UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS I (2015/1)
Prof. Dr. Alexandre da Silva Carissimi
Graduandos:
Luis P. Silvestrin (228528)
Marcos H. Backes (228483)

Trabalho Prático I:

Biblioteca Microthread

Porto Alegre, 2 de Maio de 2015

1 Introdução

No primeiro trabalho prático da disciplina de Sistemas Operacionais I foi desenvolvida uma biblioteca que implementa *threads* a nível de usuário, a *mthread*. Este documento descreve o funcionamento das primitivas que foram implementadas, bem como os testes realizados para cada uma delas.

2 Funções

Para o funcionamento correto da biblioteca *mthread*, foram desenvolvidas cinco funções que permitem a criação, manipulação e controle de *threads*. São elas: *mcreate*, *myield*, *mwait*, *mlock* e *munlock*. Todas as funções foram implementadas com sucesso.

2.1 mcreate

A função *mcreate* cria uma nova thread, recebendo por parâmetro a sua prioridade, a função que a thread irá executar e seus argumentos.

Essa função verifica primeiro se está sendo chamada pela primeira vez durante a execução. Se sim, ela inicializa o contexto do escalonador, do despachante e da *main thread*. Essa última é então adicionada à fila de threads aptas de primeira prioridade. Então, o *mcreate* cria um novo contexto com a função e argumentos recebidos por parâmetro para a thread que deve ser criada. Feito isso, essa thread é colocada na fila de aptos de sua respectiva prioridade e o contexto é chaveado para o escalonador. A função foi implementada com sucesso.

2.2 myield

A função *myield* retira a thread que está executando e a insere na fila de aptos. Depois, ela salva o contexto atual da thread e chaveia para o contexto do escalonador utilizando o *swapcontext*. A função foi implementada com sucesso.

2.3 mwait

A função *mwait* primeiramente testa se a thread a ser esperada existe ou se já possui outras threads esperando. Se algum desses casos ocorrer, a função retorna -1". Caso contrário, ela passa a thread que está executando para o estado "bloqueado", inserindo-a na fila de bloqueados juntamente com o id da thread que está sendo esperada. Finalmente, o programa chaveia para o contexto do escalonador. A função foi implementada com sucesso.

2.4 mlock

A função *mlock* recebe um *mutex* e verifica se o mesmo está livre. Se estiver, o estado do *mutex* muda para "bloqueado"e o programa segue seu fluxo normalmente. Caso contrário, a thread que está executando é transferida para a fila de bloqueados do *mutex* e o programa muda para o contexto do escalonador. A função foi implementada com sucesso.

2.5 munlock

Essa função libera a primeira thread bloqueada na fila de um *mutex*, passando-a para a fila de "aptos"e mudando o seu estado. Se, ao remover a thread a fila do *mutex* fique vazia, o estado do *mutex* muda para "livre". A função foi implementada com sucesso.

3 Testes

3.1 Teste mcreate

Neste teste, primeiro, é verificado se a função é capaz de tratar entradas inválidas e, por isso, ela é chamada duaz vezes com prioridade -1 e 3. Espera-se que a função retorne -1, mostrando que não foi possível criar threads com tais prioridades. Depois disso, são criadas 6 threads (2 threads para cada prioridade) para executar a mesma função que imprime o tid e prioridade da thread executada e a thread main aguarda o término delas. Espera-se que as 2 threads de prioridade 0 sejam executadas primeiro, seguidas das threads de prioridade 1 e por último as threads de prioridade 2.

3.2 Teste myield

Nesse teste são criadas cinco threads com prioridades variadas. A prioridade de cada thread equivale ao módulo 3 de sua ordem de criação, que varia de zero a cinco. Cada thread criada executa um *myield*, fazendo com que as threads de prioridades iguais se alternem até acabar. Assim, as threads de prioridade maior terminam primeiro e as de menor prioridade terminam por último. Antes de criar as threads, a *main* executa um *myield*, que retorna -1, pois o escalonador ainda não foi inicializado. Depois que as threads terminaram, o programa executa *myield* outra vez, e o retorno será 0, pois o escalonador já está inicializado. O programa não recebe nenhum parâmetro e imprime na tela a operação realizada por cada thread (início, *myield* e término).

3.3 Teste mwait

Esse programa, primeiramente, tenta excutar o *mwait* passando o id de uma thread inexistente, retornando -1. Em seguida, ele cria 5 threads de prioridades variadas, e cada uma dela espera outra de prioridade mais baixa. A sexta thread tem a prioridade mais baixa possível e executa um *myield*, mas logo volta a executar, pois todas as threads outras estão bloqueadas. Quando ela termina, as outras threads são liberadas

em sequência e a última thread a executar é sempre a primeira a ser criada. Após o término das threads, a *main* tenta esperar ela mesma, retornando -1.

3.4 Teste mmutex_init

Primeiro é testado se a função é capaz de tratar entradas inválidas passando como argumento o ponteiro NULL. É esperado que a função retorne -1, pois não foi possível inicializar tal ponteiro. Depois disso, tenta-se inicializar um ponteiro válido e o resultado esperado é que a função retorne 0 e que todos os atributos do mutex estejam zerados.

3.5 Teste mlock

Neste programa, primeiro, é testado chamar a função mlock antes de inicializar o escalonador. Espera-se que a função retorne -1. Depois disso, são criadas 3 threads de mesma prioridade para executar uma função que contém uma região crítica (com um myield proposital no meio) e a thread main aguarda por elas. É esperado que a primeira thread entre na região crítica, execute *myield* e que as outras duas threads sejem bloqueadas ao tentar entrar na região crítica. Depois disso, a primeira thread libera o mutex e termina de executar. Espera-se que a segunda thread entre no mutex e termine de executar e só depois a terceira thread será executada.

3.6 Teste munlock

Esse programa tenta chamar a função *munlock* antes do escalonador iniciar, o que retorna -1. Em seguida, são criadas 3 threads que compartilham uma seção crítica. Elas utilizam a função *munlock* quando deixam a seção crítica, liberando a primeira da fila, caso ela exista. No fim, o programa tenta executar um munlock com o mutex livre e o retorno é -1.

3.7 Teste primos

Programa que testa todas as funcionalidades da biblioteca. O programa recebe como entrada um numero inteiro *n* e imprime todos os números primos no intervalo [0, n). Para isso, o programa cria *n* threads. Cada thread incrementa um contador que está numa região crítica e, ainda na região crítica, cede a execução à outra thread. Depois de incrementado o contador, a thread imprime o valor do contador na tela se esse valor for um número primo.

4 Dificuldades encontradas

Houve uma dificuldade inicial de compreender como funciona o escalonador e quais deveriam ser os contextos criados. Houve também uma leve dificuldade em entender o papel da thread main. Uma vez entendios esses conceitos, a implementação da biblioteca foi simples e direta. Além disso, houve alguns

bugs no programa que foram um pouco difíceis de encontrar devido às trocas de contexto que tornam a depuração do programa mais complicada.