Relatório Trabalho 4 – Sistemas Operacionais

Aluno: Otávio Malta Borges

Matrícula: 12011BSI291

EX4

A) Implementar uma solução para o problema do jantar dos filósofos, sem o controle de impasse.

Primeiramente é iniciado os semáforos, um para cada garfo **(sem_init(&mutexGarfo[i],0,0),** e criado uma thread para cada filósofo **(pthread_create(&filosofos[i], NULL, filosofo, &filo[i]);)** , as quais executam a função 'filosofo', passando o id (de 0 a 4) de cada filósofo.

Após a execução do código, os semáforos de cada garfo são destruídos com sem_destroy(&mutexGarfo[i]); - i: de 0 a 4, representado o id dos garfos.

A função filosofo executa, em um loop, as funções:

pensar(*num) – recebendo o id do filósofo, pensar gerará aleatoriamente um número de 0 a 5, cujo representa o tempo o qual a função passará executando (sleep(time)), e printa estas informações na tela.

pegar(*num) – recebe o numero do filósofo, o qual representa o garfo a sua esquerda. Desta forma, o semáforo do garfo a esquerda de tal filosofo é bloqueado com sem_wait(&mutexGarfo[numGarfo])

pegar((*num+1)%5) – recebe o numero do garfo a sua direita. Desta forma, o semáforo do garfo direita de tal filosofo é bloqueado com sem_wait(&mutexGarfo[numGarfo])

comer(*num) – recebendo o id do filósofo, comer gerará aleatoriamente um número de 0 a 5, cujo representa o tempo o qual a função passará executando (sleep(time)), e printa estas informações na tela.

soltar(*num) – recebe o numero do filósofo, o qual representa o garfo a sua esquerda. Desta forma, o semáforo do garfo a esquerda de tal filosofo é liberado com sem_post(&mutexGarfo[numGarfo])

soltar((*num+1)%5) – recebe o numero do garfo a sua direita. Desta forma, o semáforo do garfo direita de tal filosofo é liberado com sem_post(&mutexGarfo[numGarfo])

Desta forma, haverá deadlock, pois os filósofos toma posse dos garfos esquerdos e ficam esperando até conseguem o direito, porém a não preempção gerará uma espera circular, onde nenhum filósofo conseguirá comer.

Resultado:

```
otavio@otavio:/mnt/c/users/otavi/one
Filósofo 0 esta pensando - 3 segs
Filósofo 2 esta pensando - 2 segs
Filósofo 1 esta pensando - 1 segs
Filósofo 4 esta pensando - 3 segs
Filósofo 3 esta pensando - 0 segs
```

Looping infinito, logo não há tempo de execução.

B) Implementar uma solução para o problema do jantar dos filósofos, de forma em que cada filósofo come um de cada vez (serialmente).

Diferente do exercício anterior, aqui há apenas um semáforo , relacionado ao filósofo que ira comer.

sem_init(&mutex, 0, 1);,

É criado as threads para os filósofos, e passado a função filosofo() e o id destes.

A função filosofo, portanto, não chamará a função pegar() e soltar() duas vezes, já que estas se referem diretamente aos dois garfos.

As funções pegar e soltar continuam as mesmas do exercício anterior,

sem_wait(&mutex)

E

sem_post(&mutex)

respectivamente. Porém, aqui é liberado e bloqueado o filósofo o qual comerá (o que seria os dois garfos).

Resultado:

```
Filósofo 0 esta pensando - 3 segundos
Filósofo 2 esta pensando - 2 segundos
Filósofo 1 esta pensando - 1 segundos
Filósofo 3 esta pensando - 0 segundos
Filósofo 4 esta pensando - 3 segundos
Filósofo 3 esta comendo - 0 segundos
Filósofo 3 esta pensando - 1 segundos
Filósofo 1 esta comendo - 2 segundos
Filósofo 1 esta pensando - 4 segundos
Filósofo 3 esta comendo - 1 segundos
Filósofo 3 esta pensando - 2 segundos
Filósofo 2 esta comendo - 2 segundos
Filósofo 2 esta pensando - 0 segundos
Filósofo 0 esta comendo - 4 segundos
^C
```

Looping infinito, logo não há tempo de execução.

Usando semáforos para o filósofo que irá comer, é garantido que não haja desdlock, já que não há recursos compartilhados simultaneamente, porém, por conta disso, também não há paralelismo.

C) Implementar uma solução para o problema do jantar dos filósofos, de forma em que haja um paralelismo dos recursos entre os filósofos.

Aqui é necessário semáforos para os filósofos e para os garfos. Além disso, é usado um vetor para cada filósofo para determinar seu estado (0 – pensando, 1 – com fome, 2 – comendo), para que seja analisado o estado dos filósofos do lado de um determinado.

Assim, é iniciado os semáforos dos filósofos (como no ex b) e dos gafos (como no ex c).

E como no exercício b, a função filosofo chama apenas pensar, pegar, comer e soltar. Todavia, pegar() desta vez, altera a variável estado(representa o estado do atual filósofo) para 1 (com fome) e chama a função verifica(), passando o número do filósofo como parâmetro.

A função verifica, analisa se o estado do atual filosofo é 1 (com fome) e se os filósofos a sua direita e esquerda são diferentes de 2(comendo) .

if((estado[numFil] == 1) && (estado[FILESQ] != 2) && (estado[FILDIR] != 2)) {

Caso seja verdadeiro, o estado deste filosofo passa a ser 2(comendo) e o semáforo dos garfos são liberados para ele.
Caso for falso, significa que os garfos estão sendo usado ou caso o filosofo não esteja com fome.

Quando volta para pegar(), o semáforo do filósofo é liberado, e dos garfos deste filósofo são bloqueados.

sem_post(&mutex);
sem_wait(&mutexGarfo[numFil]);

Quando é chamado a função soltar() posteriormente, é bloqueado o semáforo dos filósofos, o estado é alterado para 0(pensando) e é chamado a função verifica para os filósofos a direita e esquerda do atual, testando assim se estes estão com fome. Logo é liberado o semaforo do filosofo.

Ao criar verificar o estado do filosofo, e usar semáforos para controlar os garfos, é garantido que não haja deadlock e que ainda sim ocorra paralelismo entreeles. Isso pois, ao usar o semaforos para acessar as zonas críticas dos garfos e dos filósofos, é garantido a exclusão múta, desabilitando e habilitando interrupções dos recursos compartilhados.

Resultado:

```
O filosofo 1 não conseguiu pegar os garfos
O filosofo 1 não conseguiu pegar os garfos
O filosofo 4 não conseguiu pegar os garfos
Filósofo 4 esta pensando - 3 segundos
Filósofo 2 esta comendo - 1 segundos
Filósofo 1 esta pensando - 0 segundos
Filósofo 0 esta comendo - 3 segundos
O filosofo 2 não conseguiu pegar os garfos
O filosofo 4 não conseguiu pegar os garfos
O filosofo 2 não conseguiu pegar os garfos
Filósofo 2 esta pensando - 4 segundos
Filósofo 3 esta comendo - 0 segundos
O filosofo 5 não conseguiu pegar os garfos
O filosofo 3 não conseguiu pegar os garfos
Filósofo 3 esta pensando - 4 segundos
O filosofo 5 não conseguiu pegar os garfos
O filosofo 0 não conseguiu pegar os garfos
Filósofo 0 esta pensando - 4 segundos
Filósofo 1 esta comendo - 4 segundos
O filosofo 3 não conseguiu pegar os garfos
Filósofo 3 esta comendo - 2 segundos
O filosofo 1 não conseguiu pegar os garfos
O filosofo 3 não conseguiu pegar os garfos
Filósofo 1 esta pensando - 1 segundos
O filosofo 5 não conseguiu pegar os garfos
Filósofo 0 esta comendo - 3 segundos
Filósofo 2 esta comendo - 2 segundos
Filósofo 3 esta pensando - 1 segundos
O filosofo 2 não conseguiu pegar os garfos
O filosofo 4 não conseguiu pegar os garfos
^C
```

Looping infinito, logo não há tempo de execução.