruções dum processo são executadas 1 por unidade do tempo de simulação

Existem 7 tipos de instrução

- M n - mudar o valor da variável para o valor n
- A n - adicionar o valor n à variável
- S n - subtrair
- B - Bloquear este processo
- T - Terminar este processo
- C n - Criar um novo processo (cópia do velho) - o processo filho executa logo a seguir esta instrução enquanto o pai executa n instruções a partir desta.
- L filename - Limpar o programa atual e substituir por filename (max 15 caracteres) Terá que carregar o novo programa em memória se for necessário.

```
M 100
A 19
A 20
S 12
A 1
A 4
C 2
L filho1
C 2
L filho2
T
```

```
M 200
A 19
T
```

```
M 300
S 12
A 5
T
```

Os ficheiros : progenitor.prg, filho1.prg e filho2.prg
Exemplo de programas com 11, 3 e 4 instruções respetivamente.

Repare que este programa vai criar dois novos programas através da instrução C seguido por L (fork e exec em linux!!) carregando por exemplo os ficheiros filho1.prg e filho2.prg

Nota: neste exemplo num dado momento podemos ter 3 ou mesmo 4 processos a correr concorrentemente

Memory Model

Os programas em execução devem ser guardados num único array "memory" (cada posição dum array guardará uma instrução) que represente a memória do sistema:

```
typedef struct { char ins; int n; char nome[15] } instruction;
intruction [1000] memory;
```

Exemplo usando o exemplo anterior :

Memory[0] = { 'M', 100 , Null}
Memory[1] = { 'A', 19, Null}
..
Memory [7] { 'L', 0, "filho1" }
...
Memory[11]={'M',200,Null}
Memory[12]={'A',19,Null}
Memory[13]={'T',0,Null}
Memory[14]={'M',300,Null}
Memory[15]={'S',12,Null}
Memory[16]={'A',5,Null}
Memory[17]={'T',0,Null}

Process Control Block

Cada programa em execução (processo) necessitará uma estrutura de dados para o controlar.

Por exemplo para guardar o nome do programa, o endereço da primeira instrução, o valor da "variável", o numero do processo (PID), o PPID, prioridade, Program Counter PC, estado, … etc etc

Exemplo para 50 programas cuja soma de instruções é menor ou igual a 1000

```
typedef struct { char nome, int  start; ….. } PCB
```

Exemplo usando o exemplo anterior :

PCB[0] = { "progenitor", 0 }
PCB[1] = { "filho1", 11} – o pai tinha 11 instruções
PCB[2] = { "filho2", 14 } – o filho 1 tinha 23 instruções etc

PCB[0] = ..exemplo..
program="progenitor" , start=0
PID = 1
PPID 0
PC = 3 (a proxima instrução a executar é S 12)
Prioridade = 4
Estado=2 (→blocked)
etc

O Gestor de Processos

Inicialmente será dado a esta função uma lista de programas a correr (ficheiro plan.txt) e os tempos quando terá que os criar (tempo de chegada) e inserir na fila de processos prontos a executar.

Esta função a seguir leia comandos dum ficheiro "control.txt" ou do stdin (para debugging)

E : O simulador executará um programa para um máximo de N unidades de tempo (o Time Quantum terá que ser definido globalmente)
I : Interrompa o processo em execução e bloquei-o
D : Execute o escalonador de longo prazo (que possa desbloquear processos)
R : Chamar a função de Reportagem para imprimir estatísticas
T : Terminação do Simulador e Imprimir Estatísticas Globais (turnaround etc)

Ao terminar o simulador deverá imprimir sempre as estatísticas globais.

Exemplo dum ficheiro plan.txt

```
progenitor.prg    0
progenitor.prg    20
filho1.org        30
```

Exemplos do ficheiro control.txt / ou stdin

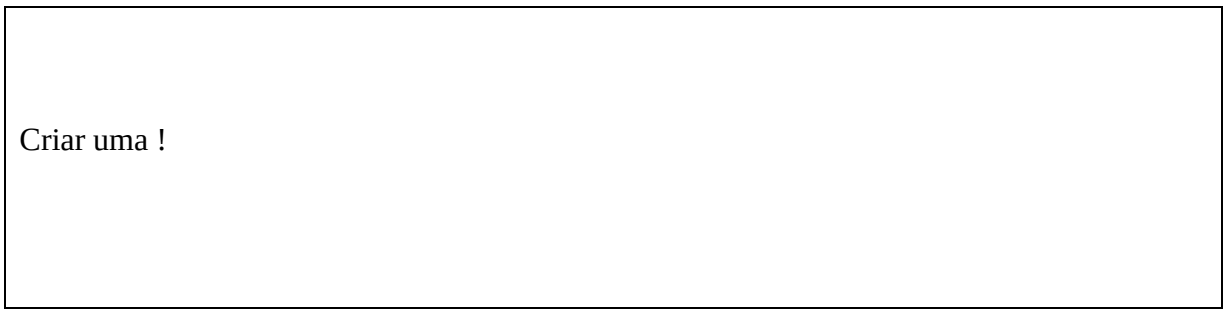
```
E
E
```

Exemplos do ficheiro control.txt /ou stdin

```
I
E
R
E
```

Também pode executar o escalonador de longo periodicamente, executar sempre, existem muitas possibilidades.

FLUXOGRAMA SIMPLIFICADO DO SIMULADOR



Estruturas de Dados da Função Gestor.

O gestor terá que manter as seguintes estruturas de dados :

- *Tempo*, CPU, *PcbTabela*, *Prontos*, *Bloqueados*, e *RunningState*.
    - *Tempo* : valor inteiro (variável global) inicializado para zero
    - *CPU: Program Counter (*variável global) do processo em execução
    - *PcbTabela* – Array/lista/fila com um entrada para cada processo em execução. Conterá informação necessária para executar um processo (PC etc)
    - *Prontos* – Fila de processos prontos a executar (fila ou fila de prioridade etc)
    - *Bloqueados* - Fila de processos bloqueados
    - *RunningState* O índice do PCBTabela do processo em execução. PID do processo em execução. PC do processo

Repare que a comutação do contexto envolve essencialmente a manipulação do PC - temos que guardar o valor atual do PC do processo a ser retirado do CPU e carregará o novo valor do PC depois de executar o escalonador do curto prazo.

Assume que : as operações do sistema operativo/simulador são instantâneas.

A Função "Execução"

Deverá "executar" um processo durante um time quantum de N unidades do tempo (menos que o processo termine/bloquei etc).

Função Do Escalonamento Do Curto Prazo

Deverá implementar várias políticas / algoritmos de escalonamento incluindo

FCFS	
Priority	o PCB vai precisar de saber a prioridade .. mais uma informação a adicionar ao ficheiro plan.txt
SJFS	repare que não há loops – sabemos quanto tempo vai durar cada processo

Podem implementar mais algoritmos, por exemplo para tempo real e processos periódicos.

- Rate Monotonic / EDF

NOTA IMPORTANTE : O escalonamento é **preemptivo** mas é relativamente trivial incluir uma opção para lidar com o caso de escalonamento ser não preemptivo.

Função Do Escalonamento Do Longo Prazo

Esta função deverá varrer a fila de processo bloqueados e muda da fila um ou mais dos processos bloqueados para a fila pronta ..

Um processo pode ser desbloqueado a partir dum processo aleatório. Poderá usara a função rand() para escolher uma função para desbloquear (Pode usar alguma distribuição probabilística (normal, poisson, exponenetial etc))

Função Do Reportagem.

A função do "Report" imprime o estado do sistema no stdout. Exemplo em Baixo

```
TEMPO ACTUAL: tempo

PROCESSO EM EXECUÇÃO:
pid, ppid, prioridade, valor da variavel, tempo do iniciação, Tempo do CPU usado etc etc

PROCESS0S BLOQUEADOS:
Fila dos processos
pid, ppid, prioridade, valor, tempo do iniciação, Tempo do CPU usado
...
pid, ppid, prioridade, valor, tempo do iniciação, Tempo do CPU usado

PROCESSOS PRONTOS A EXECUTAR
Fila dos processos
pid, ppid, prioridade, valor, tempo do iniciação, Tempo do CPU usado...
.....
pid, ppid, prioridade, valor, tempo do iniciação, Tempo do CPU usado...

PROCESSOS TERMINADOS
pid, ppid, prioridade, valor, tempo do iniciação, tempo do fim, Tempo do CPU usado
..
pid, ppid, prioridade, valor, tempo do iniciação, tempo do fim, Tempo do CPU usado
```

ASPETOS IMPORTANTES

- O Projeto tem que compilar e executar corretamente no sistema operativo POSIX Linux/MacOS e a partir da linha de comando.
- O trabalho consiste no código fonte, ficheiros de dados, Build Tool file (obrigatório) (Makefile ou Java Build File) e um relatório.
- O relatório deverá incluir um breve resumo e explicação do funcionamento do programa e os algoritmos implementados, exemplos da execução do programa e verificação com resultados teóricos. Deverá destacar o trabalho realizado pro cada elemento do grupo e incluir o url do projeto no gitlab/github.
- Não deve ser necessário incluir código no relatório apenas os protótipos das funções usadas.
- Prazos
  - Primeiro Milestone Dia 10 de Maio. Deve entregar um relatório com um resumo do estado do projeto. O simulador deve funcionar com pelo menos o algoritmo de esacalonamento FCFS implementado.
  - Segundo MileStone. 12 de Junho. Entregará (uma vez por grupo) o trabalho final no Moodle. Um ficheiro compactado (zip/tar) com o código fonte/makefile/dados e ficheiro (doc/pdf) do relatório.

INSCRIÇÃO DO GRUPO

Grupos de 2,3 ou 4 alunos : Inscrição usando este formulário

PROGRAMAÇÃO - HINTS

Muitos livros e sites da Internet disponibilizam código para implementar estruturas de dados como **filas** (FIFO) e **filas de prioridades** etc. **utilizam à vontade** .. desde que inclua as referências !

PARTE II MEMORIA (MAIS INFORMAÇÕES A SEGUIR…)

- Define-se um valor para a quantidade total de memoria e para as suas partições.
- A primeira linha de cada programa especificará o tamanho inicial de "memoria" necessária .. um valor qualquer independente de numero de instruções.
- Sendo assim - antes de colocar um programa em execução (estado ready) executaremos um algoritmo de Gestão de Memoria para saber se existe memoria suficiente.

GIT

Do WikiPedia

Git pronuncia-se /git/ (ou pronuncia-se /djit/ em inglês britânico) é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de software, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo. O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos.

- Videos : https://git-scm.com/documentation
- The simple guide for getting started with git : http://rogerdudler.github.io/git-guide/
- Git Quick Reference - http://jonas.nitro.dk/git/quick-reference.html
- Git the book (Em Português) https://git-scm.com/book/pt-br/v2

Tem que usar git – no momento da apresentação verificaremos as contribuições de cada elemento do grupo ! Se não usar e não fizeram nenehum commit é como não estivesse parte do grupo.

Devem adicionar (um d)os Professores ao seu projeto.

Usar de preferência **gitlab** (podem criar conta aqui https://gitlab.com/ ou para quem sabe ..montar e usar um servidor pessoal ou duma outra entidade)

As contas gitlab dos Professores a usar serão disponibilizados no Moodle

Qualquer divida ou esclarecimento .. usar o fórum da disciplina !

Não sub-estimar o esforço necessário para completar este projecto – a dificuldade principal é **começar** !

Agradecimientos

Este trabalho é inspirado dum trabalho proposto pelo Professor A.S.Tanenbaum. No entanto este trabalho é bastante diferente. Não usam soluções deste trabalho original que podem encontrar na Internet – tem de usar a definição descrita neste enunciado.