Relatório de implementação da biblioteca uthread.

INF01142 - Sistemas Operacionais.

Felipe Soares F. Paula – 181790
Rodrigo Freitas Leite – 160568

Plataformas de Desenvolvimento.

O programa foi desenvolvido em duas plataformas (não-virtualizadas) :

- Plataforma 1
  AMD Athlon(tm) II X2 250 Processor, dual core, sem suporte à HT.
  Distruibuição: Linux Mint 14 Nadia
  Linux version 3.5.0-17-generic

- Plataforma 2
  Intel(R) Core(TM) i3 CPU 530 @ 2.93GHz, dual core, com suporte à HT.
  Distruibuição: Ubuntu 12.04.2 LTS
  Linux version 3.5.0-28-generic
-

Ambos usando *gcc version 4.7.2*

Testes

test_max_thread.c

Testa o número máximo de threads especificado. Tenta criar 129 threads, entretanto deve falhar na última, isto é, a thread número 129 não será inserida porque a fila de aptos já contém as 128 threads (número máximo)

test_mutex.c

Testa o compartilhamento de variáveis globais protegidas por mutex. Usamos vários yields para forçar o mutex negar o acesso à uma sessão crítica e analisamos no terminal o comportamento apresentado.

mandel.c

Cria uma imagem de uma fractal a partir do conjunto de Mandelbrot. Testa se a nossa biblioteca reproduz o comportamento da pthreads.

test_function_thread.c

Verifica o funciomento da *uthread_yield* e *uthread_join*. Também é interessante porque é parecido com o exemplo da especificação do trabalho.

Funcionamento das funções da biblioteca uthread.

uthread_create

Primeiramente a nova thread e o seu contexto são alocados. Em seguida, as estruturas *u_context* são inicializadas. Nessa etapa colocamos o *uc_link* apontando para o contexto do escalonador, de maneira que quando a função alocada (pelo *makecontext*) no contexto da thread acabar, o fluxo do programa vai para o ponto especificado dentro do escalonador.

uthread_yield

Caso a thread corrente não estiver em estado de yield, o escalonador coloca essa thread no fim da fila de aptos, muda o seu estado para yield e manda o escalonador pegar a próxima thread. Quando essa thread em yield for escalonada novamente, ela vai voltar para o início da função yield e vai ter seu estado normalizado e a thread vai continuar.

uthread_join

É perguntando se a id da thread que está sendo esperada está na fila de aptos ou na fila de bloqueados. Caso esteja, é alocada uma entrada na tabela que guarda as ids das threads bloqueantes e bloqueadas, além disso, a thread recebe o estado de join. Em seguida, a thread atual é colocada na lista na bloqueados e o escalonador é chamado (pois está esperando a thread cujo id foi passado acabar). Quando a thread voltar a rodar, e a thread bloqueante não estiver na lista de aptos ou bloqueado, ela inicia no contexto do começo da função e dessa vez vai ter o estado join, por isso, o estado vai receber a flag de normal e a função termina.

uthread_lock

Quando o mutex estiver aberto, isto é, a flag estiver desligada, a variável indicadora do estado do mutex recebe a informação de fechar e o programa continua normalmente, pois a thread que chamou o lock vai fazer as modificações na área protegida. Caso a flag estiver como trancada, a thread que tentou entrar na sessão crítica vai ser colocada na fila do mutex e na fila de bloqueados , em seguida, o escalonador é chamado.

uthread_unlock

Quando o mutex estiver com a flag com valor de 1, é tirada uma thread da fila do mutex e da fila de bloqueados. Depois disso, essa thread é colocada na fila de aptos do escalonador e o programa continua normalmente.

Principais dificuldades.

Aconteceram muitos erros de segmentação e trabalhamos bastante para debugá-los. Além disso, o próprio problema proposto é bastante emanharado, então é fácil de implementar códigos compiláveis mas que não tem o comportamento previsto.

Um dos maiores problemas foi que na função uthread_join, era passado como parametro não uma thread, mas um id de thread. Isso nos dificultou porque não tinhamos um referência direta à thread. Nossas estrutura de dados é uma fila genérica logo, não podíamos comparar por valores, apenas por referência. Uma solução foi criar uma estrutura auxiliar (*table.h)* que guardava os ids das threads bloqueadas e bloqueantes.