Trabalho 1 INE5611

Alexis Armin Huf, Estagiário de Docência

12 de abril de 2017

1 Introdução

Imagine que você foi contratado pela Pedrólio Inc., uma proeminente petroleira Nigeriana. Essa petrolífera desenvolveu uma técnica revolucionária de transformar pedras em petróleo. Os detalhes são altamente confidenciais e a Pedrólio não pode compartilhar, mas o processo de produção do petróleo pode ser ilustrado pelo algorítimo da Figura 1. Na linha 1, uma pedra é retirada do Pedromágico®, e na linha 2 uma UPP (Unidade de processamento de pedras) transforma a pedra em petróleo.

```
1    rock_t rock = pd_read();
2    oil_t oil = pd_process(rock);
3    pd_deliver(oil);
4    printf("Profited %.2f Monetary Units.\n", pd_price(oil));
```

Figura 1 – Processo básico de produção de petróleo a partir de pedras

A planta de Cafundópolis é totalmente automatizada, e todos os equipamentos são controlados pela API da Pedrólio. A planta possui 8 UPPs e apenas um Pedromágico. O Pedromágico só produz uma pedra de cada vez, e a ação de obter uma pedra do Pedromágico é complexa. Portanto, pode haver apenas uma execução da rotina `pd_read` em um dado momento. As UPPs também apresentam essa limitação, mas em um dado momento é possível ter quantas execuções de `pd_process()` quanto há UPPs disponíveis. E da mesma maneira, o oleoduto saindo da planta (`pd_deliver()` só pode receber petróleo de uma UPP por vez. **É de suma importância que essas restrições sejam respeitadas, caso contrário haverá nefastas consequências!**

Apesar do grande investimento na planta de Cafundópolis, a Pedrólio está decepcionada com o retorno do investimento. Há 8 UPPs, mas o petróleo produzido corresponde a capacidade de uma única UPP. Sua tarefa é fazer a planta atingir a totalidade do seu potencial. Para melhor entender o problema, a Pedrólio compartilhou alguns detalhes da implementação atual (Seção 2). As instruções específicas são fornecidas na Seção 3.

2 Implementação atual

O código disponibilizado junto com esse documento contém os seguintes arquivos:

- `main.c` Ponto de entrada do programa de avaliação;
- `pedrolio.h` API da Pedrólio permite usar o Pedromágico e as UPPs;
- `module.h` Interface do módulo. Um módulo deve implementar as funções declaradas nesse *header*;
- `mod_control`

```c
extern void  mod_setup();
extern char* mod_name();
extern void  mod_shutdown();
```

Figura 2 – Visão simplificada de `module.h`

– `mod_control.c` Implementação atual do módulo de controle

O controle da planta é realizado por um módulo, atualmente `mod_control`, e seu objetivo é desenvolver um módulo mais eficiente (cf. Seção 3). O arquivo `module.h` funciona como uma interface Java, com o grande diferencial que o *binding* dos métodos (`mod_*`) é determinado no momento da compilação (pode haver apenas uma implementação da interface por binário compilado). Trocar `mod_control` equivale a compilar outro arquivo no lugar de `mod_control.c`.

A Linguagem C não possui objetos como Java, então para simplificar a implementação e evitar o uso de ponteiros, os projetistas da Pedrólio assumem que haverá apenas uma instância do módulo de controle, o que viabiliza o uso de variáveis globais.

Os três métodos do módulo são descritos em `module.h` (Figura 2) e podem ser sumarizadas como segue:

- `mod_setup` Inicializa o módulo, alocando os recursos necessários e iniciando o processamento. Os recursos são alocados em variáveis globais declaradas no arquivo de implementação do módulo;
- `mod_name` Retorna o nome do módulo;
- `mod_shutdown` Para o processamento de pedras de maneira ordenada e libera todos os recursos alocados em `mod_setup`.

A Figura 3 mostra a implementação de `mod_setup`. Nas linhas 1-3 são declaradas as variáveis que controlam o estado de `mod_control`. Apesar do escopo dessas variáveis ser global, elas são acessadas apenas de `mod_control.c`. Nas linhas 6-7 `mod_setup` verifica se o módulo não está sendo inicializado mais de uma vez (lembre-se, assume-se apenas instância do módulo). A *flag* `cancelled` é inicializada com zero, pois o cancelamento só ocorrerá em `mod_shutdown`.

```c
pthread_t thread;
int setup = 0;
int cancelled;

mod_t mod_setup() {
   assert(setup == 0);
   setup = 1;
   cancelled = 0;
   pthread_create(&thread, NULL, &worker, NULL);
}
```

Figura 3 – Implementação de `mod_setup` em `mod_control`.

O programa teste (`main.c`) assume que `mod_setup` não bloqueia processando pedras. Logo, a implementação da Figura 3 cria uma *thread* na linha 9 para processar as pedras. Note que o objeto `pthread_t` resultante de `pthread_create` é salvo em `thread`, para que a *thread* possa ser finalizada em `mod_shutdown`.

A Figura 4 mostra a rotina executada pela *thread* criada na Figura 3. O procedimento descrito na Figura 1 é eternamente repetido, até que um cancelamento seja solicitado. Note que essa é a única

```c
static void* worker(void* ignored) {
   while (cancelled == 0) {
      rock_t rock = pd_read();
      oil_t oil = pd_process(rock);
      pd_deliver(oil);
   }
   return NULL;
}
```

Figura 4 – Rotina `worker`

thread, e que `pd_read` será executado apenas por essa thread, assim como `pd_process` e `pd_deliver`, satisfazendo as restrições mencionadas na Seção 1.

```c
void mod_shutdown() {
   assert(setup == 1);
   cancelled = 1;
   pthread_join(thread, NULL);
}
```

Figura 5 – Rotina `mod_shutdown`

A interrupção do processamento ocorre como mostrado na Figura 5, atribuindo `1` à variável `cancelled` e realizando um *join* na *thread* que executa a rotina `worker`.

A execução do programa teste usando `mod_control` produz o resultado medíocre mostrado na Figura 6.

```
Setting up control...
control setup
control produced 8925.59 Monetary Units
```

Figura 6 – Resultado do programa teste com `mod_control`

2.1 Compilação

A implementação é compilada usando um Makefile, como mostrado na Figura 7(a). A única biblioteca utilizada é a *pthread* (linha 3). Os módulos são ligados estaticamente ao programa teste. O alvo `seq` (linha 10) gera um binário que combina `main.c`, `pedrolio.c` e `mod_control.c`.

Para testar um novo módulo hipotético `grupoX`, é necessário compilar um novo binário substituindo `mod_control.c` por `mod_grupoX.c`. Em termos do Makefile, isso é feito adicionando um novo alvo (*target*) como mostrado nas linhas 12-13 da Figura 7(b). Lembre também de adicionar o novo alvo como dependência dos alvos *all* (linha 15) e *clean* (linha 20).

3 Instruções específicas para implementação

Você deve implementar um módulo para substituir `mod_control`, que produza **no mínimo 68567 unidades monetárias** para 8 UPPs (`#define PROCESSORS_COUNT 8` em `pedrolio.h`). Soluções com resultado inferior serão avaliadas, mas serão remuneradas **com no máximo nota 2 (a implementação tem peso 4)**, de acordo com a qualidade da solução e domínio do conteúdo da disciplina. Além disso, as seguintes restrições devem ser respeitadas:

```
CC=gcc
CFLAGS=-I.
LIBS=-lpthread
DEPS = module.h pedrolio.h
OBJ = main.o pedrolio.o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

control: $(OBJ) mod_control/mod_control.o
	gcc -o $@ $^ $(CFLAGS) $(LIBS)

all: control

PHONY: clean

clean:
	rm -f *.o */*.o control
```

(a)

```
CC=gcc
CFLAGS=-I.
LIBS=-lpthread
DEPS = module.h pedrolio.h
OBJ = main.o pedrolio.o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

control: $(OBJ) mod_control/mod_control.o
	gcc -o $@ $^ $(CFLAGS) $(LIBS)
grupoX: $(OBJ) mod_grupoX/mod_grupoX.o
	gcc -o $@ $^ $(CFLAGS) $(LIBS)

all: control grupoX

PHONY: clean

clean:
	rm -f *.o */*.o control grupoX
```

(b)

Figura 7 – Makefile original e Makefile com um novo módulo, `mod_grupoX`.

- Não altere os seguintes arquivos: `main.c`, `pedrolio.h`, `pedrolio.c` e `module.h`;
- Utilize a *threads* através da biblioteca *pthread* para explorar paralelismo;
- O módulo implementado deve ter como nome `mod_grupoX`, onde X é a letra/número do grupo;
- `make grupoX && ./grupoX` deve compilar o novo módulo e executar o programa teste (veja a Subseção 2.1);
- Em um dado momento pode haver:
  - No máximo uma execução da rotina `pd_read()`;
  - No máximo uma execução da rotina `pd_deliver()`;
  - No máximo `PROCESSORS_COUNT` execuções da rotina `pd_process()`;
- `mod_shutdown` deve liberar todos recursos alocados, inclusive memória;
- Não trapaceie. Conta como trapaça:
  - Bloquear em `mod_setup` para processar pedras sem ser contabilizado;
  - Suspender a thread principal, que contabiliza o tempo;
  - Invocar `pd_deliver` mais de uma vez com o mesmo `oil_t`;
  - Fabricar instâncias de `oil_t` por outro método que não seja `pd_process`;
  - Reutilizar instâncias de `rock_t`, ou as fabricar por outro meio que não `pd_read`;
  - Essa lista não é exaustiva, casos omissos serão considerados pelo avaliador.
- Transgressões dessas restrições resultarão em desconto de pontos.

Também será avaliada a presença de *bugs* no código, boas práticas de programação e o uso correto dos conceitos vistos na disciplina.

4 Relatório

Além da implementação, entregue um relatório respondendo as seguintes perguntas:

1. Argumente como as restrições de acesso concorrente à `pd_read()`, `pd_process()` e `pd_deliver()` são respeitadas na sua implementação.
2. Explique por que a implementação original tinha um resultado monetário tão baixo;
3. Explique o que permite que a sua implementação apresente um resultado monetário melhor que a implementação original;
4. A estratégia adotada na sua implementação é a melhor possível para esse problema? Quais seriam as características da melhor solução possível? Quão próximo sua implementação está dessa solução? Considere que os parâmetros como peso das pedras, tempos do pedromágico e das Unidades de Processamento de Pedras, e a taxa de conversão das pedras em petróleo são fixos, como observados no programa de teste [1]. Os seguintes termos são dicas: ociosidade, eficiência, bloqueio.

O relatório deverá ser submetido via moodle no formato PDF incluindo os nomes dos integrantes do grupo. O código fonte da implementação deve ser submetido de maneira compilável (Subseção 2.1 e não como parte do PDF. Não é necessário apresentar provas formais ou explicar a totalidade do código.

O relatório não deve possuir mais que 2400 palavras (aprox. 6 páginas com fonte Times tamanho 12). Não há requisito específico quanto à formatação. Dica: não se alongue desnecessariamente, é possível responder completamente as quatro perguntas em menos até 3 páginas.

5 Avaliação

Cada componente do trabalho tem o seguinte peso:

1. Implementação: 4.0
2. Relatório: 3.0
    a) Restrições: 0.75
    b) Implementação original: 0.75
    c) Nova implementação: 0.75
    d) Melhor possível: 0.75
3. Apresentação (nota individual): 3.0

A nota máxima do trabalho é 10.0. O trabalho deve ser realizado em grupos de até duas pessoas. Em caso de cópia entre grupos, o componente do trabalho onde houve cópia terá a nota dividida entre todos os grupos envolvidos.

[1] Esses parâmetros estão *hard-coded* em `pedrolio.c`