RESOLUÇÃO DO CONCURSO 4

SISTEMAS OPERATIVOS 2024/2025

Feito por:
a83933 Rodrigo Linhas
2º Ano da Licenciatura de Engenharia Informática (LEI)
Regente da UC: Amine Berqia

Índice

INTRODUÇÃO

Este trabalho tem o intuito de demonstrar a minha resolução perante os exercícios propostos no Concurso 4, juntamente com a(s) referencia(s) bibliográfica(s) consultada(s) no final do documento.

As soluções foram desenvolvidas em linguagem C, testadas em ambiente Linux e validadas com capturas de ecrã para garantir conformidade com os requisitos. Os códigos usam semáforos POSIX e mutexes para sincronização de processos e threads, garantindo exclusão mútua e evitando condições de corrida, deadlock e starvation, que seram explicados com mais detalhe.

A estrutura do documento segue a numeração dos exercícios, com explicações concisas, exemplos de código e saídas geradas.

Palavras chave: semaforos, deadlock, starvation, mutex

EXERCÍCIO I: PRODUTOR-CONSUMIDOR COM SEMÁFOROS

a) Explicação e Constrangimentos

Após ter compilado o programa dado pelo enunciado obtivemos o seguinte output:

```
rodrigo@rodrigo-Aspire-XC600:~/Desktop/uni/SO/C4$ gcc ex1.c -o ex1.exe
rodrigo@rodrigo-Aspire-XC600:~/Desktop/uni/SO/C4$ ./ex1.exe
Proc. C.
Object from buffer.
Cons. objet.0
Proc P.
Production .0
Object to buffer.
Production .1
Object to buffer.
Object from buffer.
Cons. objet.-1
Production .2
Object to buffer.
Production .3
Object to buffer.
Object from buffer.
Cons. objet.-2
Production .4
Object to buffer.
Production .5
Object to buffer.
^C
rodrigo@rodrigo-Aspire-XC600:~/Desktop/uni/SO/C4$
```

tive que parar a execução do mesmo uma vez que contem ciclos infinitos, mas dá para ter uma ideia da execução do programa. Também tenho que salientar que **deixei o programa a executar por aproximadamente 1 minuto (60 segundos).** Esta métrica será necessária para fins comparativos**.**

O problema central é a falta de partilha do *i* e a a ausência de mecanismos de sincronização. Sem, isso o *buffer* não é gerência permitido condições, em inválidas (ex.: consumidor removendo de buffer vazio).

//introduzir uma explicação dos semáforos (links consultados obviamente)

Com tudo que revimos anteriormente, temos a seguinte estratégia para a continuação do exercício:
• Buffer circular com tamanho N = 10;
• Semáforo vazio (sem_empty) inicializado a N representa espaços livres;
• Semáforo full (sem_full) inicializado a 0 representa itens disponíveis;
• Mutex (sem_mutex) inicializado a 1 garante que produtor e consumidor não acedam ao buffer simultaneamente;
• Produtor: antes de inserir, faz sem_wait(&sem_empty); sem_wait(&sem_mutex); insere no buffer; sem_post(&sem_mutex); sem_post(&sem_full);
• Consumidor: antes de remover, faz sem_wait(&sem_full); sem_wait(&sem_mutex); remove do buffer; sem_post(&sem_mutex); sem_post(&sem_empty).

b) Código em C

```
rodrigo@rodrigo-Aspire-XC600:~/Desktop/uni/SO/C4$ gcc ex1-resolvido.c -o ex1-resolvido.exe
rodrigo@rodrigo-Aspire-XC600:~/Desktop/uni/SO/C4$ ./ex1-resolvido.exe
Produção: 0 no buffer[0]
Consumo: 0 do buffer[0]
Produção: 1 no buffer[1]
Consumo: 1 do buffer[1]
Produção: 2 no buffer[2]
Produção: 3 no buffer[3]
Consumo: 2 do buffer[2]
Produção: 4 no buffer[4]
Produção: 5 no buffer[5]
Produção: 6 no buffer[6]
Consumo: 3 do buffer[3]
Produção: 7 no buffer[7]
Produção: 8 no buffer[8]
Consumo: 4 do buffer[4]
Produção: 9 no buffer[9]
Produção: 10 no buffer[0]
Consumo: 5 do buffer[5]
Produção: 11 no buffer[1]
Produção: 12 no buffer[2]
Consumo: 6 do buffer[6]
Produção: 13 no buffer[3]
Consumo: 7 do buffer[7]
Produção: 14 no buffer[4]
Produção: 15 no buffer[5]
Consumo: 8 do buffer[8]
Produção: 16 no buffer[6]
Produção: 17 no buffer[7]
Consumo: 9 do buffer[9]
Produção: 18 no buffer[8]
Produção: 19 no buffer[9]
Consumo: 10 do buffer[0]
Produção: 20 no buffer[0]
Consumo: 11 do buffer[1]
Produção: 21 no buffer[1]
Consumo: 12 do buffer[2]
Produção: 22 no buffer[2]
Consumo: 13 do buffer[3]
Produção: 23 no buffer[3]
Consumo: 14 do buffer[4]
Produção: 24 no buffer[4]
Consumo: 15 do buffer[5]
Produção: 25 no buffer[5]
Consumo: 16 do buffer[6]
Produção: 26 no buffer[6]
Consumo: 17 do buffer[7]
Produção: 27 no buffer[7]
Consumo: 18 do buffer[8]
Produção: 28 no buffer[8]
Consumo: 19 do buffer[9]
```

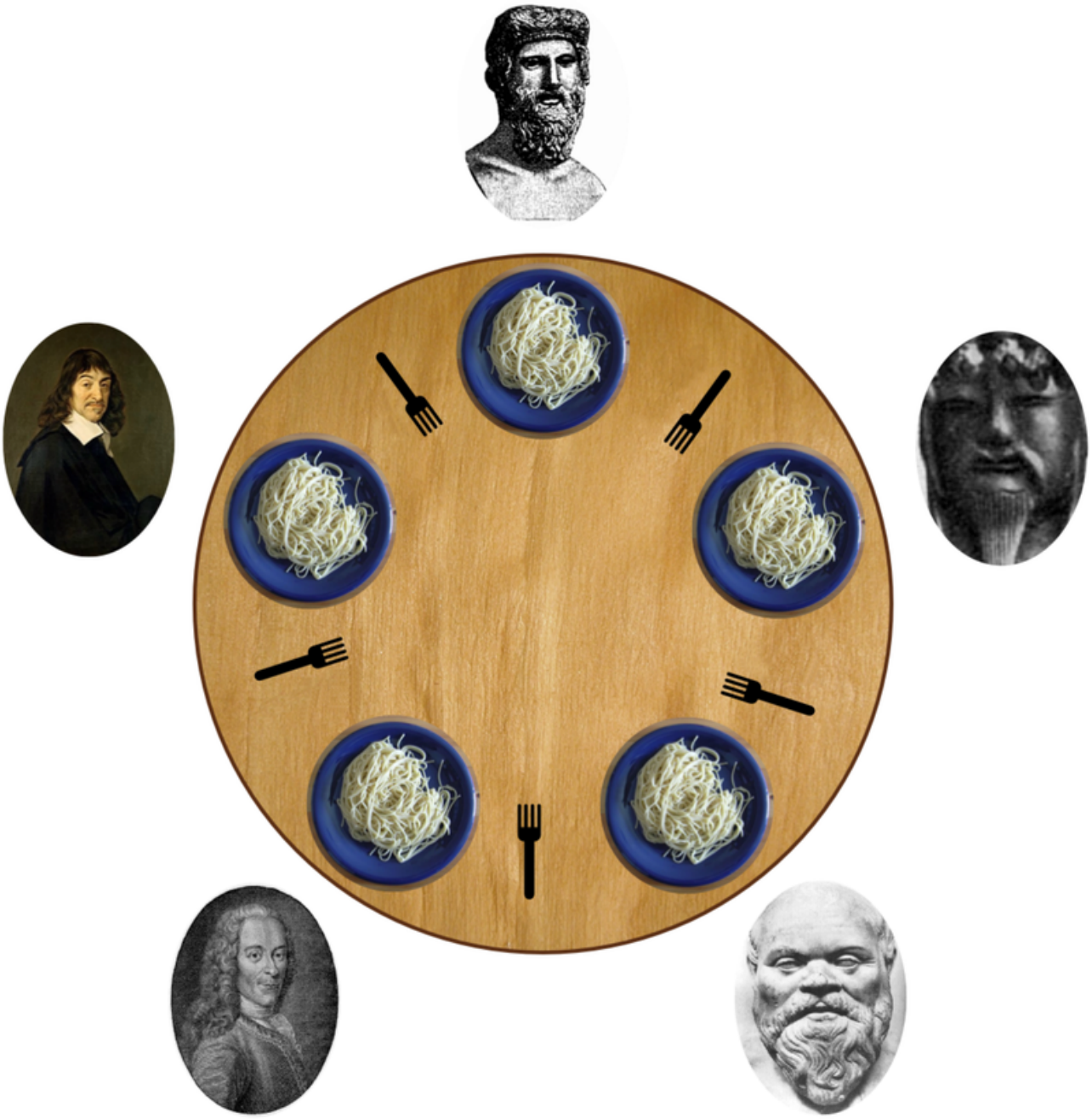
EXERCÍCIO II: JANTAR DOS FILÓSOFOS

1) Descrição do Problema

Varios filosofos sentam-se ao redor de uma mesa circular, para maior simplicidade vamos considerar apenas 5, cada um alternando entre pensar e comer. Entre cada par de filósofos há um garfo. Para comer, um filósofo precisa de dois garfos (o da sua esquerda e o da sua direita logicamente). O problema realça potencial de deadlock (todos seguram o garfo à esquerda e esperam pelo da direita) e starvation (alguns podem nunca obter dois garfos).

Resolução Concurso 4

SO 24/25



a83933 Rodrigo Linhas LEI

Regente da UC: Amine Berqia

2) Solução em C (prevenção de deadlock e starvation)

REFERÊNCIAS BIBLIOGRÁFICAS

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson.
- Man page sem_init, sem_wait, sem_post, sem_destroy.
- https://youtu.be/VSkvwzqo-Pk?si=Iz0rH6dFtnYhUDSp
- https://en.wikipedia.org/wiki/Dining_philosophers_problem