UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

SISTEMAS OPERACIONAIS I

TRABALHO PRÁTICO I

IMPLEMENTAÇÃO DE BIBLIOTECA DE THREADS

Prof. Alexandre da Silva Carissimi

Alunos:
Eduardo Sachser
Felipe Rambo Thozeski

Porto Alegre, maio de 2015

A Biblioteca mThread

mcreate: Cria uma nova thread e retorna o tid dela. Funcionando corretamente.
myield: Faz com que a thread atual entre novamente para a fila de espera. Cede voluntariamente a CPU. Funcionando corretamente.
mwait: A thread atual deve esperar para que a thread de tid passado como parâmetro execute, e ficar bloqueada até que isto aconteça. Funcionando corretamente.
mmutex_init: Inicializa o mutex com a lista de threads bloqueadas vazia. Funcionando corretamente.
mlock: Se a flag estiver desativada, é ativada. Caso já esteja, coloca a thread atual em estado bloqueado até que a flag seja desativada. Funcionando corretamente.
munlock: Desativa a flag do mutex passado como parâmetro. Além disso, coloca em apto todas as threads bloqueadas. Funcionando corretamente.

Testes Realizados

teste_create_yield.c

Testa o funcionamento das funções *mcreate* e *myield*. Cria duas threads, dá um yield para que as threads sejam executadas. Na primeira thread, faz yield para rodar a segunda thread. Após ela rodar, vai para a main, que novamente faz yield, para que a thread 1 termine sua execução. Por fim, finaliza a main. A saída da execução foi a seguinte:

```
Iniciando operacao.
Criacao da thread PID = 1
Criacao da thread PID = 2
Cedendo a CPU.

Funcao F1 em execucao.

Funcao F2 em execucao.
Funcao F2 encerrando.

Main cedendo a CPU novamente.

Funcao F1 encerrando.

Concluido main.
```

teste_wait.c

Cria 6 threads, depois faz a main esperar pelo fim da thread 1. A thread 1, por sua vez, por ter prioridade menor, deixa outras threads executarem. Por fim, todas executam, testando a funcionalidade do *mwait*. Abaixo, a execução do programa.

```
Iniciando operacao.
Criacao da thread PID = 1
Criacao da thread PID = 2
Criacao da thread PID = 3
Criacao da thread PID = 4
Criacao da thread PID = 5
Criacao da thread PID = 6
ThreadMain aguardando Thread2.

Thread4 em execucao.

Thread4 aguardando Thread2.

Thread4 encerrando.

Thread6 em execucao.

Thread6 aguardando Thread5.

Thread2 em execucao. 2

Thread2 encerrando.

ThreadMain aguardando Thread1.

Thread5 em execucao.

Thread5 encerrando.

Thread6 encerrando.

Thread1 em execucao.

Thread1 cedendo CPU.
```

```
Thread3 em execucao.

Thread3 aguardando Thread4.

Thread3 encerrando.

Thread1 encerrando.

ThreadMain encerrando.
```

teste_mmutex_init.c

Testa criação do mutex, para ver se seus dados estão criados corretamente. A seguir a execução.

```
Iniciando operacao.
Flag: 0
Inicio Lista: (nil)
Fim Lista: (nil)
Encerrando main.
```

teste_mlock_munlock.c

Testa se, quando uma thread faz o bloqueio, a outra cede o funcionamento para que ela execute. A seguir o teste.

```
Iniciando operacao.
Entrando na f1
Area protegida f1
Fazendo yield para testar
Entrando na f2
Saindo da f1
Area protegida f2
Saindo da f2
Encerrando main.
```

Dificuldades Encontradas

As principais dificuldades encontradas para execução do trabalho foram no uso dos contextos, pois estes tornam um código aparentemente linear, em algo que pode

gerar loops, erros, segmentation faults, etc… Depois de conseguir dominar um pouco melhor as funções para criação do contexto, tudo se torna mais tranquilo. Isto, é claro, se trabalhar com ponteiros é algo tranquilo. Não tivemos muitas dificuldades com ponteiros. A lógica das funções não é muito complexa, e, por isso, mesmo tendo alguns erros durante a implementação, estes (erros de lógica) foraam relativamente fáceis de serem contornados.

A solução básica para todas as dificuldades encontradas neste trabalho foram pesquisa, testes e mais testes, e pensamento sobre o problema. Em alguns casos, também trocamos ideias com nossos colegas sobre como encontrar determinadas soluções.