Programação Concorrente – Aula 8

Gilmar Gimenes Rodrigues

1 Semáforos

Tudo que vimos tem buzy-waiting!

- Algoritmo de Dekker
- Algoritmo de Peterson(Tie Breaker)
- Laço de busy-waiting usando test and set
- Algoritmo de Ticket
- Algoritmo de Bakery

Dijkstra (~1965)
Tipos de dados abstrados
Encapsula um inteiro, que é o número de unidades disponíveis do recurso controlado pelo semáforo.
Outra maneira de pansar: Quantas threads ainda podem "passar".

1.1 Duas operações.

Elas podem ser:

P, ou down ou wait ou acquire:
V, ou up ou signal ou release

```
P:
    P(s) equivale <espera s> 0; s = s - 1;>

V:
    V(s) equivale <s = s + 1;>

P(s) {
    entra_RC();
    while(s <= 0) {
        sai_RC();
        entra_RC();
    }
    s = s -1;
    sai_RC();
}

V(s) {
    entra_RC();
    s = s + 1;
```

```
    sai_RC();
}
```

Oi Professor. Falta um desenho aqui. To fazendo.....
Semáforo s tem um contador "conta" e uma fila de threads bloqueadas "fila".

```
P(s) {
    entra_RC;  // Para acessar "conta" e "fila"
    if(s.conta > 0) {
        s.conta--;
        sai_RC;
    }
    else {
        coloca thread atual em s.fila;
        sai_RC();
        tira a thread do estado "em execução";
    }
}

V(s) {
    entra_RC();
    if(s.filaestá vazia) {
       s.conta++;
    }
    else {
        tira proxima thread de s.fila;
        coloca na fila de threads prontas para execução;
    }
    sai_RC();
}
```

Solução para o problema da região crítica usando semáforo.

```
semaforo mutex = 1; //O contador do semáforo começa com 1.

thread_i() {
    for(;;) {
        P(mutex);
        regiao_critica();
        V(mutex);
        regiao_nao_critica();
    }
}
```

Problema do Produtor - Consumidor com semáforos.

```
1 thread produtora
1 thread consumidora

tipo buf;

Semaforo buf_vazio = 1;
Semaforo buf_cheio = 0;

produtor() {
    for(;;) {
        produz_informacao();
        P(buf_vazio);
        coloca_informacao_no_buffer();
        V(buf_cheio);
    }
}

consumidor() {
    for(;;) {
        P(buf_cheio);
        coloca_informacao_nobuffer();
        V(buf_cheio);
    }
}
```

Produtor - Consumidor (Problema do bounded buffer

```
tipo buf[N];
int inicio;
int final;

semaforo espaco = N;
         item = 0;

produtor() {
    for(;;) {
        produz_dado();
        P(espaco);
        buf[final] = dado;
        final = (final + 1) % N;
        V(item);
    }
}

consumidor() {
    for(;;) {
        P(item);
        dado = buf[inicio];
        inicio = (inicio + 1)% N; // Pode haver mais de um consumidor
                                  // querendo fazer isso.
        V(espaco);
```

```
        processa_dado();
    }
}


tipo buf[N];
int inicio;
int final;

semaforo espaco = N;
semaforo item = 0;

semaforo mutex_final = 1;
semaforo mutex_inicio = 1;

produtor() {
    for(;;) {
        produz_dado();
        P(espaco);
        P(mutex_final);
        buf[final] = dado;
        final = (final + 1) % N;
        V(mutex_final);
        V(item);
    }
}

consumidor() {
    for(;;) {
        P(item);
        P(mutex_inicio);
        dado = buf[inicio];
        inicio = (inicio + 1)% N;
        V(mutex_inicio);
        V(espaco);
        processa_dado();
    }
}
```

E se usarmos um semáforo para o buffer:

```
semaforo mutex_buf = 1;

produtor() {
    for(;;) {
        produz_dado();
        P(mutex_buf);
        P(espaco);
        buf[final] = dado;
        final = (final + 1) % N;
        V(item);
```

```
        V(mutex_buf);
    }
}

consumidor() {
    for(;;) {
        P(mutex_buf);
        P(item);
        dado = buf[inicio];
        inicio = (inicio + 1)% N;
        V(espaco);
        V(mutex_buf);
        processa_dado();
    }
}
```