+---------------------------+

| EEL770 |

| PintOS |

| DOCUMENTION |

+----------------------------+

Grupo:

Anni Provietti <[anniprovietti@gmail.com](mailto:anniprovietti@gmail.com)>

Arthur Pezzin <[arthurton2005@poli.ufrj.br](mailto:arthurton2005@poli.ufrj.br)>

Preliminares:

Fontes e anotações

<https://web.stanford.edu/class/cs140/projects/pintos/pintos_6.html>

<https://github.com/arthurpezzinton/trabalho_SO>

JOIN

====

---- DATA STRUCTURES ----

Para o projeto 1, alteramos o arquivo timer.c adicionando um ponteiro para uma variável da estrutura *thread* que vai apontar para a thread atual; adicionamos uma variável do tipo *int64\_t* que vai receber o momento de “acordar” a thread atual em *ticks*, modificamos a função *timer\_sleep* para ser executada como o nosso raciocínio da questão e usamos a *timer\_interrupt* para servir como “contador”.

---- ALGORITHMS ----

*struct thread\* thread\_pointer;*

Aqui é onde criamos o ponteiro que vai servir como uma variáve auxiliar, apontando para a thread atual

*int64\_t wake\_tick;*

Aqui criamos a variável auxiliar do tipo int64\_t que receberá o valor em ticks do tempo que deve decorrer até “acordar” a thread

Na função *timer\_sleep* nossas alterações foram:

*timer\_sleep (int64\_t ticks)*

*{*

*int64\_t final\_tick = timer\_ticks () + ticks;*

Aqui é onde dizemos para a variável final\_ticks quantos ticks ela vai valer, que será o tempo necessário para a thread "acordar"

*thread\_pointer = thread\_current();*

Definimos que o ponteiro para a variável "thread" se refere a nossa thread atual

*ASSERT (intr\_get\_level () == INTR\_ON);*

*enum intr\_level old\_level = intr\_disable ();*

Para as Interrupções

*wake\_tick = final\_tick;*

Atribuo o valor dos ticks para acordar à variável do thread\_pointer à nossa variável auxiliar criada anteriormente

*thread\_block();*

Thread bloqueada

*intr\_set\_level (old\_level);*

Para Interrupções

*}*

Para o nosso “contador”, usamos a *timer\_interrupt*:

*static void*

*timer\_interrupt (struct intr\_frame \*args UNUSED)*

*{*

*ticks++;*

*thread\_tick ();*

Até aqui, é a função original, atuando como um contador de ticks.

*if(thread\_pointer){*

Aqui, nós verificamos se há alguma thread sendo apontada pelo ponteiro que criamos.

*if(wake\_tick < ticks){*

Se o *if* anterior for verdadeiro, verificamos se o tempo final de “acordar” a thread passado à nossa variável auxiliar já foi decorrido.

*thread\_unblock(thread\_pointer);*

Caso o *if* anterior seja verdadeiro, aqui é o momento de desbloquear a thread anterior, ou seja, “acordando-a” depois do tempo em ticks decorrido corretamente.

*}*

*}*

*}*

---- RATIONALE ----

Parar o projeto 1, nós pensamos que poderíamos criar um ponteiro para a nossa thread e uma variável que guardasse o tempo em que essa thread deveria acordar e depois a retornaríamos para atividade quando o tempo correto fosse decorrido.

Para isso, seguindo orientações do próprio professor, pensamos em utilizar uma outra função que fosse executada periodicamente para ser o nosso “contador”. Com buscas na internet e vendo a estrutura da própria função, observamos que a *timer\_interrupt* poderia ser usada como nosso contador e implementamos a contagem nela.

Na função *timer\_sleep*, nós atribuímos ao ponteiro o endereço da nossa thread atual e o tempo final à variável auxiliar *wake\_tick*. Habilitamos as interrupções e bloqueamos a thread. Depois disso, na função *timer\_interrupt*, a cada vez que a função é chamada, fazemos a verificação se há alguma thread sendo apontada pelo ponteiro auxiliar que criamos e utilizamos o contador da própria função para ver se o tempo decorrido era maior ou igual ao tempo final para reativar a thread.

Nos nossos testes, tivemos problema relacionado com TIMEOUT nos testes. Procuramos algumas soluções na internet, mas foram ineficientes.

Em relação às desvantagens do nosso projeto, podemos apontar que a nossa variável auxiliar só pode receber e apontar para uma thread por vez, sendo assim, se fossem requeridas mais threads, somente o endereço e tempo para acordar da última thread seriam registrados. Assim, para uma melhor implementação, poderia ter sido utilizada uma lista que guardasse as threads e seus devidos tempos de acordar, mas não tivemos condições de pensar e implementar isso no trabalho.

Um esboço dessa solução proposta acima seria criar uma matriz de ponteiros para threads e uma matriz de *ticks int64\_t*, assim como um contador de quantas threads poderiam existir na lista, e assim, na *timer\_sleep* a inserção seria sequencial na matriz, incrementando o contador, e a cada utilizaçºão da função *timer\_interrupt*, seria feita a verificação de toda a matriz até o contador de quantas threads há nela, e se uma thread tivesse seu tempo decorrido, ela seria desbloqueada, removida da matriz auxiliar criada e a matriz seria reorganizada para inserção de novas threads.

Além deste documento, nosso progresso com o trabalho pode ser consultado no projeto do github exposto na referência acima.