Sistema Computacional
Escalonador EDF

Gustavo Olegário
Thales Alexandre

October 2016

Índice

1 Introdução

Durante o segundo semestre de 2016, na disciplina INE 5412 - Sistemas Operacionais I, solicitou-se aos alunos que aos mesmos que realizassem uma implementação de um Sistema Computacional. O professor da disciplina disponibilizou vários temas para o projeto. A dupla deste trabalho resolveu por escolher o escalonador EDF. O escalonador implementado foi feito para funcionar em um simulador feito pelo próprio professor da disciplina. Na sequência, será apresentado maiores informações sobre o simulador, o escalonador EDF e os requisitos sendo cumpridos.

2 Simulador

Como já foi dito, o escalonador implementado foi feito para funcionar dentro de um simulador feito pelo professor da disciplina. O simulador simula um computador por completo, desde de seus componentes de mais baixo nível, como CPU, DMA e HDD até as abstrações de processos, threads e gerenciador de memória. O funcionamento base do simulador é emular um Sistema Operacional que roda uma aplicação simples, demonstrando o funcionamento correto do projeto. O simulador é configurável, no sentido de, para cada grupo específico ele pode ser ajustado para verificar se o projeto em questão está funcionando corretamente. Essa configuração é ajustada no arquivo "Trais.h". Todo o código do simulador está sendo fornecido junto com este relatório.

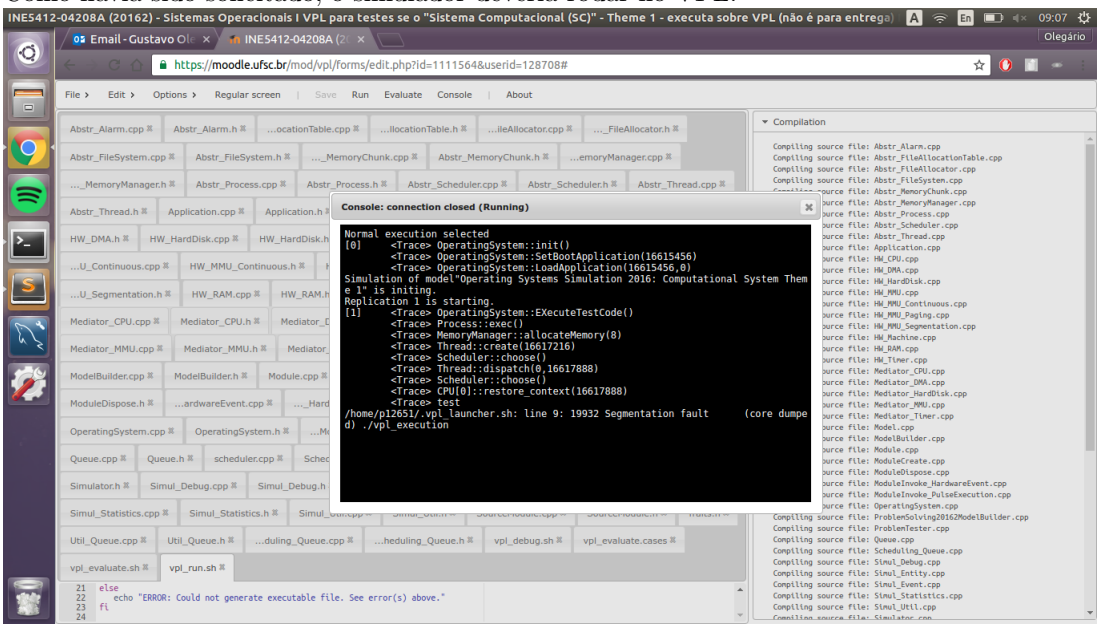
3 Escalonador EDF

O escalonador EDF (Earliest Deadline First) foi projetado em especial para sistemas de tempo real e é um escalonador do tipo preemptivo, ou seja ele tem permissão para remover a tarefa que está rodando sem que ela tenha terminado ou tenha sido bloqueada. Basicamente, como o seu próprio nome já sugere, o seu critério para escalonamento é o tempo da Deadline da thread. Deadline, é o tempo máximo em que tarefa deve estar pronta. O escalonador funciona da seguinte forma, o escalonador sempre escolherá a tarefa com a Deadline mais próxima. É importante lembrar que o escalonador mantém sempre a fila ordenada, pelo menos no caso usado no simulador.

4 Requisitos

4.1 O sistema deve executar no VPL do Moodle

Como havia sido solicitado, o simulador deveria rodar no VPL.



4.2 Coletar estatísticas sobre o escalonamento

Para a coleta de estatísticas, criou-se novos atributos nas classes "Thread" e "Process". Dentro da "Thread" temos o tempo de uso da CPU, tempo de espera e tempo bloqueada e cada uma com suas auxiliares para o cálculo do tempo.

```
double _cpuTime;
double _waitingTime;
double _blockedTime;
```

Quando a thread for então eliminada, todas essas suas estatísticas serão acumuladas no processo pai que possui os mesmos atributos. Sendo assim, quando o processo for finalizado, teremos todas as suas estatísticas. O cálculo é feito de forma que a thread sempre que passa por um evento de escalonamento (bloqueada, esperando, executando) ela guarda o horário em que o evento ocorreu e ao sair do estado faz a diferença pra saber quanto tempo ficou no estado.

```
// calcula quanto tempo ficou esperando (bloqueada)
i->_accountInfo._blockedTime += (Simulator::getInstance()->getTnow() - i->_accountInfo._blockedAux);
```

Ao final todos os processos tem suas estatísticas mostradas dentro da classe simulator.

```cpp
void Simulator::_showSimulationStatistics() {
    auto list_proc = Process::getProcessesList();
    for (auto i = getAllProc()->front(); i != getAllProc()->back(); ++i) {
        auto statistics = i->getCpuTime();
        std::cout << "Statistics from process " << i->getId() << ".\n";
        std::cout << "CPU time: " << statistics._CPUTime << ".\n";
        std::cout << "Waiting time: " << statistics._waitingTime << ".\n";
        std::cout << "Blocked time: " << statistics._blockedTime << ".\n";
    }
}
```

```
Waiting time: 466.
Blocked time: 0.
Statistics from process 0.
CPU time: 466.
Waiting time: 0.
```

4.3 Implementar testes unitários

Outro requisito foram os testes unitários. O VPL é capaz de atribuir nota a trabalhos se ele possuir testes. O arquivo de testes está junto ao código fonte do sistema com o nome de "vpl_evaluate.cases". Cada case do teste corresponde a um case da classe "OperatingSystem" no método "ExecuteTestCode()".

Scheduler.h × Scheduling_Queue.h × Scheduling_Que

```
case=1
input=1
output=/(.)+Process 1 created!(.)+/
grade reduction=10%

case=2
input=2
output=/(.)+Thread 3 created!(.)+/
grade reduction=10%

case=3
input=3
output=/(.)+Thread 5 executed yield!(.)+/
grade reduction=10%

case=4
input=4
output=/(.)+Thread 7 executed join!(.)+/
grade reduction=10%

case=5
input=5
output=/(.)+Process 5 executed exit!(.)+/
grade reduction=10%

case=6
input=6
output=/(.)+Thread 14 executed exit!(.)+/
grade reduction=10%

case=7
input=7
output=/(.)+Thread 14 executed sleep!(.)+/
grade reduction=10%
```

4.4 Implementar o algoritmo de escalonamento

O algoritmo EDF(Earliest Deadline First) sempre escalona primeiro a thread com a deadline mais próxima do horário atual, que ainda não tenha sido ultrapassado. Para implementar o EDF tivemos que criar um atributo "Deadline time" no "Account Information" da classe "Thread". Quando uma thread é criada ela tem a deadline de sua "Account Information" atribuida da seguinte forma:

```
_accountInfo._deadLineTime = Simulator::getInstance()->getTnow() + DEAD_LINE;
```

Onde a deadline será a soma entre o tempo atual do sistema (primeiro argumento da soma) e o tempo limite que a thread tem para realisar sua atividade (segundo argumento da soma). A deadline nada mais é do que um número aleatório gerado entre 0 a 100. Logo depois de ter sua deadline atribuída, a prioridade da thread é atribuida com o mesmo valor, bastando ao escalonador organizar a fila por prioridade com menor valor para que o escalonador sempre escolha a thread com a deadline mais baixa.