INF01142: Sistemas Operacionais I N

Trabalho #1

24 de Abril de 2014

Fernando Bombardelli da Silva(218324), William Bombardelli da Silva(218324)

Questão 2

- Plataforma 1:
  - Processador:
    - Intel Core i3 @2,30 GHz
    - 2 cores com suporte a HT (4 thread cores)
  - Sistema Operacional:
    - GNU/Linux CentOS 6.5
    - Kernel Linux 2.6.32 (x86_64)
  - Compilador:
    - GCC 4.4.7
  - Ambiente não virtualizado
- Plataforma 2:
  - Processador:
    - Intel Core i3 @2,30 GHz
    - 2 cores com suporte a HT (4 thread cores)
  - Sistema Operacional:
    - GNU/Linux Ubuntu 12.04
    - Kernel Linux 3.2.0 (x86_64)
  - Compilador:
    - GCC 4.6.3
  - Ambiente não virtualizado

Questão 3

Função	**Correção**
mcreate	Correto
myield	Correto
mjoin	Correto
mmutex_init	Correto
mlock	Correto
munlock	Correto

Questão 4

ListElem

Estrutura utilizada pelo módulo de lista encadeada para armazenar o dado da lista e controlar o encadeamento.

```
typedef struct SListElem ListElem;

struct SListElem{
 void *e;
 ListElem *prev;
 ListElem *next;
};
```

List

Estrutura utilizada como descritor da lista encadeada, contém o apontador de início e fim da lista.

```
typedef struct SListDesc{
 ListElem *begin;
 ListElem *end;
} List;
// Operacoes:
List* newList();
void freeList(List* listDescriber);
void listCheckRep(List* listDescriber);
void listAdd(List* listDescriber, void* e, int (*comparator) (void*, void*));
void listAppend(List* listDescriber, void* e);
void* listGet(List* listDescriber, void* e, int (*comparator) (void*, void*));
void listRemove(List* listDescriber, void* e, int(*comparator)(void*,void*));
```

OrderedQueue

O tipo *OrdereQueue* é uma estrutura de dados criada para controlar a fila de *threads* aptas a rodar.
A operação *Enqueue* insere um elemento na fila, porém sua posição depende do parâmetro *comparator* informado, que vai determinar por comparação com os outros nós qual a posição do elemento entrante. Esta operação permite o escalonamento das *threads* pela política Shortest Process Next.
A operação *Dequeue* remove o primeiro elemento da lista.
De fato o descritor é idêntico ao descritor da lista encadeada.

```
typedef List OrderedQueue;
// Operacoes:
OrderedQueue* newOrderedQueue();
void freeOrderedQueue(OrderedQueue* queue);
boolean orderedQueueEmpty(OrderedQueue* queue);
void orderedQueueEnqueue(OrderedQueue* queue, void* e, int (*comparator) (void
    *, void*));
void* orderedQueueDequeue(OrderedQueue* queue);
```

Questão 4 continuação na próxima página...

mmutex_t

Estrutura da variável de controle mutex.
O campo *flag* é uma enumeração que pode assumir o valor *Locked* ou *Free* indicando o estado da seção crítica que a variável controla.
O campo *ownerThread* contém o ID da *thread* que detém o *lock* da variável. Esse campo é utilizado para evitar que uma *thread* diferente da que tem o *lock* tente liberar a seção crítica.
O campo *waitingQueue* mantém a fila dos IDs das *threads* esperando pela liberação do *lock*.

```
typedef enum {Locked, Free} MutexFlag;

typedef struct Smmutex_t{
    MutexFlag flag;
    int ownerThread;
    OrderedQueue* waitingQueue;
} mmutex_t;
```

Questão 5 - myield

Momento	Chamada de Sistema	Descrição
T1	getcontext	Salva o contexto da *thread main*. É executada apenas uma vez durante toda a execução do programa, com o intuito de criar as estruturas necessárias para realizar o escalonamento da *thread main* juntamente com as demais.
T2	clock_gettime	Consulta o *timer* do sistema para calcular o tempo executado pela *thread* corrente.
T3	getcontext	Salva o contexto da *thread* corrente na sua estrutura TCB. Quando esta for chamada novamente, sua execução retornará a esse ponto. Note que neste caso o escalonamento não será executado em função do controle da variável global *yielding*.
T4	clock_gettime	Salva o tempo atual em um campo do TCB da *thread* que será posta em execução pelo escalonador. Esse tempo será usado posteriormente para definir o tempo executado pela *thread*.
T5	swapcontext	Troca o contexto atual pelo contexto da *thread* selecionada para executar. A partir desse ponto a próxima *thread* entrará em execução.

Questão 6 - mjoin

Momento	Chamada de Sistema	Descrição
T1	getcontext	Salva o contexto da *thread main*. É executada apenas uma vez durante toda a execução do programa, com o intuito de criar as estruturas necessárias para realizar o escalonamento da *thread main* juntamente com as demais.
T2	clock_gettime	Consulta o *timer* do sistema para calcular o tempo executado pela *thread* corrente.
T3	getcontext	Salva o contexto da *thread* corrente na sua estrutura TCB. Quando esta for chamada novamente (após o fim da *thread* a ser esperada), sua execução retornará a esse ponto. Note que neste caso o escalonamento não será executado em função do controle da variável global *yielding*.
T4	clock_gettime	Salva o tempo atual em um campo do TCB da *thread* que será posta em execução pelo escalonador. Esse tempo será usado posteriormente para definir o tempo executado pela *thread*.
T5	swapcontext	Troca o contexto atual pelo contexto da *thread* selecionada para executar. A partir desse ponto a próxima *thread* entrará em execução.

Note que caso uma *thread* execute *mjoin* para esperar por uma *thread* que ja tenha sido encerrada, *mjoin* retorna código de erro -1, seguindo especificação. Note ainda que este é o mesmo comportamento do caso de o id recebido por parâmetro em *mjoin* não seja válido (não exista *thread* com tal id).

Questão 7 - mlock

Momento	Chamada de Sistema	Descrição
T1	getcontext	Salva o contexto da *thread main*. É executada apenas uma vez durante toda a execução do programa, com o intuito de criar as estruturas necessárias para realizar o escalonamento da *thread main* juntamente com as demais.
T2	getcontext	Salva o contexto da *thread* corrente na sua estrutura TCB. Quando esta for chamada novamente sua execução retornará a esse ponto. Note que neste caso o escalonamento não será executado em função do controle da variável global *yielding*.
T3	clock_gettime	Salva o tempo atual em um campo do TCB da *thread* que será posta em execução pelo escalonador. Esse tempo será usado posteriormente para definir o tempo executado pela *thread*.
T4	swapcontext	Troca o contexto atual pelo contexto da *thread* selecionada para executar. A partir desse ponto a próxima *thread* entrará em execução.

Note que os passos dos tempos T2, T3 e T4 executam enquanto a variável *mutex* em questão não estiver livre. Ou seja, se na chamada para *mlock* o *mutex* estiver livre, as chamadas de sistema nos tempos T2, T3 e T4 não serão executadas.

Questão 8 - munlock

Momento	Chamada de Sistema	Descrição
T1	getcontext	Salva o contexto da *thread main*. É executada apenas uma vez durante toda a execução do programa, com o intuito de criar as estruturas necessárias para realizar o escalonamento da *thread main* juntamente com as demais.
T2	clock_gettime	Consulta o *timer* do sistema para calcular o tempo executado pela *thread* corrente. Após isto, muda o estado de uma *thread* que estava esperando pelo *unlock* para apto.

Note que o passo do tempo T2 executa apenas se houver alguma *thread* na fila dos bloqueados por este lock.

Questão 9

Os testes da biblioteca seguiram os seguintes passos:

1. Testes unitários nos módulos de lista encadeada e de fila da biblioteca.
2. Debug passo a passo para funcionalidades essenciais das funções oferecidas pela biblioteca e das funções do escalonador. Nesse passo foi utilizado o GDB integrado à IDE NetBeans.
3. Testes unitários das funções oferecidas pela bilbioteca.
4. Testes de sistema utilizando a biblioteca *libmmthread.a* finalizada juntamente com programas chamadores que exploram diversas funcionalidades.

Questão 10

Dificuldade	**Descrição**	**Solução Utilizada**
Análise do problema	A análise foi o ponto que exigiu o maior esforço de raciocínio. Uma boa análise e planejamento implicariam certamente uma boa execução do desenvolvimento.	Para facilitar a análise foi utilizado diagramas de classe UML (para descrever os módulos) e diagramas de sequência (para descrever as chamadas de funções internas à biblioteca). A saber, a ferramenta utilizada foi o Astah Community.
Chamada de sistema	Uma dificuldade bem específica encontrada foi a utilização das chamadas de sistema que facilitassem as trocas de contexto. Primeiramente utilizou-se *setcontext* para carregar um novo contexto, porém em algumas circunstâncias ocorria uma exceção de ponto flutuante dentro da função *setcontext*. Mais detalhes no Manual da Intel página 369.	Utilizar *swapcontext* para a troca de contexto.
Depuração	Erros inesperados naturalmente ocorreram, dado que é muito difícil prever todas as situações no momento da análise.	Para depurar o código e resolver tais erros foi utilizado o GDB integrado à IDE Netbeans.
Testes	Outra dificuldade encontrada foi desenvolver casos de testes capazes de comprovar a correção e a completude, ou seja, a partir da especifiação ela faz o que é proposto de meneira correta.	Tentou-se projetar casos de testes de maneira a alcançar um maior *coverage* de código, e mostrar que não há contradição entra a especificação e a implementação.