Pedro Henrique Marra Araújo (12011BCC008)

Jantar dos Filósofos

Uberlândia, Minas Gerais
2022

Sumário

Repositório

Todos os códigos foram disponibilizados em um repositório no GitHub, que pode ser acessado clicando aqui.

(1) Primeira Implementação

A implementação da questão 1, sem usar qualquer mecanismo de exclusão mútua, está em `main1.c`. Para rodá-lo, execute os seguintes comandos: `gcc main1.c -l pthread` e `./a.out`. As partes importantes do código (digo, a resolução lógica do problema, excluindo a criação das estruturas, o *set* dos valores iniciais, etc) são:

```
// OBS: retirei os comentários para ficar menor, ele
// todo comentado pode ser encontrado em main1.c

int come(int* filosofo, int* ge, int* gd){
    if(*filosofo == COMENDO) return -1;

    int r = 1 + (rand() \% TEMPO);
    int ge_anterior = *ge;
    int gd_anterior = *gd;

    if(ge_anterior == DESOCUPADO){
        *ge = OCUPADO;
        printf("Garfo esquerdo foi pego.\n");
    }
    else{
        printf("Garfo esquerdo ocupado.\n");
        return 0;
    }

    if(gd_anterior == DESOCUPADO){
        *ge = OCUPADO;
        printf("Garfo direito foi pego.\n");
    }
    else{
        printf("Garfo direito ocupado.\n");
```

```
        return 0;
    }

    *filosofo = COMENDO;

    sleep(r);

    return 1;
}

int pensa(int* filosofo, int* ge, int* gd){
    if(*filosofo == PENSANDO) return -1;

    int r = 1 + (rand() %% TEMPO); // Pensa por [1,TEMPO] s

    *ge = DESOCUPADO;
    printf("Garfo esquerdo foi colocado na mesa.\n");

    *filosofo = PENSANDO;

    *gd = DESOCUPADO;
    printf("Garfo direito foi colocado na mesa.\n");

    sleep(r);

    return 1;
}
```

Pode-se verificar o problema da inconsistência no uso dos recursos compartilhados, pois, ao executar esse código, ele, infinitamente ([...]), imprimirá algo parecido com o seguinte trecho:

```
Filosofo 2 juntou-se à mesa!

Garfo esquerdo foi pego.
Garfo direito foi pego.
Filosofo 3 juntou-se à mesa!
```

```
Garfo esquerdo foi pego.
Garfo direito foi pego.
Filosofo 1 juntou-se à mesa!

Garfo esquerdo foi pego.
Garfo direito ocupado.
Filósofo 1 não pôde comer.
Filósofo 1 pensou.

Garfo esquerdo ocupado.
Filósofo 1 não pôde comer.
Filósofo 1 pensou.

Garfo esquerdo ocupado.
Filósofo 1 não pôde comer.
Filósofo 1 pensou.

Garfo esquerdo ocupado.
Filósofo 1 não pôde comer.
Filósofo 1 pensou.

Filosofo 5 juntou-se à mesa!

Garfo esquerdo foi pego.
Garfo direito ocupado.
Filósofo 5 não pôde comer.
Filósofo 5 pensou.

Garfo esquerdo ocupado.
Filósofo 5 não pôde comer.
Filósofo 5 pensou.

Garfo esquerdo ocupado.
Filósofo 5 não pôde comer.
Filósofo 5 pensou.

Garfo esquerdo ocupado.
Filósofo 5 não pôde comer.
```

```
Filósofo 5 pensou.

[...]
```

Ou seja, em algum momento nenhum filósofo poderá comer, pois os garfos estarão nas mãos dos outros filósofos de maneira que nenhum come (ficando com um garfo na mão e todos somente pensando eternamente).

(2) Segunda Implementação

A implementação da questão 2, usando exclusão mútua, está em `main2.c`. Para rodá-lo, execute os seguintes comandos: `gcc main2.c -l pthread` e `./a.out`. As partes importantes do código (digo, a resolução lógica do problema, excluindo a criação das estruturas, o *set* dos valores iniciais, etc) são:

```
// OBS: retirei os comentários para ficar menor, ele
// todo comentado pode ser encontrado em main2.c

void* jantar(void* f_numero){
    int f_n = *(int*) f_numero;
    printf("Filosofo \%d juntou-se à mesa!\n\n", f_n+1);
    int *filosofo = &filosofos[f_n];
    *filosofo = PENSANDO;
    struct sem_garfo *ge = &garfos[f_n], *gd = &garfos[(f_n+1)
    ↪  \% QTD_FIL];
    int comeu, pensou;

    while(TRUE){
        if((comeu = come(filosofo, ge, gd)) == 0)
            printf("Filósofo \%d não pôde comer.\n", f_n+1);
        else if(comeu == 1) printf("Filósofo \%d comeu.\n\n",
        ↪  f_n+1);

        printf("Filósofo \%d pensou.\n\n", f_n+1);
    }
}
```

```
int come(int* filosofo, struct sem_garfo* ge, struct sem_garfo*
↪  gd){
    if(*filosofo == COMENDO) return -1;

    int r = 1 + (rand() \% TEMPO);

    sem_wait(&ge->sem);
    int ge_anterior = ge->garfo;
    if(ge_anterior == DESOCUPADO){
        ge->garfo = OCUPADO;
        printf("Garfo esquerdo foi pego.\n");
    }
    else{
        printf("Garfo esquerdo ocupado.\n");
        return 0;
    }
    sem_post(&ge->sem);

    sem_wait(&gd->sem);
    int gd_anterior = gd->garfo;
    if(gd_anterior == DESOCUPADO){
        gd->garfo = OCUPADO;
        printf("Garfo direito foi pego.\n");
    }
    else{
        printf("Garfo direito ocupado.\n");
        sem_post(&gd->sem);
        return 0;
    }
    sem_post(&gd->sem);

    *filosofo = COMENDO;

    sleep(r);

    return 1;
}
```

```
int pensa(int* filosofo, struct sem_garfo* ge, struct
↪  sem_garfo* gd){
    if(*filosofo == PENSANDO) return -1;

    int r = 1 + (rand() \% TEMPO);

    sem_wait(&ge->sem);
    ge->garfo = DESOCUPADO;
    printf("Garfo esquerdo foi colocado na mesa.\n");
    sem_post(&ge->sem);

    *filosofo = PENSANDO;

    sem_wait(&gd->sem);
    ge->garfo = DESOCUPADO;
    printf("Garfo direito foi colocado na mesa.\n");
    sem_post(&gd->sem);

    sleep(r);

    return 1;
}
```

Pode-se verificar o problema do *deadlock*, pois, ao executar esse código, ele, em algum momento (relativamente rápido) parará de imprimir as atualizações:

```
Filosofo 1 juntou-se à mesa!

Garfo esquerdo ocupado.
Filósofo 1 não pôde comer.
Filósofo 1 pensou.

Filosofo 2 juntou-se à mesa!

Filosofo 5 juntou-se à mesa!

Garfo esquerdo ocupado.
```

```
Filósofo 5 não pôde comer.
Filósofo 5 pensou.

Filosofo 3 juntou-se à mesa!

Garfo esquerdo ocupado.
Filósofo 3 não pôde comer.
Filósofo 3 pensou.

Filosofo 4 juntou-se à mesa!

Garfo esquerdo ocupado.
Filósofo 4 não pôde comer.
Filósofo 4 pensou.

Garfo esquerdo ocupado.
Filósofo 2 não pôde comer.
Filósofo 2 pensou.
```

(3) Terceira Implementação

A implementação da questão 3, aproveitando a lógica do minha implementação anterior e utilizando a solução do Tanenbaum (usando exclusão mútua para os filósofos — isto é, cada filósofo tem um semáforo, ou seja, cada filósofo é uma `struct sem_filosofo` — e exclusão mútua para a mesa — com um semáforo `sem_t mutex`), está em `main3.c`. Para rodá-lo, execute os seguintes comandos: `gcc main3.c -l pthread` e `./a.out`. As partes importantes do código (digo, a resolução lógica do problema, excluindo a criação das estruturas, o *set* dos valores iniciais, etc) são:

```c
// OBS: retirei os comentários para ficar menor, ele
// todo comentado pode ser encontrado em main3.c

void* jantar(void* f_numero){
    int f_n = *(int*) f_numero;
    printf("Filosofo \%d juntou-se à mesa!\n\n", f_n);
```

```
    while(TRUE){
        pensa(f_n);
        pega_garfos(f_n);
        come(f_n);
        larga_garfos(f_n);
    }
}

void seta_infos(int f_n, struct sem_filosofo** filosofo, struct
↪  sem_filosofo** fe, struct sem_filosofo** fd, int** ge,
↪  int** gd){
    *filosofo = &filosofos[f_n];
    *ge = &garfos[f_n], *gd = &garfos[(f_n+1) \% QTD_FIL];
    *fe = NULL, *fd = NULL;

    if(f_n == 0) *fe = &filosofos[QTD_FIL-1];
    if(f_n == QTD_FIL-1) *fd = &filosofos[0];
    if(*fe == NULL) *fe = &filosofos[f_n-1];
    if(*fd == NULL) *fd = &filosofos[f_n+1];
}

void teste(int f_n){
    struct sem_filosofo *filosofo, *fe, *fd;
    int *ge, *gd;
    seta_infos(f_n, &filosofo, &fe, &fd, &ge, &gd);

    if(filosofo->estado == FOME && fe->estado != COMENDO &&
    ↪  fd->estado != COMENDO){
        filosofo->estado = COMENDO;
        *ge = OCUPADO;
        *gd = OCUPADO;
        sem_post(&filosofo->sem);
    }
}

void pega_garfos(int f_n){
    sem_wait(&mutex);
```

```
    struct sem_filosofo *filosofo, *fe, *fd;
    int *ge, *gd;
    seta_infos(f_n, &filosofo, &fe, &fd, &ge, &gd);

    filosofo->estado = FOME;
    teste(f_n);
    sem_post(&mutex);
    sem_wait(&filosofo->sem);
}

int f_esq(int f_n){
    if(f_n == 0) return QTD_FIL-1;
    else return (f_n - 1);
}

int f_dir(int f_n){
    if(f_n == QTD_FIL-1) return 0;
    else return (f_n + 1);
}

void larga_garfos(int f_n){
    sem_wait(&mutex);

    struct sem_filosofo *filosofo, *fe, *fd;
    int *ge, *gd;
    seta_infos(f_n, &filosofo, &fe, &fd, &ge, &gd);

    filosofo->estado = PENSANDO;
    *ge = DESOCUPADO;
    *gd = DESOCUPADO;
    teste(f_esq(f_n));
    teste(f_dir(f_n));

    sem_post(&mutex);
}

void come(int f_n){
    int r = 1 + (rand() \% TEMPO);
```

```
    printf("Filosofo \%d comeu.\n\n", f_n+1);

    sleep(r);
}

void pensa(int f_n){
    int r = 1 + (rand() \% TEMPO);

    printf("Filosofo \%d pensou.\n\n", f_n+1);

    sleep(r);
}
```

Percebe-se, pelo trecho abaixo, que agora não temos mais o *deadlock*, pois a exclusão mútua foi implementada corretamente. Assim, os filósofos comerão e pensarão indefinitivamente ([...]), sem nunca ficarem em *starvation*:

```
Filosofo 1 juntou-se à mesa!

Filosofo 1 pensou.

Filosofo 2 juntou-se à mesa!

Filosofo 5 juntou-se à mesa!

Filosofo 2 pensou.

Filosofo 3 juntou-se à mesa!

Filosofo 5 pensou.

Filosofo 3 pensou.

Filosofo 4 juntou-se à mesa!

Filosofo 4 pensou.
```

```
Filosofo 2 comeu.

Filosofo 4 comeu.

Filosofo 1 comeu.

Filosofo 5 comeu.

Filosofo 3 comeu.

Filosofo 4 pensou.

Filosofo 1 pensou.

Filosofo 2 pensou.

Filosofo 5 pensou.

Filosofo 3 pensou.

Filosofo 2 comeu.

Filosofo 4 comeu.

Filosofo 5 comeu.

Filosofo 2 pensou.

Filosofo 3 comeu.

Filosofo 1 comeu.

Filosofo 4 pensou.

[...]
```