UC Sistemas Operacionais
ICT/UNIFESP

Prof. Bruno Kimura
bruno.kimura@unifesp.br
05/06/2018

LAB 3: Deadlock

Metodologia: Trabalho individual ou em grupo de no máximo 3 (três) alunos a ser desenvolvido em laboratório de informática através de codificação na linguagem C.

Data de entrega: 14/06/18

Forma de entrega: Código .c deve ser enviado no SEAD. Insira como comentário no código o nome e matrícula de cada integrante do grupo.

Observação: Somente serão aceitos trabalhos **autênticos**. Cópias (entre grupos e/ou de fontes da Internet) serão anuladas.

Descrição:

A espera circular é uma condição para ocorrência de *deadlocks*. Um *deadlock* ocorre se existir um encadeamento circular de dois ou mais processos/threads, em que cada um deles encontra-se à espera de um recurso que está sendo usado pelo membro seguinte dessa cadeia. Conforme Seção 3.4.1 (*Detecção de deadlocks com um recurso de cada tipo*) da bibliografia básica (A. Tanembaum), a construção de um grafo de recursos possibilita identificar *deadlocks*, conforme ilustra da Figura 1. Um *deadlock* existe se o grafo contiver um ou mais ciclos. A abordagem de identificação de ciclos em grafos dirigidos pode ser implementada, por exemplo, através de uma estrutura de dados do tipo lista.

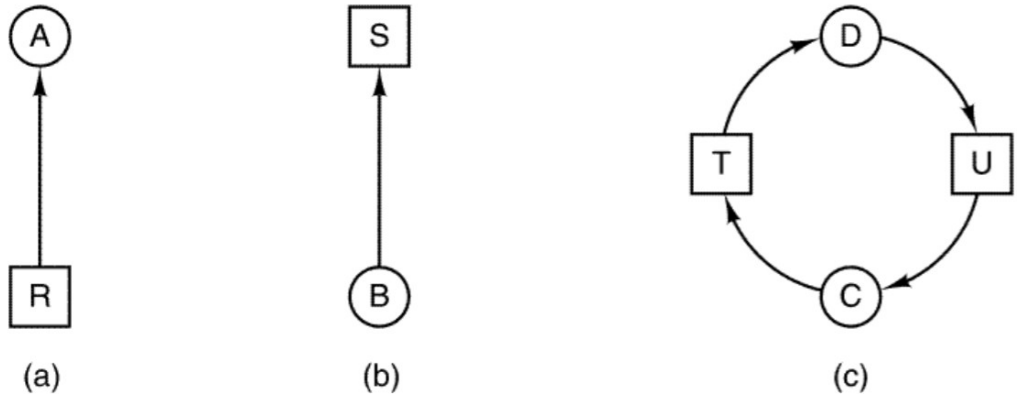

Figura 1: (a) Recurso R alocado exclusivamente ao processo A. (b) Processo B solicita acesso exclusivo ao recurso S. (c) Espera circular onde os processos D e C estão em *deadlock* sobre os recursos U e T.

Neste exercício, elabore um mecanismo de detecção de *deadlock* e o implemente na Linguagem C. Esse mecanismo deve ser transparente à aplicação. Isso requer que sua implementação ocorra dentro das funções que manipulam semáforos, como `sem_wait()` e `sem_post()`. Uma vez que um semáforo *s* determina o acesso a um recurso *r*, o par (*r*, *s*) é inseparável. Como um recurso pode ser algo abstrato, assuma que o semáforo *s* é o identificador do recurso *r*. Para que a operação do

mecanismo seja transparente em tempo de execução, é necessário re-implementar as funções de interesse, `sem_wait()` e `sem_post()`, com o devido suporte interno à detecção de *deadlock* e, então, realizar a sobreposição das funções antigas (sem suporte) pelas novas (com o suporte). Para tal, o mecanismo pode ser implementado de forma não-intrusiva através da técnica de sobreposição de chamadas com LD_PRELOAD, conforme o exemplo abaixo.

Assim, diferente das funções originais, as novas funções que implementam detecção de *deadlocks* podem retornar com erro (específico de *deadlock*) em vez de bloquearem no semáforo. Como exemplo, considere o código abaixo, *my_semaphore.c*. Nele há um exemplo de re-implementação da função `sem_wait()`. Observe que o ponteiro para função `_sem_wait` aponta para a função original `sem_wait()`. Assim, dentro da nova função `sem_wait()`, tem-se a liberdade de implementar o que quiser.

(***my_semaphore.c***)

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <dlfcn.h>

int (*_sem_wait)(sem_t *) = NULL;

int sem_wait(sem_t *sem) {
        int r;
        if (!_sem_wait) {
                _sem_wait = dlsym(RTLD_NEXT, "sem_wait");
                        /* Irá apontar para o sem_wait original*/
        }
        printf("\t Dentro da sem_wait()... faça o que quiser aqui!\n");
        r = _sem_wait(sem);
        return(r);
}
```

Considere uma aplicação abaixo, com exemplo de aplicação que faz uso de semáforo.

(***aplicacao.c***)

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>

int main(void) {
        sem_t mutex;
        sem_init(&mutex, 0, 1);
        sem_wait(&mutex);
        printf("\t Na main da aplicacao depois do sem_wait()\n");
        return 0;
}
```

Para compilar no terminal, siga os comandos abaixo:

(***terminal***)

```
terminal:~$ gcc -Wall -o aplicacao aplicacao.c -lpthread
terminal:~$ gcc -Wall -shared -o my_semaphore.so my_semaphore.c -ldl -fPIC
terminal:~$ LD_PRELOAD=./my_semaphore.so ./aplicacao
```

Por fim, para desenvolver está prática, considere as funções de interesse da biblioteca `semaphore.h` abaixo.

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
       /* Inicializa o semáforo apontado em sem, indicando seu valor inicial
       especificado em value. O argumento pshared indica se o semáforo será
       compartilhados entre threads de um processos (pshared=0) ou entre
       diferentes processos (pshared<>0)*/

int sem_wait(sem_t *sem);
       /* Decrementa (lock) o semáforo apontado em sem. Se valor do semáforo
       for maior que zero, decrementa. Se igual a zero, bloqueia quem chamou
       sem_wait(). */

int sem_post(sem_t *sem);
       /* Incrementa (unlock) semáforo apontado em sem. Se valor do semáforo
       for zero, ao incrementar, desbloqueia outras threads/processos que
       estão bloqueados em sem_wait(). */

int sem_getvalue(sem_t *sem, int *sval);
       /* Copia o valor atual do semáforo apontado em sem para o inteiro
       apontado em sval. Chamada não é bloqueante. */
```